INTERFACES PARA AMBIENTES DE PROGRAMAÇÃO: PROBLEMAS E PERSPECTIVAS

INTERFACES FOR PROGRAMMING ENVIRONMENTS: PROBLEMS AND PERSPECTIVES

André Luis R. G. CARVALHO*
Profª Drª Heloísa Vieira da ROCHA**

ABSTRACT

Despite the importance normally assigned to programming, it is still very difficult to learn how to write computer programs. There has always been a considerable difficulty in interating with computers. Traditionally, these difficulties are not restricted only to interating in programming enviromments. However, in other domains, one can presently observe significant changes in these aspects. These changes can be seen as a very positive response to the increasing improvement in the man-machine interaction style that has been occuring lately in these domains. Despite of the many currently, available programming tools, programming is still an activity that is essencially done in an old fashioned way. Recent publications show that for many researchers that is a very worth-studying issue and reveal many inovative proposals. These proposals are based on Artificial Intelligence (AI) research results an are related mainly with the nature of the programming activity, problem solving and human interaction in computer environments. A prelliminary observation of these proposals can show clearly that a revolution is occuring, not only in the programming environments interaction style, byt also in the paradigms that support them, causing significant changes in what is meant by programming. The objective of this article is to explore a bit these issues, allowing this way further and deeper analysis of the several aspects concerned to it.

RESUMO

Apesar da enorme importância que é atribuida à atividade de programar, aprender a programar ainda é uma tarefa muito difícil. Embora essa dificuldade de apropriação tradicionalmente nunca tenha sido restrita ao âmbito dos ambientes de programação, na atualidade podem-se observar conquistas muito significativas em outros domínios da computação, frutos de uma maior preocupação com o estilo de interação a ser mantido com o usuário. Apesar de se poder observar a disponibilização de muitas ferramentas de apoio à programação, programar ainda é uma atividade que se faz à moda antiga. Literatura recente mostra que essa questão tem constituído preocupação de muitos pesquisadores, e que propostas bastante inovadoras têm surgido. Essas propostas são baseadas em resultados de pesquisas em Inteligência Artificial (IA) quanto à natureza da atividade de programação, resolução de problemas e interação com ambientes de computação. A observação preliminar dessas propostas evidencia uma revolução não somente no estilo de interação com ambientes de programação, mas também nos paradigmas que os suportam, mudando significamente o que se entende por programar. O objetivo deste trabalho é trazer à tona toda essa problemática, de modo a servir de subsídio para um trabalho de análise dos diversos aspectos envolvidos na questão.

(*) Pesquisador do Centro de Pesquisa e Desenvolvimento da TELEBRÁS - Professor do I.I. - PUCCAMP e do CTC - UNICAMP. Mestrando em Ciência da Computação - DCC - IMECC - UNICAMP
(**) Professora do Departamento de Ciência da Computação - DCC - IMECC - UNICAMP. Pesquisadora do Núcleo de Informática Aplicada à Educação - NIED - UNICAMP.

1. INTRODUÇÃO

Todos sabemos da extrema importância da atividade de programar computadores, que não só constitui meio rico para formalização e depuração de conceitos e idéias sobre resolução de problemas, mas que em última instância, representa a única maneira possível de se ter controle pleno sobre a máquina.

Em contrapartida, qualquer um que tenha passado pela experiência de ensinar programação, pode dizer sobre quão árduo é o processo de aprendizagem, sobre quão íngreme e tortuoso é o caminho a ser percorrido. Mas afinal, porque é tão difícil aprender a programar?

Se deixarmos de lado, por um instante, o domínio da programação, e nos lembrarmos um pouco de como eram, há algum tempo atrás, as coisas neste aspecto no domínio dos aplicativos, certamente nos recordaremos da aversão pré-concebida que sempre existiu, por parte das pessoas, à simples idéia de ter que interagir com um sistema de computação.

Podemos observar que isto andou mudando e, que hoje, já não mais se verifica de forma generalizada aquela rejeição que outrora percebíamos. Naturalmente não era gratuito o estabelecimento daquela situação, assim como certamente também não o são, as mudanças que ora podemos ali verificar, o que nos leva a crer - e agora voltamos a nos lembrar de que falávamos de programação - que a trajetória evolutiva por que passou o domínio dos aplicativos tem muito para nos ensinar.

O que se verificou no domínio dos aplicativos, foi o desenvolvimento de novas formas de interação homem-computador, o que resultou em *interfaces* modernas, das quais as pessoas se apoderam com extrema facilidade. O mesmo não se deu no domínio da programação, onde poucas mudanças são percebidas, e com o qual, ainda hoje, se interage de maneira essencialmente textual.

Um exemplo ilustrativo dessa problemática, é a linguagem Logo. A linguagem Logo é fortemente calcada em objetivos educacionais (no sentido de ser uma linguagem fácil de ser aprendida), e sua "porta de entrada" tradicionalmente sempre foi seu ambiente gráfico de programação, aliado à exploração de atividades espaciais. Essencialmente, através de não mais que uma dezena de comandos e da estrutura de procedimentos, resultados gráficos extremamente interessantes são conseguidos muito rapidamente.

Se outrora suas capacidades gráficas constituíram seu principal atrativo, hoje em dia, com o surgimento de aplicativos gráficos sofisticados, com *interfaces* de última geração, muito fáceis de serem aprendidos, e que produzem resultados muito atraentes também muito rapidamente, Logo corre o risco de perder seu "carro chefe".

Tendo em vista a já discutida importância da atividade de programar, simplesmente substituir o ambiente gráfico Logo por um sofisticado editor não constitui solução aceitável. Acredita-se na necessidade de buscar novas alternativas de *interfaces* para programação, e pode-se observar em literatura recente, que esta é a preocupação de um grande número de pesquisadores.

Propostas bastante inovadoras têm sido apresentadas. De uma análise preliminar dessas propostas, podemos observar que não só no estilo de interação são necessárias mudanças, mas que aliadas a estas, surgem também novos paradigmas de programação, mudando significativamente o que se entende por atividade de programação.

Portanto, a proposta de uma nova *interface* para um ambiente de programação deve estar calcada em três estudos básicos: nas mudanças na forma de interação com o computador, nos objetivos da atividade de programar no ambiente a ser alterado, e na análise das novas propostas e paradigmas de programação que vêm surgindo como resultado de pesquisas recentes.

2. O DOMÍNIO DOS APLICATIVOS

Há bem pouco tempo atrás, não somente aprender a programar acontecia de maneira traumática. As pessoas tinham uma resistência muito grande ao uso de programas de computador de uma forma geral. Quando interrogadas sobre as razões dessa resistência, as pessoas se justificavam dizendo que não se sentiam capazes de aprender a lidar com algo "tão complicado", que se sentiam inseguras, e que tinham muito medo de cometer erros.

Certamente não podemos culpá-las. Isso reflete claramente não só o hermetismo da área, os tabus que a envolvem, as idéias mistificadas que as pessoas têm dela, mas também, e principalmente, a pouca atenção que sempre se deu à questão da qualidade da interação entre as pessoas e os sistemas de computação.

Entretanto, nos últimos anos, muitos esforços têm sido feitos no sentido de melhorar a qualidade da interação entre o homem e o computador. Podemos ver resultados desses esforços sendo colocados no mercado o tempo todo pelos fabricantes de *software*, que de certo modo, estão aprendendo a respeitar cognitivamente o ser humano que interage com seus produtos.

Assim, vimos surgirem os *menus* de função em substituição à escrita de comandos e,sempre que possível, à especificação dos dados;vimos surgirem as interfaces gráficas, onde o usuário deixa de caminhar por um *menu* em busca de opções, e passa a apontar o que deseja, e onde formas icônicas substituem a interação

textual; tudo isto levando os programas a exigirem menos da memória de seus usuários, possibilitando a ocorrência de poucos erros, e fazendo com que os usuários se sintam mais confortáveis e confiantes.

No entanto, à medida que mais poder vai-se conferindo aos programas, também mais complexos eles vão se tornando. As aplicações vão passando a englobar tantas facilidades, que se torna praticamente impossível conhecê-las e empregá-las todas. Muitos usuários se sentem confusos e, de modo geral, não conseguem dominar por completo o produto que utilizam. Apesar disso, com toda a certeza, continuarão sempre a sentir falta desta ou daquela facilidade ainda não contemplada no produto.

Por um lado, parece possível convencer-se de que não é possível fechar um conjunto de facilidades que venha a atender plenamente a todas as necessidades dos usuários; por outro lado, vemos que na tentativa de se aproximar desta cobertura total, acaba-se tendo aplicações com *interfaces* carregadas e difíceis de serem aprendidas.

Um problema de que sofre praticamente todo sistema de computação é que suas *interfaces* não são modificáveis ou ampliáveis, mesmo nas mãos de usuários experientes e que potencialmente poderiam fazê-lo (Baecker 1987). Esta limitação acaba por fadar as aplicações, mesmo aquelas que possuem uma *interface* fácil de ser aprendida e usada, a eventualmente se tornarem incômodas. Isto porque elas não crescem com seus usuários, e nem podem facilmente ser adaptadas às necessidades individuais destes.

Pode-se observar que na tentativa de minimizar o impacto desse problema, vem se tornando mais corriqueiro o uso de um recurso relativamente antigo conhecido por *macro*. Este recurso tem sido empregado nas *interfaces* modernas da seguinte maneira: o usuário coloca a aplicação em um modo de operação no qual ela registra todas as ações do usuário e as unifica em uma seqüência identificável e passível de ser ativada posteriormente pelo usuário.

Apesar de efetivamente representarem algum progresso no sentido da extensibilidade da *interface, macros* são extremamente frágeis (Lieberman 1992), pois são limitadas a repetir exatamente a seqüência de operações registrada pela aplicação, sendo via de regra muito sensíveis a detalhes irrelevantes do contexto em que foram definidas.

Existe um potencial muito grande no sentido de uma nova classe de aplicações, mais expressiva, mais ampliável e que respeite mais a imaginação do usuário (Eisenberg 1991). A idéia central consiste em se ter nas aplicações, tanto *interfaces* de manipulação direta com capacidade de aprendizado, como um interpretador para pelo menos uma linguagem de programação.

Com isto, os usuários passariam a ter acesso a certos elementos de linguagens de programação que normalmente estão ausentes das *interfaces* que atualmente existem.

Um exemplo de aplicação destinada a usuários finais que fez uma tentativa neste sentido, é o Guide 3.1. Trata-se de um sistema de autoria de *hipermidia*; foi o primeiro disponível para microcomputadores IBM e Macintosh. Um documento do Guide é chamado *guideline*, e é uma mistura de texto e gráficos. Certas palavras, frases, ou objetos gráficos, podem funcionar como botões que provêm acesso a *guidelines* subjacentes. Existem diversos tipos de botão, cada um com uma funcionalidade diferente, e.g., botões de expansão, de nota, de referência e de comando.

Botões de comando são justamente a porta de acesso à programação. Este tipo de botão deve ser associado a um programa escrito em uma linguagem de programação própria, de alto nível, bastante poderosa e muito semelhante a Pascal. O pressionamento de um botão deste tipo, tem como efeito a execução do programa que lhe foi associado.

Como dissemos acima, a linguagem usada para escrever programas é bastante semelhante a Pascal, ou seja, trata-se de uma linguagem textual. Ao construir aplicações com esta ferramenta, podemos sentir claramente a quebra de contexto que acontece, quando do ambiente de autoria de *hipermidia*, passamos para um editor de textos, a fim de escrever o texto do programa a ser associado a um botão de comando do hiperdocumento em que trabalhamos.

Enquanto nas *interfaces* modernas o usuário interage com seus objetos de interesse, num ambiente gráfico de manipulação direta, para ampliar as funcionalidades da *interface*, normalmente é requerido do usuário o domínio de uma linguagem de programação convencional, como C ou Lisp, por exemplo. A distância que existe entre estes dois ambientes constitui uma espécie de "Muro de Berlim" (Lieberman 1992), impedindo que o usuário tenha controle pleno sobre suas aplicações.

É importante ressaltar que nos exemplos que temos atualmente, ao inserirmos uma linguagem de programação no domínio dos aplicativos, com ela trazemos também toda problemática que existe no domínio da programação e sobre a qual já comentamos anteriormente.

3. O DOMÍNIO DA PROGRAMAÇÃO

Como vimos, no que diz respeito à deselitização do uso do computador, são notórias as conquistas que têm ocorrido no domínio dos aplicativos, o que nos mostra claramente que as mudanças de estilo de interação ali

promovidas são um estrondoso sucesso, e nos convida à reflexão e à análise. Surge assim o interesse de procurar promover modificações semelhantes na forma atual de interação no domínio da programação.

Sem dúvida nos ambientes de programação têm ocorrido inovações, e hoje dispomos de muitas facilidades das quais há bem pouco tempo atrás não dispunhamos. São editores sensíveis à sintaxe, que vão nos advertindo sobre construções mal elaboradas à medida que vamos introduzindo nossos programas.

São ambientes integrados de edição, compilação e execução, que apontam erros de sintaxe e de execução, permitem que sigamos nossos programas diretamente no código fonte, e que fazem a ligação de código de biblioteca automaticamente, sem que precisemos nos preocupar em especificar de que bibliotecas precisamos ou suas localizações.

São sistemas de *help on-line* sofisticados, sensíveis ao contexto e baseados em tecnologia de hipertexto, que permitem diversas formas de acesso à informação, fornecendo ajuda descritiva, procedimental, interpretativa, navegacional e orientada por objetivo, inclusive dando exemplos que podem ser cortados e colados na área de trabalho.

São linguagens não procedimentais, onde descrevemos não o procedimento a ser seguido para a obtenção de um certo resultado, mas sim o resultado que se visa obter, e a ferramenta produz o resultado desejado, seja pela interpretação direta da especificação, seja pela geração de código que o faça. Ferramentas deste tipo existem para alguns domínios de aplicação, e. g., consultas a bancos de dados, geração de relatórios e geração de *interfaces* homem-computador.

Enfim, é toda uma gama de ferramentas sofisticadas que embora facilitem a tarefa de desenvolver programas, se limitam a ajudar na depuração, em prover um acesso imediato e rápido a informações de manual ou a diminuir o volume de programação, mas que não promovem nenhuma inovação no que diz respeito ao estilo de interação com a atividade de programar. Assim, apesar da modernidade que podemos identificar neste domínio, e em geral em todas as áreas de aplicação, escrever programas ainda é uma tarefa que se faz à moda antiga.

Achamos importante que fique claro que quando falamos em mudanças, não nos referimos a esse tipo de mudanças que mencionamos acima, e que podemos observar no domínio da programação atualmente. Precisamos sim, de mudanças que façam uso qualitativo da moderna tecnologia de *interfaces*, no sentido de facilitar a apropriação por parte do usuário.

Um exemplo ilustrativo desta problemática é a linguagem Logo. A linguagem Logo foi desenvolvida por volta de 1968, e a idéia de seu projeto surgiu inicialmente com o propósito de criar uma linguagem que pudesse vir a substituir o Basic.

Assim, foi concebida para ser uma linguagem poderosa, não apenas com capacidade de processar listas e de permitir a criação de novos procedimentos, caraterísticas que não eram exibidas pelo Basic, mas fundamentalmente, que fosse de fácil aprendizado.

Originalmente, Logo não dispunha de facilidades gráficas, já que os computadores da época não possuíam capacidades desta natureza. Seymour Papert, através de intensa utilização da linguagem, e de inúmeras pesquisas, conseguiu dar ao Logo uma nova roupagem e uma estrutura filosófica (Valente 1991).

É universalmente sabido que Logo é uma linguagem fortemente calcada em objetivos educacionais, o que a leva a ser amplamente utilizada não só para introduzir programação em cursos de informática dos mais diversos níveis, mas também, e principalmente, com crianças, com o intuito de estudar os impactos da tecnologia da programação sobre o desenvolvimento cognitivo.

Suas capacidades gráficas, utilizadas principalmente para explorar atividades espaciais, historicamente têm constituído a "porta de entrada" para o aprendizado de programação, através de um contato imediato do aprendiz com o computador.

Se outrora essas capacidades representaram o principal atrativo da linguagem, hoje em dia, com o advento das *interfaces* gráficas, e de sua exploração por inúmeras aplicações sofisticadas e de fácil inserção, isso tem deixado de ser verdade, movendo o foco de interesse dos aprendizes na direção de editores gráficos e de outras aplicações do gênero, fazendo assim com que Logo perca seu "carro chefe".

Somos forçados a concordar que o "conforto" da interação provida por essas aplicações não se compara àquele do ambiente Logo, onde para obter qualquer resultado, por mais simples que seja, é preciso escrever código de programação. Por outro lado, sabemos da extrema importância da atividade de programar. Assim, nos parece claro que o atual estilo de interação com o ambiente Logo precisa ser repensado, posto que se mostra antigo e ultrapassado, o que pode tornar desmotivante seu uso.

Buscar respostas para a problemática da interação com as linguagens de programação constituem presentemente preocupação de muitos pesquisadores, que vêm apresentando propostas inovadoras para o encaminhamento da solução desta problemática.

Da observação dessas propostas, pode-se verificar que a efetividade de uma proposta está ligada a três grandes aspectos: ao domínio da programação propria-

mente dito, à tecnologia de *interfaces* homem-computador, e ao suporte de paradigmas de programação adequados. Assim, surgem com elas, não somente mudanças na forma de interagir com o domínio da programação, como também novos paradigmas de programação.

O exame de algumas dessas novas propostas e paradigmas, nos dá uma idéia do fervilhar que ocorre presentemente na área, o que coloca em clara evidência a necessidade e o interesse de pesquisa existente na mesma.

3.1. NOVOS PARADIGMAS DE PROGRAMAÇÃO

Papert coloca que os paradigmas de programação estruturam a atividade de programar, e que implicam na maneira pela qual o programador pensa sobre essa tarefa (Baranauskas 1993).

Assim, parece natural que, em resposta à demanda por mudanças no domínio da programação, surjam novos paradigmas de programação, trazendo com eles novas formas de estruturar a atividade de programar, e em última instância, novas concepções acerca desta tarefa.

3.1.1. ORIENTAÇÃO A OBJETOS

O paradigma de orientação a objetos já é um paradigma relativamente antigo, sendo hoje praticamente considerado um clássico e seu uso vem aumentando com o tempo.

Esse paradigma propõe que os programas simulem o mundo real. Assim, posto que o mundo real é constituído nem por dados, nem por processos, mas sim por coisas que o paradigma chama de objetos, propõe que estes sejam os elementos constituintes dos programas.

O paradigma envolve dois aspectos, um modelo de computação e uma filosofia de desenvolvimento.

O modelo de computação proposto, prevê um mundo todo constituído apenas por objetos, que trabalham e cooperam entre si através da troca de mensagens.

Todo objeto possui um estado interno, que é representado em uma memória local inacessível e indevassável por outros objetos. Além de um estado interno, todo objeto possui também um comportamento, que é representado por um repertório de ações de que o objeto dispõe a fim de responder a demandas (mensagens) externas ou mesmo internas ao próprio objeto.

No modelo, todo processamento é ativado pela troca de mensagens, e todo objeto, ao receber uma mensagem, passa a executar uma ação específica em reação à mensagem recebida, que poderá resultar em alterações em seu estado interno, no envio de novas mensagens, e mesmo na criação de novos objetos. Assim, um programa pode ser visto como uma sociedade de objetos que interagem trocando mensagens, e que reagem em resposta a mensagens recebidas.

A filosofia de desenvolvimento proposta prevê que os objetos tendem a exibir facetas similares, o que nos permite agrupá-los em classes, de modo a fatorar propriedades a serem herdadas por todos seus irmãos de classe. Analogamente, classes podem ser agrupadas em superclasses, etc. Assim, desenvolvimento orientado por objetos resume-se na identificação de objetos e no agrupamento deles em classes (Takahashi 1988).

Dentre os ambientes que suportam o paradigma, podemos destacar: o SmallTalk, que é considerado um ambiente clássico de orientação a objetos; os diversos ambientes C++, que são inúmeros hoje em dia; o Visual Basic, que valida o paradigma de orientação a objetos ao moderno paradigma da programação visual, sobre o qual em seguida falaremos um pouco mais, e por fim ambientes como o Turbo Pascal, que tradicionalmente não foram projetados para seguir esse paradigma, e que hoje, em suas versões mais modernas, passam a incorporá-lo. Mesmo para a linguagem Logo tem-se o Object Logo da Aplle e MicroWorlds da Logo Computer Systems Inc., que implementam orientação a objetos.

3.1.2. ORIENTAÇÃO A EVENTOS

O paradigma de orientação a eventos é um dos paradigmas que surgiram à sombra do paradigma de orientação a objetos. Entende-se por evento qualquer ação reconhecida por um objeto e para o qual se pode escrever código de tratamento.

Eventos podem ter origem em ações de usuário, podem ser provocados via programação, ou ainda, podem ser disparados pelo sistema. Assim, tudo se passa como se a sociedade dos objetos se mantivesse permanentemente à espreita, à espera da ocorrência de eventos que os sensibilizem. Quando isso ocorre, o objeto dispara uma reação em resposta ao evento e volta ao estado inicial.

Trata-se de um paradigma bastante adequado para modelar uma série grande de tipos de problemas, em especial, problemas com caraterísticas de tempo real. Tem sido sobremaneira usado em sistemas gerenciadores de *interface* homem-computador, onde os objetos são os elementos de *interface*, e.g., botões, caixas de texto, etc. Dentre os sistemas que seguem este paradigma, podemos destacar o MS Windows e o X Windows, ambos sistemas gerenciadores de *interface*.

3.1.3. PROGRAMAÇÃO VISUAL

O paradigma da programação visual também é um dos paradigmas que surgiram à sombra do paradigma de orientação a objetos. Ele pressupõe a existência de uma *interface* de manipulação direta com capacidade de manipulação de uma série de objetos com aparência e comportamento pré-definidos e/ou configuráveis, que possam ser acoplados na construção de objetos de mais alto nível.

Tem sido extensivamente usado em sistemas geradores de *interface* homem-computador, onde os objetos são os elementos de *interface*, e.g., botões, caixas de texto, etc. Seu uso neste contexto é bastante natural, posto que elementos de *interface* são de natureza totalmente visual, e é ideal que possuam aparência e comportamento pré-definidos e configuráveis.

Nesse contexto, o paradigma é normalmente usado aliado ao paradigma de orientação a eventos, que como já dissemos, é muito usado como base de sistemas gerenciadores de *interface*, que costumam ambientar não só esse tipo de aplicativo, mas ainda os sistemas cuja *interface* é gerada por eles. Dentre os sistemas que seguem este paradigma, podemos destacar o Visual Basic e o Visual C++ para ambiente PC, e o TeleUSE em ambiente de estação de trabalho, todos sistemas geradores de *interface*.

3.1.4. PROGRAMAÇÃO POR EXEMPLOS

O paradigma de programação por exemplos constitui uma inovação mais radical. A tônica deixa de ser uma forma a ser empregada na escrita de programas, posto que não mais se configura a atividade de escrever programas.

Assim, ao programar o computador utiliza-se a metáfora de "ensinar" tarefas ao computador, o que se dá através de exemplos e contra-exemplos. A interação se estabelece da seguinte forma: o usuário apresenta ao computador uma série de instâncias da tarefa que lhe deseja "ensinar", e este abstrai dos exemplos os procedimentos necessários para a realização da tarefa desejada (Bauer 1979).

Não dispomos comercialmente de nenhum ambiente de programação de propósito geral que suporte este paradigma. O que existe, são diversos protótipos de pesquisa, incluindo propostas de aplicações como editores de texto, ou editores gráficos, e que incorporam características de aprendizado através de exemplos em suas *interfaces*.

3.2. NOVAS PROPOSTAS PARA AMBIENTES DE PROGRAMAÇÃO

3.2.1. BOXER

Programação é quase sempre vista como um meio que os especialistas têm de fazer com que os computadores realizem tarefas complicadas de modo eficiente e confiável. Boxer propõe uma visão alternativa: programação como uma atividade cotidiana para a maioria das pessoas; programação como uma forma de usuários não especialistas controlarem um meio reconstrutível.

Um meio reconstrutível deve não somente servir para especialistas desenvolverem produtos profissionais, mas também atender às necessidades de pessoas comuns, e.g., crianças, professores, e outros usuários de computador não especialistas, e nele, qualquer usuário, sem distinção de nível de perícia, deve ter acesso às mesmas ferramentas de trabalho.

O meio reconstrutível deve ser programável e aberto, o que implica que mesmo produtos produzidos profissionalmente passam a ser mutáveis, adaptáveis, fragmentáveis, e compartilháveis. Assim, não somente profissionais são capazes de construir seus produtos, nele, mas qualquer um é capaz de reconstruir versões personalizadas destes mesmos produtos, o que estabelece uma situação bastante contrastante com a situação atual dos aplicativos de *software*.

Desse modo, necessidades e restrições tradicionalmente impostas às linguagens de programação, e.g., simplicidade formal, eficiência, verificabilidade, e uniformidade convivem com outras imposições, tais como ser de fácil compreensão, parecer familiar, ser expressiva, e ser extremamente interativa.

Assim, Boxer baseia-se em dois princípios fundamentais: metáfora espacial e realismo ingênuo. Todos os objetos computacionais são representados em termos de caixas, que são regiões da tela que podem conter texto, gráficos e outras caixas (o que estabelece uma estrutura de hierarquia).

Dessa forma, uma variável é uma caixa que contém seu valor; estruturas de dados compostas são caixas que contém outras caixas (como uma variável que contém outras variáveis); um programa é uma caixa que contém o texto do programa; e subprogramas e variáveis são representados como sub-caixas.

Realismo ingênuo é uma extensão do conceito *what you see is what you have*. Assim, o usuário pode pensar que o que ele vê na tela é seu mundo computacional inteiro, e.g., (1) todo texto que aparece na tela, seja ele produzido pelo sistema, entrado pelo usuário, produzido

por um programa, pode ser movido, copiado, modificado, ou, no caso do texto ser um programa, executado; (2) pode-se alterar o valor de uma variável simplesmente alterando o conteúdo de sua caixa na tela, e se uma variável tem seu valor atualizado por outros meios, o conteúdo de sua caixa automaticamente espelha isto.

Boxer é uma extensão da linguagem Logo; pequenos procedimentos Boxer, especialmente aqueles que trabalham com gráficos, se parecem muito com procedimentos Logo (diSessa 1986).

Podemos observar no Boxer o uso do paradigma de programação visual aliado ao paradigma de programação procedimental. O paradigma visual é usado na elaboração da estrutura hierárquica da aplicação, que se dá através do aninhamento de caixas; e o paradigma procedimental é usado para escrever texto de código para caixas que representam ações.

Existem versões comerciais de Boxer para computadores da linha Macintosh, da linha PC e também para estações de trabalho SUN.

3.2.2. MONDRIAN

Mondrian é um editor gráfico orientado a objetos com capacidade de "aprender" novos procedimentos gráficos através do paradigma de programação por exemplos.

O usuário pode demonstrar uma seqüência de comandos de edição gráfica através de um exemplo concreto com o propósito de ilustrar o funcionamento esperado do novo procedimento.

Um agente da *interface* grava os passos do procedimento em forma simbólica, usando técnicas de aprendizagem do domínio da Inteligência Artificial (IA), observando relacionamentos entre objetos gráficos e dependências entre operações da *interface*.

O agente generaliza então um programa que pode ser usado para resolver exemplos análogos. A heurística de generalização não é comparável às tradicionais *macros*, que se limitam apenas a repetir a exata seqüência de passos que a definiu.

Como já dissemos, o sistema utiliza o paradigma da programação por exemplos, e representa internamente todas as operações através de seqüências pictóricas de exemplos, aliando assim o poder da representação procedimental e facilidade de uso das *interfaces* gráficas. (Lieberman 1992)

3.2.3. METAMOUSE

Metamouse é um ambiente de edição gráfica onde um agente ajuda o usuário através da automatização de tarefas de edição repetitivas. O objetivo do projeto foi tornar a tarefa de programar o mais parecida possível com a tarefa de editar, com uma quantidade mínima de interação extra, com a finalidade de dirimir situações de ambigüidade.

Para especificar um procedimento o usuário constrói um exemplo, provavelmente desenhando construções gráficas, de modo a tornar as relações explícitas. O sistema generaliza então esta seqüência em um programa com variáveis. Quando o usuário repete parte da seqüência, Metamouse prevê e realiza automaticamente o restante da mesma.

O usuário pode introduzir novas ações, que tomarão a forma de ramos condicionais. O usuário pode também corrigir erros no programa através da demonstração do exemplo correto ou indicando ícones que mostrem inferências alternativas.

O usuário expressa suas intenções através da metáfora de "ensinar" uma tartaruga gráfica chamada Basil. O uso deste tipo de metáfora foi que motivou o uso de um agente e de técnicas de aprendizado incremental.

O sistema faz uso do paradigma de programação por exemplos, aliado a uma *interface* de manipulação direta. (Maulsby 1993)

3.2.4. MICROWORLDS

MicroWorlds é a mais nova versão do ambiente Logo e se encontra disponível para computadores Macintosh. O estilo de interação com sua *interface* é totalmente gráfico, de conformidade com o padrão Macintosh.

Um projeto feito neste ambiente é organizado em páginas, e nelas pode-se utilizar recursos como desenho, animação, som. etc.

Desenhos podem ser feitos através do Centro de Desenhos, onde se dispõe de recursos no estilo dos editores gráficos modernos, como o PaintBrush por exemplo. Assim, temos facilidades para desenhar formas geométricas (linhas, retângulos sólidos e vasados, elipses sólidas e vasadas, etc), para fazer desenhos a mão livre, para colorir (palheta de seleção de cores, pintura e *spray*). etc.

Pode-se realizar programação de cores, i.e., programar ações a serem disparadas quando for pressionado o botão do *mouse* com o cursor apontando uma determinada cor.

É possível se ter várias tartarugas simultaneamente em uma página, sendo permitido criar novas tartarugas e remover tartarugas dinamicamente. Toda tartaruga tem um nome, e é possível através dele endereçar comandos a elas. Tartarugas podem assumir não só a

forma de tartaruga, mas ainda diversas outras formas. Para se lidar com formas, dispõe-se do Centro de Formas, onde é possível escolher uma forma para a tartaruga, estampar a forma de uma tartaruga na página como um carimbo, editar novas formas, etc.

Pode-se fazer com que tartarugas se comportem como se fossem botões, i.e., pode-se fazer com que seja disparado um conjunto de ações quando do pressionamento do botão do *mouse* com o cursor apontando uma delas. Para tanto, devem-se programar as ações desejadas no quadro de diálogo que possui toda tartaruga. Desta forma, podem-se construir animações programando o quadro de diálogo de uma tartaruga para que ela continuamente se mova e/ou troque de forma.

Além da possibilidade de fazer com que tartarugas se comportem como se fossem botões, dispõe-se também de botões propriamente ditos, e ainda de outros elementos de *interface*, como por exemplo, caixas de texto, molduras, mecanismos deslizantes de controle, etc.

Outra ferramenta disponível no ambiente é o Editor de Melodias. Nele pode-se editar uma melodia fazendo uso dos seguintes recursos virtuais: teclado, controlador de volume, controlador de tempo de notas e pausa, e palheta de instrumentos (onde pode-se escolher o instrumento que executará a melodia editada - violino, clarineta, etc). As melodias editadas podem ser gravadas sob algum nome, e executadas tanto no próprio editor, como no decorrer de um programa, através de instruções específicas.

A todo projeto pode ser associado um programa, que pode ser escrito na página de procedimentos que todo projeto tem. Além disto, toda página pode ser impressa, e ainda pode-se importar e exportar figuras. Por fim, todos os recursos de um projeto, tais como páginas, melodias, procedimentos e sons, podem ser compartilhados por diversos projetos. ("MicroWorlds" 1993)

O sistema faz uso dos paradigmas procedimental e funcional, que suportam a linguagem Logo, bem como do paradigma de orientação a eventos a nível de *interface.* Ainda, toda interação com o sistema se dá através de uma *interface* gráfica de manipulação direta.

4. CONCLUSÃO

Apesar da enorme riqueza intrinsecamente inerente à atividade de programar, essa atividade tem sido reservada para "iniciados", e por isso, ou para isso, a despeito de já há algum tempo estar sendo feito muito esforço no sentido de melhorar a qualidade da interação com os usuários de sistema de computação, e conseqüentemente, no sentido de tornar a computação acessível a um maior número de pessoas, a mesma preocupação tradicionalmente não se observa no domínio dos ambientes de programação.

Pode-se observar resultados significativos destes esforços em praticamente todas as áreas da computação, o que culmina numa crescente deselitização da computação e conseqüentemente em sua penetração em praticamente todos os segmentos da atividade humana.

Haja vista ser o único meio que possibilita exploração total dos recursos da máquina e controle completo sobre a mesma, a importância da atividade de programar tem sido resgatada. Ambientes de programação são então trazidos para o universo dos não "iniciados", seja simplesmente como um ambiente de programação em sua forma mais pura, seja embutido na *interface* de programas aplicativos.

A aliança desses dois domínios se configura muito poderosa, mas existem "efeitos colaterais". Os ambientes de programação adentram o universo dos não especialistas e trazem consigo todos os problemas não resolvidos em seu domínio. Assim, apesar de sua disponibilidade para usuários finais, sua apropriação não ocorre.

Na pequena amostra apresentada neste texto, as propostas que vêm surgindo apontam para muitos sentidos diferentes. Faz-se mister colecionar, analisar e compilar as propostas que vêm surgindo, de modo a identificá-las e a lhes apreender os fundamentos, a aplicabilidade e a abrangência, gerando assim a cultura necessária à obtenção de resultados.

Concluindo, a efetiva deselitização da atividade de programar exige mudanças importantes nesta atividade. Esta questão tem sido do interesse de muitos pesquisadores, que têm feito propostas inovadoras no sentido de tornar programação acessível em âmbito mais geral.

Pode-se observar que essas propostas se apóiam significativamente em resultados de IA com relação aos mecanismos de aprendizagem e aos aspectos cognitivos envolvidos na atividade de programar.

O que se objetiva é empregar as modernas tecnologias de projeto de *interfaces* nos ambientes de programação, de modo a minimizar o esforço mental requerido do usuário na aprendizagem da atividade de programação e no desempenho dessa atividade.

REFERÊNCIAS

(Bae87) Baecker,R. M.: and Buxton,W. A. S., "Research Frontiers and Unsolved Problems" in **Readings in Human-Computer Interaction: A Multidisciplinary**

Approach. Baecker, R. M.; and Buxton W.S. eds., Morgan Kaufmann, California, 1987.

(Bar93) Baranauskas, M. C. C., "Procedimento, Função, Objeto ou Lógica? Linguagens de Programação vistas pelos seus paradigmas" in **Computadores e Conhecimento - Repensando a Educação**. Valente, J. A. org., NIED-UNICAMP, Campinas 1993.

(Bau79) Bauer, M. A., "Programming by Example" in **Artificial Intelligence**, 12:1, May, 1979.

(dA86) diSessa, A. A.,and Abelson, H., "Boxer: A Reconstructible Computational Medium" in **Communications of the ACM**, 9:29, September, 1986.

(Eis91) Eisenberg, M., "Programmable Applications: Interpreter meets Interface".MIT Laboratory for Computer Science, 1991.

(LCS93) "MicroWorlds How To". Logo Computer Systems Inc., 1993.

(Lie92) Lieberman, H., "Mondrian :A Teachable Graphical Editor" in **Visible Language Workshop**. MIT Media Laboratory 1992.

(Mau93) Maulsby, D., and Witten, I. H., "Metamouse: An Instructible Agent for Programming by Demonstration" in **Watch What I Do: Programming by Demonstration**. , A.; Halbert, D.C. ...(et al) eds. The MIT Press, Cambridge, Massachussets; London, England, 1993.

(Tak88) Takahashi, T., "Introdução à Programação Orientada a Objetos", III EBAI-Escola Brasileiro-Argentina de Informática, 1988.

(Val91) Valente,J. A., Logo: mais do que uma linguagem de programação" in **Liberando a Mente - Computadores na Educação Especial.** Valente, J. A. org., NIED - UNICAMP Campinas, 1991.