Visualização sistemática de programas

Maria João Varanda Pereira [a] , Pedro Rangel Henriques [a]

[a] *gEPL / Dep. de Informática*
Universidade do Minho – Braga, Portugal
mjoao@ipb.pt, prh@di.uminho.pt

Resumo

Este artigo surge no âmbito de desenvolvimento do sistema Alma — sistema de animação de programas escritos em diferentes linguagens.

Irá ser feita uma breve apresentação da arquitectura do sistema, focando com mais detalhe os aspectos que permitem garantir o carácter genérico e versátil do sistema; irão ser apresentados alguns exemplos de programas, sua representação interna e visualizações geradas a partir dessas representações internas.

O estudo destes exemplos tem por objectivo encontrar as visualizações adequadas, para cada tipo de linguagem, tentando na medida do possível homogeneizar os desenhos gerados sem perder contudo a expressividade necessária à compreensão dos programas.

A escolha da representaçao visual a usar não está apenas relacionada com o tipo de linguagem mas também com as diferentes vistas sobre variáveis, controlo de fluxo e de dados que devem ser postos à disposição do utilizador. O sistema deve permitir que o utilizador escolha não só o programa que quer animar mas também o tipo de visualizações que necessita. Faz parte então da arquitectura do sistema uma interface que permite programar facilmente a animação de qualquer programa.

1 Introdução

Os sistemas de animação de programas construídos até ao momento demonstraram um elevado grau de especificidade, sendo normalmente dependentes de linguagens e muitos deles associados a determinados algoritmos. Alguns sistemas criavam a animação com base em anotações o que implicava alterar o código fonte. Neste artigo irá ser apresentada 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*

⋆ O trabalho da M. João é suportado parcialmente pelo programa PRODEP, acção 5.2 da medida 5 - doutoramentos

durante uma simulação da sua execução. Este sistema pretende demonstrar 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, recorre-se 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.

A generalidade e versatibilidade pretendida é conseguida usando, como representação interna dos programas, uma ASAD (Árvore de Sintaxe Abstrata Decorada) e um conjunto de ferramentas de animação independentes das linguagens utilizadas nesses programas e que tiram partido da ASAD. Um *Tree walker animator* efectua uma travessia na árvore e vai criando representações visuais para cada nodo, aglomerando os desenhos de forma a obter no final um retrato do programa num determinado instante. O processo repete-se sempre que uma regra de reescrita é aplicada à árvore, obtendo-se assim a animação (através da sequência de desenhos gerados em instantes consecutivos).

Uma das questões chave por trás da abordagem que apresentamos é a proposta das tabelas (Nodo,Figura Local) e (Nodo,Regra de Reescrita). Refere-se ainda que o estudo efectuado sobre tipos de visualizações a gerar pelo sistema Alma tem o objectivo de sintonizar os ditos *mappings* de acordo com os paradigmas da programação e com os níveis de visualização.

Como aplicações possíveis do Alma, destacamos: a animação de algoritmos, como apoio ao ensino da programação e como instrumento da didáctica da matemática (onde o principal uso, poderá ser na explicação visual dos princípios de cálculo descritos pelo algoritmo em análise); a análise de resposta, para apoio à correcção de provas de avaliação; a interpretação (visual) de documentos anotados.

Aqui apenas se irá abordar o caso de aplicação do sistema ao ensino da programação, isto é, a utilização do Alma com linguagens de programação. No entanto, os programas a animar poderão ser escritos em linguagens de diferentes paradigmas.

Este artigo está dividido em quatro secções além desta: uma secção sobre sistemas de animação, onde não se fala só do sistema Alma, mas também de sistemas já existentes; uma outra secção apenas sobre o sistema Alma — breves considerações sobre o seu caracter genérico, a sua arquitectura, alguns detalhes de implementação relativos à construção da animação e respectivas visualizações; uma outra secção sobre a interacção com o utilizador (interfaces utilizadas e importância pedagógica do sistema); e, por último as conclusões sobre o estado de implementação do sistema e trabalho futuro.

2 Sistemas de animação de algoritmos

A animação de um algoritmo é um tipo de visualização dinâmica das principais abstracções expressas pelo algoritmo subjacente a um programa; a animação é uma forma natural de representar comportamentos.
A importância da animação de algoritmos reside na habilidade de retratar a essência da lógica do programa. Um dos papeis fundamentais das linguagens visuais é facilitar a percepção e a exploração de informação complexa. Os professores podem usar o poder expressivo das representações visuais para ajudar os alunos a entenderem os algoritmos e o comportamento dos programas através da animação desses mesmos algoritmos.
É necessário ver a animação como um processo dinâmico e complexo, caracterizado por uma dimensão temporal, com misturas de mudanças contínuas e mudanças ocasionais que produzem uma evolução concorrente de vários objectos gráficos.
Para que se possa fazer um estudo comparativo, apresenta-se 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 que permitiu a programação textual de animações de algoritmos, surgiu em 1981, e tornou-se num modelo seguido nos trabalhos posteriores. Neste sistema alguns pontos estratégicos do algoritmo a animar eram anotados com chamadas a procedimentos de animação. Em 1987, surgiu o sistema ANIMUS [Dui98] que, sendo também um sistema de programação textual de animações, inclui restrições temporais na construção de animações de algoritmos. Mais tarde, em 1989, apareceu o sistema ALADDIN [HHR89] que permite especificar visualmente a animação de programas textuais. TANGO [Sta90] surge em 1990 com um modelo de animação com uma semântica precisa baseada no paradigma Path Transition. Este paradigma baseia-se no movimento contínuo e suave de uma imagem, considerando conceptualmente todos os tipos de animação como uma imagem ao longo de sucessivas alterações. O sistema TANGO cria a animação indicando passo a passo as modificações a efectuar. Existe também o sistema XTANGO [Sta] que permite construir animações coloridas em tempo real. Neste sistema, a construção da animação consiste em implementar o algoritmo a animar em C; decidir quais os eventos importantes, ou seja, aqueles a serem retratados durante a execução do algoritmo; estes eventos activam rotinas de animação que são implementadas num ficheiro à parte; as transições dos objectos incluem movimento, mudança de cor, de tamanho ou de conteúdo.
A seguir ao sistema XTANGO surgiu o sistema POLKA [Sta99a] (sistema para construir animações) que, sendo mais poderoso e flexível, tem vindo a ser actualizado e inclui um *front-end* chamado SAMBA. Enquanto os sistemas apresentados são utilizados por alguem que quer explicar algoritmos, o sistema SAMBA tem por objectivo permitir que sejam os próprios alunos a criar animações de modo a entenderem o algoritmo subjacente a essas animações.

Os autores deste sistema baseiam-se na ideia que a animação de um algoritmo é uma representação gráfica e dinâmica de dados e operações que tornam o algoritmo mais concreto e mais fácil de entender. SAMBA [Sta99b] é um interpretador interactivo de animações que lê comandos ASCII e produz acções de animação. Um programa escrito em qualquer linguagem pode ser anotado para gerar esses comandos.
O sistema POLKA, tal como o anterior (XTANGO), obriga a que o código fonte seja alterado para que o programa possa ser animado. A animação é programada textualmente (através de anotações), associando a cada anotação desenhos e imagens que se julguem necessárias.
Mais recentemente, foram apresentados trabalhos que têm vindo a dar o seu contributo no sentido de criar um sistema que evite uma programação exaustiva e inflexível de animações. Em [BNR97] é apresentado um sistema que usa tipos de dados especiais (aos quais chamam auto-animados) para criar animações de programas escritos em Java. Este sistema embora inovador nao evita que o código fonte seja alterado.
Em relação à animação de linguagens visuais existe um estudo de implementação do paradigma *Path Transition* na programação deste tipo de animações. Neste trabalho[CBC96], a animação é programada passo a passo usando uma linguagem visual. Existe ainda um outro modelo de programação visual de animações para interfaces [Vod97], tendo como base o mesmo paradigma. Este trabalho consiste em conseguir manusear objectos gráficos usando também uma linguagem visual.
Todas as referências apresentadas falam de sistemas que, de uma forma ou de outra, pretendem criar animações de algoritmos subjacentes a programas. No entanto, existem outros trabalhos que discutem questões relacionadas com a aplicação desses e de outros sistemas idênticos no ensino da programação. Alguns exemplos podem ser consultados em [McW96], [Mic96], [Sta96] e [SBC96].
O projecto que envolve o desenvolvimento do sistema Alma, consiste em explorar a importância da animação e da visualização de programas quer a nível científico, quer a nível pedagógico e alargar estas facilidades às linguagens visuais de programação reforçando o interesse de uma nova representação da informação: a representação visual.
Pretende-se, no contexto do projecto em causa, caracterizar detalhadamente estes domínios (visualização/animação, interpretação e aprendizagem) de modo a poder-se estudar a viabilidade de um ambiente genérico para resolução sistemática deste tipo de problemas.
O sistema Alma fornece um conjunto de facilidades para visualizar o fluxo de controlo e de dados dos algoritmos subjacentes aos programas em análise (visão mais virada para a semântica operacional). Pretende-se também criar visualizações relacionadas com os conceitos subjacentes a esses programas (visão mais próxima da semântica declarativa).
O sistema Alma pretende ser usado em programas escritos em diversos paradigmas e consegue essa generalidade porque se baseia numa representação interna universal. Para além disso, este sistema distingue-se dos apresentadas atrás na

medida em que não é necessário alterar o código a animar adicionando-lhe primitivas de desenho, ou usando tipos de dados especiais.

3 O sistema Alma

Nesta secção apresenta-se o sistema Alma, a sua arquitectura e as suas características principais. São ainda discutidas técnicas e estruturas usadas na sua implementação, bem como algumas das visualizações geradas pelo sistema para cada tipo de construtores típico das linguagens de programação.

3.1 Introdução sobre o caracter genérico do sistema

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). Utiliza-se o mesmo *back-end*, ou seja, a animação é gerada sempre da mesma forma independentemente da linguagem, o qual trabalha sempre sobre a mesma representação interna (que terá de ser produzida pelos diferentes *front-end*). Nesta forma de abordar a concepção/desenho do sistema é que reside, essencialmente, a generalidade que temos apregoado.
Pretende-se que o sistema Alma use a mesma representação interna para todos os paradigmas, e o mesmo princípio de funcionamento, embora a visualização gerada para cada um deles possa (e provavelmente deva) ser diferente, para a sua completa adequação, pois é certo que a generalidade dum sistema opõe-se à sua expressividade.
As interfaces do sistema devem permitir que o utilizador interactue facilmente com o sistema. Era desejável que o Alma fornecesse um editor de texto. Ora 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 deve permitir também 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 determinadas estruturas, as visualizações irão basear-se em representações

visuais das estruturas de dados e do controlo de fluxo do programa, 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.

3.2 Características da arquitectura do sistema que permitem essa generalidade

Para simular a execução, permitindo um controlo total sobre as variáveis e blocos a inspeccionar, decidimos construir um animador baseado na àrvore 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[WG84] para reconhecer a estrutura do programa fonte (reconhecer símbolos e as regras de derivação; calcular atributos) e construir a árvore decorada. A arquitectura do sistema Alma está esquematizada na figura 1. Inspirados na tecnologia tradicional da

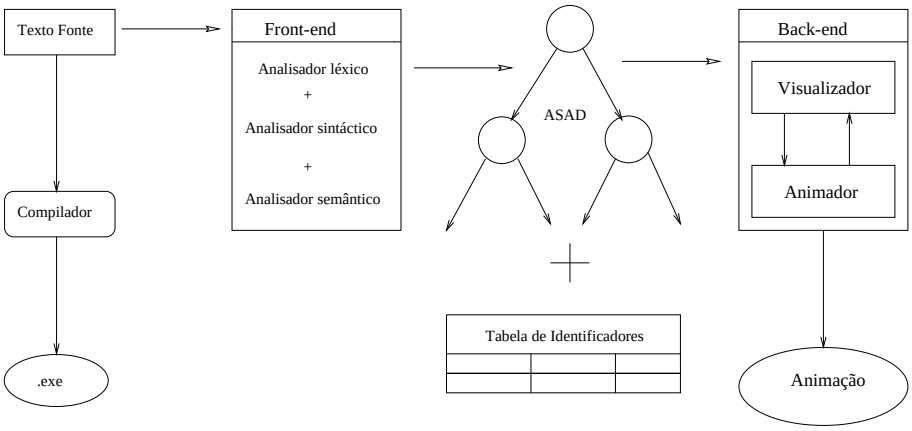

Figura 1. Arquitectura do Sistema Alma

compilação (bem conhecida e bem fundada), o princípio em que se vai basear o sistema Alma é de que a Àrvore de Sintaxe Abstracta Decorada (ASAD) juntamente com a Tabela de Símbolos 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 sistemática, 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*.

3.3 Uso de ASAD nos diferentes paradigmas de programação a animar

Dentro de um conjunto possível de finalidades do sistema Alma, o ensino da programação tem sem dúvida prioridade. O importante é abranger o maior número de tipos de linguagens para conseguirmos cobrir o maior número de casos de aprendizagem. A arquitectura do sistema Alma, tal como já foi dito na secção anterior, permite obtermos essa generalidade.
Nesta secção pretende-se mostrar (através de alguns exemplos) a representação interna gerada pelo *front-end* e explicar como é usada pelo *back-end* para criar a animação desejada. A ideia é mostrar como, independentemente do paradigma da linguagem (imperativas, lógicas, funcionais, etc) irá ser feita a travessia da árvore de sintaxe abstrata decorada de modo a produzir a animação desejada.
O *front-end* gera a ASAD cujos tipos de nodos serão conhecidos do *back-end*. Ou seja, o *back-end* tem uma lista de nodos que poderão ser gerados pelo *front-end* para os diversos tipos de linguagens. É possível que um tipo de nodo seja comum a vários paradigmas.
A estrutura de um nodo típico de uma árvore de *parsing* da gramática concreta está representada na figura 2. Cada nodo contém o nome do símbolo que representa, o número da produção da gramática desse símbolo, um conjunto de atributos e um conjunto de apontadores para os simbolos da lado direito da produção.

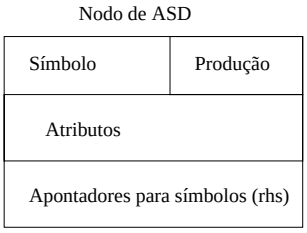

Figura 2. Estrutura dos nodos da ASD

O *back-end* terá um conjunto de primitivas de visualização a associar a alguns dos nodos da árvore. A travessia da árvore (efectuada pelo *back-end*) tem por objectivo criar um desenho que represente (estaticamente) o programa a animar. Ao longo da travessia são usados atributos previamente calculados (para obter valores de variáveis e decidir a sequência de instruções a executar) e, feito o agrupamento de figuras (locais aos nodos) de modo a obter-se o desenho completo do programa. Esse desenho, ao qual chamamos **global** é criado ao colecionar as pequenas figuras associadas a determinados nodos,

às quais chamamos **figuras locais**. Essa recolha é feita por uma travessia *Top-Down* (essencialmente *Post-fix*) da árvore, sintetizando-se o desenho global por agrupamento das figuras locais num processo *Bottom-Up*, tal como no trabalho descrito em [KA95].
Terminada a travessia que constrói o desenho global, será usado um processo de reescrita como forma de simular a execução necessária à animação. Após cada reescrita, obtém-se um novo desenho à custa de uma outra travessia da árvore, agora com nova forma e com os valores dos atributos, eventualmente alterados, já actualizados.
Podemos então afirmar que o *back-end* será constituido por um *visualizador* e por um *animador*. O visualizador produz o desenho do programa num determinado instante e o animador aplica uma regra de reescrita de árvores à ASAD (como no AGG [MRRT99]). A ASAD transformada gerará através das mesmas travessias um novo desenho global. A apresentação dos sucessivos desenhos constitui a animação. Essas regras de reescrita permitem programar a animação não actuando directamente no desenho mas na árvore que produz esse desenho. Sendo todo este processo independente da paradigma da linguagem fonte.
A figura 3 mostra toda a linha de transformações entre o texto fonte e a animação pretendida.

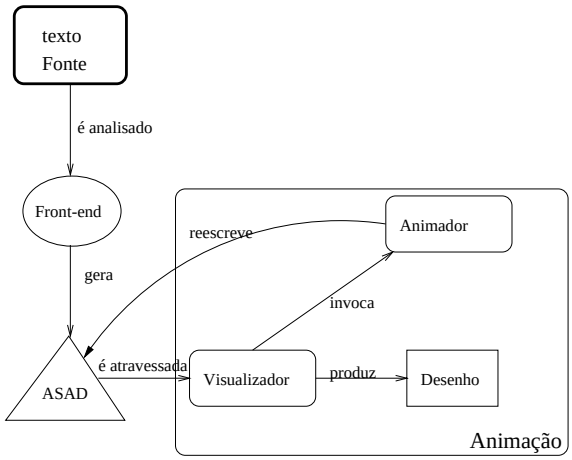

Figura 3. Linha de produção da animação a partir do texto fonte

Para que o *visualizador* e o *animador* do *back-end* efectuem as suas tarefas, precisam de armazenar informação sobre a forma de primitivas de desenho para cada tipo de nodo e regras de reescrita de árvores.
Um exemplo de ASAD é apresentado na figura 4. O extracto de programa (escrito em Pascal) que deu origem a esta árvore contem as seguintes instruções:

```
...
read(a);
read(b);
if (a>b) then a:=a-b
```

```
        else write(b/2);
...
```

A estrutura dos nodos da ASAD foi simplificada para diminuir a complexidade da figura apresentada. Algumas das figuras locais que poderão ser associadas

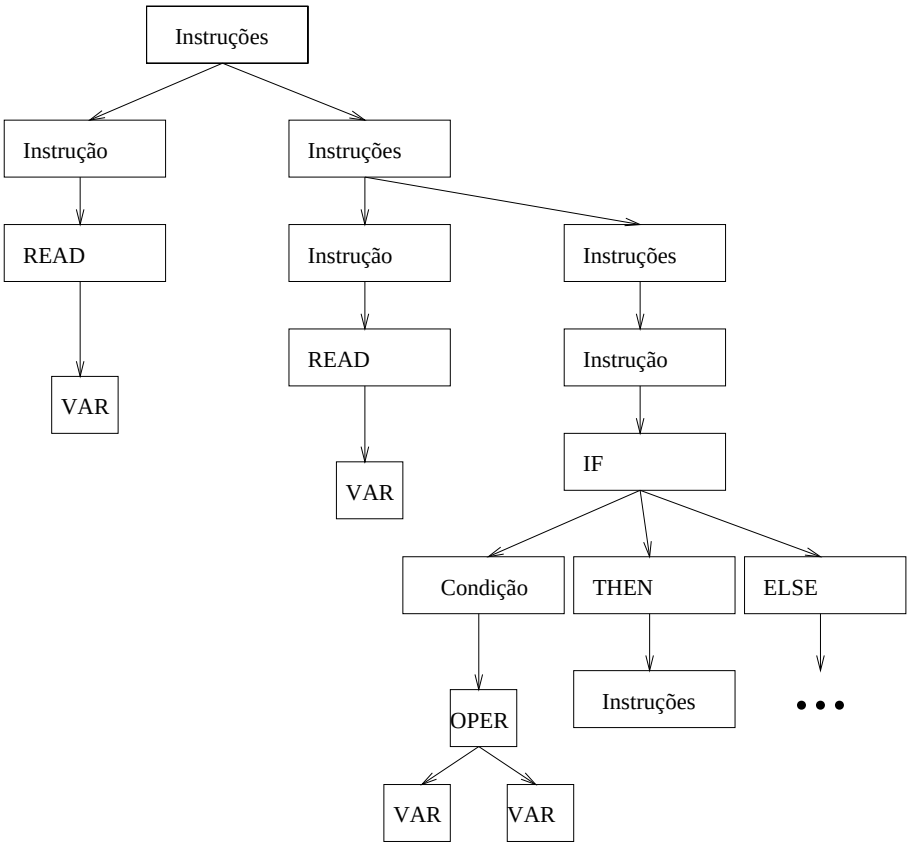

Figura 4. Árvore de Sintaxe Abstracta Decorada

a alguns nodos são apresentadas na figura 5 (visão parcial do *mapping* entre nodos da ASAD e representações visuais, referido na introdução).

Aplicando regras de reescrita, como a apresentada na figura 6, a árvore da figura 4 vai sendo modificada, após o que será redesenhada. Essas regras tem por objectivo traduzir a semântica do programa; como resultado da sua aplicação a ASAD vai sendo simplificado, prosseguindo o processo até nada mais haver para reescrever.

As travessias dessas árvores têm como resultado visualizações semelhantes às apresentadas na subsecção seguinte.

3.4 Visualizações adequadas para cada tipo de linguagem

Optamos, a título de exemplo, por uma visualização muito simples onde é possível, para cada instrução, ver as variáveis (e os seus valores) que são usadas e as variáveis que são alteradas.

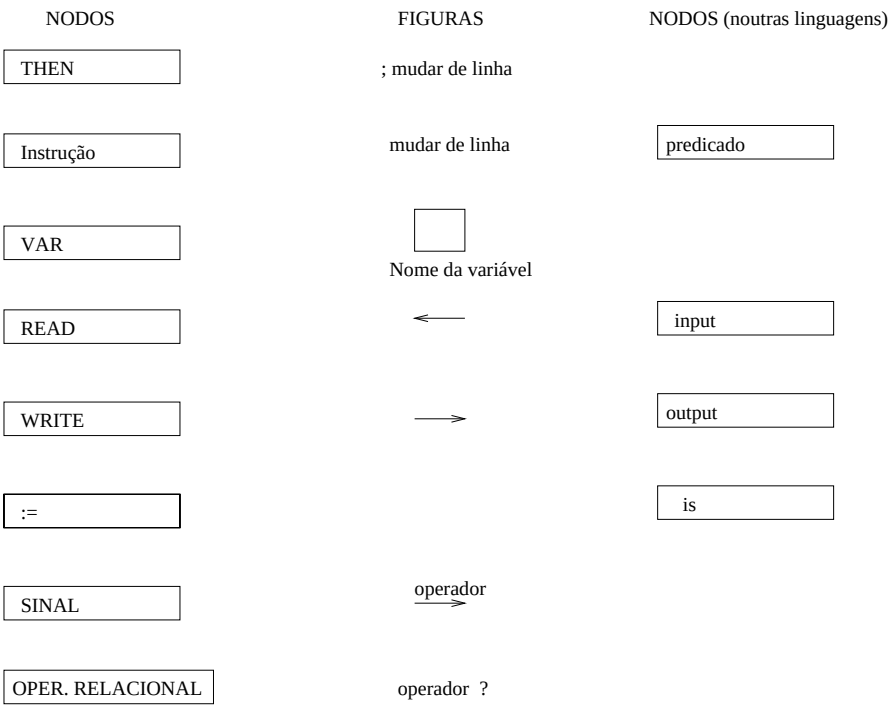

Figura 5. Alguns nodos e suas figuras locais

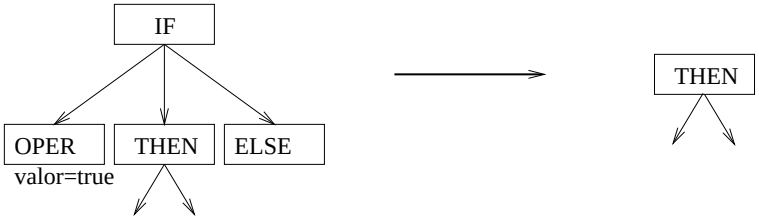

Figura 6. Novo desenho do programa

Os nodos mais elementares da ASAD, que normalmente correspondem a variaveis ou a constantes, devem ter associado um figura local simples. Essa figura é um rectangulo etiquetado com o nome da variável (se for o caso) e dentro do rectângulo estará o valor da variável ou da constante. Cada nodo de operação tem associada uma figura com o formato de uma seta que estará etiquetada com o sinal da operação. Os operadores relacionais estarão representados pelo próprio sinal e aparecerão em notação *post-fix*, tal como as operações aparecerão depois de todas as figuras dos seus operandos e tal como o nome do predicado (no paradigma lógico) aparecerá a seguir aos seus argumentos.
Existem, em nodos superiores, figuras que representam as instruções condicionais (ligação entre a condição e as instruções) ou as instruções cíclicas (divisão entre a condição de paragem e as diversas iterações). As classes (no paradigma orientado ao objecto) serão representadas por etiquetas com o nome da instância e o nome do método a ser executado. As instruções que constituiem o método serão tratadas como no paradigma imperativo.
Para ilustrar as ideias descritas nos parágrafos anteriores mostrar-se-á um

exemplo na figura 7. O desenho apresentado é o resultado que se pretende obter (com o sistema Alma) para o programa em Pascal apresentado atrás, cuja ASAD foi parcialmente ilustrada na figura 4.

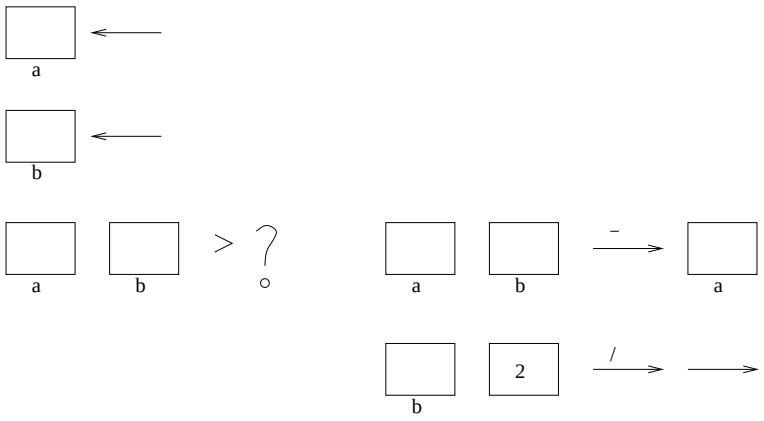

Figura 7. Desenho global do programa

O programa ilustrado na figura 7 efectua a leitura de duas variáveis e a animação deste programa traduz-se na actualização dos valores das variáveis em função das entradas. Neste caso, a animação é constituída pela figura 7 e pela figura 8, alcançada após algumas iterações de reescrita sobre a ASAD que deu origem ao desenho da figura 7.

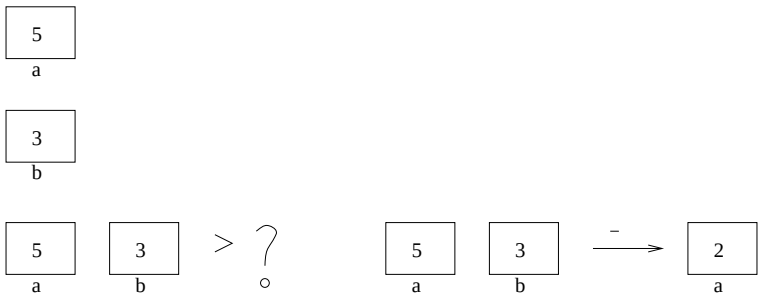

Figura 8. Novo desenho do programa

Este tipo de animação (gerada como se acabou de explicar) permite uma análise operacional dos programas em estudo. A simulação da execução dos programas segue uma abordagem imperativa, com o objectivo de mostrar, instrução após instrução, os valores das variáveis (como foi ilustrado na figura 8). No entanto, pretende-se permitir a criação de outro tipo de animação, facultando outro tipo de análise do programa. Por exemplo, para um pequeno programa escrito em Prolog com os seguintes predicados:

```
mae(alda,joana).
mae(joana,joao).
pai(luis,pedro).
pai(luis,joana).
pais(M,P,E) :- mae(M,E), pai(P,E).
?-pais(M,P,joana).
```

poderiamos obter, com o sistema Alma, vários tipos de visualizações, nomeadamente as das figuras 9 e 10. As visualizações mostradas constituem o resul-

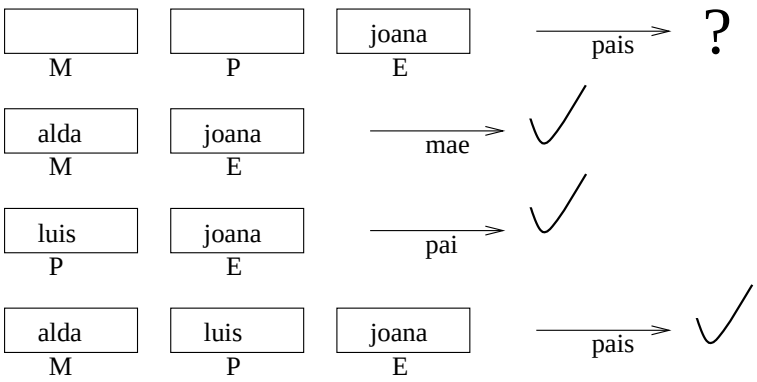

Figura 9. Visão operacional do programa Prolog (construção da resposta)

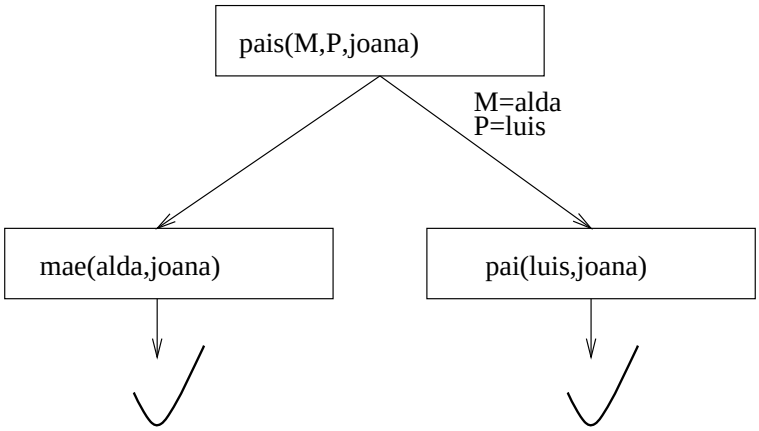

Figura 10. Visão declarativa do programa Prolog (Árvore de Prova)

tado de associar desenhos e as regras de reescrita a cada nodo da árvore. Os *mappings* que definem tais associações permitem, não só controlar a percurso efectuado pelo *Tree Walker Animator* (porque modifica a árvore sobre a qual este trabalha), mas também, associar um só desenho a um conjunto de nodos, de forma a gerar uma visão, mais orientada à variável, ou mais orientada ao conceito.
Concluindo, afinando e coleccionando um maior número de regras e tendo uma tabela de figuras locais cada vez mais completa, pretende-se abranger o maior número de casos possível para aplicar o sistema Alma.

4 Interacção com o utilizador

Tendo sido projectado o sistema Alma para ser usado como ajuda a tarefas de programação, a sua interface deve ser de utilização fácil e intuitiva. A interacção do sistema com o utilizador é um factor importante a ter em conta

na implementação deste sistema. Esta secção pretende tecer algumas considerações sobre as características das interfaces e sobre o futuro papel deste sistema nas aulas de programação.

4.1 Interfaces para inserção de programas e escolha de visualizações

O *back-end* deve produzir uma interface adequada, para permitir escolher facilmente os blocos a analisar e as variavéis a visualizar. O utilizador poderá visualizar a evolução das várias instruções ou apenas os conteúdos das variáveis.
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. Um primeiro protótipo pode ser visto na figura 11. A figura mostra como é possível indicar qual o subprograma a visualizar. Para cada subprograma são apresentadas ao utilizador as instruções ser executadas e simultaneamente o valor das variáveis aquando dessa execução. É necessário também criar uma janela de inserção de valores de variáveis como é o caso da variável a do programa. Este primeiro protótipo

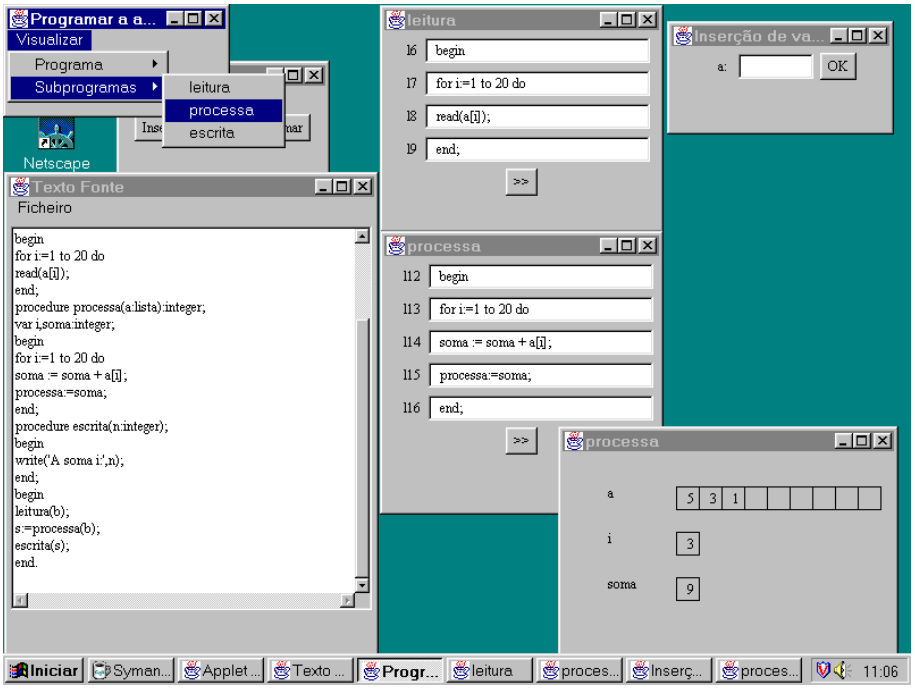

Figura 11. Aspecto Geral das Interfaces usadas para Visualização

foi construído apenas para estudar questões relacionadas com a interacção do sistema.
O processo de construção das interfaces usa a *Tabela de Identificadores* para ter acesso a nomes e tipos de variáveis usadas no programa e a informação relativa a subprogramas. Essas informações serão usadas para construir as

várias opções dos menus. É também necessário ter em conta, em todo este processo, os nodos de definição e uso das variáveis para poder eventualmente detectar variáveis não declaradas e redeclarações.

4.2 Importância pedagógica do sistema ALMA

Na medida em que o sistema Alma também está vocacionado para o ensino da programação, tem por objectivo ajudar, de alguma forma, os alunos a compreenderem os programas escritos por eles e pelo professor. A animação pode ser criada pelo próprio aluno ou então pelo professor se este quiser incluir a visualização dos programas no plano da aula.
As animações contribuiem para o sucesso da aprendizagem visto ser mais fácil e intuitivo o desenho do que a linguagem textual usada para programar. Para o aluno tirar proveito da animação deve compreender o mapeamento entre o algoritmo e a sua representação gráfica. Uma simples apresentação e narração da animação não permite que os alunos construam conexões referenciais entre as duas representações. Defendemos então que é necessária a interacção animação/utilizador usando várias estratégias de sucesso consoante o nível do aluno.
Com o sistema Alma o aluno pode visualizar o conteúdo das variáveis e a sua alteração sempre que uma instrução é executada. Esta facilidade vai permitir que o aluno compreenda o que faz cada operação, a evolução dos valores das variáveis ao longo da execução e o objectivo final do programa.

5 Conclusão

Tomando por base a reconhecida importância da animação / visualização de programas para apreensão dos algoritmos subjacentes, defendeu-se a necessidade de um sistema genérico e apresentámos uma proposta.O trabalho descrito neste artigo teve grande importância na definição de necessidades mais concretas relativas à construção das visualizações / animações.
A implementação do sistema Alma implicou o estudo de estruturas como árvores de sintaxe abstracta decoradas (para serem usadas como representação interna) e técnicas relacionadas com travessias de árvores. Na criação de animações foram precisas técnicas de reescrita de grafos, ou, no caso do sistema Alma, reescrita de árvores.

5.1 Estado de implementação

Após o desenho da arquitectura do sistema Alma, a sua implementação foi dividida em várias fases. Neste momento estuda-se com detalhe métodos e técnicas a utilizar em cada um dos seus módulos (essencialmente do *back-end*), com o objectivo de confirmar a eficiência da arquitectura proposta, optimizar o seu

esquema de funcionamento e tentar identificar ferramentas já desenvolvidas que sejam utilizáveis nalguma fase de implementação.
O estudo que envolveu a criação e optimização da arquitectura do sistema levou à construção de alguns protótipos parciais, nomeadamente, a construção de uma interface para editar programas e programar animações.

5.2 Trabalho futuro

O trabalho a desenvolver consiste essencialmente na conclusão de implementação do sistema Alma: existem algumas ferramentas que deverão ainda ser exploradas para possível integração no sistema; irá ser feito um estudo de viabilidade sobre o uso do sistema proposto para animar programas visuais. Poderão ser ainda identificadas novas situações onde o sistema poderá ser usado e fazer um estudo das alterações que este terá que sofrer para cobrir um maior número de necessidades.

Referências

[BNR97] M. H. Brown, M. A. Najork, and R. Raisamo. A java-based implementation of collaborative active textbooks. In *VL'97 - IEEE Symposium on Visual Languages*, pages 376–384. IEEE, September 1997.

[CBC96] Paul Carlson, Magaret Burnett, and Jonathan Cadiz. A seamless integration of algorithm animation into a visual programming language. In *AVI'96 - International Workshop on Advanced Visual Interfaces*. acm, May 1996.

[Dui98] R. A. Duisberg. Animation using temporal constraints: An overview of the animus system. *Human-Computer Interaction*, 3(3):275–307, August 1998.

[HHR89] E. Helttula, A. Hyrskykari, and K. Raiha. Graphical specifications of algorithm animations with aladdin. In *22nd Hawaii International Conference on System Sciences*, January 1989.

[KA95] Hideki Koike and Manabu Aida. A bottom-up approach for visualizing program behavior. In *IEEE Workshop on Visual Languages*. IEEE, October 1995.

[McW96] Jeffrey D. McWhinter. Algorithm explorer: A student centered algorithm animation system. In *VL'96 - IEEE Symposium on Visual Languages*, pages 174–181. IEEE, September 1996.

[Mic96] Amir Michail. Teaching binary tree algorithms through visual programming. In *VL'96 - IEEE Symposium on Visual Languages*, pages 38–45. IEEE, September 1996.

[MRRT99] Boris Melamed, Michael Rudolf, Olga Runge, and Gabriele Taentzer. The attributed graph grammar system - homepage. http://tfs.cs.tu-berlin.de/agg/, 1999.

[SBC96] John T. Stasko, D. Michael Byrne, and Richard Catrambone. Do algorithm animations aid learning? Technical Report GIT-GVU-96-18, Georgia Institute of Technology, Atlanta, August 1996.

[Sta] John Stasko. Xtango. http://www.cc.gatech.edu/gvu/softviz/algoanim/xtango.html.

[Sta90] John T. Stasko. Simplifying algoritm animation with tango. In *IEEE Workshop on Visual Languages*. IEEE, October 1990.

[Sta96] John T. Stasko. Using student-built algorithm animations as learning aids. Technical Report GIT-GVU-96-19, Georgia Institute of Technology, Atlanta, August 1996.

[Sta99a] John Stasko. Polka. http://www.cc.gatech.edu/gvu/softviz/parviz/polka.html, 1999.

[Sta99b] John Stasko. Samba. http://www.cc.gatech.edu/gvu/softviz/algoanim/samba.html, 1999.

[Vod97] D. Vodislav. A visual programming model for user interface animation. In *VL'97 - IEEE Symposium on Visual Languages*, pages 348–357. IEEE, September 1997.

[WG84] William M. Waite and Gerhard Goos. *Compiler Construction*. Springer-Verlag, 1984.