GRELHA DE COMPARAÇÃO PARA LINGUAGENS VISUAIS [1]

Carla Gonçalves[2]
Mário Rui Gomes[3]
Joaquim Jorge[4]
INESC, Rua Alves Redol nº 9, LISBOA

Sumário

Idealizado no fim da década 60, o conceito de linguagem visual (LV) tem ganho cada vez mais importância como base para ferramentas de produtividade na construção de sistemas.

Com o desenvolvimento deste novo conceito surge a necessidade de definir uma (ou mais) métricas para analisar tanto os produtos comerciais como os protótipos, resultantes de trabalhos avançados de investigação.

Neste artigo propõe-se uma grelha de caracteristicas usada para comparar diversos sistemas de programação baseados em linguagens visuais. Essa grelha será aplicada a dois produtos comerciais (Prograph, LAbVIEW) e aos trabalhos do grupo de investigação da Universidade de Washington, na qual o primeiro autor realizou recentemente um estágio de investigação[5] (Hyperflow, ainda em desenvolvimento, e Show and Tell).

1. Introdução

"Uma imagem vale mil palavras". É inegável a importância das imagens na comunicação. Elas tanto podem ser usadas para descrever paisagens como para estruturar raciocínios.

É muito comum programadores esboçarem diagramas antes de iniciarem a implementação de um sistema numa linguagem textual. Os diagramas bidimensionais descrevem a estrutura do sistema, sendo por este facto mais fáceis de compreender que as listagens do código. Os diagramas podem ter a vantagem adicional de destacarem a informação mais relevante.

Por Programação Visual (PV) entendemos uma programação que usa representações gráficas, contrariamente à textual que usa apenas palavras.

As LVs são consideradas boas ferramentas para resolução de problemas de programação complexos devido à sua capacidade de representar os modelos mentais do utilizador [Eisenstadt 90], à facilidade na aprendizagem, à independência da linguagem natural (tal como inglês, português ou japonês) e à alta "densidade" semântica [Raymond 91]. Wayne

[1] Este trabalho foi parcialmente suportado pela Junta National de Investigação Científica e Tecnológica.
[2] mcg@minerva.inesc.pt
[3] mrg@minerva.inesc.pt
[4] jaj@minerva.inesc.pt
[5] Laboratory for Pen-based Silicon Paper Technology and Visual Programming, Wash.Univ., St.Louis, EUA

Citrin argumenta que a apresentação das LVs aumenta a rapidez de implementação dos modelos, enquanto que a densidade semântica aumenta a rapidez de compreensão de um programa [Citrin 93].

À medida que as representações visuais melhoram e o processo de programação é simplificado, o tempo dispendido a programar passa a depender mais da interface homem-máquina (IHM) das ferramentas de PV, factor fundamental que vai ser contemplado na definição da grelha a apresentar.

Requisitos de grandes espaços de ecran, criação de representações visuais e qualidade dos dispositivos físicos (por ex. resolução dos ecrans e dados de entrada) são cruciais nas LV.

Podendo a ambiguidade ser um ponto fraco das representações usadas, estas têm que ser cuidadosamente escolhidas para que o programa seja usado universalmente por pessoas de níveis culturais diferentes, sem que haja confusão ou má compreensão. Consideremos a fig 1. Enquanto que algumas pessoas podem identificar esta imagem como um sinal de estacionamento proibido, outras indentifican-na como a cabeça de um parafuso!

Figura 1: O que representa esta figura?

A secção 2 descreve sucintamente os sistemas estudados, à qual se segue a comparação dos mesmos. Por último faz-se um resumo e retiram-se alguma conclusões sobre o dominio de aplicação das linguagens e aplicações que geram.

2. Sistemas estudados

Esta secção descreve sucintamente cada um dos sistemas estudados por ordem cronológica.

2.1 Show and Tell

Show and Tell (ST) foi desenvolvido por Kimura, aparecendo pela primeira vez em 1986 como um sistema de programação com uma interface gráfica evoluida e simples de manipular [Kimura 86] [Kimura 90]. Show and Tell é uma Linguagem de Programação Visual (LPV) genérica originalmente desenvolvida para crianças. A construção de um programa não requer o uso de teclado, excepto para entrada de caracteres, sendo o rato o principal meio de interacção.

Os programas consistem num conjunto ordenado de caixas e setas. A semântica é baseada no conceito de *dataflow* e consistência.

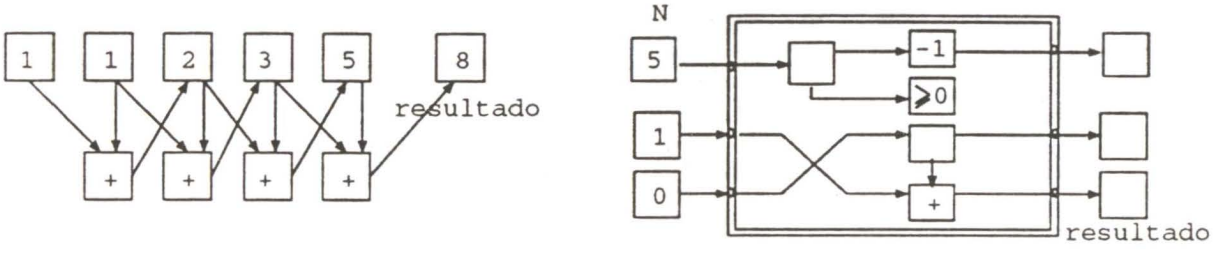
Em programação por *dataflow* os dados são activos, eles fluem pelo "código do programa", activando as operações assim que todas as suas entradas chegarem.

A consistência é usada no controlo dos programas de Show and Tell, emulando o *if* de uma linguagem procedimental, tal como o C ou o Pascal, sem, contudo, explicitar as ideias de fluxo de controlo. Uma caixa fica inconsistente se existir algum conflito entre os dados

que fluem para ela. Nesse caso, ela desactiva as suas saídas, que deixam de existir para o resto do programa.

Em Show and Tell as funções, constantes, variáveis, iteradores e contentores de inconsistência são todos representados por caixas. As setas indicam o fluxo de dados entre as caixas.

A fig. 2 mostra um programa para calcular o 5º número de Fibonacci, usando uma estrutura sequencial e uma caixa de iteração (caixa com bordadura dupla). No primeiro caso os dados fluem pelas caixas até que o resultado é apresentado na caixa final (resultado). As caixas com números representam caixas de memória e as com o sinal "+" adicionam os dados de entrada. A caixa de iteração é um ciclo infinito que pára apenas quando se torna inconsistente. O diagrama que se observa dentro da caixa é executado uma vez em cada iteração.



a) Estrutura sequencial b) Caixa de iteração

Figura 2: 5º número de Fibonacci (Show and Tell).

2.2 Hyperflow

Hyperflow (HF) é uma extensão de Show and Tell e foi pela primeira vez publicada em 1992 [Kimura 92] [Sengupta 94] [Apte 94]. É uma LPV desenvolvida para um sistema multimedia baseado em caneta, e como uma ferramenta para IHM.

A semântica de Hyperflow é baseada em BoxGraph [Kimura 94], um modelo formal de *dataflow*. O elemento básico no programa, *vip* (*visually interactive process*) é um processo concurrente. A funcionalidade de um vip é representada por um diagrama de caixas hierarquicamente colocadas, comunicando por setas. As caixas representam processos e as setas dados que fluem entre eles. Os vips comunicam através de mensagens, permitindo computação distribuida.

As caixas em Hyperflow podem representar fontes de dados ou operações aritméticas, assim como estruturas de controlo, esquemas para bases de dados, objectos de IHM e animações.

As setas podem transportar tipos de dados continuos ou discretos. Os primeiros são utilizados no processamento multimedia, como som e animação, e para processamento de traços de caneta ("pen-strokes"), tais como reconhecimento de comandos.

Hyperflow tem um mecanismo de segurança para controlar a acessibilidade (*visibility*) e a modificabilidade (*locking*) [Kimura 94] de modo a uniformizar a LV para diferentes níveis de utilizadores. A acessibilidade dos objectos é controlada independentemente dos utilizadores, existindo para cada 10 níveis de segurança. A modificabilidade é também controlada por níveis, sendo independente da acessibilidade: os objectos podem ser vistos sem serem modificados.

A fig. 3 mostra um programa para calcular o 5º número de Fibonacci, usando uma caixa de iteração (caixa com bordadura dupla). Pode-se controlar a acessibilidade de modo a que os calculos realizados sejam escondidos. Assim, por exemplo, um professor pode fornecer aos seus alunos um programa para calculo do n^{ésimo} número de Fibonacci, sem contudo mostrar como os calculos são efectuados. Para isso teria que, usando o programa da fig. 3 a), tornar acessível a caixa que controla o ciclo, que seria a entrada do sistema, e a caixa resultado que mostra o n^{ésimo} número de Fibonacci.

Se usarmos uma estrutura sequencial para calcular o 5º número de Fibonacci chegamos a um diagrama igual ao de Show and Tell (fig. 2 a)).

a) Parte visivel do professor b) Parte visivel dos alunos

Figura 3: 5º número de Fibonacci (Hyperflow).

2.3 Prograph

Prograph é comercializado pela Prograph International Inc. e apareceu pela primeira vez em 1985 [Prograph]. Uma das caracteristicas mais importantes de Prograph é a integração da programação visual baseada em *dataflow* com técnicas de programação orientadas por objectos [Brown 94].

Na programação orientada para objectos (O.O.) [Booch 94] os programas são organizados como colecções de objectos cooperantes. Todos os objectos são instâncias de uma dada classe, sendo caracterizados por propriedades (atributos) e comportamento (métodos).

As classes podem ser organisadas numa hierarquia, podendo as subclasses herdar todos os atributos e métodos conhecidos pela classe base. Acrescentando novos atributos e métodos e redefinindo métodos herdados nas subclasses especializamo-las.

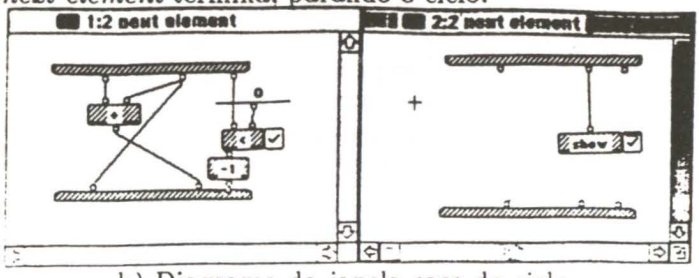
Prograph tem um toolkit completo para construção de aplicações, onde todos os elementos tomam a forma de icons nas janelas do ecran, quer sejam classes, operações (primitivas ou definidas pelo utilizador) ou secções (módulos). Inclui ainda um módulo para compilação. As aplicações compiladas correm mais depressa e ocupam menos espaço que a versão interpretada, apesar de não serem tão facilmente modificáveis.

As classes são representadas num *browser* através de icons, estando a sua hierarquia sempre em concordância com o código! Esta caracteristica reduz o tempo dispendido no desenvolvimento, e manutensão da documentação.

Cada método é representado por um icon. Um método é definido por uma ou mais janelas *case* contendo um diagrama constituido por operações ligadas entre si por *datalinks*. Uma *datalink* é um segmento entre um terminal de saida e outro de entrada, podendo transportar qualquer elemento base (como inteiros e cadeias de caracteres) ou objectos complexos, tais como instâncias de classes.

A figura 4 mostra a implementação do calculo do 5º número de Fibonacci. Quando a 3ª entrada fica negativa, a janela *case* seguinte (2:2) é executada e o resultado (8) aparece numa janela diferente da implementação do programa. Finalmente, e devido ao terminal associado a *show* (√) a função *next element* termina, parando o ciclo.

a) Ciclo



b) Diagrama da janela case do ciclo.

Figura 4: 5º número de Fibonacci (Prograph).

Prograph inclui mecanismos pré-definidos de iteração e paralelismo, incluindo operadores que aplicam uma função a todos os elementos de uma lista, retornando essa mesma lista.

A grande deficiência de Prograph são os minimos requisitos de *hardware*: processador 68020, 7M de memória real, e 15M de espaço em disco.

2.4 LabVIEW

LabVIEW, Laboratory Virtual Instrument Engineering Workbench [Vose 86], [LabVIEW] é um programa para desenvolvimento de aplicações que recorre a uma linguagem de programação visual, G, para a criação dos programas numa forma de diagrama de blocos.

LabVIEW, desenvolvido para engenheiros e cientistas com pouca experiência em programação tradicional, tem como objectivo simplificar a escrita de programas de calculo, controlo de processos e aplicações para teste e medição.

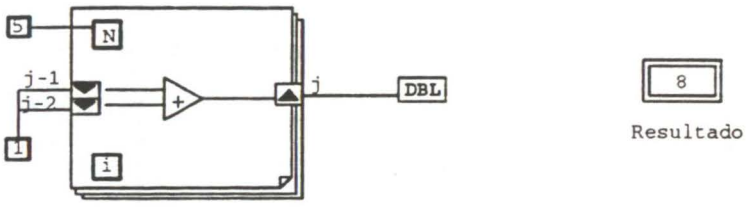
Um programa em LabVIEW é um VI, *virtual instrument*. Tal com o nome indica, o VI simula um instrumento de laboratório, tal como um termómetro ou um osciloscópio. Os VIs são composições de outros VIs, possuindo um painel frontal e um diagrama de blocos, que define o seu comportamento.

A LPV adoptada é uma linguagem *dataflow*, à qual se adicionou um conjunto especial de estruturas de controlo, como por exemplo manipulação de vectores. Também inclui, entre outras, funções matemáticas pré-definidas tais como geração de nºs aleatórios, funções trignométricas e logarítmicas e transformadas (ex.: Transf. de Fourier); funções de acesso a ficheiros; ferramentas de IHM para recolha e mostra de dados.

Na fig 5 pode-se observar a implementação do 5º número de Fibonacci, onde N representa o nº de iterações do ciclo *for* e j-n o registo com o valor da iteração j-n. Note-se que os registos j-1 e j-2 tiveram que ser inicializados a 1. O resultado (8) aparece numa janela diferente da implementação do programa e é identificada em 5 a) pelo icon rectangular DBL.

3. Comparação

Nesta secção faz-se uma comparação das várias linguagens estudadas. A grelha vai ser apresentada progressivamente, juntamente com os resultados, explicando-se porque os



a) Diagrama de blocos b) Painel frontal

Figura 5: 5º número de Fibonacci (LabVIEW).

critérios adoptados na sua escolha.

3.1 Linguagem

3.1.1 Semântica base da linguagem

O uso de linguagens baseadas em dataflow é mais intuitivo [Hills 92]. Experiências com a linguagem Show and Tell [Kimura 92] provaram que o conceito de dataflow é mais facilmente assimilável por crianças do que o conceito de *control flow*[6].

O uso de uma linguagem O.O. pode trazer várias vantagens [Booch 94]. Adicionalmente, os conceitos base na programação O.O., tais como a noção de classe, são muito bem aceites por principiantes e não-programadores.

Hyperflow é uma linguagem O.O. sem classes. Os objectos, VIPs, são criados por duplicação de outros, chamados protótipos. Os novos objectos criados são independentes do seu protótipo. Tal como Prograph, Hyperflow não tem nenhum mecanismo de herança múltipla.

	Prog	LabV	HF	ST
Dataflow	S	S	S	S
O.O.	S	N	S	N
IHM | estr. prog	S	S	S	N

3.1.2 Separação entre a interface e a estrutura do programa

A separação entre a interface (IHM) e a estrutura do programa permite alterações na estrutura sem alterar a interface e vice-versa. Este conceito torna a estrutura do programa mais complexa e abstracta, sendo contraprodutivo no caso do sistema ser dirigido a crianças.

Este tipo de separação diminui o problema de falta de espaço no ecran.

3.1.3 Tipos de dados

Cada linguagem tem o seu conjunto próprio de tipos de dados (TD) disponiveis ao programador, pelo que o vamos descriminá-los aqui. Uma linguagem pode ainda ser ou não fortemente tipificada. Prograph por exemplo só faz faz verificação dos tipos dos dados quando o programa executa (por exemplo, por exemplo na compilação não há verificação

[6] Em *control flow* especifica-se explicitamente a sequência de operações, em vez dos dados necessários para que uma operação execute. Este procedimento é muito semelhante ao dos fluxogramas.

dos argumentos de uma operação e as listas podem ter elementos de tipos complectamente diferentes).

A representação de tipos de dados continuos (dados que representam algo que é continuo como o video ou som) torna-se fundamental para a integração de aplicações multimedia e para reconhecimento de gestos.

Dados	Prog	LabV	HF	ST
Tipificado	S (fracamente)	S	S (sem *cast*)	N
Discretos				
Booleanos	S	S	N	N
Caracteres, Inteiros e Reais	S	S	S	S
Imagem	S	N	N	S
Vectores	N	S	N	U
Listas	S	S	N	S
Classes	S	N	N	N
Definidos pelo Utilizador	S	S	N	N
Continuos	S	S	S	S

U - definido pelo utilizador.

3.1.4 Variáveis

O conceito de variável existe em algumas linguagens baseadas em *dataflow*, embora não seja obrigatória. Considerou-se três tipos de variáveis: locais, globais e estáticas. As variáveis estáticas são variáveis reconhecidas apenas pela classe ou modulo a que pertencem.

A estrutura *persistent* de Prograph representa uma variável global que mantem o valor de uma execução para outra.

Em Hyperflow e Show and Tell não existe o conceito explicito de variável. Estes sistemas têm "caixas" de memória, que contêm valores que podem ser acedidos e alterados por qualquer ligação. Por esta razão consideram-se variáveis.

Variáveis	Prog	LabV	HF	ST
local	S	S	S	S
global	S	S	S	S
estática	S	N	N	N

3.1.5 Controlo

Alguns sistemas baseados em *dataflow* adoptam incluir operações para controlo de fluxo de modo a possibilitar a definição de precedências ou sequências de certas operações. Listam-se em seguida os tipos que consideramos mais importantes.

Estruturas de controlo tais como sequência, ciclos (*do - while*, *for*, ...), *if - then - else* e *switch* são suportadas por todas as linguagens.

Algumas linguagens visuais permitem a formação de ciclos de ligações fechados (CLF). Outras não o fazem de modo a garantir que nunca aconteçam *dead locks*! Este conceito é dificil de transmitir a, por exemplo, uma criança.

O conceito de subrotina, de compreensão não muito simples, ajuda a modularizar e estruturar um programa. Este conceito é suportado por todos os sistemas.

Prograph permite a programação de chamadas de excepção.

Segundo Myers, [Myers 92], o poder e expressividade das abstrações parametrizáveis só está completamente disponível quando existe recursividade. Todas as linguagens modernas providenciam alguma forma de recursividade.

Todas as linguagens, excepto Show and Tell, têm um processo de interrupção de ciclos ou rotinas: uso de *break* ou *return*.

O processamento de vectores é uma caracteristica muito útil no caso do LabV, uma vez que os VIs exigem em geral uma carga elevada de computação. Este processamento é útil numa linguagem genérica onde a simplicidade não é crítica. O aumento na complexidade do sistema é perfeitamente superável pelo possivel aumento de desempenho.

Prograph têm um mecanismo de tratamento de listas. O programador pode transformar uma operação num ciclo que: quebra a lista e executa-a para cada elemento; ou controi uma lista a partir das múltiplas execuções dessa operação.

A existência de subrotinas dentro de outras subrotinas, melhora a modularização de um programa.

Controlo	Prog	LabV	HF	ST
sequência	S	S	S	S
ciclos	S	S	S	S
if then - else	S	S	U	U
switch	S (a)	S (b)	N	U
CLF	N	N	S	N
subrotina (SR)	S	S	S	S
recursividade	S	S	S	S
break/return	S	S	S	N
outros	trata. de listas SR de SR(c)	proc. de vectores	N	N

(a) qualquer tipo

(b) numerais

(c) **A** é SR da SR **B** se **A** é uma SR só reconhecida em **B**.

3.1.6 Paralelismo

Paralelismo existe quando duas ou mais operações são executadas ao mesmo tempo. No caso de concorrência, os processos partilham o mesmo processador. Cada um deles possui um tempo de processamento definido segundo um algoritmo de prioridades.

Dataflow implica paralelismo, uma vez que as operações devem disparar assim que todas as entradas estejam disponíveis. No entanto, isso aumenta a complexidade da implementação do sistema, pelo que muitas vezes não existe paralelismo, havendo um algoritmo para decidir a ordem de execução das operações, o sequenciador (*scheduling*).

Surge então o problema de saber o que acontece quando duas ou mais operações estão activas (prontas a executar). Executam ao mesmo tempo? Se não, qual é a primeira?

Como se impõe a execução de uma operação depois de outra?

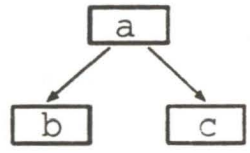


Figura 6: Qual das operações, **b** ou **c**, é executada primeiro?

Existem várias aproximações para determinar qual das operações activas executa. Uma é o uso da informação da posição. Um exemplo seria dar prioridade à operação com o menor valor em XX'. Caso isso não seja suficiente, escolhe-se entre as operações de igual valor de x, a que tiver a menor coordenada em YY'. Nesse caso, considerando a figura 6, **c** só é executada depois de **b** acabar a sua execução, embora sejam activadas simultaneamente. Hyperflow e Show and Tell adoptam esta aproximação, diminuindo a prioridade da esquerda para a direita e de cima para baixo.

Prograph tem uma ligação especial (*synchronize link*) que impõe a sequência de execução de duas operações entre si. Prograph permite a observação da ordem de execução das operações, ajudando o programador a decidir no estabelecimento desta ligação.

Em LabVIEW o programador dispõe de um mecanismo de prioridades com 5 níveis. O tempo de processamento é partilhado entre os VIs de prioridade mais elevada. Depois de todos os VIs de maior prioridade concluirem a execução, começam a executar-se os de prioridade imediatamente inferior e por aí adiante.

Paralelismo	**Prog**	**LabV**	**HF**	**ST**
paralelismo	S	S	S	N
scheduling	S (a)	S (b)	N	N

(a) através de ligações de sincronismo
(b) através de 5 níveis de prioridade

3.1.8 Outras caracteristicas

A utilização de rotinas noutras linguagens tem a vantagem de permitir uma rápida integração de bibliotecas já existentes. Por outro lado, também reflete a opinião de que algumas coisas são melhor implementadas em linguagens textuais do que em visuais, tais como em situações criticas de desempenho.

A possibilidade de manipular facilmente uma base de dados ou ficheiros de dados é uma caracteristica interessante encontrada em quase todos os sistemas.

O *inject* de Prograph permite usar na execução de um programa uma cadeia de caracteres (introduzida pelo utilizador ou não) numa entrada que espere uma função.

O *inject* serve para uma introdução mais simples da noção de ponteiros para funções. A versão interpretada de um programa em Prograph emite uma mensagem de erro caso o ponteiro não exista na execução ou os argumentos estejam incorrectos. Deste modo os erros são detectados mais facilmente e a compreensão do conceito é facilitada.

Numa análise experimental dos sistemas devem ainda ser considerados os tempos de resposta e a extensibilidade (tanto do código como do sistema).

A compreensibilidade da linguagem não deve ser descurada, sendo só efectivamente conhecida através da análise estatistica de experiências conduzidas com vários utilizadores. No entanto pode-se extrair algumas conclusões pelo tipo de icons e diagramas utilizados conjuntamente com conhecimentos adquiridos de IHM. Falaremos sobre isso na secção de interfaces.

Outras caracteristicas	Prog	LabV	HF	ST
rot. noutras linguagens	S	S	S	N
base de dados	S	S	S	S
ficheiros	S	S	S	N
inject	S	N	N	N

3.2 Interface

Na utilização dos sistemas com LPV a interface é da máxima importância pelo que também se propõe uma grelha para este caso. A introdução de diagramas, bem como o processo de desenvolvimento e correcção dos programas é muito importante uma vez que tem um impacto directo e decisivo na aceitação da linguagem. Os jogos de computadores são um exemplo vivo disso. O mesmo jogo pode cativar universalmente todo o tipo de utilizadores pela diversão, mas também simplicidade de utilização. Será que alguém utiliza um jogo que é dificil de compreender e/ou controlar?

Não podemos também esquecer o poder e funcionalidade das aplicações para o utilizador final. Falarei ainda de algumas particularidades interessante dos vários sistemas.

3.2.1 Icons

O aumento do vocabulário de uma linguagem visual (icons, cores, ...) não aumenta, necessariamente, o domínio de aplicação, mas, pode tornar a comunicação mais eficiente, simples e precisa. Em contrapartida, aumenta o tempo de aprendizagem. Também aumenta o tempo de interacção se o sistema for baseado em menus, pois o programador tem que aceder mais vezes aos menus e leva em geral mais tempo a escolher o icon que pretende. Além disso o grau de complexidade dos icons pode limitar a gama de utilizadores.

Um sistema que use icons e conceitos que sejam difíceis de compreender vê diminuida a sua aceitação. Tomemos como exemplo LabVIEW. Pode ser muito útil para um engenheiro fazer um processamento de imagem, mas não é adequado a crianças, já que o número de icons é grande, necessitando muitos deles de fortes conhecimentos matemáticos (ex.: icon da transformada de Fourier).

Finalmente, o aumento do número de icons aumenta a complexidade da forma dos icons, pois têm que se distinguir uns dos outros. Isso aumenta o seu tempo de reconhecimento, e dificulta a construção de um bom dicionário e de um sistema de ajuda.

A sobreposição de icons tanto pode ser positiva como negativa. De facto, o uso de caixas sobrepostas na programação é mais intuitivo para algumas pessoas. Portanto, no caso de Hyperflow e Show and Tell é um factor positivo, uma vez que a sintaxe se baseia na sobreposição de caixas. No caso de Prograph, como a sintaxe não é baseada na sobreposição de icons, esta pode confundir o programador, tornando a depuração de código mais dificil.

	Prog	LabV	HF	ST
Icons				
número	≈ 50	> 100	20	3
sobreposição	S	N	S	S
conhecimento base	médio	elevado	baixo	baixo
Uso de côr	S	S	N	N

3.2.2 Côr

A côr pode ser usada para aumentar e melhorar a transmissão de informação, além de tornar a interface mais atractiva. Mas, sendo a visão particularmente estimulada pela côr, o seu uso excessivo pode distrair e desorientar o utilizador [Chang].

Há ainda a ponderar os aspectos culturais. A mesma côr pode ter significados diferentes consoante o meio em que se insere: um chinês comum associa imediatamente sorte ao vermelho, e um português comum associa-o a sangue ou perigo. Há que ser muito cuidadoso no uso de significado nas cores.

Tanto Prograph como LabVIEW fazem bom uso das cores.

3.2.3 Entradas e saídas

Embora a linguagem em si não dependa dos dispositivos de entrada e saída (E/S), o uso destes influência o desempenho tanto dos programadores e utilizadores finais, como o domínio das aplicações.

Vários investigadores [Kimura 93a] [Greenstein 90] afirmam que a caneta (usada com uma tablete digitalizadora) é mais indicada para desenhar e esboçar que qualquer outro dispositivo, sendo o rato preferido para apontar e seleccionar.

Através de experiências conduzidas com Show and Tell, Kimura conclui que o rato é mais difícil de utilizar por crianças do que a caneta [Kimura 93a].

E / S	Prog	LabV	HF	ST
teclado	S	S	S	S
rato	S	S	N	S
caneta	N	N	S	N
portos E/S	N	S	N	N

Uma das caracteristicas mais interessantes do LabVIEW é a possivel aquisição de dados de uma grande variedade de fontes, utilizando portos E/S.

3.2.4 Depuração de código

As facilidades para depuração de código encontradas são:

Depuração de código	Prog	LabV	HF	ST
breakpoint	S	S	N(a)	Y
passo a passo (local, global, continuo)	S	S	N(a)	N
observação de valores	S	S	S	S
ajuda disponível	Bom	Bom	N	N
documentação	Bom	Bom	Médio	Médio

(a) A ser implementado.

Como seria de esperar, os produtos comerciais possuem um melhor conjunto de facilidades para ajuda e depuração de erros. Todas elas são úteis no desenvolvimento de um programa. Enquanto que ajuda disponível ("*on line help*") tem mais peso para programadores não familiarizados com o sistema, as facilidades de depuração tornam-se mais importantes à medida que os programas e conceitos implementados aumentam de complexidade. Também são importantes na aprendizagem da linguagem e da programação: nada melhor para aprender recursividade do que observar o *stack*, o código do programa com a operação activa evidenciada e os valores das várias variáveis, à medida que o programa vai sendo executado, como acontece em Prograph.

A documentação fornecida em Prograph é mais acessivel que a de LabVIEW, lendo-se mais facilmente.

3.2.5 Comentários

Os comentários são uma grande ajuda na organização e documentação do código não só para o próprio programador, como para compreensão futura.

Em Prograph e LabVIEW os comentários são cadeias de caracteres que estão associadas ao objecto (AO). Em ambos os sistemas pode-se esconder parte ou a totalidade dos comentários.

Em Hyperflow e Show and Tell os comentários são introduzidos como Texto Livre ou Desenho Livre (TDL), que não tem qualquer ligação com objectos.

Comentários	Prog	LabV	HF	ST
tipo	AO	AO	TDL	TDL
podem-se esconder?	S	S	S	N

3.2.6 Alguns pontos fracos e fortes

Descreve-se a seguir alguns pontos fracos e fortes e possiveis consequências na aceitação dos sistemas devido às limitações e restrições, bem como caracteristicas interessantes dos mesmos.

Prograph requere muita memória, sendo muito lento no carregamento, salvaguarda e compilação de ficheiros, especialmente se for usada a biblioteca para construção de interfaces ABC (Prograph's Application Builder Classes).

Prograph apresenta problemas com a salvaguarda de ficheiros quando a memória é menor que 7Mbyte. Existem ainda algumas inconsistências na parte da IHM e as funções de ajuda têm pesquisas muito lentas, que não se podem ser interrompidas.

Do ponto de vista de interface, notou-se no LabVIEW a falta do comando *undo*. Quanto às ferramentas disponíveis para interfaces, seria útil a existência de objectos para entrada e saída de dados. LabVIEW tem icons para entrada (*control*) ou saída (*indicator*) de dados, mas não para ambas as funcionalidades.

Os controladores digitais têm uma caracteristica interessante: podem-se programar para aceitar dados numa determinada gama, ignorando aqueles que não pertencem a essa gama, aproximando-os a um dos valores permitidos, ou suspendendo a execução.

Ao sairmos do LabVIEW surge uma janela com as opções de guardar (ou não) e cancelar, e ainda outra que dá a lista de modificações realizadas não guardadas e suas possiveis consequências.

O problema de Hyperflow é o reconhecimento de caracteres, que podia ser mais rápido e preciso. Duas caracteristicas interessantes de Hyperflow são o estabelecimento vários níveis de utilizador e sobreposição de "janelas" numa só, tal como se faz com transparências, permitindo a reutilização de esquemas.

O problema do Show and Tell foi o uso de rato. A simplificação dos icons (existem apenas 3 distintos e muito simples) diminui muito o tempo de aprendizagem da linguagem, no entanto pode aumentar o tempo de percepção do diagrama em alguns casos.

Tanto Hyperflow como Show and Tell são muito simples de manipular.

4. Resumo

Prograph permite uma introdução à programação O.O. mais simples e interessante do que qualquer outra linguagem textual, tal como C++. As quatro razões principais são: suporte das caracteristicas normalizadas de O.O., correspondência entre o código e diagrama de classes, funções de ajuda e depuração código fornecidas. A depuração do código com interfaces gráficas é mais simples e mais produtiva do que a simples listagem de mensagens textuais, uma vez que se observam os valores e o fluxo dos dados directamente no código do programa (diagrama). Finalmente, e tal como qualquer outra LV, torna a programação muito mais interessante e divertida, e consequentemente menos cansativa.

Prograph é um sistema muito adequado a não programadores, mas não é aconselhável para o desenvolvimento de grandes programas, por razões de desempenho. Quanto a crianças não é recomendável por duas razões: complexidade da linguagem e uso de rato.

LabVIEW tem uma grande biblioteca de funções de conveniência, que atrai engenheiros e cientistas por um lado, mas afasta estudantes de liceu e outros por outro. Isso não constitui um grande problema, uma vez que o sistema é destinado ao primeiro grupo de utilizadores. A especialização excessiva de um sistema pode reduzir o número de utilizadores, mesmo que possa ser usado numa maneira mais geral.

Usar uma linguagem próxima de outra conhecida por Eng^{os} Electrotécnicos é vantajoso para estes. No entanto a linguagem simbólica é normalmente diferente de um ramo científico para outro. Também aqui o sistema de ajuda facilita muito o desenvolvimento de programas.

Seria necessária uma análise mais cuidada para determinar a utilidade de LabVIEW, uma vez que a maioria dos programas requerem um processamento "pesado" e não foi efectuada qualquer análise quantitativa de tempos de processamento. Mas a linguagem e a interface são muito úteis e fáceis de manipular. A existência de níveis de utilizadores regidos pela dificuldade dos conceitos disponíveis ajudaria na utilização desta linguagem por estudantes em geral.

Show and Tell foi uma linguagem muito avançada no seu tempo e tem a particularidade de ser muito simples e ao mesmo tempo completa. A simplicidade de ambos os icons e opções disponíveis, e os procedimentos na programação, torna-a muito interessante e acessível a todos, principiantes ou peritos, novos ou não, ingleses ou portugueses. Experiências demonstraram que crianças são capazes de programar usando conceitos tão sofisticados como recursividade sem qualquer problema.

Não existe nenhum mecanismo para melhorar o processamento. No entanto, a sintaxe e semântica desta linguagem servem perfeitamente o seu objectivo (construção de programas por e para crianças).

O problema na aceitação deste sistema deve-se à sua interface: uso de rato. O Homem apresenta maior destresa no uso da caneta do que do rato, devido a aspectos intrinsecos da sua fisionomia.

Hyperflow resolveu os problemas de Show and Tell. Demonstrou-se que Hyperflow é potencialmente capaz de fornecer uma ferramenta uniforme para o desenvolvimento de diferentes níveis de programação, desde o desenvolvimento de IHM até à programação de aplicações e sistemas [Kimura 93b] com suporte para vários *media*. Uma aplicação em Hyperflow pode ter texto, imagem, som e animação.

Embora os utilizadores aceitem e gostem mais de usar a caneta que o rato e o teclado, a imprecisão do ecran e especialmente do reconhecedor de caracteres e a expectativa de uma maior rapidez de processamento tem afastado os utilizadores.

A sua simplicidade permite uma compreensão rápida e fácil do sistema, permitindo que seja usada por crianças. Ajuda principiantes a apreender conceitos base de programação, uma vez que vêm a execução do programa e fluxo de dados. Quanto aos outros utilizadores tem a vantagem de permitir uma rápida implementação de programas sem uso de teclado, além de não exigir conceitos base de programação.

5. Conclusão

LabVIEW é indicado para aplicações que simulem instrumentos e exijam calculos científicos. Prograph é indicado para todas as aplicações em geral com arquictetura O.O.. Hyperflow é uma linguagem, que pela sua simplicidade pode ser usada por uma gama muito maior de utilizadores, desde crianças até pessoal qualificado. Tem o poder de uniformizar vários níveis de programação e de possibilitar a extensão a computação distribuida. O uso de caneta torna-a ainda mais poderosa, uma vez que pode ser usada em situações onde o teclado não pode ser usado, como visitas médicas ou inspecções. Além disso a escrita é mais rápida. No entanto tem a desvantagem do reconhecedor de caracteres ainda não ser suficientemente rápido e preciso. Show and Tell prova como uma linguagem simples com apenas 3 icons diferentes, exceptuando a seta, pode ser completa, servindo de base para outras LV.

As LPV são bem sucedidas numa variedade de situações, quer como linguagens muito especificas quer como linguagens mais gerais, tornando o desenvolvimento das aplicações mais rápido e mais simples, sendo acessível a não programadores. Se bem que ainda há muito por descobrir nesta área, as LVP irão provavelmente ser o paradigma principal da programação ainda antes do virar do século.

Agradecimentos do primeiro autor

Gostaria de agradecer ao Professor Kimura pela orientação e toda a amizade que demonstrou durante a estadia nos E.U.A. e depois, sem a qual não existiria este artigo.

De realçar e agradecer amizade do Professor Katsuo Uota, Eng^{os} Samudra e Masheh e as criticas construtivas do Eng^{o} Rui Guerreiro.

Finalmente gostaria de agradecer a todos que de uma forma ou de outra coontribuiram para a realização do estágio nos E.U.A..

Bibliografia

[Apte 94] Ajay Apte, "Form/Formula (ff) Paradigm: A Visual Programming Approach For User Definable User Interfaces", tese de mestrado, Washington University, Saint Louis, 1994.

[Booch 94] Grady Booch, "Object Oriented Analysis and Design with applications", The Benjamin Publishing Company, Inc, Santa Clara, California, 1994.

[Brown 94] Timothy B. Brown and T.D. Kimura, "Completeness of a Visual Computation Model" Soft - Concepts and Tools, 1994.

[Burnett 93] Margaret M. Burnett, "Types and Type Inference in a Visual Programming Language", 1993 IEEE Workshop on Visual Languages.

[Caron 80] J.P. Caron et al., "Evaluating Pictograms Using Semantic Differencial and Classification Techinques", Ergonomics, vol. 23, pg 137-147, 1980.

[Chang] Shi-Kuo Chang, "Principles of Visual Programming Systems", Prentice-Hall International Editions.

[Chang 87] Shi-Kuo Chang, "Visual Languages: A Tutorial and a Survey", IEEE Soft. Jan 1987.

[Citrin 93] Wayne V. Citrin, "Requirements for Graphical Front Ends for Visual Languages", IEEE Symposium on Visual Languages, pg 142-150, Ago 1993, Norway.

[Eisenstadt 90] M. Eisenstadt, J. Domingue, T. Rajah and E. Motta, "Visual Knowledge Engineering", IEEE Trans. on Soft Engineering, vol 16(10), pg 1164-1177. Out 1990.

[Glinert] E.P. Glinert, "Towards Software Metrics for Visual Programming" Tecnical Report, n 87-16, Rensselaer Polytecnic Institute, Dept of CS.

[Greenstein 90] J.S. Greenstein and L.Y. Arnaut, "Input Devices", Handbook of Human-Comp Interaction, M. Helander publ, North Holland, Amsterdam, pg 495-519, 1990.

[Hills 92] D.D. Hills, "Visual Languages and Computing Survey: Data flow Visual Programming Languages", Jornal of Visual Languages and Computing, vol 3 n 1, 1992, pg 69-101.

[Hirakawa 94] Masahito Hirakawa and Tadao Ichikawa, "Visual Language Studies - A Prespective", ATT BSTJ, vol 15, n 2, pg 61-67, 1994.

[Kimura 86] T.D. Kimura, J.W. Choi and J.M. Mack, "A Visual Language for Keyboarless Programming", Washington University Technical Report, Jun 1986.

[Kimura 90] T.D. Kimura, J.W. Choi and J.M. Mack, "Show and Tell: A Visual Programming Language", Invited paper in Glinert EP, Editor, Visual Computing Environments, IEEE Comp Society Press, Washington DC, pg 397-404, 1990.

[Kimura 92] T.D. Kimura, "Hyperflow: A Visual Programming Language for Pen Computers", 1992 IEEE Workshop on Visual Languages, Seatle, Washington, pg 125-132.

[Kimura 93a] T.D. Kimura, "Hyperflow: A Uniform Visual Language for Different Levels of Programming", Washington University Technical Report, Mar 1993.

[Kimura 93b] T.D. Kimura, "Potencials and Limitations of Pen-Based Computers", CSC, Fev 1993.

[Kimura 94] T.D. Kimura, Samudra Sengupta and Adjay Apte, "A Graphic Diagram Editor for Pen Computers", Sotware - Concepts and Tools, 1994.

[Kimura 94] T.D. Kimura and Timothy B. Brown, "BoxGraph: A Two-Dimensional Visual Computation Model" Washington University Technical Report, Jan 1994.

[LabVIEW] G.M. Vose and G. Williams, "LabVIEW: User Manual", "LabVIEW: User Guide"

[Myers 92] A. Myers, Jones and Bartlett Publishers, "Languages for developing user interfaces", chap. 6, 1992.

[Ota 87] Y. Ota, "Pictogram design", Kashiwa Shobo Pub., Tokyo, 1987.

[Prograph] Manuais de Prograph: "Prograph User Guide", "Prograph Tutorial", "Prograph ABC Reference" e "Extensions Reference"

[Rasure 90] John Rasure, Danielle Argiro, Tom Sauer and Carla Williams, "Visual Language and Software Development Environment for Image Processing", International Journal of Imaging Systems and Techonology, vol2, pg 183-199, 1990.

[Raymond 91] D. R. Raymond, "Caracterizing Visual Languages", 1991 IEEE Workshop on Visual Languages, pg 176-182, Out 1991, Kobe, Japan.

[Sengupta 94] Samudra Sengupta, "Modularity and Abstraction in a Graphical Diagramming Environment", tese de mestrado, Washington University, Saint Louis, 1994.

[Shu 88] Nan C. Shu, "Visual Programming", 1988.

[Sutherland 66] W. Sutherland, "On-line Graphical Specification of Computer Procedures" , tese de Doutoramento, M.I.T., Cambridge, MA.

[Vose 86] G.M. Vose and G. Williams, "LabVIEW: Laboratory Virtual Instrument Engineering Workbench", Byte 11:9, pg 84-92, 1986.