

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

RENY CURY FILHO

**INTERFACES GRÁFICAS REFERENCIALMENTE CLARAS
E SUA UTILIZAÇÃO NA CRIAÇÃO DE LABORATÓRIOS PARA A
APRENDIZAGEM A DISTÂNCIA *ON-LINE***

**Uberlândia
2008**

RENY CURY FILHO

**INTERFACES GRÁFICAS REFERENCIALMENTE CLARAS
E SUA UTILIZAÇÃO NA CRIAÇÃO DE LABORATÓRIOS PARA A
APRENDIZAGEM A DISTÂNCIA *ON-LINE***

**Tese apresentada ao Programa de Pós-graduação em
Engenharia Elétrica da Universidade Federal de
Uberlândia, como parte dos requisitos para obtenção
do título de Doutor em Engenharia Elétrica.**

Área de concentração: Processamento da Informação.

Orientador: Professor Dr. Luciano Vieira Lima.

**Uberlândia
2008**

Dados Internacionais de Catalogação na Publicação (CIP)

- C982i Cury Filho, Reny, 1955-
Interfaces gráficas referencialmente claras e sua utilização na criação de laboratórios para a aprendizagem a distância on-line / Reny Cury Filho. - 2008.
84 f. : il.
- Orientador: Luciano Vieira Lima.
Tese (doutorado) – Universidade Federal de Uberlândia, Programa de Pós-Graduação em Engenharia Elétrica.
Inclui bibliografia.
1. Ensino a distância - Teses. 2. Engenharia de software auxiliada por computador - Teses. 3. Linguagens de programação (Computadores) - Teses. I. Lima, Luciano Vieira. II. Universidade Federal de Uberlândia. Programa de Pós-Graduação em Engenharia Elétrica. III. Título.

CDU: 37.018.43

RENY CURY FILHO

**INTERFACES GRÁFICAS REFERENCIALMENTE CLARAS
E SUA UTILIZAÇÃO NA CRIAÇÃO DE LABORATÓRIOS PARA A
APRENDIZAGEM A DISTÂNCIA *ON-LINE***

**Tese apresentada ao Programa de Pós-graduação em
Engenharia Elétrica da Universidade Federal de
Uberlândia, como parte dos requisitos para obtenção
do título de Doutor em Engenharia Elétrica.**

Área de concentração: Processamento da Informação.

Uberlândia, 18 de junho de 2008

Banca Examinadora

Prof. Dr. Luciano Vieira Lima – FEELT/UFU

Prof. Dr. Gabriel Wellesley de Barros Ferreira - IFETTM

Prof. Dr. Pedro José Abrão - CEFET-GO

Prof. Dr. Waldenor Barros Moraes Filho – ILEEL/UFU

Prof. Dr. Adriano Alves Pereira – FEELT/UFU

Dedico este trabalho às crianças e adolescentes,
homens e mulheres do amanhã.

AGRADECIMENTOS

Agradeço a Deus pela oportunidade, a minha família pelo incentivo, aos professores Eduardo Costa e Luciano Lima pela amizade e aos amigos do laboratório de IA pela convivência.

“... a criança não é uma tábula rasa, que pode ser descrita exteriormente como um recipiente vazio e que deve ser preenchido com matéria estranha, para se conseguir algo; mas sim como uma força real, viva e autônoma, que já no primeiro instante da sua existência age coordenando organicamente seu próprio desenvolvimento e expansão...”

Johann Heinrich Pestalozzi.

RESUMO

Este trabalho mostra a importância da Educação a Distância no que diz respeito à aprendizagem a distância *on-line*. Trata-se de discutir as possibilidades desse tipo de aprendizagem: a credibilidade das plataformas de educação a distância, o uso de metáforas na criação de interfaces gráficas referencialmente claras, a construção de programas educativos utilizando bibliotecas gráficas e linguagens de programação eficientes e, ainda, a proposta de uma plataforma de aprendizagem a distância *on-line* monitorada pelo aprendiz. Conclui-se que a utilização de laboratórios a distância é uma forma de quebrar barreiras, de criar condições para que o ensino *on-line* possa ser um sistema educacional que seja aceito e promova uma aprendizagem efetiva. Além disso, a proposta de criação de uma plataforma de aprendizagem a distância *on-line* monitorada pelo aprendiz pode ser uma importante contribuição para a melhoria do processo ensino-aprendizagem.

Palavras-chave: educação a distância, aprendizagem a distância *on-line*, interface gráfica do usuário, laboratório *on-line*, sistema de gestão da aprendizagem, linguagens de programação, engenharia de software, *software* educativo, metáfora, aprendiz.

ABSTRACT

This study demonstrate the importance of Distance Education as respects on-line distance learning. It intends to discuss the possibilities of this type of learning: the credibility of distance education platforms, the use of metaphors in the creation of referentially explicit graphic interfaces, the construction of educational programs utilizing graphic libraries and efficient program languages and, further, the proposal for an on-line distance learning platform monitored by the apprentice. It concludes that the utilization of distance laboratories is a way of breaking down barriers, creating conditions that make on-line teaching an educational system that will be acceptable and will promote efficiente learning. Besides this, the proposal to create an on-line distance learning plataform monitored by the apprentice can be an important contribution toward improving the teaching-learning process.

Palavras-chave: distance education, on-line distance learning, graphical user interface, on-line laboratory, learning management system, programming languages, software engineering, educational software, metaphors, apprentice.

LISTA DE FIGURAS

Figura 1 - <i>Google</i> – Mecanismo de busca.....	1
Figura 2 - Circuito para aquisição de dados - Temperatura.	11
Figura 3 - Interfaces antes do surgimento das GUI.....	13
Figura 4 - Metáfora do Painel de Controle.....	19
Figura 5 - Metáfora do Menu	20
Figura 6 - Metáfora do Formulário e Documento	21
Figura 7 - Metáfora da Console e do Diálogo Serial	22
Figura 8 - Caso de Uso da Plataforma Padolma.....	26
Figura 9 - Funcionalidades da plataforma Padolma disponíveis para o aprendiz.	30
Figura 10 - Funcionalidades da plataforma PADOLMA – lado servidor.....	31
Figura 11 - Um aprendiz pratica usando um <i>Cessna Flight Simulator</i>	31
Figura 12 - Estudantes trabalham em uma maquete para avião.	32
Figura 13 - Painel criado para ensino do Superlogo, baseado na Metáfora do ADC.....	41
Figura 14 – APLIP Laboratório de Aprendizagem de Programação <i>On-line</i>	42
Figura 15 – APLIP: esquema das áreas.	43
Figura 16 - CLEAN - programa usando biblioteca OPENGL.	46
Figura 17 - OCaml - programação usando biblioteca TK.....	49
Figura 18 - Programa em OCaml que faz desenho utilizando a Biblioteca TK.	49
Figura 19 - Arquivos do pacote TCL/TK necessários para execução.	51
Figura 20 - BIGLOO – GUI do programa stm.scm usando biblioteca GTK.	53
Figura 21 – Programa stp.scm (Bigloo/GTK) – Jogo que demonstra o uso do mouse.....	55
Figura 22 – Arquitetura da Biblioteca CAW.....	58
Figura 23 - Animação em CLEAN utilizando biblioteca JAPI.....	58
Figura 24 - Pacote LEMAC, disponibilizado no site surceforge.net	62
Figura 25 - Pacote STALINGUI, disponibilizado no code.google.com	62
Figura 26 - Pacote CAW, disponibilizado no code.google.com	63

LISTA DE FONTES DE PROGRAMAS

Listagem 1 - Algoritmo de Comunicação em Bigloo (Scheme)	11
Listagem 2 – Programa em Clean com interface gráfica	37
Listagem 3 – Programa em OCaml que cria Interface Gráfica de forma simplificada	38
Listagem 4 - Listagem do programa mundos.icl em Clean	48
Listagem 5 - Fonte do programa em OCaml com a Biblioteca TK.....	51
Listagem 6- Fonte do programa em Bigloo stm.scm com a biblioteca GTK.....	54
Listagem 7 - Fonte do programa stp.scm em Bigloo com a biblioteca GTK.....	56
Listagem 8 - Fonte do programa Clean com a biblioteca CAW.	59

SUMÁRIO

1. INTRODUÇÃO.....	1
2. CREDIBILIDADE DA APRENDIZAGEM A DISTÂNCIA.....	5
2.1 Fundamentos	5
2.2 Tutoriais On-line.....	7
2.3 Vivenciando experiências	8
2.4 Um experimento: Ilhas de Calor.....	9
2.5 Laboratórios a distância.....	12
3. INTERFACES GRÁFICAS.....	13
3.1 Dificuldades para o usuário final.....	14
3.2 Dificuldades para o desenvolvedor	15
3.3 Bibliotecas para execução de <i>Softwares</i>	17
3.4 Requisitos necessários às Interfaces Gráficas.....	18
4. METÁFORAS.....	19
4.1 Painel de controle.....	19
4.2 Menus	20
4.3 Formulários e documentos	21
4.4 Console e diálogos seriais	22
5. PLATAFORMA DE APRENDIZAGEM A DISTÂNCIA ON-LINE MONITORADA PELO APRENDIZ.....	24
5.1 Gerenciamento da utilização feita pelo aprendiz	26
5.2 Programas disponibilizados para uso do aprendiz	27
5.3 Inserção de programas na plataforma	27
5.4 Protocolos da plataforma	28
5.5 Árvore de Conhecimento.....	29
5.6 Construindo programas para a plataforma Padolma	29
5.7 Kits para uso do aprendiz	31
5.8 Repositório.....	32
6. APLICAÇÕES GRÁFICAS	34
7. LABORATÓRIO PARA APRENDIZAGEM A DISTÂNCIA	39
7.1 Execução de Experimentos a Distância.....	39
7.2 Aprendizagem de Linguagem de Programação	40
7.3 Interface gráfica ADC	41
7.4 Interface Gráfica para o Ensino do Logo	42
8. LINGUAGENS, BIBLIOTECAS E PACOTES	44
8.1 Linguagens de Programação e Bibliotecas	44

8.2	Uso do Clean com a biblioteca gráfica Opengl	44
8.3	OCaml utilizando a biblioteca gráfica TK.....	48
8.4	BIGLOO utilizando a Biblioteca gráfica GTK	51
8.5	Utilização da biblioteca GTK pelo Bigloo.....	52
8.6	CLEAN utilizando a biblioteca JAPI.....	57
8.7	Clean com a biblioteca gráfica CAW	58
8.8	Pacotes.....	60
8.8.1	Componentes do pacote	61
<i>CONSIDERAÇÕES FINAIS</i>		64
<i>REFERÊNCIAS</i>		66
<i>GLOSSÁRIO</i>		70

1. INTRODUÇÃO

O *Google*, uma das maiores empresas de *software* da atualidade, detentora de uma gama de prêmios (GOOGLE, 2007), tem o seu ponto forte no mecanismo de busca, o qual procura por páginas existentes na *Internet* que, encontradas, são catalogadas e disponibilizadas para consulta.

Entende-se que o que está por trás do valor do *Google* e de outros *sites* de busca é a sua capacidade de fornecer informação de maneira rápida, fácil e gratuita. Ao se pesquisar no *Google* por *Lorem Ipsum* encontramos aproximadamente 5.550.000 páginas referenciadas, conforme pode ser visto na Figura 1. Outra pesquisa feita com a palavra “biologia”, por exemplo, trouxe como resultado aproximadamente 27.900.000 páginas.

É fácil encontrar qualquer tipo de assunto, mas quem vai garantir a veracidade dos conteúdos disponibilizados nas páginas apresentadas? Não existem pessoas designadas para funcionarem como revisores de todas as páginas existentes na *Internet*. Ao realizar uma pesquisa para aprender determinado conteúdo, pode-se obter uma grande quantidade de páginas com erros de conteúdo, *links* inexistentes e outros mais. As páginas encontradas também podem ser verdadeiras armadilhas, pois ao se clicar na página pode-se ter o computador infectado por vírus de computador.

A Figura 1, a seguir, mostra a página do mecanismo de busca do *Google*.

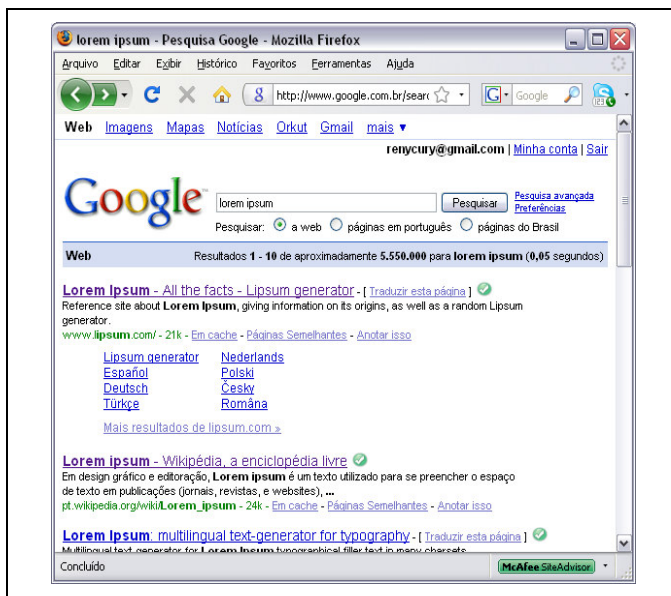


Figura 1 - *Google* – Mecanismo de busca.

O *Google* tem seu foco na informação digital, surgida com o aparecimento da *Internet*. A cada dia apresenta novas propostas de *softwares* que procuram agilizar a vida do usuário. Contrariamente a todas as perspectivas empresariais, o *Google* foi pensado para concorrer com as grandes empresas da época, como *Yahoo* e *Altavista*, que dominavam os mecanismos de busca. A criação de um diferencial foi fundamental para o crescimento do *Google*. Mas apesar de tudo, percebe-se que o conteúdo disponibilizado hoje na *Internet* é totalmente fragmentado. Os conteúdos relacionados à educação também não estão agrupados em um único lugar o que não permite se possa encontrar:

- Conteúdos organizados por assunto ou conceito;
- conteúdos certificados;
- programas que garantam uma aprendizagem eficaz do conteúdo desejado;
- programas que façam a avaliação durante a aprendizagem.

Assim como a *Internet* possibilitou ao *Google* uma oportunidade de crescimento, ela, segundo Moran (2005), começa a provocar mudanças profundas na educação presencial e na aprendizagem a distância. Gadotti (2000), corroborando Moran, afirma que, com o advento da *Internet* surgiram novas perspectivas educacionais que já promovem grandes mudanças na forma como se aprende atualmente. Como pode ser visto em Chaves (1999), uma das grandes mudanças é o surgimento do Ensino a Distância, também conhecido como Educação a Distância (EAD) ou Aprendizagem a Distância (destaque-se que existem diversas correntes a favor de um ou de outro termo).

A fim de que as idéias veiculadas neste trabalho possam ser mais bem compreendidas se faz necessário esclarecer o porquê da utilização de determinadas palavras e quais significados são mais adequados ao que se quer transmitir.

As diferenças entre as palavras *aluno*, *estudante* e *aprendiz* podem parecer, à primeira vista, insignificantes, mas *aluno* e *estudante* pressupõem a necessidade de um professor, que será responsável por transmitir o conhecimento; *aprendiz* é utilizado, neste trabalho, para designar uma pessoa que aprende de acordo com o seu interesse, tendo inclusive a autonomia para definir quem poderá saber o que conhece. O *aprendiz* será a pessoa que irá utilizar os programas disponibilizados na plataforma.

Quanto à palavra *aprendizagem*, ela foi utilizada ao invés de *educação* e de *ensino*, por significar um processo no qual o aprendiz é o ator principal que vai definir o quê, o porquê, o onde, o como e quando aprender. A aprendizagem pode ser entendida como sendo a forma através da qual os seres humanos adquirem novos conhecimentos e desenvolvem ou

aprimoram suas competências. Qualquer que seja a definição que se dê, ela estará baseada na visão do homem, de como ele aprende e na sociedade onde ele vive.

Diversos questionamentos podem ser feitos sobre a *Aprendizagem a Distância*, dentre estes se podem destacar:

- Ela é garantia de uma aprendizagem eficiente, confiável e aceita em todos os países?
- Tem como promover aprendizagem utilizando a *Internet* e ao mesmo tempo avaliá-la e registrá-la?
- Tem como o aprendiz gerenciar a sua aprendizagem?
- Existe alguma forma de Aprendizagem a Distância cujo foco não seja o professor?
- É viável o uso intensivo de laboratórios na Aprendizagem a Distância?
- Existem parâmetros que possibilitem o uso de interfaces gráficas simplificadas na construção de programas para serem utilizados na Aprendizagem a Distância?

Estes questionamentos são objetos de pesquisa de diversos grupos de pesquisadores interessados no tema. Como pode ser visto em Rived (2007), o Ministério da Educação e Cultura do Brasil (MEC) tem um sítio que nos dá uma perspectiva de uma das vertentes educacionais que está sendo trabalhada no âmbito do governo federal:

“... a produção de conteúdos pedagógicos digitais, na forma de objetos de aprendizagem. Tais conteúdos primam por estimular o raciocínio e o pensamento crítico dos estudantes, associando o potencial da informática às novas abordagens pedagógicas. A meta que se pretende atingir disponibilizando esses conteúdos digitais é melhorar a aprendizagem das disciplinas da educação básica e a formação cidadã do aluno. Além de promover a produção e publicar na *web* os conteúdos digitais para acesso gratuito”.

Programas e mais programas são hoje disponibilizados a todos os tipos de usuários, mas o que eles têm em comum? A maioria deles tem funções e mais funções agregadas a cada versão, ou seja, os programas estão cada vez mais complexos, mais difíceis de se utilizar. Onde está a simplicidade e a clareza das interfaces?

A Aprendizagem a Distância, o *software* educativo, a Internet, a convergência de mídias e outras novas possibilidades tecnológicas são hoje motivo de estudos pelas mais diversas áreas e podem ainda serem considerados "jovens", tendo um largo caminho a percorrer para serem consolidados.

É importante ressaltar que esta tese irá investigar a possibilidade de se utilizar laboratórios a distância em plataformas de Aprendizagem a Distância; de avaliar a possibilidade de se criar um novo olhar que permita se tenha mais credibilidade para com a

Aprendizagem a Distância; de se definir a criação de parâmetros para a construção de interfaces gráficas; de se analisar linguagens modernas e bibliotecas eficientes para a construção de programas que corroborem a Aprendizagem a Distância.

A proposta geral desse estudo é mostrar a importância da Aprendizagem a Distância no que diz respeito ao atendimento de demandas, à credibilidade e uso de ferramentas, programas e utilização de laboratórios.

São objetivos específicos desse trabalho, mostrar a importância:

- da utilização de interfaces gráficas referencialmente claras na construção de *softwares* educativos;
- da construção de bibliotecas gráficas que auxiliem na elaboração de *softwares* educativos;
- do uso de laboratórios remotos a distância para a aprendizagem a distância *on-line*.
- da presença de kits (materiais a serem trabalhados pelo aprendiz) para que os aprendizes possam construir seus próprios experimentos;
- de se ter uma plataforma de aprendizagem a distância *on-line* monitorada pelo aprendiz;
- da distribuição de pacotes de distribuição (*softwares*, bibliotecas, tutoriais e outros) a todos os interessados em construir novos programas educativos.

Para uma melhor organização desse trabalho, optou-se por dividi-lo em oito capítulos. O primeiro, esta introdução, apresenta algumas considerações sobre o assunto e os objetivos do trabalho. No capítulo dois, é feita uma avaliação sobre a credibilidade da aprendizagem a distância e uma proposta sobre o uso de laboratório na aprendizagem a distância *on-line*. A seguir, no capítulo 3 sobre as interfaces gráficas é feita uma descrição da evolução das Interfaces e de como elas podem facilitar ou dificultar a construção e utilização de programas pelos usuários. No capítulo 4 sobre as Metáforas é feita uma descrição daquelas mais utilizadas na área de informática. No capítulo 5 é discutida a criação de uma plataforma de aprendizagem a distância *on-line* monitorada pelo aprendiz. No capítulo 6 é discutido a importância das aplicações gráficas. A seguir, no capítulo 7, intitulado “Laboratório para Aprendizagem a Distância” é construído um exemplo de um laboratório para o ensino de programação. Finalmente é apresentada uma forma de se documentar e disponibilizar os programas, em forma de pacotes de distribuição, para que outros programadores possam deles fazer uso para a confecção de novos programas.

2. CREDIBILIDADE DA APRENDIZAGEM A DISTÂNCIA

Neste capítulo discutem-se as críticas feitas aos sistemas de Aprendizagem a Distância e são apresentadas sugestões que permitem avançar na construção de novas propostas para a sedimentação desta tecnologia educacional, o que permitirá se tenha uma maior credibilidade deste sistema junto à comunidade educacional.

Incontri (1996) nos mostra que Pestalozzi mais do que Comenius, insiste em colocar o sujeito em contato direto com o objeto de aprendizagem. Para Pestalozzi, há sempre uma intermediação humana no processo de aprendizagem. Com o advento do computador, são abertas novas perspectivas para se pensar a educação. Existem novos pensares sobre a possibilidade da aprendizagem a distância *on-line*. A possibilidade de se pesquisar novas formas de interatividade entre o homem e o computador é urgente. A globalização faz com que o conhecimento se torne, cada vez mais, a moeda de maior valor para um estado. Este, para ser adquirido, necessita de que o homem tenha contato direto com experiências concretas, se possível, realizadas no mundo real.

Destaque-se que o trabalho em laboratório é parte fundamental para o aprendizado das Ciências Naturais. Nas Ciências Humanas, o trabalho prático previne erros em projetos. Por exemplo, para que aprendizes possam reconstruir a história de registros sedimentares, eles têm que examinar registros paleobiológicos; coletar amostras ou examinar amostras disponíveis em repositórios; têm que fazer observações e, muitas vezes, utilizar sistemas de Coleta de Dados.

Poucos sistemas de Educação/Aprendizagem a Distância lidam com a observação, com a prática, com a coleta de amostras e com experimentos realizados em laboratório. A falta de experiências e conhecimentos práticos permite críticas aos sistemas de Educação/Aprendizagem a Distância. Por isso, muitas instituições que oferecem graduação à distância não têm seus cursos reconhecidos em muitos países. Pode-se avaliar melhor esta temática nos próximos parágrafos.

2.1 Fundamentos

Diversas pesquisas na *Web* foram feitas à procura de Sistemas de Ensino/Aprendizagem a Distância oficialmente certificados nos Estados Unidos, Canadá, e Austrália. Guidetoonlineschools (2007) concentra diversas informações sobre programas de graduação *on-line* e aprendizagem a distância. Neste e em outros sítios pesquisados

encontram-se análises que mostram que a maioria dos Sistemas de Ensino/Aprendizagem a Distância das instituições desses países limitam suas atividades aos ramos do comércio, do direito, dos estudos religiosos e serviços humanos. Nos Estados Unidos existem pouquíssimos cursos de Engenharia autorizados pela Comissão de Educação e Treinamento a Distância (*Commission of the Distance Education and Training Council* - DETC). Esta comissão é reconhecida pelo Departamento de Educação Norte-Americano como uma agência de certificação.

Os catálogos das instituições que oferecem este tipo de educação registram apenas programas de cursos associados, o que significa que seus alunos trabalharão sob a supervisão de um Engenheiro ou Mestre em Engenharia.

Os críticos do Ensino/Aprendizagem a Distância, os provedores de Educação *On-line* e as comissões de certificação concordam que as engenharias, as ciências naturais, e outros assuntos nos quais o trabalho em laboratório é imperativo, requerem a residência na graduação (cursos presenciais).

O racional é que a aprendizagem a distância não pode fornecer a experiência prática necessária para algumas competências. Por outro lado, não existe nenhuma contestação de que o trabalho e a prática em laboratório são necessários ao ensino das ciências. Corroborando este pensamento, o Conselho Federal de Biologia do Brasil CFBIO (2008) argumenta que “não há como formar um biólogo para atuar na área de ambiente sem ter vivenciado experiências e situações que as atividades laboratoriais e de campo podem oferecer” e por isto resolve:

Vetar expressamente o registro perante os Conselhos Regionais de Biologia dos portadores de diplomas dos egressos dos cursos de Educação a Distância (EAD) em Ciências Biológicas e/ou Biologia e do Programa Especial de Formação Pedagógica de Docentes (Lei Nº 9.424, de 24/12/1996).

O que se pretende refutar é que a aprendizagem a distância não possa fornecer o indispensável para que os aprendizes possam apreender os conceitos necessários da Natureza. Rudde (1979 e 2007) escreve sobre as muitas razões que possibilitam o uso de experimentos em laboratório em cursos de Ciência, quais sejam:

- para demonstrar conceitos e princípios da Ciência;
- para ensinar métodos e atitudes científicas;
- para manter o assunto fundamentado na realidade.

Outras razões que Rudde não contemplou em seus escritos são discutidas por Dawkins (1998) em seu polêmico livro e examinadas em detalhes por Matteoli (2004):

Nós podemos então escrever um manual de como martelar um prego, com todos os detalhes relacionados de como pregar um prego, como a profundidade, o som, a resistência etc. O manual... pode ser útil para reduzir o ciclo de aprendizagem, mas se você comparar a habilidade de um carpinteiro que pregue 100 pregos, mas não leu o manual, com a habilidade de outro que leu somente o manual, mas nunca golpeou um prego, você verá que o que aprendeu fazendo é muito mais capaz do que o que aprendeu somente pela leitura.

Dawkins (1998) declara que aprender através da experiência prática cria um tipo de conhecimento digital, enquanto a linguagem tende a ser analógica. O carpinteiro experiente presta atenção a metas que significam tudo ou nada. Por exemplo: uma dessas metas poderia ser a de ter a cabeça do prego nivelada com a prancha. Não importa qual seja a causa para o sucesso da aprendizagem através da experiência prática, o que realmente importa para o que se tem como propósito é que a Aprendizagem a Distância não seja excluída das pesquisas a serem realizadas.

Outra crítica negativa e recorrente sobre o Ensino/Aprendizagem a Distância é que a exigência de residência coloca o estudante em contato com assuntos correlatos. Por exemplo, o conhecimento de Latim não é fundamental para um botânico ou um advogado. Entretanto, isso é tão útil que a maioria dos botânicos e advogados chegam ao ponto de adquirirem algum conhecimento dessa língua, se a escola onde estudam tiver oferecido aulas opcionais. Muitas pessoas que elaboram programas de aprendizagem a distância não estão especialmente interessadas pelos Clássicos, é provável que elas não acrescentem aulas de Latim ao currículo. O resultado é um estudante com uma formação muito estreita. Estas críticas são rebatidas a seguir no intuito de mostrar as possibilidades concretas da Aprendizagem a Distância.

2.2 Tutoriais On-line

Nada é mais fácil do que postar um tutorial na World Wide Web (WWW). Não se fala sobre a qualidade do material: criar um bom texto é sempre difícil. Todavia, há vinte anos, se alguém quisesse oferecer um curso de, diga-se, Filosofia Romana, ele/ela precisaria de um lugar para os estudantes se encontrarem, imprimiria o material do assunto de interesse, e assim por diante. Hoje em dia tudo o que um professor precisa para oferecer um curso é conhecer o assunto. Utilizando a Internet o professor terá o mundo inteiro para procurar os estudantes para preencher sua classe.

Na Internet têm-se disponíveis milhares de cursos, bons livros estão prontamente disponíveis para serem lidos, a maioria deles com notas de gramática e tradução interlinear.

Assim, Latim ou qualquer conhecimento auxiliar está mais facilmente acessível *on-line* do que em uma universidade ou colégio. A conclusão é que um estudante que faz um curso de Direito ou Botânica pode aprender Latim tão bem como um jovem matriculado no Departamento de Clássicos de Cornell.

2.3 Vivenciando experiências

Há dois argumentos principais em defesa das exigências para graduação, mas novas tecnologias tornaram ambos sem sentido. O primeiro argumento é que o aluno deve obter informação e conselho de seu/sua professor (a), da equipe escolar e através das bibliotecas. Esse argumento não resiste a um exame minucioso. Afinal poucas bibliotecas físicas são tão boas quanto às modernas bibliotecas virtuais e uma comunicação aluno/professor pode ser alcançada através de salas de bate-papos, *webcams* e fóruns. Enciclopédias *on-line* são bem completas e com uma riqueza de detalhes que competem com bons livros-textos. Existem também muitos livros-textos *on-line*. Finalmente, não há nada que impeça um estudante de comprar livros, eletrônicos ou não, em livrarias *on-line*, em vez de visitar uma loja de compras. Em um relatório para o Presidente dos EUA, Reddy et al. (2001) declaram:

Nós estamos especialmente satisfeitos por enviar o relatório “Digital Libraries: Universal Access to Human Knowledge”, por causa da profunda relevância dessa tecnologia em promover a qualidade da educação em cada escola, nos centros de aprendizagem e nas residências do país.

O outro argumento contra a Aprendizagem a Distância é que os aprendizes precisam de prática, trabalho em laboratório e ensaio. O trabalho em laboratório e a aquisição de dados podem ser executados com vantagem ao utilizar um computador. Com exceção de algumas áreas como a dança, educação física e artes marciais há, também, a necessidade de contato físico entre os praticantes. Os resultados coletados podem alimentar um repositório na *World Wide Web*; onde um professor ou o computador pode então usar os dados para verificar o trabalho dos aprendizes. Destaque-se a existência de duas abordagens básicas para equipar laboratórios para aprendizagem a distância. Na abordagem mais popular, conhecida como Experimentação Remota, os estudantes têm disponíveis robôs controlados remotamente conforme Cohen et al. (2002). Uma exemplificação dos procedimentos para usar as instalações de Cristalografia Macromolecular da Stanford University é descrita por Gonzalez (2007) a seguir:

- Descubra a pessoa a ser contatada para seu experimento, consultando a agenda do pessoal de suporte.

- Faça o *download* e instale o *software* livre *NX client*. Verifique que o cliente de NX possa ser usado com sucesso para acessar os computadores da SMB antes da experiência.
- Usuários que utilizarão pela primeira vez o controle remoto devem fazer antes um treinamento com o orientador anteriormente designado.
- Leia cuidadosamente as instruções para preparar e enviar amostras para SSRL. Preste especial atenção ao seguinte:
 - Só use modelos de conectores que são compatíveis com o robô de SSRL (SAM).
 - Conheça as práticas de segurança do SSRL para controlar nitrogênio líquido.
 - Substitua o nitrogênio líquido no recipiente de carga assim que qualquer gelo se forme.

Há muitas instalações de laboratório baseadas em robôs disponíveis na *WEB*, além das Instalações do *Macromolecular Crystallography de Stanford*. Por exemplo, a Universidade Federal de Santa Catarina oferece o Laboratório de Experimentação Remota, segundo Bosco (2007) para práticas simples à distância. Também há suporte por parte de empresas comerciais para a prática de experimentação à distância.

Em uma primeira abordagem para prover laboratórios e aquisição de dados para eventos de Aprendizagem a Distância, o experimento está em um sítio distante da área do aprendiz. Em uma segunda abordagem, o experimento está perto do aprendiz. Essa segunda abordagem é preferível, uma vez que aprendizes que usam somente experiências remotas em sua formação têm freqüentemente medo de manusear máquinas e equipamentos. Há vendedores que fornecem materiais educacionais e equipamentos para K-12 (designação norte-americana para educação primária e secundária) cobrindo áreas como Biologia, Química, Ciências da Terra e Físicas. Se os estudantes, ainda jovens, começarem a lidar com equipamentos de laboratório, eles estarão qualificados para lidarem com diferentes tipos de experimentos. Então, as experiências projetadas para K-12 são muito importantes em programas de aprendizagem a distância. Na seqüência, algumas considerações são apresentadas sobre este tipo de experimento.

2.4 Um experimento: Ilhas de Calor

Professores envolvidos com aprendizagem a distância devem que projetar trabalhos de laboratório que aproveitem a localização do estudante, do ambiente físico ou social. Um exemplo explicará melhor este ponto. Muitos estudantes vivem em grandes cidades. Alguém poderia pensar que esses estudantes não precisam de aprendizagem a

distância, desde que há escolas convencionais disponíveis para eles. Contudo, bons colégios nem sempre estão localizados em grandes centros urbanos. Além disso, escolas K-12 são frequentemente localizadas em locais perigosos em muitos países como Estados Unidos, Brasil, Paquistão, etc. Desse modo, é proposto um experimento para estudantes que vivem em uma grande cidade.

Uma ilha de calor urbana é uma área da cidade que é mais quente do que seus arredores. Esse fenômeno afeta diretamente o bem estar dos moradores da cidade. Somente nos Estados Unidos centenas de pessoas morrem a cada ano por causa do calor extremo. Um aprendiz residente na cidade certamente sentirá o fenômeno, e amiúde mostrará entusiasmo com a probabilidade de estudá-lo. Alguém poderá estudar a ilha de calor através das imagens infravermelhas prontamente disponíveis por satélite. Neste tipo de imagens, cada cor representa uma temperatura diferente; contudo, o aprendiz pode construir um mapa que mostre a relação entre cor e temperatura. O aprendiz poderá obter o mapa coletando as temperaturas em diferentes locais de sua cidade, comparando o resultado com a cor da mesma localização presente na imagem. Tecnicamente, esse tipo de observação é chamado de verdade terrestre.

Na Figura 2 é mostrado o diagrama de um equipamento usado para coletar informações sobre verdade terrestre para o estudo das ilhas de calor urbanas. Esse equipamento é de fácil construção e baixo custo (menos de vinte dólares a unidade). Uma vez ligado ao computador do aprendiz, o equipamento coleta dados da verdade terrestre e os envia via WEB para ser utilizado. Por ser muito utilizada no universo das Ciências da Computação a linguagem *Scheme* foi utilizada para exemplificar o algoritmo de comunicação mostrado na Listagem 1

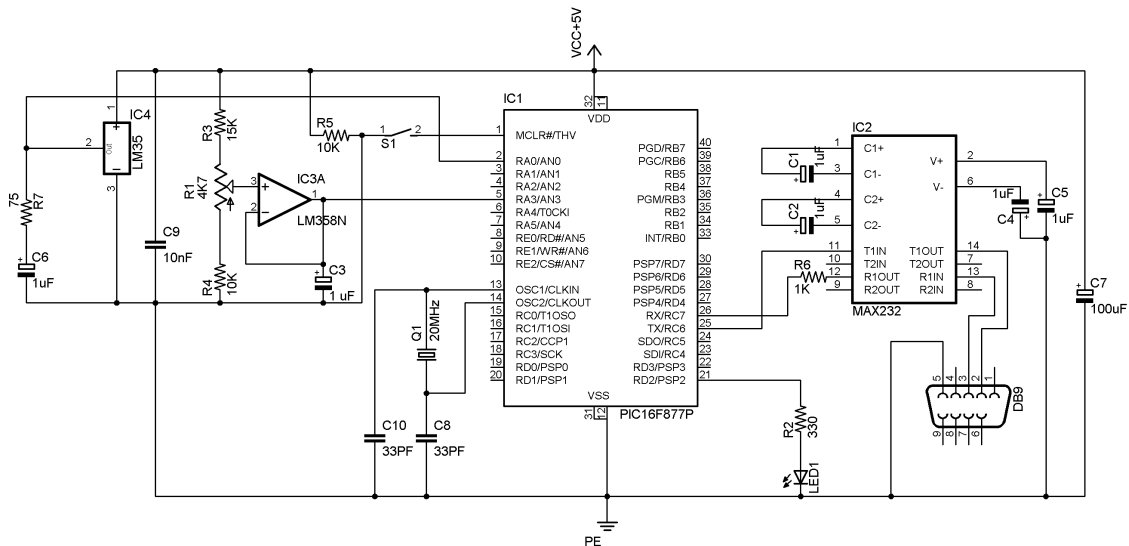


Figura 2 - Circuito para aquisição de dados - Temperatura.

O diagrama-exemplo mostra o diagrama de um *hardware* bem simples utilizado em um experimento, cujos resultados podem ser usados em estudos reais sobre aquecimento global. A verdade terrestre pode ser coletada por estudantes de Nairóbi, Jacarta, São Paulo, Cidade do México, etc. O aprendiz pode construir sozinho o *hardware* da Figura 2 e escrever o programa da Listagem 1.

```
(let*
  ( (pout (open-output-file "/dev/ttyS0"))
    (pin (open-input-file "/dev/ttyS0"))
    (s (make-client-socket
        "10.23.63.139" 8080: buffer #f))
    (p (socket-output s)))

  (write-char #\a pout)
  (flush-output-port pout)

  (let ((ans (read-chars 19 pin)))
    (display ans p)
    (newline p)
    (flush-output-port p) )
  (read-line)
  (socket-shutdown s))
```

Listagem 1 - Algoritmo de Comunicação em Bigloo (Scheme)

2.5 Laboratórios à distância

A conectividade proporcionada pela Internet não somente aumentará o número de assuntos disponibilizados *on-line*, mas também criará uma tendência para programas de graduação oferecidos *on-line* que ultrapassaram em número os tradicionais. Afinal, lições *on-line* melhoram a atenção do aprendiz, e programas de graduação a distância são economicamente atraente, desde que eles não desloquem as pessoas e não lhes exija que mudem para alojamentos. Considerando que a tendência é para que programas de graduação à distância se tornem cada vez mais comuns, os professores/conteudistas têm que desenvolver a tecnologia necessária para prover trabalho de laboratório para aprendizes, não importa onde eles estejam.

Conforme pode ser visto em Vide (2004) laboratórios de controle remoto já são bem comuns, e o alcance das aplicações vai dos genomas até a climatologia. Pode-se dizer que grande parte dos cientistas de hoje trabalham distantes dos laboratórios. Por exemplo, a maioria dos biólogos utiliza repositórios de genomas, ou dados de taxonomia. Climatologistas e astrofísicos trabalham com experiências de satélite executadas por equipamentos e pessoas de apoio. Então, pode-se concluir que os aprendizes terão um ganho ao lidar com as Ciências Naturais utilizando experimentos remotos.

3. INTERFACES GRÁFICAS

Os *mainframes*, computadores enormes, reinaram absolutos até o final dos anos 70. Por serem caros, somente algumas poucas empresas podiam investir alguns milhões de dólares para tornar mais eficientes alguns processos internos e melhorar o fluxo de informações. Segundo o *Computer History Museum*, o primeiro PC - computador pessoal foi o Kenbak-1, lançado em 1971. Tinha 256 bytes de memória e foi anunciado na revista *Scientific American*. Todavia, não possuía CPU e era projetado para uso educativo (ou seja, demonstrar como um "computador de verdade" funcionava). Surge em 1975, o Altair 8800, um PC baseado na CPU Intel 8080. Inicialmente foi vendido através da revista norte-americana *Popular Electronics* como um kit de montar.

Nos anos seguintes, surgiram dezenas de novos computadores pessoais como o Radio Shack TRS-80 (o TRS-80 foi comercializado com bastante sucesso no Brasil pela Prológica com os nomes de CP-300 e CP-500), Commodore 64, Atari 400 e outros com sucesso moderado.

Conforme pode ser visto na Figura 3, a tela dos computadores pessoais era escura ou esverdeada. Em 1984 surge o Macintosh, o primeiro computador de sucesso com uma interface gráfica direcionada ao usuário, usando ícones, janelas e mouse. No princípio, as Interfaces Gráficas do Usuário - GUI (*Graphical User Interfaces*) eram de fácil utilização.

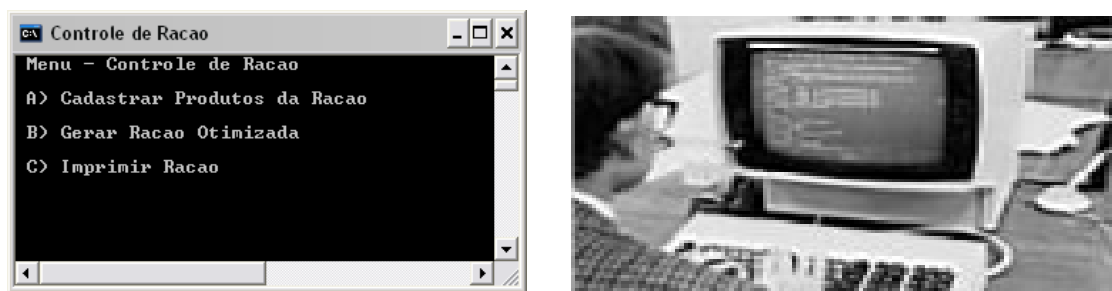


Figura 3 - Interfaces antes do surgimento das GUI

Com o tempo, as GUI começaram a sofrer pressão daquilo que os americanos chamam de *craving for features*, ou seja, um desejo intenso por novas características. Usuários finais começaram a preencher listas de solicitações, com propostas de recursos que, na opinião deles, tornariam a interface mais prática.

Antes das interfaces gráficas, os usuários finais utilizavam uma console para controlar o computador por meio do teclado.

Se o usuário escolhesse a opção **A** e apertasse a tecla **Enter**, a aplicação leria a chave do registro a ser incluído, e apresentaria um novo menu com as opções da modificação.

Com a introdução das interfaces gráficas, a tela do computador passou a ser povoada de janelas cheias de dispositivos de controle denominados *widgets* (elementos gráficos). Entre estes dispositivos figuram botões, menus que se abrem, árvores de opções, campos de edição, pastas, etc. Todos esses *widgets* são ativados pelo *mouse* ou outro dispositivo apontador.

Com a necessidade dos usuários pelos menus controlados por teclado, o surgimento das teclas aceleradoras foi inevitável. Muitos usuários pediram as consoles de volta, pois achavam que, em muitas situações, digitar comandos era uma forma mais direta, rápida e segura de controlar a máquina do que procurar em janelas e pastas a aplicação ou o *widget* que se desejava disparar. Com isto, as interfaces gráficas receberam não apenas consoles, mas também macros e linguagens de *script* em tudo semelhante às velhas linguagens de processamento *batch*. Interessante notar que os programas mais populares da atualidade utilizam as interfaces gráficas apenas para chamar uma console. Mesmo assim, os usuários consideram as interfaces gráficas mais amigáveis. Um exemplo concreto deixa esse ponto mais claro: o usuário utiliza um menu ou um ícone na barra de ferramentas para disparar um programa de bate-papo. Acontece que o programa de bate-papo é uma console, em nada diferente daquelas em uso no início da década de 80.

3.1 Dificuldades para o usuário final

Em resumo, as interfaces gráficas modernas tornaram-se de difícil utilização, pois os fabricantes começaram a dotá-las de uma profusão de *widgets* e recursos redundantes. Por exemplo, para disparar um editor de textos, pode-se usar:

- **Ícone** - que pode ser do *desktop*, de uma *toolbox*, de uma árvore de navegação, de uma janela de pasta, de um navegador, da barra de *status*, etc.
- **Menu** - que pode ser o menu **Iniciar**, um menu *popup* disparado com o botão direito do *mouse*, o menu de janela de pasta, o menu do navegador, etc.
- **Console** - que pode ser do sistema, ou disparada de um aplicativo de busca.
- **Diálogo** - como é o caso do diálogo **executar** do Windows.

Vale ressaltar que não é fácil, para o usuário final, aprender todas essas opções; também não é fácil encontrar a sequência correta de ativações devido à grande quantidade de *widgets*, menus, ícones, e diálogos.

Finalmente, o comportamento inconsistente dos diversos dispositivos de controle leva, além de um aprendizado difícil, a frequentes desastres cibernéticos. Por exemplo, eis algumas das formas de compactar uma pasta:

- Chamar o programa compactador da linha de comando, usando como argumentos o nome da pasta a ser compactada e o nome do arquivo resultante da compactação;
- abrir um diálogo do programa compactador e apertar a sequência correta de botões, ícones, árvores de navegação, etc;
- arrastar a pasta até o ícone do compactador. Com frequência, o usuário arrasta a pasta não para o ícone do compactador, mas para o programa de criptografia. Nesse caso, a pasta torna-se inacessível! Aliás, basta que se arraste a pasta para outra pasta, para a qual não tenha direitos de acesso, para que a pasta arrastada fique inacessível.

O problema aqui está nas diferentes semânticas da ação de arrastar a pasta. De fato, podemos estar arrastando a pasta para transferi-la, compactá-la, gravá-la, criptografá-la, etc.

3.2 Dificuldades para o desenvolvedor

Se o comportamento caótico das interfaces gráficas torna sua utilização difícil para o usuário final, elas provocam um trabalho hercúleo para o desenvolvedor da aplicação: vão em contraposição ao que diz Nielsen (1993), que define os cinco atributos que caracterizam o nível de facilidade de uso de um sistema:

- **Aprendizagem:** o sistema deve ser fácil de aprender de forma que o usuário possa rapidamente obter resultados.
- **Eficiência:** o uso do sistema deve ser eficiente, uma vez que o usuário aprendeu a usá-lo, será possível obter um alto grau de produtividade.
- **Memorização:** o sistema deve ser fácil de recordar, para que um usuário eventual seja capaz de retornar ao sistema sem ter que reaprender tudo novamente, depois de algum período sem utilizá-lo.
- **Erros:** o sistema deve ter uma baixa taxa de erros, de forma que os usuários cometam poucos erros ao usá-lo e, caso os cometam, sua correção seja simples, pois não devem ocorrer erros catastróficos.

- **Satisfação:** o sistema deve ser agradável de usar, de modo que os usuários se sintam satisfeitos ao utilizá-lo.

Considerando que uma das aplicações da programação é pedagógica, já que vários pedagogos e filósofos da Educação consideram o exercício da programação saudável, as dificuldades de organizar as interfaces gráficas dificultam o uso do computador em aplicações educacionais. De fato, mesmo programadores tarimbados têm dificuldades em criar interfaces. A abertura de uma janela exige dezenas de parâmetros que controlam processos; criação de identificadores de dispositivos, descrição de componentes, funções restauradoras de aspecto, posicionamento de componentes, grades, etc.

Para facilitar a programação, duas classes de ferramentas foram criadas. A primeira é a programação visual de interfaces e a outra são os objetos de Entrada/Saída. Na programação visual, utilizam-se ferramentas que criam ícones, *containers* e diagramas que representam os vários componentes da interface. Essas ferramentas permitem que o projetista construa uma representação gráfica de como será a interface. A grande falha da programação visual é que ela é de difícil documentação, ou seja, ela impede a descrição precisa da estrutura e do funcionamento da interface. A programação visual também é dependente da história do desenvolvimento da aplicação. Em outras palavras, o resultado da programação visual depende dos passos e da ordem da montagem efetuada pelo desenvolvedor: se um módulo ou um componente for acrescentado na ordem errada, nem a interface, nem o programa funcionarão. O que é pior: não existem ferramentas simples e de fácil uso para documentar a construção de interfaces visuais.

Em geral, interfaces visuais são difíceis de aprender, de utilizar e, principalmente, documentar. Paradoxalmente, elas passam a impressão de tornar fácil a implementação de aplicações GUI. A razão para o paradoxo é que a criação de uma aplicação básica em interface visual, ou seja, abrir um *frame* e arrastar componentes para ela é muito fácil. Em consequência, os estudantes entusiasmam-se e encantam-se com interfaces visuais muito facilmente, desconhecem os problemas de documentação e manutenção que enfrentarão no futuro. Aliás, provavelmente não enfrentarão, pois a maioria dos profissionais ignoram e não tentam resolver tais problemas.

Quanto aos objetos de Entrada/Saída, utilizados em linguagens como *Java* e *Clean*, elas permitem uma boa documentação e manutenção. Entretanto, as ferramentas baseadas em objetos de Entrada/Saída exigem uma programação prolixa e enorme esforço de

memória, pois o programador precisa descrever em texto não apenas a estrutura da interface, mas cada aspecto do comportamento dinâmico.

É interessante notar que a obtenção e processamento do conteúdo da interação entre a interface e o usuário final é difícil e tem uma curva de aprendizado abrupta, tanto no caso da programação visual quanto nos objetos de Entrada/Saída. A maneira de obter os dados digitados em um campo de edição (*edit field*) e utilizá-los em um botão não é intuitiva, assim como o processamento de eventos com funções de *callback*.

A utilização de bibliotecas gráficas pode de alguma forma contribuir para a construção de GUI? O seu uso é acessível a todos os desenvolvedores?

3.3 Bibliotecas para execução de *Softwares*

A necessidade de se utilizar bibliotecas gráficas é um problema que tem de ser enfrentado por quem vai utilizar interfaces gráficas. Tal problema não se limita à portabilidade dos aplicativos entre fabricantes de diferentes interfaces. Mesmo ao se limitar a um único fabricante, existem pelo menos quatro sistemas de interfaces diferentes; e em um único sistema operacional de uma instalação típica há, no mínimo, dez sistemas de interfaces diferentes. O desenvolvedor deve obter as bibliotecas de desenvolvimento da interface de sua escolha e instalá-las. Esta tarefa não é simples, e até programadores tarimbados têm dificuldades em realizá-la, mas como o programador é um profissional, pode-se admitir que ele tenha obrigação de saber como resolver tal classe de problemas.

Vale ressaltar que a pior fraqueza das interfaces gráficas tradicionais é que até o usuário final precisa instalar bibliotecas para executar a aplicação final. Novamente, mesmo limitando-se a um único fabricante, o usuário precisa instalar outras bibliotecas além da distribuída com a máquina.

Essas bibliotecas exigem o *download* e instalação, às vezes, de dezenas de *megabytes* de código. Hoje, com a banda larga (*Internet* rápida), talvez isto não seja problema, mas mesmo assim, a instalação de bibliotecas não é trivial e freqüentemente apresenta problemas, pois falha em definir o caminho/endereço (*path*). Há também o problema da existência de diferentes versões, incompatíveis, da mesma biblioteca. Por isso, a instalação de uma biblioteca gráfica de acesso direto para executar uma dada aplicação pode tornar inoperante uma outra aplicação, que exige uma versão mais antiga da mesma biblioteca.

Quanto ao usuário final, ele também encontra problema em descobrir, na *Internet*, o sítio no qual as bibliotecas são distribuídas. Esse problema é agravado porque várias bibliotecas têm distribuído somente o código fonte, e é necessário descobrir quem as distribui em binário, visto que a compilação, realizada com ajuda de *scripts* de configuração e *makefiles* não são triviais nem mesmo para profissionais de informática.

3.4 Requisitos necessários às Interfaces Gráficas

Uma interface gráfica que pretenda resolver os problemas expostos deve possuir os seguintes requisitos:

- Compacta, ou seja, ela não deve aumentar significativamente o tamanho da aplicação. Deve também ser portátil de um sistema operativo para outro e não exigir instalação de bibliotecas além das básicas que já estão instaladas no sistema operativo;
- fácil de programar, aumentando a produtividade do desenvolvedor;
- fácil de aprender a programar para quem não é profissional de informática, para que seja utilizada em educação auxiliada por computador e no desenvolvimento mental de estudantes de diversas áreas;
- fácil de aprender a utilizar para usuários que querem executar aplicações finais.

Para que se possa entender melhor como construir interfaces gráficas que privilegiem o usuário, se faz necessário conhecer um pouco mais sobre as metáforas. O próximo capítulo descreve e exemplifica algumas das metáforas mais utilizadas na informática.

4. METÁFORAS

Poppendieck (2003), na introdução de seu livro, destaca os problemas da Engenharia de *Software*, que podem ocorrer quando se faz uso incorreto das metáforas. O entendimento das metáforas é importante para uma escolha correta.

A seguir serão analisadas algumas metáforas utilizadas na criação de interfaces gráficas – GUI, que, essas para serem de fácil utilização, precisam fazer uso de situações e objetos com os quais o usuário final esteja familiarizado. Por exemplo, a maioria das pessoas sabe utilizar equipamentos, máquinas elétricas e eletrônicas. Então, uma boa metáfora é a do painel de controle. Outras metáforas possíveis e comuns são aquelas baseadas na utilização de documentos, formulários, console e diálogos seriais.

4.1 Painel de controle

Um painel de controle possui usualmente botões, teclado e mostradores (*displays*). Por exemplo, um televisor pode possuir um controle remoto, que nada mais é do que um teclado com botões para selecionar canais, para regular o volume, para ajustar a luminosidade, para ligar ou desligar, etc. Outros equipamentos de larga utilização que contêm painéis de controle são: celulares, rádios, aparelhos de som, *dvd player*, impressoras, teclados musicais, fornos de microondas, telefone, etc. Pode-se, então, afirmar que as pessoas estão familiarizadas com painéis de controle e com seus componentes ativos, que são: botões, teclado e mostradores. Existem outros tipos de componentes, como o dial e ponteiros, cada vez menos utilizados com o domínio crescente da eletrônica digital. Na Figura 4 é mostrado um exemplo de um objeto do mundo real e sua metáfora.



Figura 4 - Metáfora do Painel de Controle

Uma interface gráfica baseada na metáfora do painel de controle deve ter, no mínimo (e idealmente manter-se neste mínimo para melhorar a curva de aprendizado e destreza do usuário), botões, teclados e mostradores. O usuário entra com dados nos

mostradores, que são chamados de campos (*edit fields*). A entrada de dados pode ser realizada por meio do teclado normal, ou por meio de um teclado virtual. O teclado virtual faz sentido no caso em que a entrada de dados é pequena. Evidentemente, no caso de grandes quantidades de dados, o usuário pode recorrer a arquivos de texto, ou a pesquisa em bases de dados. De qualquer forma, os três tipos de componentes citados devem ser suficientes para uma boa interface baseada na metáfora do painel de controle.

4.2 Menus

Outra metáfora provada e comprovada é constituída de menus e árvores de opções. Em restaurantes e em outras situações da vida em sociedade, as pessoas acostumaram-se com menus, que são listas de opções onde são realizadas as escolhas dos itens disponíveis, o que pode ser visto na Figura 5. Então, não se pode ficar admirado de que a metáfora do menu tenha se tornado ubíqua nas interfaces gráficas. O menu evita que o usuário tenha de digitar o comando em um campo de edição, com as conseqüentes possibilidades de erro. Por isto, os menus tornaram-se complementares das interfaces baseadas em painel de controle e são utilizados sempre que o número de opções de comando é pequeno o bastante para se enquadrar no paradigma do menu.

Cumpra-se notar que os menus são comuns também em aparelhos eletrônicos, como tocador de DVD e televisão. A metáfora do painel de controle compreende botões, teclados, mostradores e menus. Árvores de seleção são menus organizados em forma de árvore, para facilitar a localização de um item. Em um computador, as árvores de seleção são freqüentemente utilizadas para navegar entre pastas de armazenamento de arquivos.

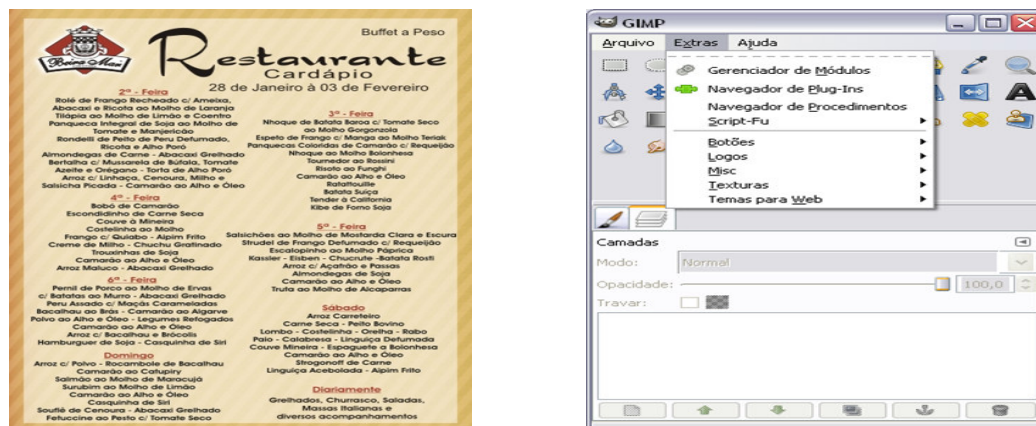


Figura 5 - Metáfora do Menu

4.3 Formulários e documentos

Documentos, cadastros, questionários e formulários são utilizados pela maioria das pessoas. Afinal, é difícil encontrar pessoas que ainda não tenham preenchido formulários de imposto de renda, cadastros de crédito, pesquisas de opinião e questionários médicos. Então, é natural que uma interface em forma de formulários esteja entre as mais populares da atualidade, conforme pode ser visto na Figura 6. Esta popularidade é, em grande parte, consequência da *Internet*, onde o sistema de comunicação serial de grandes volumes de dados não permite, no estágio atual da tecnologia, a utilização ampla da metáfora do painel de controle. Em consequência, a comunicação pela *Internet* é realizada em grande parte por formulários, documentos, questionários, etc.

Figura 6 - Metáfora do Formulário e Documento

Dois componentes fundamentais nas interfaces baseadas na metáfora do formulário/documento são o editor de texto e o analisador léxico/sintático (*parser*). Já os documentos, utilizados na comunicação com a máquina ou para recebimento/envio de mensagens por rede, podem ser de três tipos:

- documentos de formato livre, escritos em uma linguagem formal ou natural. Neste caso, a interpretação do documento é realizada por analisadores léxicos, sintáticos e semânticos construídos com técnicas linguística;
- formulários, que são documentos com campos fixos, para entrada de dados formatada. Nas interfaces que utilizam a metáfora do documento, devem-se utilizar máscaras para digitação, de modo a evitar que o usuário cometa erros no preenchimento.

- hipertextos, ou seja, documentos organizados através de *links*, com o objetivo de facilitar a navegação. Interfaces baseadas nesse tipo de documento foram amplamente divulgadas pela *Internet*, e são as fontes de consulta mais utilizadas atualmente.

4.4 Console e diálogos seriais

Considerando que os seres humanos se comunicam por meio de diálogos seriais, realizados em línguas humanas como o grego clássico (a citação se deve ao fato dos gregos terem sido os primeiros a notar esta característica da comunicação humana e serem os pioneiros no estudo das propriedades do diálogo), uma das primeiras metáforas em que se pensou quando surgiu o problema da comunicação homem/máquina foi a **metáfora do diálogo serial**.

Muito antes da construção dos primeiros computadores, os homens já haviam criado máquinas para facilitar a comunicação serial entre eles. Uma dessas primeiras máquinas foi o telégrafo, o qual pode ser visto na Figura 7, que permitia o diálogo serial entre dois operadores, que utilizavam uma codificação especial, conhecida como Código Morse, em honra a seu inventor Samuel Morse. Os antigos persas e os franceses utilizavam também um sistema ótico de telegrafia, baseado em sinais visuais realizados em torres sob visão direta uma da outra, conforme explica o romancista Alexandre Dumas, no livro *O Conde de Monte Cristo*.



Figura 7 - Metáfora da Console e do Diálogo Serial

Tanto o telégrafo de Morse quanto o dos antigos persas apresentavam o inconveniente de exigirem longos e extenuantes treinamentos dos operadores. Então, não demorou muito para que alguém imaginasse um sistema de teletipo, em que os diálogos eram impressos em uma fita, usando a linguagem corrente (um tanto quanto abreviada por razões

de concisão e praticidade, como a linguagem dos adolescentes de hoje na *Internet*) e o alfabeto latino. O teletipo foi desenvolvido a partir do telegrama, e por isto foi o precursor das interfaces baseadas em documentos.

Tão logo o computador foi inventado, o teletipo foi utilizado para a comunicação com as novas máquinas. Na console, a interface utilizada é a metáfora do teletipo e da comunicação serial humana. O homem e a máquina dialogam por meio de uma sequência de comandos e respostas digitadas em uma janela com rolagem, conhecida como console.

Considerando que essa metáfora é a que mais se aproxima da comunicação humana, era de se esperar que ela se tornasse imensamente popular. Entretanto, como o computador ainda não domina as complexidades da língua humana, a comunicação por console deve ser realizada por meio de linguagens formais. Pouca gente consegue dominar estas linguagens formais e, como resultado, a metáfora da console tornou-se apenas um apêndice complicador da metáfora do painel de controle. Porém, consoles ainda são a principal metáfora da interface de programas imensamente populares, como o Matlab, o Maxima e os programas de bate-papo. É possível, quando os problemas de comunicação serial forem resolvidos, que os computadores do futuro utilizem interfaces baseadas na console.

5. PLATAFORMA DE APRENDIZAGEM A DISTÂNCIA ON-LINE MONITORADA PELO APRENDIZ

Ensino a Distância, Educação a Distância ou Aprendizagem a Distância: designações que levam a uma série de questionamentos, uns a favor e outros contra. Dentre todas as perspectivas existentes será focada neste capítulo uma proposta alternativa centrada na Aprendizagem a Distância *On-line* cujo ator principal será o aprendiz.

Em geral, as plataformas de Ensino a Distância, pagas ou gratuitas, têm todo o controle feito pelos coordenadores, professores e ou tutores. O estudante é simplesmente um mero participante: não define regras de permissões de acesso às suas atividades realizadas, não escolhe qual conteúdo irá fazer e muito menos quem pode saber o que ele sabe.

É apresentada neste capítulo a Plataforma de Ensino a Distância *On-line* Monitorada pelo Aprendiz, denominada de Padolma. Nela não existe o controle, pelo coordenador, professor e/ou tutor, do que o aprendiz vai aprender, estudar, pesquisar, desenvolver, construir, etc. Ela se propõe a ser um caminho alternativo para que as pessoas, de acordo com a sua vontade, possam construir e registrar o seu conhecimento teórico/prático. O conhecimento registrado é de total controle do aprendiz: ele pode disponibilizar o seu currículo para os pais, para as instituições não governamentais, para as escolas, para as empresas, para governos, etc.

A plataforma é um ambiente administrado pelos aprendizes (qualquer pessoa que se inscreve para aprender). Os pais, as escolas, os amigos, governos e outros somente terão acesso ao conhecimento do aprendiz se este o permitir. A plataforma por ser centrada no aprendiz lhe permite definir os passos do seu caminhar. Os registros de todo o caminhar do aprendiz nunca serão apagados. Cada aprendiz terá um código único e válido para realizar todas as atividades escolares e administrativas a ele pertinentes.

Os programas poderão ser disponibilizados por qualquer pessoa ou instituição cadastrada na plataforma. O foco da plataforma é a aprendizagem a distância *on-line* focada no uso de laboratório presencial ou remoto. O aprendiz tem na plataforma uma forma de realizar a aprendizagem pelo uso de experimentos presenciais ou remotos, podendo ou não necessitar de kits. Cada programa é responsável pela avaliação e registro do aprendizado obtido pelo aprendiz. A validação do aprendiz, definida por níveis de segurança, é feita por módulos específicos e fornecidos pela plataforma, devendo ser incluídos nos programas disponibilizados. Outro módulo que os programas têm que incluir é o que possibilita a interação entre os aprendizes. Os programas devem manter os textos, frases e palavras em

arquivos individuais a fim de permitir a tradução para outros idiomas. Cada programa tem um registro do tempo de validade do conteúdo. Uma vez esgotado este tempo, o conteúdo terá que ser novamente refeito pelo aprendiz.

Quanto aos programas, eles poderão ser disponibilizados de duas formas: como domínio público e outro para uso restrito de quem o disponibilizou. No caso de uso restrito existirá uma pessoa (que foi a responsável pela inclusão do programa) para gerenciar quais aprendizes poderão ter acesso ao programa. Daí para frente à responsabilidade pela divulgação do conhecimento continua sendo do aprendiz.

Todo programa em domínio público terá os códigos fontes, bibliotecas e outros disponibilizados, em forma de pacote, para *download*. Isto possibilita que outras pessoas possam utilizá-los como modelo a fim de construir novos programas para os mais diversos conteúdos. Os programas antes de serem disponibilizados serão analisados quanto aos conteúdos. A classificação de um programa é associada a um dado conteúdo. Os conteúdos e programas associados são disponibilizados através de uma estrutura denominada árvore do conhecimento. Esta árvore serve para que o aprendiz escolha o conteúdo que deseja aprender. A árvore do conhecimento tem como base a folha, que representa cada conteúdo específico.

As principais características apresentadas pela plataforma Padolma são descritas a seguir:

- Gerenciamento dos conteúdos aprendidos feita pelo aprendiz;
- Programas disponibilizados para uso do aprendiz;
 - experimentos presenciais (com ou sem kits);
 - experimentos remotos (com ou sem kits);
 - realizam a avaliação do conteúdo.
- Inserção de programas na plataforma;
 - analisados antes da liberação;
 - possibilitam uso em diversos idiomas;
 - podem ser de domínio público/uso restrito;
 - autoria do programa/analista disponibilizada;
 - prazo de validade.
- Protocolos de comunicação programa-plataforma;
 - registro da avaliação;
 - validação do aprendiz;
 - interação entre aprendizes.
- Árvore do conhecimento;

- busca de conteúdo pela Árvore;
- programa designado para um dado conteúdo base (Folha);
- registro do tempo de validade do conteúdo.

Durante os próximos tópicos serão feitas as considerações necessárias para que se possa ter uma melhor compreensão de cada item descrito acima.

5.1 Gerenciamento da utilização feita pelo aprendiz

A utilização da plataforma é definida pelo próprio aprendiz, que escolhe o conteúdo, o programa e o momento para realizar o aprendizado. O aprendiz pode também interagir com outros aprendizes/tutores da plataforma quando e se o desejar. Um dos pontos que mais determinam o foco no aprendiz é a possibilidade de escolha que lhe é dada para determinar qual conteúdo de seu domínio poderá ser do conhecimento de terceiros.

Uma visão geral da plataforma Padolma é mostrada na Figura 8. Para uma melhor visualização da plataforma foi utilizada a descrição de um caso de uso, padrão UML.

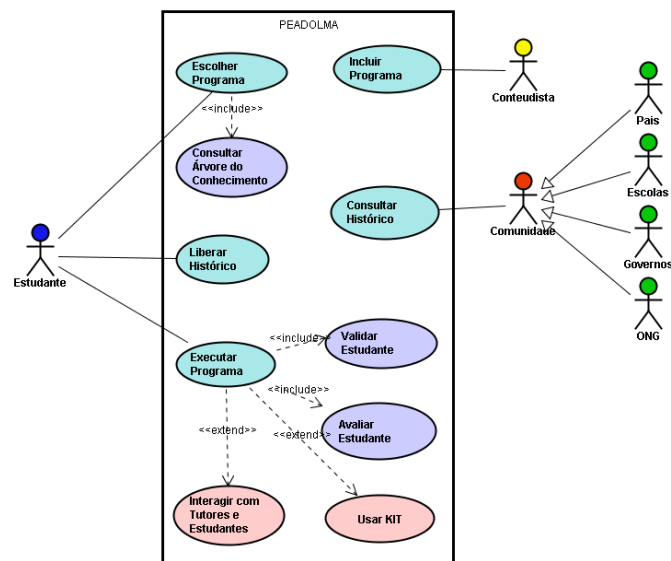


Figura 8 - Caso de Uso da Plataforma Padolma

Uma síntese das funções que o aprendiz pode realizar na Plataforma Padolma é mostrada a seguir:

- escolher o conteúdo a ser aprendido, consultando a Árvore do Conhecimento;
- escolher um ou mais programas disponibilizados para aquele conteúdo;
- poder fazer ou refazer determinado conteúdo quantas vezes o desejar;

- determinar, se o desejar, o momento de interagir com outros aprendizes/tutores;
- escolher o momento para realizar os experimentos definidos pelo programa;
- ter conhecimento instantâneo da avaliação executada pelo programa;
- definir o momento e as pessoas que poderão ter acesso ao seu histórico;
- receber informação de que determinado conteúdo necessita ser refeito.

5.2 Programas disponibilizados para uso do aprendiz

Os programas a serem disponibilizados para o aprendiz têm seu foco no uso de laboratórios. Os experimentos a serem realizados pelo aprendiz podem ser realizados com o uso de laboratório presencial ou então remotamente. O formato presencial é aquele determinado pelo experimento realizado pelo aprendiz no local onde se encontra, podendo ou não utilizar kits de aprendizagem. O formato remoto é determinado pelo uso de um laboratório que se encontra em outro local.

Os programas são responsáveis por realizar a avaliação do aprendizado durante o processo ou no momento em que o aprendiz solicitar. O aprendiz fornece os dados coletados durante o experimento e o programa retorna a avaliação do aprendizado.

5.3 Inserção de programas na plataforma

Os novos programas disponibilizados para a plataforma têm que ter o seu conteúdo analisado por especialistas da área correspondente. O aval do conteudista é de fundamental importância para o sucesso da plataforma Padolma.

Os programas têm que prever o uso de arquivos externos ou outras formas que possibilitem a sua utilização nos mais diversos idiomas.

Os programas podem ser de domínio público ou ter o seu uso restrito. Os de domínio público têm o seu uso liberado de acordo com as licenças disponibilizadas pela plataforma. Os de uso restrito serão liberados para uso de um grupo específico de aprendizes por um prazo determinado. Após este prazo ele se torna de domínio público.

A autoria do programa e da pessoa que deu o seu aval para utilização na plataforma é registrada e fica associada ao programa, podendo ser consultada a qualquer instante.

O prazo de validade do programa é definido para garantir que o conhecimento não seja esquecido depois de decorrido um determinado tempo. O prazo de validade é opcional e deve ser informado quando da inclusão do programa na plataforma. O prazo de validade é

definido quando do *design* do programa: ele tem que levar em conta o tempo que o aprendiz conseguirá reter o conteúdo estudado. Por exemplo, tendo o aprendiz executado somente um programa de um determinado conteúdo e tendo o programa um prazo de validade de 6 meses, isto fará com que o conteúdo, depois de decorridos os 6 meses, seja marcado como “provavelmente esquecido”. Decorrido o prazo de validade o aprendiz que executar o programa novamente a fim de marcar o conteúdo como aprendido. O conteúdo não será marcado como provavelmente esquecido se o aprendiz tiver feito outros programas para o mesmo conteúdo da árvore do conhecimento.

5.4 Protocolos da plataforma

A plataforma necessita de protocolos que possibilitem o registro das avaliações, das validações, das interações, dos programas trabalhados; etc. Tais protocolos devem ser disponibilizados em forma de módulos e inseridos nos programas a serem disponibilizados para os aprendizes.

O módulo de registro das avaliações fica responsável por armazenar os dados das avaliações que o aprendiz realizou:

- o tempo de uso do programa pelo aprendiz;
- o registro das avaliações de conteúdo realizadas pelo aprendiz;
- os dados porventura coletados.

Outro protocolo é o das interações. Ele armazena as interações realizadas com outros aprendizes/tutores. O registro das interações permite consultas posteriores e a criação de perguntas e respostas a serem colocadas junto a cada programa.

Um outro protocolo que garante uma maior aceitação das plataformas de Aprendizagem a Distância é o que realiza a validação do aprendiz. A validação, além da informação do código do aprendiz e da senha, pode ser feita em diversos níveis e de forma progressiva. Desta forma, será possível a implementação de novas formas de identificação física que forem surgindo. Com isto, os programas poderão ter os níveis alterados durante a sua vida útil. Se a validação inexistir o nível é zero, se for feita por um analisador de íris o nível é maior, se feito com o uso de biometria associada com o analisador de íris o nível é maior ainda, e assim por diante.

O registro da validação do aprendiz ao acessar determinado programa, necessita ser armazenada na plataforma para poder compor o histórico do aprendiz.

5.5 Árvore de Conhecimento

A Árvore de Conhecimento é uma base de dados que define como o conhecimento está organizado. Ela é composta de níveis hierárquicos, sendo que o nível mais baixo é denominado folha. A folha da árvore do conhecimento representa um conteúdo específico. Em cada folha ficarão armazenados os programas que irão trabalhar o conteúdo. Cada programa poderá pertencer a uma e somente uma folha da árvore do conhecimento.

Toda a árvore de conhecimento pode ser pesquisada por qualquer aprendiz. Não haverá regras para o acesso do aprendiz. O acesso aos programas de domínio público é total; a restrição de acesso a programas ocorre para os programas não-públicos, que dependem das regras de acesso definidas por quem fez o *upload*.

5.6 Construindo programas para a plataforma Padolma

A idéia de programa na plataforma é a de não padronização, ou seja, cada programa deve ser elaborado de acordo com o conteúdo que pretenda trabalhar. Além do *design* que será discutido adiante, todo programa precisa ter os seguintes módulos incorporados:

- **comunicação** - Permite ao programa interagir com a plataforma, enviando ou recebendo dados;
- **validação** - Valida tanto *logins* como a identificação física do usuário;
- **interação** - Permite ao usuário interagir com outros aprendizes que estejam aprendendo sobre aquela folha da árvore do conhecimento e também com o responsável pelo conteúdo trabalhado pelo programa;
- **avaliação** – O sistema de avaliação de cada programa é fundamental para a Padolma, pois a avaliação é que será registrada e definirá os conhecimentos já vivenciados pelo aprendiz.

Os módulos a serem incorporados aos programas utilizam algoritmos de criptografia e ficam disponíveis na plataforma.

Quanto à utilização dos programas poderá ou não haver necessidade de kits. Por exemplo, um programa que vá treinar um médico para dissecar um músculo poderá necessitar de um kit específico que deverá estar de posse do aprendiz para a sua execução.

Cada programa é responsável por executar toda a sistemática de avaliação automaticamente, avaliando e anotando o desempenho do aprendiz (notas e dados coletados). Os dados coletados pelo programa, além de serem enviados para a plataforma, terão que ser

gravados localmente. Dados de experimentos coletados poderão ou não ser enviados (como parte da avaliação) e armazenados em local próprio.

Como usuário do Padolma, o aprendiz tem, obrigatoriamente, acesso a funcionalidades presentes em todos os programas disponibilizados pela plataforma. Algumas delas podem ser vistas na Figura 9.

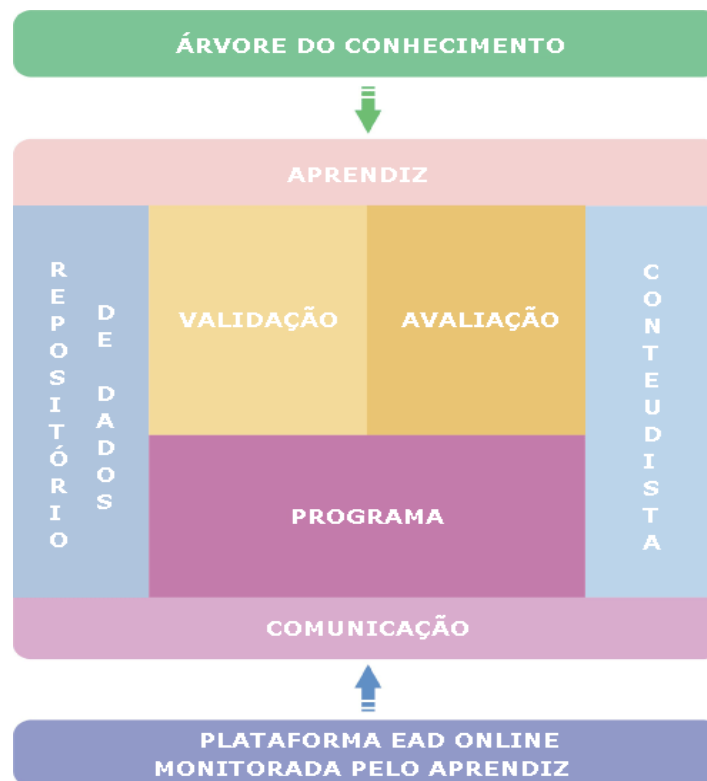


Figura 9 - Funcionalidades da plataforma Padolma disponíveis para o aprendiz.

Outras funcionalidades que ficam disponíveis do lado servidor podem ser vistas na Figura 10, a seguir



Figura 10 - Funcionalidades da plataforma PADOLMA – lado servidor.

5.7 Kits para uso do aprendiz

Determinados programas poderão fazer uso de um kit específico, o que vai implicar que o aprendiz para fazer uso do programa terá que estar de posse do kit, o que deverá ser construído de forma a atender a necessidade da parte prática do conteúdo, da coleta de dados a ser realizada com algum experimento. Na Figura 11 e na Figura 12, disponibilizadas em Soon (2008), vêm-se exemplos de dois kits nos quais os aprendizes realizam experimentos. Não é discutido o valor ou a sofisticação dos kits, mas a sua necessidade para que o aprendizado ocorra de forma concreta.



Figura 11 - Um aprendiz pratica usando um *Cessna Flight Simulator*.



Figura 12 - Estudantes trabalham em uma maquete para avião.

O planejamento e a elaboração de programas que utilizem kits é que fará com que a plataforma consiga promover o aprendizado de uma forma mais concreta, podendo oferecer uma maior credibilidade para a Aprendizagem a Distância. A integração do programa com o kit, a interação do aprendiz com outros que estejam utilizando o mesmo programa, a interação com os responsáveis pelo conteúdo, a avaliação em tempo real, a transmissão dos dados para a plataforma garantem ao aprendiz uma maior facilidade no aprendizado do conteúdo desejado.

5.8 Repositório

O repositório da plataforma é o local onde cada aprendiz terá armazenado, pelo programa, diversos tipos de conteúdos, tais como:

- avaliações de desempenho;
- conteúdos que o aprendiz aprendeu;
- dados coletados;
- textos elaborados;
- sons;
- fotos, etc.

O armazenamento pode ser feito de forma intencional pelo aprendiz ou a sua revelia durante a execução do programa. O programa fará o armazenamento de acordo com a

sua própria lógica ou devido à inclusão de módulos/bibliotecas da plataforma como o de Validação de Execução.

No próximo capítulo é descrita a definição das interfaces gráficas - GUI propostas para auxiliar na construção de programas que visem à aprendizagem a distância.

6. APLICAÇÕES GRÁFICAS

Em geral, desenvolver aplicações gráficas é uma tarefa difícil, pois quase todo projetista de sistemas de *widget* acaba cedendo às pressões dos usuários por novas funcionalidades. De fato, basta mostrar uma interface gráfica para alguém e começa a proposta de acrescentar isto ou aquilo.

Desenvolvido em OCaml (CURY, 2007) o editor *Emacs widget*¹ foi disponibilizado para teste, tendo sido enfatizado que ele deveria ser limitado a dois tipos de *widgets* apenas, pois era desejado manter a programação simples, assim como o processo de aprendizagem de como usar a aplicação final. De fato, se o sistema tem poucos *widgets*, o usuário final aprenderá a utilizá-lo mais facilmente e, se os *widgets* forem suficientemente poderosos, como é o caso de *widgets* baseados no **Emacs**, poderá fazer o que precisa de forma rápida e eficiente. Foi explicado que a aparência é questão de gosto e não é possível satisfazer todos os desejos por novos requisitos.

Todos os usuários que testaram o programa concordaram com isso, pois tinham passado pelos problemas trazidos pelas dificuldades em criar programas nos sistemas de *widgets* atuais, assim como os problemas logísticos que surgem da interação de centenas de *widgets*. Concordaram que aspecto é questão de gosto e que era urgente limitar-se a um único aspecto do grafismo, e a uma única filosofia de entrada de dados, para unificar, facilitar a programação e o uso.

Assim, todos concordaram quanto à origem do problema da programação gráfica e nos métodos de resolvê-la:

- do lado do usuário, manter apenas um paradigma de *widget* e padronizar os métodos de entrada de dados;
- do lado do desenvolvedor, padronizar a criação das estruturas de *widgets*, e criar um método de documentar as referidas estruturas. Além disto, a criação das estruturas não pode ser influenciada pela história, ou seja, não pode haver sequência de ações do programador que influenciem no comportamento da interface e que não apareçam no programa. De fato, há sistemas de programação visual em que é preciso criar uma fórmula clicando opções em um diálogo e, depois de criada a fórmula, arrastar *widgets* para dentro dela. Em seguida, é preciso escolher a opção *build* para construir

¹ O projeto *Emacs widget*, denominado **LEMAC** foi desenvolvido em **OCaml** pelo laboratório de Inteligência Artificial da Faculdade de Engenharia Elétrica da UFU em 2007. Os fontes do projeto **LEMAC** são acessíveis para download em <http://lemac.sourceforge.net/>.

a classe da fórmula e os objetos correspondentes. Finalmente, cria-se o objeto da fórmula, clicando na árvore de projetos. Foi descrita uma interface mínima, extremamente simples. Mesmo assim, há várias ações do programador não documentadas e de documentação difícil, como criar uma fórmula, clicar e arrastar componentes, fazer um *build*, escolher uma opção em um dado menu, selecionar o *code expert* na árvore de projetos, etc. Ora, fazer a manutenção desse sistema sem saber a história do desenvolvimento é extremamente difícil. Em geral, na maioria dos sistemas de programação visual de alguma complexidade, apenas o programador que participou da criação da interface é capaz de dar manutenção.

Houve um consenso quanto à origem dos problemas e como combatê-los. Entretanto, todos, sem exceção, queriam que o aspecto e o tipo de *widget* fosse aquele de sua preferência ou aquele que melhor se adequasse para a aplicação em que trabalhavam. Eis alguns comentários e as tentativas de atender as sugestões, na ordem em que foram dadas:

- **Sugestão 1** - Seria muito interessante se esta interface gráfica tivesse ícones. Por exemplo, botões gráficos e barra de ferramentas; ou pelo menos um botão transparente. Esta proposta foi atendida, introduzindo botões gráficos, *bitmap* de fundo e outras coisas.
- **Sugestão 2** - Botões gráficos e *bitmap* de fundo são problemáticos, como prova o fato de nenhum programa sério utilizá-los. Deixá-los à solta vai acarretar sérios problemas. É melhor eliminá-los. Foi apresentado para o autor da sugestão 2 que já havia sido feita a sugestão 1, e os botões gráficos estavam ali para atendê-lo. Resposta: Então pelo menos acrescente uma barra de ferramentas, e limite os botões à barra; mas faça um protocolo na criação de barras, para evitar a proliferação delas.
- **Sugestão 3** - Seria melhor que houvesse menus, em vez de botões. O usuário trabalharia com as teclas do **Emacs widget** e, se não se lembrasse de alguma operação, usaria um menu, como no **Emacs**, que não tem botões. Foi mostrado que os botões tiram o incentivo em utilizar apenas o teclado e causam vários problemas, como LER - Lesão por Esforço Repetitivo. O que se deve fazer é retirar os botões e não introduzir *menus*.

As sugestões apresentadas, com exceção da última, fazem propostas de *widgets* que parecem mais naturais em uma dada aplicação, como música, mas não são gerais, ou seja, não podem ser utilizadas em qualquer situação. Em música, o *widget* gráfico, representando notas musicais e claves, parece mais natural do que um texto com o nome da nota. Entretanto

os ícones pedidos na Sugestão 1 não são apropriados para outras aplicações, como entrada de dados de formulários financeiros e de cadastramento, etc.

A idéia do **Emacs widget** e de toda padronização que vise simplificar o uso é escolher um denominador comum para qualquer aplicação. Observe que botões de texto também são desnecessários, visto que o **Emacs widget** pode tomar conta de qualquer situação, como prova o **Emacs** real. Entretanto o autor da Sugestão 2 não protestou contra eles, porque em sua aplicação (pesquisa de opinião), botões de texto são normalmente utilizados em questões de múltipla escolha. Já a Sugestão 3 foi dada por um educador, preocupado em fazer os aprendizes escreverem, em vez de clicar botões. Embora ele pareça concordar, o faz apenas porque o tipo de interface proposta coincide inteiramente com o usual em seu ramo de atuação.

Quanto mais facilidade se deseja no desenvolvimento de um programa, na sua utilização e no aprendizado do usuário, deve-se limitar a dois *widgets* apenas, sendo um deles poderoso o bastante para atender todas as necessidades, e o outro suficientemente difundido para trazer a familiaridade necessária aos primeiros contatos com a aplicação.

O *widget* para atender qualquer situação é o Emacs widget; os *widget* suficientemente difundidos são os botões de texto. Os menus são também bastante difundidos, mas como não mostram claramente as opções que, para serem vistas, é necessário desenrolar o menu, tornando a navegação mais difícil. Os botões, sendo sempre visíveis e ocupando espaço na janela, onde espaço é um recurso limitado, fazem com que ofereçam apenas as opções necessárias ao principiante, que deve aprender as teclas do **Emacs widget** para operações mais complexas.

Se for dada muito flexibilidade no projeto da estrutura de uma interface gráfica, a programação tornar-se-á forçosamente complexa. Ao se analisar o código de dois programas pode-se perceber a diferença. Será analisada a construção de uma interface gráfica em **Clean**, uma linguagem funcional pura, e em **Emacs widget**. Um problema bem simples, que é ler um número de um campo de edição e encontrar o fatorial, foi utilizado para facilitar a análise.

```
module fns
import StdEnv, StdIO

::DIA= {ed::Id, d::Id}

Start w= startIO SDI {ed= wId, d=dId} win
      [ProcessClose closeProcess] w2
where
  (wId, w1)= openId w
```

```

(dId, w2)= openId w1

win pst={ls}= pst2
where
  (ok, pst2)= openWindow 0 j pst
  j= Window "Functions"
    ( EditControl "" (PixelFormat 100) 1
      [ControlId ls.ed]
      :+: ButtonControl "Ok" [ControlFunction roda])
    [WindowId ls.d]

roda (local, ps=: {ls, io})= (local, {ls= ls, io=io1})
  where
    (Just win, newio)= getWindow ls.d io
    (ok, maybeText)= getControlText ls.ed win
    io1= setControlText ls.ed (txt maybeText) newio

txt (Just txt)= toString(fact(toInt txt))
txt Nothing= ""

fact n
  |n < 2= 1
  | otherwise= n*(fact (n-1))

```

Listagem 2 – Programa em Clean com interface gráfica

Como pode ser visto no exemplo acima, **Clean** e a maioria das interfaces gráficas exigem que se inicie um processo de entrada e saída com pelo menos cinco argumentos, a saber:

- Informar se a interação é *SDI - Single Document Interface*, ou se é *MDI- Multiple Document Interface*;
- o estado inicial do processo;
- função de *callback*, que abre janela;
- atributos do processo;
- janela-mãe ou processo-mãe.

A abertura da janela exige descrição de funções de *callback*, assim como da estrutura da janela/diálogo e dos componentes.

Finalmente, a retirada e a inserção do texto requerem um complicado gerenciamento de identificadores. Na Listagem 3 vê-se um programa utilizando interface gráfica baseada na biblioteca do Emacs Widget.

```

let rec factorial n= if n<1 then 1 else n*factorial(n-1) ;;

let facfun txt= match Wid.getFloatList txt with
  (x::xs) -> string_of_int(factorial (int_of_float x))
  | any -> " ";;

let main () =
  Wid.open_gr "eMacs like Editor - Gui Example";
  let ed = Wid.mk_edt ~txt:(Wid.nlines 20) ~fn:facfun "factorial"
  and ext= Wid.exit_button() in
  while true do
    Wid.refresh ();
    ed.Wid.process_event();
    ext.Wid.process_event()
  done;
  Wid.close_gr();

  main ();;

```

Listagem 3 – Programa em OCaml que cria Interface Gráfica de forma simplificada

A estrutura do programa ficou mais inteligível ao permitir a criação de janelas e botões com uma sintaxe de melhor visualização. O código bem reduzido possibilita uma maior compreensão de suas funcionalidades.

7. LABORATÓRIO PARA APRENDIZAGEM A DISTÂNCIA

Neste capítulo são feitas duas propostas de metáforas para o uso de laboratórios a distância, uma para experimentos no mundo real e outra para o mundo virtual. Inicialmente é mostrada uma interface para a utilização de um experimento para o aprendizado de um conteúdo de biologia e posteriormente um para o ensino da linguagem Superlogo criada pelo núcleo de informática educacional (NIED) da UNICAMP (2007).

7.1 Execução de Experimentos a Distância

Um outro exemplo de interface gráfica que pode auxiliar a construção de laboratórios para a Aprendizagem a Distância é o LADOL (Laboratório de Aprendizagem a Distância *On-line*). A construção deste tipo de laboratório se justifica pela existência de alguns fatores que impedem o acesso dos aprendizes aos laboratórios convencionais que são:

- complexos na sua elaboração;
- caros na aquisição e manutenção;
- podem provocar problemas de saúde pelo manuseio de equipamentos/produtos;
- necessitam de pessoal de apoio qualificado.

A construção de uma interface gráfica que possibilitasse ao aprendiz reunir em um mesmo contexto todas as facilidades de um laboratório foi efetivada e denominada de LADOL. Ela foi criada para que se possam fazer programas para a Aprendizagem a Distância *On-Line*, que tenham como proposta maior permitir ao aprendiz conhecer a teoria, exercitar a construção de experimentos, realizarem observações, construir teorias, fazer demonstrações e realizar a defesa do aprendizado.

A interface gráfica Ladol é composta de diversas áreas gráficas, o que facilita o aprendizado do conteúdo. Algumas das áreas que podem compor a Ladol permitem ao aprendiz:

- conhecer a teoria;
- conhecer o experimento a ser realizado;
- conhecer a meta/objetivo a ser alcançada;
- dirimir dúvidas sobre conteúdos/conceitos necessários ao experimento;
- informar os resultados coletados;
- verificar se os resultados coletados são condizentes com o esperado;

- comunicar com outros aprendizes/outros que estejam realizando o mesmo experimento;
- verificar os demais resultados já coletados e informados.

7.2 Aprendizagem de Linguagem de Programação

A interface gráfica APLIP (Laboratório de Aprendizagem de Linguagem de Programação) se difere da LADOL por focar o ensino de uma linguagem de programação. É, portanto, uma interface gráfica voltada ao mundo digital, ou seja, a sua construção e aplicabilidade se dão no mundo digital.

A APLIP se baseia na necessidade que um aprendiz tem ao aprender uma linguagem de programação. Normalmente ele utiliza diversos aplicativos com diferentes tipos de interfaces gráficas, que são abertas e consultadas, para a construção deste conhecimento. A descrição das funções das diversas áreas que compõe a interface gráfica APLIP são:

- define a meta a atingir;
- lista os comandos² executados até o momento;
- mostra o resultado dos comandos executados até o momento;
- escreve o comando a ser executado;
- explica o uso de um determinado comando;
- possibilita ao aprendiz interagir com outros aprendizes.

A APLIP possibilita ao aprendiz reunir em uma mesma interface gráfica todas as facilidades de um laboratório para a aprendizagem de uma linguagem de programação. A APLIP é composta das seguintes áreas:

- área de execução - onde é visualizado o comando executado;
- área de meta/objetivo - onde é visualizado o que se pretende alcançar (objetivo);
- área de ajuda do comando - composta pelo campo *List Edit*, nesta área são apresentados exemplos de funcionamento do comando, nesta área é utilizada a interface gráfica ADC mostrada no próximo tópico;
- área de comando - na qual é colocado o próximo comando a ser executado;
- área da lista de comandos - mostra todos os comandos executados até o momento;
- área de comunicação - funciona como um *chat*, chamada através de botão e permite a interação com outros aprendizes;
- área de botões auxiliares - com botões disponibilizando diversas funcionalidades.

² Comandos aqui podem ser entendidos também como procedimentos, métodos, funções, etc.

7.3 Interface gráfica ADC

Ao ministrar um treinamento da linguagem de programação denominada de *Superlogo* (*software* criado pelo Nied (2007)) para professoras que desejavam trabalhar com os aprendizes uma ferramenta de desenvolvimento intelectual associada a uma possibilidade de inclusão no mercado de trabalho, foi elaborada uma proposta metodológica para o ensino da linguagem Superlogo que associasse o lúdico ao aprendizado de comandos da linguagem.

A definição dessa interface gráfica foi elaborada pensando-se não em um objeto real, mas em uma realidade composta de 3 momentos: o **antes**, o **depois** e o **comando**. O comando é aquele que leva a uma mudança do estado **antes** ao estado **depois**.

Foram criados painéis com os diversos comandos. Cada painel foi composto de três áreas:

- situação antes da execução do comando;
- situação depois da execução do comando;
- comando a ser executado.

A área superior à esquerda apresenta a situação em que a tela (desenho) se encontrava, antes da execução do comando; a área inferior mostra o comando a ser executado; a área superior à direita é a que contém o resultado final: o comando depois de executado.

A interface gráfica ADC é uma contribuição que pode ser usada para o ensino de programação. Com ela o aprendiz tem uma visão gráfica/visual de um determinado comando, o que facilita bastante o aprendizado. A seguir, na Figura 13, tem-se um exemplo de um dos painéis criados para o treinamento.

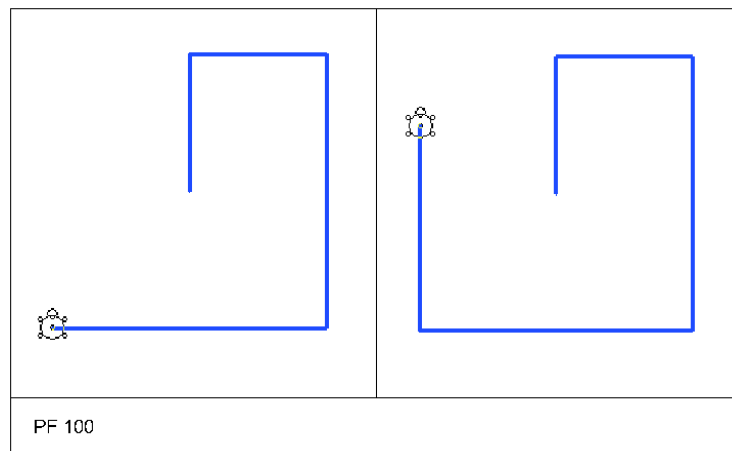


Figura 13 - Painel criado para ensino do Superlogo, baseado na Metáfora do ADC

7.4 Interface Gráfica para o Ensino do Logo

A Figura 14 mostra uma possibilidade de GUI baseada na interface gráfica APLIP, voltada para o ensino de uma linguagem de programação. Ela é composta de um mínimo de elementos definidos pela APLIP. Na figura pode ser vista a meta que o aluno deve atingir e realizar o desenho que se encontra no lado superior à direita. O resultado da execução do comando tem uma área ao lado esquerdo da meta a ser alcançada. Na área inferior à esquerda tem-se os comandos executados e o comando a ser executado. Na área de ajuda, a interface gráfica ADC foi utilizada para mostrar a execução de um comando. Na parte inferior existe uma área de comunicação pela qual o aprendiz pode interagir com outros aprendizes que estejam utilizando o mesmo programa.

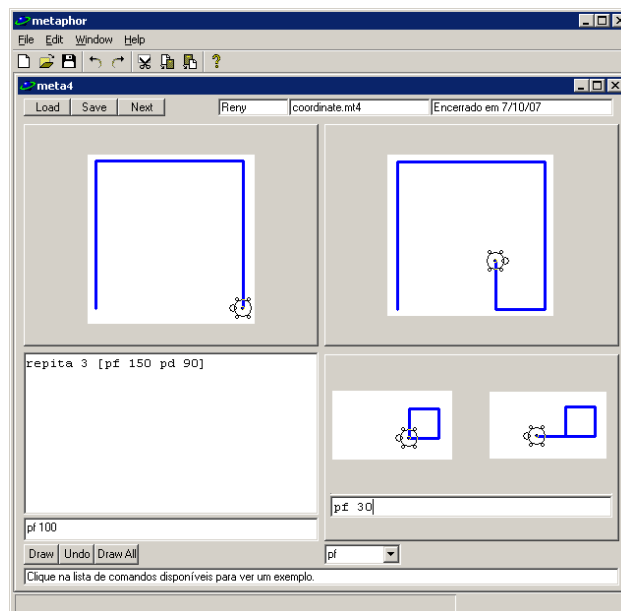


Figura 14 – APLIP Laboratório de Aprendizagem de Programação *On-line*

Algumas das áreas que compõem a interface gráfica APLIP podem ser mais bem visualizadas no esquema mostrado na Figura 15.



Figura 15 – APLIP: esquema das áreas.

8. LINGUAGENS, BIBLIOTECAS E PACOTES

Neste capítulo são descritas as pesquisas realizadas com o fito de se encontrar uma forma de construir e disponibilizar bibliotecas que permitam a construção de interfaces gráficas voltadas ao usuário final. São analisadas linguagens modernas para a criação de bibliotecas e programas educativos. É apresentada uma forma de se criar pacotes que possibilitem a manutenção ou a replicação de programas a curto, médio e longo prazo. Os pacotes contendo bibliotecas, fontes e manuais são disponibilizados e servem como modelo para a criação de novos programas para a plataforma Padolma.

8.1 Linguagens de Programação e Bibliotecas

Foram pesquisadas diversas bibliotecas gráficas populares, construídas outras com os próprios recursos da linguagem e uma utilizando o AWT da máquina Java. Foram testadas algumas das linguagens mais eficientes, conforme o *benchmark* disponibilizado em Debian (2007). O conjunto biblioteca-programa foi pensado para tornar a programação mais simplificada. Os modelos criados foram colocados a disposição para serem replicados e utilizados pelos aprendizes da plataforma Padolma.

8.2 Uso do Clean com a biblioteca gráfica OpenGL

Esta parte da pesquisa procurou associar uma linguagem eficiente, segura com uma biblioteca também consagrada. Clean foi a linguagem escolhida. Ela é uma das que se encontram entre as mais velozes de acordo com Debian (2007). A biblioteca OPENGL foi a escolhida por estar presente no sistema operacional Windows e ter seu uso bastante consagrado.

As principais vantagens da linguagem de programação funcional Clean são descritas no sítio de seus criadores Clean (2007) e traduzidas a seguir:

A Functional Programming Language Clean é baseada no conceito de funções matemáticas. Clean é uma linguagem puramente funcional, não existe o conceito de atribuição, com isto a função não pode ter um efeito colateral. A função Clean é referencialmente clara, ou seja, o resultado da função só depende do valor dos argumentos da função e nada mais. Estes conceitos têm consequências importantes, pois depois que uma função funcionar, nunca deixará de fazê-lo. A função irá reagir sempre da mesma maneira, independentemente de quando e em que contexto ela é utilizada. Pode-se raciocinar sobre funções e programas em Clean como um todo (o que é uma aplicação da função), usando um raciocínio matemático tradicional: substituição uniforme e indução matemática.

Embora Clean não tenha atribuições, os objetos podem ser atualizados destrutivamente. Clean é a única língua funcional no mundo que tem um tipo especial de sistema: tipos únicos. Este tipo de sistema foi desenvolvido em Nijmegen. Ela permite a atualização de argumentos da função destrutivamente conservando a pureza da linguagem.

Clean é *lazy*, pura, a programação é feita em uma linguagem funcional de ordem superior explícita com semântica explícita de reescrita gráfica. O que pode definir explicitamente o compartilhamento de estruturas (como estruturas cíclicas) na linguagem. Ela pode se tornar uma linguagem *strict* para obter uma ótima relação tempo/espço: funções podem ser definidas *lazy*, assim como (parcialmente) *strict* em seus argumentos. Clean é uma linguagem fortemente tipada e baseada em uma extensão conhecida como Milner/Hindley/Mycroft esquema de inferência/verificação.

Devido à forte tipagem de Clean e a obrigação de inicializar todos os objetos que estão sendo criados em tempo de execução, erros podem ocorrer em um número muito limitado de casos. Quando funções parciais são chamadas como argumentos para fora do seu domínio (por exemplo, divisão por zero), quando arrays são acessados com índices fora da série e quando não tem memória suficiente (espço em *heap* ou *stack*) atribuída a aplicação Clean.

Tipos de classes e construtores de classes estão previstos para fazer uso de sobrecarga de funções e possíveis operadores. Clean oferece os seguintes tipos predefinidos: inteiros, reais, booleanos, caracteres, *strings*, listas de tuplas, registros (atualizáveis destrutivamente), arrays e arquivos; Clean tem casamento de padrão, compressão de listas, *arrays* e um modo sensível ao *lay-out*.

Interfaces gráficas baseadas em programas escritos em Clean e utilizando a biblioteca I/O 0.8 podem ser transferidas sem modificação do código-fonte para qualquer uma das muitas plataformas suportadas.

O esforço para se criar ambientes 3D em CLEAN é extremamente complexo, o que pode ser verificado ao se procurar por bibliotecas no sítio do desenvolvedor, onde foi possível encontrar a biblioteca **Clean Object I/O 1.2 Library**, com a qual se pode criar somente aplicações 2D. Esta biblioteca foi desenvolvida por WIERING (2008) como parte de sua tese. Sobre a biblioteca ele afirma que a **Clean Game Library** é um sistema completo de desenvolvimento para jogos 2D utilizando a linguagem funcional Clean. Embora Clean não seja fácil de aprender (é muito diferente de linguagens imperativas, como Pascal e C), uma vez que você programe em Clean, você pode facilmente criar jogos com a **Clean Game Library**.

Além de ser para ambientes 2D, essa biblioteca é disponível somente para a plataforma **Windows** e necessita da instalação do **DirectX**. Não existindo uma biblioteca 3D, foi desenvolvida em Clean uma estratégia que permitisse o uso da OpenGL já padrão no **Windows**. Isto levou a criação de programas eficientes e extremamente compactos, cujo resultado pode ser visto na Figura 16. O código-fonte do programa feito em Clean é mostrado na Listagem 4.

Na Figura 16 são apresentados três momentos da execução do programa *mundos* (colocados sobrepostos para melhor visualização). O mundo azul gira em torno do amarelo e o verde em torno do azul.

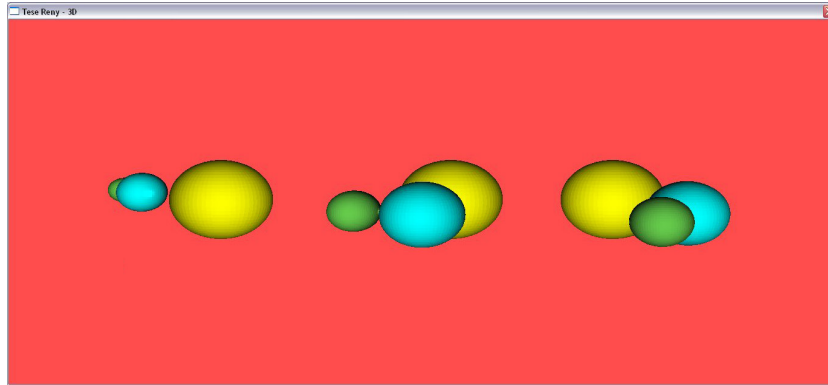


Figura 16 - CLEAN - programa usando biblioteca OPENGGL.

```

module mundosgirando
import primitivas

Start world
# a = (esphereClose 16 14.0 (0.0,0.0,0.0))
# m1= makeFaces ((esphereClose 36 5.0 (0.0,0.0,0.0)))
    {materialAmbient = (Color3 1.0 1.0 0.0),
      materialDiffuse = (Color3 1.0 1.0 0.0),
      materialSpecular = (Color3 1.0 1.0 0.0),
      materialShininess = 50,
      materialShader = FlatShader}
# m2 = makeFaces ((esphereClose 36 3.0 (0.0,0.0,0.0)))
    {materialAmbient = (Color3 0.0 1.0 1.0),
      materialDiffuse = (Color3 0.0 1.0 1.0),
      materialSpecular = (Color3 0.0 1.0 1.0),
      materialShininess = 100,
      materialShader = FlatShader}
# m3 = makeFaces ((esphereClose 36 2.0 (0.0,0.0,0.0)))
    {materialAmbient = (Color3 0.3 0.8 0.3),
      materialDiffuse = (Color3 0.4 0.8 0.3),
      materialSpecular = (Color3 0.5 0.8 0.3),
      materialShininess = 100,
      materialShader = GouraudShader}
# (gl, world) = createGL "Mundos" 800 600 32 32 False world
# gl = int gl
# gl = rungl False 0.0 0.1 0.0 0.0 0.0 0.0 m1 m2 m3 0 0 gl
= destroyGL gl world

rungl fullscreen alfa dalfa t tx ty tz m1 m2 m3 i1 i2 gl
# gl = glClearColor 1.0 0.3 0.3 1.0 gl
# gl = glClear (GL_COLOR_BUFFER_BIT bitor
              GL_DEPTH_BUFFER_BIT) gl
# gl = glMatrixMode GL_MODELVIEW gl
# gl = glLoadIdentity gl
# alfaFix1 = [0.0,0.05..360.0]
| i1 == ((length alfaFix1) - 3) = (rungl fullscreen

```

```

        (alfa + dalfa) dalfa (t + reciprocal 50.0)
        tx ty tz m1 m2 m3 0 (i2+1) gl)
# alfa1 = (1.0*alfaFix1!!i1)/t * 10.0
# raio0 = 4.0 // raio de m2 em relação a m1
# raio1 = 10.0 // raio de m1 ao centro
# gl = drawgl m gl
# gl = glRotate (alfa1*5.0) 0.0 1.0 0.0 gl
# gl = glTranslate raio1 0.0 raio1 gl
# gl = drawgl m1 gl
# gl = glRotate (alfa1*10.0) 0.0 1.0 0.0 gl
# gl = glTranslate raio0 0.0 raio0 gl
# gl = drawgl m2 gl
# (ok, gl) = windowGL gl
# (continue, keys, gl) = runGL gl
| keys.[27] = gl
| continue = rungl fullscreen (alfa + dalfa) dalfa
        (t + reciprocal 50.0)
        tx ty tz m1 m2 m3
        (i1 + 1) (i2+1) gl
= gl

where
drawgl [g:gs] gl
# gl = drawGeometry g gl
= drawgl gs gl
drawgl _ gl = gl
drawglM2 [g:gs] gl
# gl = drawGeometry g gl
= drawglM2 gs gl
drawglM2 _ gl = gl

int gl
# gl = glHint GL_PERSPECTIVE_CORRECTION_HINT GL_NICEST gl
gl = glShadeModel GL_SMOOTH gl
gl = glLightModel GL_LIGHT_MODEL_COLOR_CONTROL
        GL_SEPARATE_SPECULAR_COLOR gl
gl = glLightModel GL_LIGHT_MODEL_TWO_SIDE
        GL_TRUE gl
gl = glLightModel GL_LIGHT_MODEL_LOCAL_VIEWER GL_TRUE gl
gl = glLightModelv GL_LIGHT_MODEL_AMBIENT {15, 0, 0, 0} gl
gl = glEnable GL_CULL_FACE gl
gl = glEnable GL_NORMALIZE gl
gl = glEnable GL_LIGHTING gl
gl = glEnable GL_LIGHT0 gl
gl = glEnable GL_DEPTH_TEST gl
gl = glClearColor 0.5 0.5 0.5 1.0 gl
gl = glClearDepth 1.0 gl
(ok, gl) = fullscreenGL gl

(ok, gl) = windowGL gl

```

```
gl = glMatrixMode GL_PROJECTION gl
gl = glLoadIdentity gl
gl = gluPerspective 50.0 (4.0 / 4.0) 5.0 100.0 gl
gl = glTranslate 0.0 -5.0 -50.0 gl
= gl
```

Listagem 4 - Listagem do programa mundos.icl em Clean

A construção de programas que permitam ao aprendiz realizar explorações do conteúdo estudado é de fundamental importância para a fixação de novos conceitos. Em um ambiente 3D essas explorações serão ainda mais efetivas.

Uma biblioteca que possibilite a construção de programas 3D em uma linguagem como Clean abre uma grande frente de pesquisa para a implementação de novas idéias que revolucionem a utilização de laboratórios para a aprendizagem a distância.

Vale destacar que a associação de uma linguagem segura e rápida como o Clean e uma biblioteca 3D com funções já conhecidas é altamente viável e permite a construção de softwares educativos numa perspectiva educacional que privilegia a exploração.

Outras possibilidades de linguagens de programação e bibliotecas serão examinadas nos próximos tópicos.

8.3 OCaml utilizando a biblioteca gráfica TK

Outra pesquisa realizada envolveu a linguagem de programação OCaml com a biblioteca Gráfica TK³. A exploração do potencial desta biblioteca gráfica resultou na construção de diversos programas educativos.

Tk é um *toolkit* gráfico desenvolvido inicialmente para a linguagem de programação Tcl. Atualmente, Tk é utilizado como interface gráfica padrão por várias outras linguagens dinâmicas de programação. Tk permite o desenvolvimento de aplicativos com interface gráfica que rodam em diferentes plataformas, como Linux, Windows, Mac OS X, e muitas outras. Um dos diversos *softwares* educativos desenvolvidos com a biblioteca gráfica TK é mostrado na Figura 17.

³ Maiores informações sobre a biblioteca gráfica TK pode ser obtida em <http://www.tcl.tk/about/>

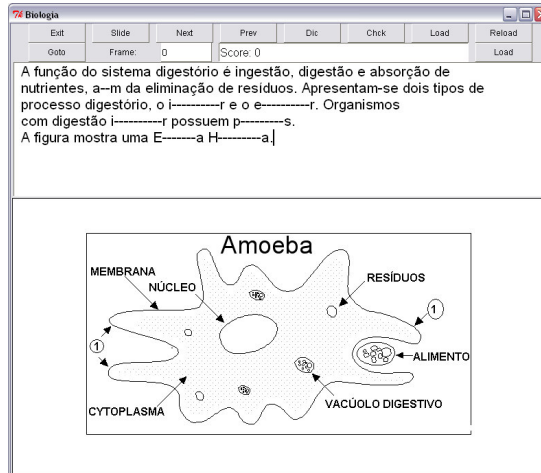


Figura 17 - OCaml - programação usando biblioteca TK.

Outro exemplo pode ser visto na Figura 18. Parte do código-fonte do programa é mostrado na Listagem 5. A construção de interfaces gráficas pode ser bem simples, como neste exemplo, onde o uso de botões possibilita o acesso a todas as funcionalidades e torna o aprendizado do programa bem rápido, permitindo a sua utilização por qualquer aprendiz.

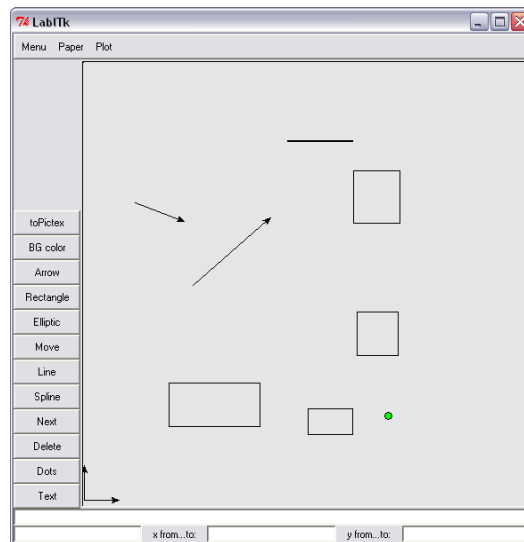


Figura 18 - Programa em OCaml que faz desenho utilizando a Biblioteca TK.

```
( Para compilar:
ocamlopt -I +labltk labltk.cmxa str.cmxa graf.ml -o graf.exe
ou, então (se você não tem gcc instalado):
ocamlc -I +labltk labltk.cma graf.ml -o graf.exe
Para executar: graf.exe
)
exception Dim of string;;
exception Vazio;;
```

```

(* Inicializar Tk *)
let top = Tk.openTk ();;

let pictop = Toplevel.create ~name:"pictex" top;;
(* Crio um campo de entrada de dados *)

let tex = Text.create pictop ~width:64 ~height:8 ~setgrid:true
  ~wrap:`Word ~padx:4 ~font:"Arial 16";;

(* As dimensões da quadra *)
let qwidth, qheight= (ref 500),(ref 500);;

let frame= Frame.create top;;
let picFrame= Frame.create top;;

let lines= ref [];;

let (* Criar quadra *)
  quadra = Canvas.create ~width:qwidth.contents
    ~height:qheight.contents ~background:(Color "gray90")
    ~borderwidth:1 ~relief:`Sunken top;;

let makeFig tela posx posy =
  Canvas.create_image tela
    ~x: posx
    ~y: posy
    ~anchor: `Center;;

let fig= ref (makeFig quadra (!qwidth / 2) (!qheight / 2));;

let pontos= ref [];;

let _ =
  Tk.bind ~events:[`ButtonPress] ~fields:[`MouseX; `MouseY]
    ~action:mmouse
    quadra;;

(* Colocar a quadra e o botão na janela *)
Tk.pack ~side:`Left ~expand:true ~fill:`X [ Tk.coe entrada];;
Tk.pack ~side:`Left ~expand:true ~fill:`X
  [Tk.coe xfromto_label];;
Tk.pack ~side:`Left ~expand:true ~fill:`X [Tk.coe xfromto];;
Tk.pack ~side:`Left ~expand:true ~fill:`X [Tk.coe yfromto_label; Tk.coe yfromto];;

Tk.pack ~side:`Bottom ~expand:true ~fill:`X [Tk.coe frame] ;;
Tk.pack ~side:`Bottom ~expand:true ~fill:`X [Tk.coe entradatext] ;;
Tk.pack ~side:`Left ~anchor:`S [Tk.coe picFrame; Tk.coe quadra];;

Tk.pack [genPictex; bckGround; arrowRead; rectangleDraw; ellipseDraw;
  moveRead; lineRead; splineRead;

```

```

nextElem; deletElem; rdDots; textRead];;

Tk.pack [tex];;

(* Chamar mainLoop *)
let _ = Tk.mainLoop ();

```

Listagem 5 - Fonte do programa em OCaml com a Biblioteca TK

Para que programas que utilizem a Biblioteca TK possam funcionar é necessário ter, no micro, instalados diversos arquivos do pacote TCL/TK. Uma listagem deste pode ser vista na Figura 19. Além de estarem presentes, existe a necessidade de se colocar o caminho de onde se encontram nas variáveis de ambiente do sistema operacional. Isso pode causar transtornos na execução do programas. O usuário tem que ter uma certa experiência para lidar com a instalação desta biblioteca.

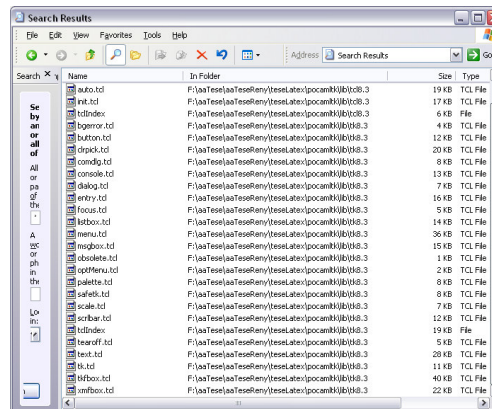


Figura 19 - Arquivos do pacote TCL/TK necessários para execução.

8.4 BIGLOO utilizando a Biblioteca gráfica GTK

Bigloo é uma implementação da linguagem de programação Scheme, desenvolvida no French IT Research Institute - INRIA. Sua orientação é no sentido de proporcionar ferramentas para a geração de código eficiente e diversificado que pode igualar o desempenho do C ou C++. Bigloo contém um compilador que pode gerar código C, *bytecode* Java ou .NET. Tal como acontece com outros dialetos Lisp, ele contém um intérprete.

Bigloo é uma implementação Scheme voltada a um objetivo: permitir a programação no estilo Scheme onde C++ é usualmente necessária. Dentre alguns dos aplicativos feitos em Bigloo temos o HOP (motor de aplicação web 2.0) e o Roadsend PHP foram escritos em Bigloo.

GTK, ou o GIMP Toolkit, é um *cross-platform widget toolkit* para a criação de interfaces gráficas. É um dos mais populares *toolkits* para o Sistema X Window, juntamente com Qt e Motif. O GTK foi criado inicialmente para o GNU *Image Manipulation Program* - Gimp, um editor gráfico *raster*, em 1997 por Spencer Kimball e Peter Mattis, membros da eXperimental Computing Facility – XCF de Berkeley. Licenciado sob a LGPL, GTK+ é um software livre (*open source*) e faz parte do Projeto GNU.

8.5 Utilização da biblioteca GTK pelo Bigloo

A Figura 20 mostra uma interface gráfica, escrita utilizando a linguagem Bigloo e a Biblioteca Gráfica GTK, onde se vê a utilização de um tipo só de *widget*. A interface gráfica foi mantida simples, para facilitar o entendimento das funções.

Algumas instruções, bem simplificadas, possibilitam a utilização do programa pelo aprendiz. Exemplos de instruções a serem seguidas pelo aprendiz são: digite o texto e para salvar clique no botão próprio. Para carregar um arquivo, escreva o nome do arquivo e clique em *Load*, se clicar em *Browser* pode navegar pelo computador, a fim de encontrar um local para salvar o seu arquivo fonte. Pode-se usar o *widget browser* para percorrer as pastas do computador e ao escolher o arquivo o navegador será encerrado, e o arquivo carregado para a área de entrada de dados. Para pesquisar escreva o texto que se deseja pesquisar no campo específico, e pressione *Search*. Se quiser encontrar a próxima instância, pressione *NxtSrc*. Para verificar se os parênteses estão abrindo e fechando corretamente, selecione a expressão, e clique no *widget* `((()))`. Se há parênteses desemparelhados, a expressão irá tornar-se vermelha. Pode-se gerar um código em **Latex** de seu programa. Se iniciar a linha com `;` ela é interpretada como comentário. Se pressionar o botão LaTeX ele irá gerar um documento em **LaTeX**.

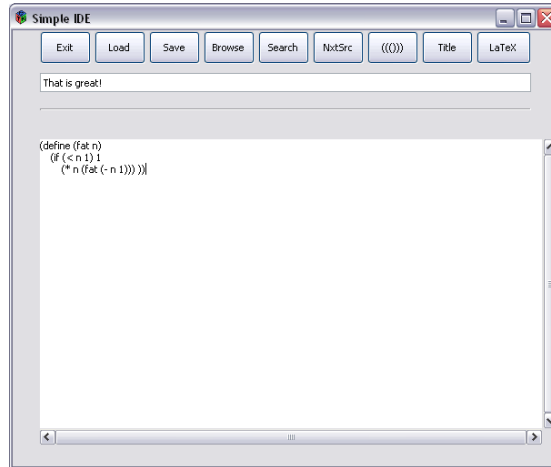


Figura 20 - BIGLOO – GUI do programa stm.scm usando biblioteca GTK.

Na Listagem 6 do código-fonte do programa stm.scm, escrita em Bigloo utilizando a biblioteca gráfica GTK, pode-se observar como é feita a criação de uma interface gráfica de forma simplificada. A dificuldade de se utilizar programas que incorporem a biblioteca GTK está na necessidade de se encontrar a versão adequada da biblioteca bem como em instalá-la corretamente.

```
:: Compile with:
:: bigloo -o stm stm.scm -copt -w -lgtk-server
:: This program shows how to write a simple GUI for Bigloo.

(module test
  (include "bind.sch")
  (extern (macro gtk::string (::string) "gtk"))
  (main main))

(define (main arqv)
  (gtk-init)
  (let* ( (win::string (gtk-window-new 0))
    ( _ (gtk-window-set-default-size win 600 480))
    (fileWin (gtk-window-new 0))
    ( _ (gtk-window-set-default-size fileWin 400 200))
    (fileTable (gtk-table-new 10 20 0))
    (tbl::string (gtk-table-new 20 20 1))
    (fromit (gtk-frame-new))
    (startit (gtk-frame-new))
    (endit (gtk-frame-new))

    (mrk #f)
    (ext::string (gtk-button-new-with-label "Exit"))
    (sav::string (gtk-button-new-with-label "Save"))
    (lod::string (gtk-button-new-with-label "Load"))
```

```

(brs (gtk-button-new-with-label "Browse"))
(ok (gtk-button-new-with-label "Ok"))

(src (gtk-button-new-with-label "Search"))
(nsr (gtk-button-new-with-label "NxtSrc"))
(par (gtk-button-new-with-label "((( )))"))
(sTt (gtk-button-new-with-label "Title"))
(ltx::string (gtk-button-new-with-label "LaTeX"))

(gtk-box-pack-start hfx enFile 0 0 1)
(gtk-box-pack-start hfx ok 0 0 1)
(gtk-table-attach-defaults fileTable hfx 0 10 0 1)
(gtk-table-attach-defaults fileTable filsel 0 19 2 10)
(gtk-container-add fileWin fileTable)
(gtk-file-filter-set-name fileFilter "all")
(gtk-file-filter-add-pattern fileFilter " *.*")
(gtk-file-chooser-add-filter filsel fileFilter)
(gtk-text-buffer-create-tag txt "fstcolor" "foreground" "red")
(do ((event (gtk-server-callback)
            (gtk-server-callback)) )
    ((equal? event ext))
    (cond ( (equal? event sav)
            (set! fname (gtk-entry-get-text en))

(define (wrap-verbatim xs)
  (append '("\begin{verbatim}") xs '("\end{verbatim}") ))
(define (wrap-document docTitle xs)
  (append (latex-header docTitle) xs '("\end{document}"))))

```

Listagem 6- Fonte do programa em Bigloo stm.scm com a biblioteca GTK

No exemplo da Figura 21 vemos um jogo. O código-fonte do programa é mostrado na Figura 7. A interface gráfica é de fácil entendimento, por utilizar somente botões.

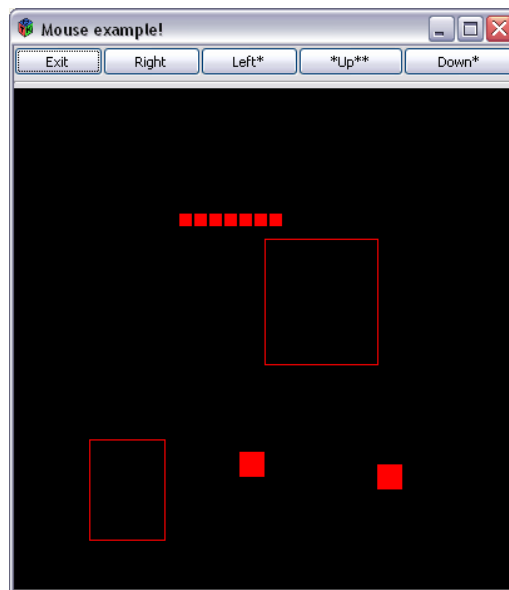


Figura 21 – Programa stp.scm (Bigloo/GTK) – Jogo que demonstra o usa do mouse.

O Código fonte do programa stp.scm é mostrada na Listagem 7, nela se vê as chamadas da biblioteca GTK. Elas são iniciadas com gtk_, o que facilita o entendimento do código-fonte do programa.

```
;; Compile with:
;; bigloo -o stp stp.scm -copt -w -lgtk-server

(module mus
  (include "bind.sch")
  (extern (macro gtk::string (::string) "gtk"))
  (main main))

(define (main args)
  (gtk-init)
  (let* ( (win (gtk-window-new 0)) ; 0 means TOP WINDOW
    ;; Create widget to display image
    (image (gtk-image-new))
    ;; Create eventbox
    (EBOX (gtk-event-box-new))
    (_ (gtk-container-add EBOX image))
    (xsize 400)
    (ysize 400)
    (winId ""))
    (ext (gtk-button-new-with-label "Exit"))
    (rt (gtk-button-new-with-label "Right"))
    (lt (gtk-button-new-with-label "Left*"))
    (up (gtk-button-new-with-label "**Up**"))
    (dn (gtk-button-new-with-label "Down*"))
```

```

(hbx (gtk-hbox-new #f 0))
(vbx (gtk-vbox-new #f 2))
(sep (gtk-hseparator-new))

(_ (pack-homogeneously hbx ext rt lt up dn))
(_ (pack-pack vbx hbx sep EBOX))
(_(gtk-container-add win vbx ))
(_(gtk-widget-show-all win) )
(gdkwin (gtk-widget-get-parent-window image))
(PIX (gdk-pixmap-new gdkwin xsize ysize -1))
(GC (gdk-gc-new PIX))
(_ (gtk-image-set-from-pixmap image PIX))
(COLOR (gtk-frame-new)))
(gtk-window-set-resizable win #t)
(gtk-window-set-default-size win 600 400)
(gtk-window-set-position win 1)
(gtk-window-set-title win "Mouse example!")
(gtk-server-connect EBOX "button-press-event canvas")
(gtk-server-connect EBOX "motion-notify-event motion")
(gtk-server-connect win "show winId")
(gtk-server-timeout 200 win)
(init-worm)
(play-with event until (or (equal? event ext)
                           (end-of-game crashed)
                           (draw-obstacles PIX GC)
                           (draw-food PIX GC)
                           (draw-worm PIX GC)
                           (gtk-image-set-from-pixmap image PIX) ])
);close play-with
);close let*
); close define

(define crashed #f)
.(define obstacles
  ' ((200 120 90 100)
    (60 280 60 80)))

(define food
  ' ((180 290 20 20)
    (290 300 20 20)))

(define (update-energy x)
  (cond ( x (set! cor "#ff0000")
         (set! energy (+ energy 40)))
        (else (set! energy (- energy 1))))
);close cond
);close define

```

Listagem 7 - Fonte do programa stp.scm em Bigloo com a biblioteca GTK

Para que programas que utilizem a biblioteca GTK possam funcionar é necessária a presença de diversos arquivos do pacote get-server: `gtk-server.cfg`, `gtk-server.dll` e `bind.sch`. A necessidade de se ter que assinalar as variáveis de ambiente com os endereços desses arquivos, torna a execução dos programas extremamente complexa.

8.6 CLEAN utilizando a biblioteca JAPI

A Biblioteca JAPI tem como base a AWT (*Abstract Window Toolkit*) do Java. Com isto qualquer máquina que tenha a máquina JVM instalada poderá executar os programas criados com essa biblioteca. Na pesquisa a Biblioteca JAPI foi utilizada inicialmente com a linguagem de programação OCaml, posteriormente com Bigloo (Scheme) e por último com Clean.

A Biblioteca **JAPI** é a biblioteca da classe java que implementa a interface do *applet* através de janelas. O *applet* pode ser implementado para ser executado fora da página web e por isso, a AWT pode ser usada para implementar interface de aplicativos escritos em Java.

A AWT é um conjunto de recursos gráficos comumente conhecidos pelos sistemas de interface usando janelas. Com ela é possível criar janelas, botões, janela de diálogos, etc. Ela é baseada na interface orientada a eventos. A programação orientada a eventos é uma técnica de programação na qual o programa é construído através de regras (métodos) a serem executados a cada vez que um evento diferente ocorrer. Por exemplo, pressionar teclado, mover *mouse*, clicar no botão, receber dados pela rede, todos são eventos. Cada vez que um desses eventos ocorrer, o método correspondente é ativado. Os elementos da AWT são na maioria, componentes, isto quer dizer que eles podem ser adicionados numa janela (*frame*) ou painel (*panel*). Quando ocorre qualquer evento no componente da AWT, o método denominado de *action* é chamado. É necessário implementar o método *action* para identificar o evento ocorrido e manipulá-lo corretamente. Essa é uma breve descrição de como funciona a AWT.

A implementação proposta por Joost (2001) fazendo a ligação da linguagem C com a biblioteca AWT, contornando a orientação a objeto da linguagem Java, permitiu os experimentos feitos com linguagens modernas e eficientes aqui descritas. A biblioteca Japi evoluiu para CAW, uma vez que foram feitas diversas modificações e melhorias, por isto em alguns momentos é feita a referência como JAPI e em outros como CAW.

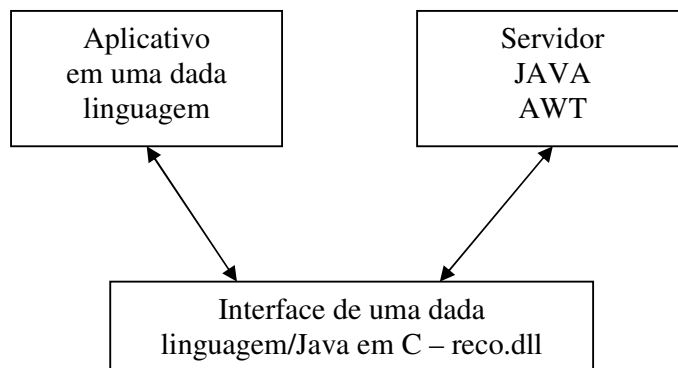


Figura 22 – Arquitetura da Biblioteca CAW

A API gráfica AWT tem uso disseminado nas mais diversas plataformas, sendo muito utilizada no Windows, Linux e outros, pois faz parte da máquina Java instalada na maioria das máquinas atuais.

Quando existe uma chamada para a dll **reco**, criada para fazer a ligação da linguagem de programação com a máquina Java, ela é carregada uma única vez, permanecendo na memória até a máquina ser desligada. O servidor monitora a porta 5089 em busca de instruções, que são dadas em forma de números inteiros que estão associados às funções disponíveis na Biblioteca CAW.

O C é responsável por iniciar e manter a comunicação com o servidor Java. Ele faz a passagem do identificador da função passada pelo programa base para o Java.

8.7 Clean com a biblioteca gráfica CAW

A biblioteca JAPI/CAW, além da linguagem de programação Clean mostrada a seguir, foi testada com outras diversas linguagens como OCaml e Bigloo. Ao se criar um programa em Clean, o programador utiliza as funções gráficas fornecidas pelo conjunto C/Java.

Esse programa mostra uma figura fixa e o fundo em movimento. São mostrados na Figura 23 três momentos da interface gráfica do programa durante a execução.

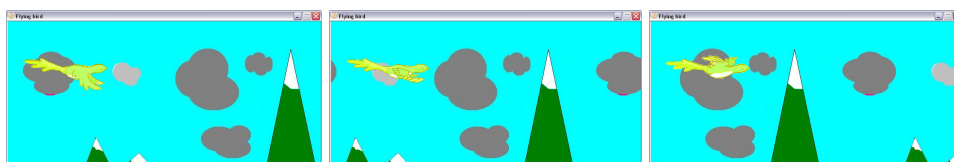


Figura 23 - Animação em CLEAN utilizando biblioteca JAPI.

Na Listagem 8 é mostrada a forma como são feitas as chamadas do Programa Abird para a API AWT.

```
module abird
import StdEnv, caw

/* Another animation, where the sprite is
static, and the background moves. */

Start world
# (js, world) = begin_caw world
(sky, js) = jloadimage "sky.jpg" js
(w, h, js) = imagesize sky js
(frm, js) = jnewFrame (WH w 400) "Flying bird" js
(cv, js) = jnewCanvas (XYWH 6 30 w 400) frm js
(img_a, js) = load_images js
(frm, js) = jshow frm js
js = loop frm 0 0 cv js with
  loop fr sel dx cv js
  | dx < ~w = loop fr sel 0 cv js
  # (x, js) = jsleep 250 js
  # (na, js) = jgetaction x js
  | na == (act fr)
    # js = jdispose cv js
    = jquit js
  # cv = jdraw (Img (dx, 0) sky) cv
  # cv = jdraw (Img (dx+w, 0) sky) cv
  # cv = jdraw (Img(30, 0) img_a.[sel]) cv
  # js = jsync js
  # sel = (sel+1) rem (size img_a)
  = loop fr sel (dx+10) cv js
= end_caw js world

where
load_images :: !JS -> (!{#Int}, !*JS)
load_images js
# (b1, js) = jloadimage "bird1.gif" js
(b2, js) = jloadimage "bird2.gif" js
(b3, js) = jloadimage "bird3.gif" js
(b4, js) = jloadimage "bird4.gif" js
= ({b1, b2, b3, b4}, js)
```

Listagem 8 - Fonte do programa Clean com a biblioteca CAW.

O uso do **Clean** com a biblioteca **CAW** tornou o código fonte mais compreensível, o que possibilitou a construção de programas gráficos mais eficientes e corretos.

8.8 Pacotes

De que adianta construir programas, se após algum tempo a manutenção ou atualização se torna quase impossível? Como possibilitar a manutenção e/ou a construção de novos programas utilizando o conhecimento já construído?

Para responder a estas perguntas, vale a pena conhecer as idéias do filósofo Aristóteles no que se refere às “quatro causas”. Ele faz uma exposição sobre a sua Teoria da Causalidade em Física (L.II, 3,7), e volta ao tema, no I livro da Metafísica (Met., V, 2. 1013a 25-35). Tem-se também detalhes sobre causa no Organon (II analíticos 98b-99b).

A Doutrina das Quatro Causas, também conhecida como Teoria da Causalidade é uma reflexão que Aristóteles faz na busca de um pensar sistemático sobre Ontologia (tema de reflexão a respeito do ser).

A teoria aristotélica da Causalidade vale para toda a Natureza. Segundo Chauí (1991), para Aristóteles existem quatro causas implicadas na existência/transformação de um objeto/corpo:

- A causa material - matéria da qual o corpo é constituído: terra para o tijolo;
- A causa formal - a forma que a matéria possui para constituir um corpo determinado: o tijolo para a terra;
- A causa motora ou eficiente - a ação ou operação que possibilita que a matéria passe a ter uma determinada forma: o trabalho do oleiro;
- A causa final - o motivo ou razão pela qual uma determinada matéria passa a ter a forma escolhida: o tijolo para construir uma casa para servir de moradia.

A causa material refere-se a um elemento disponível na natureza: a terra, existente no chão. É estática e, portanto, poderia ser substituída por qualquer outro elemento também existente na natureza.

A segunda, a causa formal, pressupõe uma concepção interna, a elaboração de uma idéia, o delineamento de uma decisão sobre que forma deve ser conferida ao material apropriado. Aqui já se pode notar a existência da consciência do homem, que irá conduzir a ação a partir de um pensamento deliberado e dirigido para orientar sua atividade. No caso da explicação de Aristóteles, a forma será o tijolo no caso da terra. Ou seja, a terra transformada em tijolo.

A causa eficiente ou motriz, responde a pergunta: quem vai ser o responsável para que esta transformação aconteça? Que ingrediente será necessário? Resposta: o ingrediente

necessário será o trabalho, e o responsável será o trabalhador, o operário especializado, o oleiro.

A transformação foi feita e o movimento assegurado. Resta saber, nesse caso, por que o tijolo? Para responder esta indagação é preciso destacar a última causa. A causa final representa o motivo ou a razão pela qual determinada matéria passa a ter uma forma definida. Como já se disse, nesse caso, foi o tijolo, um objeto a ser utilizado em uma casa, na tentativa de atender uma necessidade de uma dada pessoa.

8.8.1 Componentes do pacote

De acordo com Pereira; Grings e Santos (2007) que aplicam a Teoria da Causalidade de Aristóteles à computação, um *software* para ser documentado tem que contemplar as quatro causas de Aristóteles:

- A Causa Formal em um programa é o caminho da estrutura do fluxo dos dados. A Causa Formal responde à pergunta: *Qual é a forma do objeto?*
- A Causa Material especifica as linguagens, recursos de *hardware*, bibliotecas e algoritmos para um produto de software. A Causa Material responde a pergunta: *Do que o objeto é feito?*
- A Causa Eficiente, também chamada de Causa Motora, define o desenvolvimento de um *software*; o controlador fornece a Causa Eficiente. A Causa Eficiente responde às perguntas: *O que dirige o objeto? Em que ordem as operações estão acontecendo?*
- A Causa Final explica a função de um objeto. *Responde qual é a utilização deste programa?*

Ainda, segundo Pereira, Grings e Santos (2007), a documentação necessita ainda incluir o Chão de Fábrica, que pode ser entendido como as ferramentas necessárias para percorrer e navegar pelo código, algoritmos, bibliotecas e especificações lógicas, tendo também que ter ferramentas para localizar códigos que produzam um determinado comportamento.

De que adianta construir programas, se após algum tempo a manutenção ou atualização se torna quase impossível? Como possibilitar a manutenção e/ou a construção de novos programas utilizando o conhecimento já construído?

A criação de pacotes é uma proposta que se mostra eficiente na construção e manutenção de programas. Ela tem que contemplar as causas descritas acima, ou seja, tem que ter além das fontes, as bibliotecas utilizadas, manuais, ferramentas utilizadas e outros que

se façam necessário para a completa reprodução do ambiente de desenvolvimento no instante exato da criação do programa. Alguns pacotes criados durante as pesquisas realizadas podem ser visualizados no Sourceforge e no Google. São disponibilizados diversos pacotes, como exemplo, com o fito de permitir a programadores o entendimento dos programas desenvolvidos e a construção de novos programas para os mais diversos conteúdos.

O projeto LEMAC mostrado na Figura 24 tem exemplos que utilizam a biblioteca gráfica criada na própria linguagem OCaml e tem também o pacote que tem a linguagem Bigloo com a biblioteca GTK.

Na Figura 25 apresenta-se o pacote Stalingui, com exemplos utilizando a linguagem Stalin com a Biblioteca JAPI e, finalmente, na Figura 26 o pacote CAW, composto por exemplos de programas em Clean com a Biblioteca CAW.

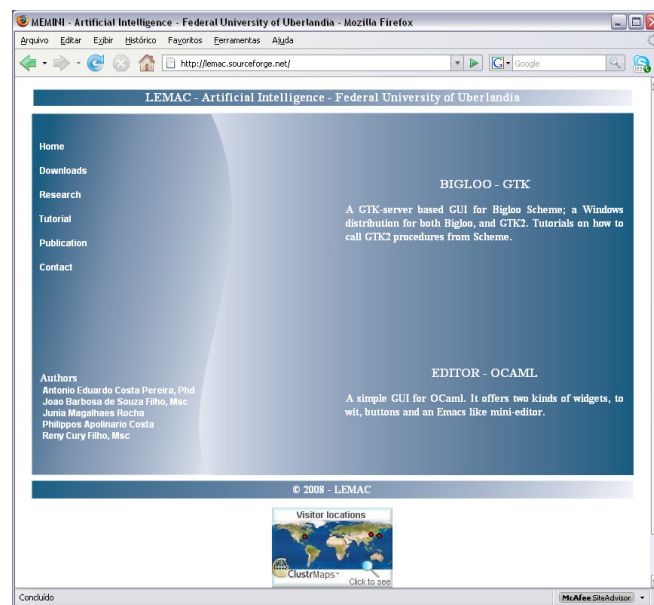


Figura 24 - Pacote LEMAC, disponibilizado no site sourceforge.net.



Figura 25 - Pacote STALINGUI, disponibilizado no code.google.com.

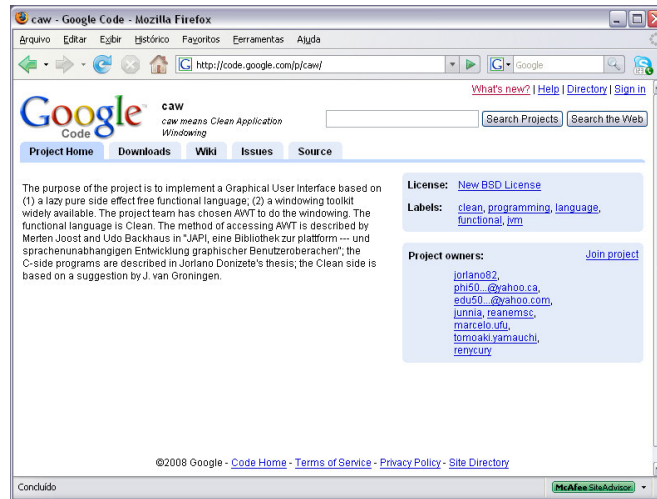


Figura 26 - Pacote CAW, disponibilizado no code.google.com.

Nos pacotes descritos acima pode-se ver a importância de uma descrição da linguagem, da instalação da linguagem e das bibliotecas necessárias, além de como escrever e compreender um programa.

Para a plataforma Padolma a forma de distribuição de programas utilizando pacotes é de vital importância, uma vez que permite a replicação do conhecimento contido na construção dos programas disponibilizados para os mais diversos conteúdos.

A plataforma Padolma e exemplos de pacotes, fontes de programas, bibliotecas gráficas, documentação de programas e outros poderão ser encontrados no site www.padolma.com.br.

CONSIDERAÇÕES FINAIS

O surgimento da *Internet* cria uma nova área educacional: a Aprendizagem a Distância *On-line*. Esse tipo de aprendizagem a distância *on-line* abre um leque enorme de pesquisas.

Nesta pesquisa foi realizado um estudo das interfaces gráficas; das metáforas mais utilizadas na computação; da criação de uma metáfora para o uso de laboratórios na Aprendizagem a Distância; das bibliotecas gráficas mais conhecidas, o aprimoramento de umas e a criação de outras; de uma plataforma de Aprendizagem a Distância na qual o aprendiz é o coordenador de seu aprendizado e da construção de diversos programas educativos envolvendo linguagens de programação modernas utilizando diversas bibliotecas gráficas.

Como a proposta da pesquisa é mostrar a importância da aprendizagem a distância *on-line* no que diz respeito ao atendimento de demandas e uso de ferramentas, interfaces gráficas, programas e laboratórios ressaltam-se a figura do aprendiz que, diante de programas educativos, é o centro do processo de ensino-aprendizagem.

Como o mecanismo de pesquisa do **Google** é hoje um projeto que deu certo, a plataforma **Padolma** é um projeto que pode percorrer os mesmos caminhos, com um diferencial importante: o conhecimento disponibilizado é validado por pessoas com conhecimento na área. Esse conhecimento não será disponibilizado simplesmente para ser lido, visto ou ouvido. O aprendiz, ao utilizar o programa para aprender determinado conceito, fará observações, pesquisas, construirá experiências e conseguirá avaliar seu aprendizado.

É importante ressaltar que no decorrer dos capítulos foram demonstradas diversas técnicas e ferramentas para as construções dos programas que irão alimentar a plataforma proposta **Padolma**. O programa, nesta proposta, tem que ser mais eficiente, utilizar interfaces claras, trabalhar cada conteúdo com programas que tenham como premissa primeira o uso de laboratório do mundo real, de maneira a prover ao aprendiz um caminho agradável para o aprendizado dos conteúdos de seu interesse.

A plataforma **Padolma**, definida na perspectiva do aprendiz, com proposta de utilização de interfaces gráficas claras, voltadas para o auto-aprendizado, a auto-avaliação e o uso intensivo de laboratórios. E ainda uma árvore do conhecimento que serve de estrutura para o aprendiz buscar o conhecimento que lhe é necessário e do qual ele possa ser o administrador.

Um outro ponto importante da plataforma **Padolma** é o registro de todas as atividades, ou seja, ficarão registrados todos os conteúdos aprendidos ou não, todas as experiências e observações praticadas pelo aprendiz. Os registros são gerenciados pelo aprendiz e ele poderá fazer uso deles quando julgar conveniente, podendo até usá-los como prova da sua aprendizagem.

Destacam-se também a disponibilização das bibliotecas gráficas, fontes de programas e de tutoriais para o usuário. Isso irá permitir construção de interfaces gráficas e de programas mais eficientes que irão possibilitar ao aprendiz o acesso a programas educativos mais adequados ao aprendizado dos conteúdos desejados.

Outro destaque que deve ser ressaltado neste trabalho é a necessidade de prover trabalho de laboratório (localmente ou remotamente) para os aprendizes.

Por fim, necessário se faz acreditar na nova área educacional, a aprendizagem a distância *on-line* que, com o uso de laboratório irá permitir ao aprendiz o conhecimento da teoria, a construção de experimentos e outros facilitando o aprendizado de novos conteúdos.

Uma das possibilidades para trabalhos futuros é ampliar os estudos sobre o uso de laboratório na aprendizagem a distância. Estas pesquisas tentarão definir os parâmetros necessários para a criação de uma **WebClean**, ou seja, uma *web* onde o conteúdo disponibilizado seja confiável, validado por pedagogos e profissionais especializados nos mais diversos conteúdos. Outra possibilidade é a continuidade da pesquisa sobre bibliotecas gráficas enfocando ambientes 3D. A construção de novos programas voltados para uma nova prática pedagógica, onde o aprendiz possa mergulhar em um mundo virtual 3D a fim de realizar as suas experiências educacionais é outra possibilidade de pesquisa. Criar uma política de cotas que possibilite a participação nos lucros, que porventura surgirem, para os colaboradores que disponibilizarem e ou participarem das análises dos programas para uso da plataforma Padolma é também mais uma possibilidade de trabalhos futuros.

REFERÊNCIAS

BOSCO, João. *Remote Experiment Lab*. Florianópolis SC, Brasil, 1999. Acesso em: 12 Nov. 2007. Disponível em: <<http://www.inf.ufsc.br/~jbosco/lexrem1i.htm>>.

CFBIO. *Conselho de Biologia não aceita ensino a distância*. Jornal Zero Hora, 11 Jun. 2008. Caderno Educação Acesso em: 20 Jun. 2008. Disponível em: <<http://zerohora.clicrbs.com.br/zerohora/jsp/default2.jsp?uf=1&local=1&source=a1959983.xml&template=3898.dwt&editio n=10045§ion=67>>

CHAUÍ, M. *O que é ideologia*. São Paulo: Brasiliense, 1991.

CHAVES, E. O. C. *Tecnologia na Educação, Ensino a Distância e Aprendizagem Mediada pela Tecnologia: Conceituação Básica*. Revista da Faculdade de Educação da Pontifícia Universidade Católica de Campinas São Paulo, 1999. Acesso em: 12 Nov. 2007. Disponível em: <<http://www.chaves.com.br/TEXTSELF/EDTECH/EAD.htm>>.

CLEAN. *Overview Clean Language Features*. Acesso em: 10 Dez. 2007. Disponível em: <http://clean.cs.ru.nl/About_Clean/Clean_Language_Features/clean_language_features.html>

COHEN, Aina E.; ELLIS, Paul J.; MILLER, Mitchell D.; DEACON, Ashley M.; PHISACKERLEY, R. P. *An automated system to mount cryo-cooled protein crystals on a synchrotron beamline, using compact samples cassettes and a small scale robot*. Journal of Applied Crystallography, 2002. p. 720-726.

COHEN, Marcelo; MANSSOUR, Isabel Harb. *OpenGL: uma abordagem prática e objetiva*. São Paulo: Novatec Editora, 2006.

COSTA, E.; BARBAR, J. S.; CURY, Reny; MAGELLAN, Junia. Participation of Distance Learning Students in Experiments. In: *Encyclopedia of Distance and Online Learning*. IGI-Global. No prelo.

COSTA, E.; CURY FILHO, Reny; MAGELLAN, Junia. Self-normalizing Distance Learning Tools. In: *Encyclopedia of Distance and Online Learning*. IGI-Global. No prelo.

CURY FILHO, Reny; SOUZA, João Barbosa F. *Emacs widget: A data entry oriented GUI*. Acessado em: 12 Dez. 2007. Disponível em <<http://sourceforge.net/projects/lemac/>>.

DAWKINS, Richard. *Unweaving the rainbow*. Boston: Houghton Mifflin, 1998.

DEBIAN. *The Computer Language Benchmarks Game*. Acesso em: 14 Jan. 2007. Disponível em: <<http://shootout.alioth.debian.org/gp4/benchmark.php?verbtest=all&lang=all>>.

DOMINGUES, Delmar Galisi. *O Uso de Metáforas na Computação*. 2001. Dissertação (Mestrado em Ciências da Comunicação) - Escola de Comunicação e Artes da Universidade de São Paulo, São Paulo, 2001. Acesso em: 10 Set. 2007. Disponível em: <http://www.teses.usp.br/teses/disponiveis/27/27143/tde-06052004-193951/publico/O_Uso_de_Metaforas_na_Computacao.pdf>.

ERIKSSON, Thomas. *Remote Unix Desktop*. 2005. Acesso em: 01 Set. 2007. Disponível em: <http://smb.slac.stanford.edu/facilities/%remote_access/remote_desktop/>

GADOTTI, Moacir. *Perspectivas atuais da educação*. São Paulo Perspec. [online]. 2000, v. 14, n. 2. Acesso em: 25 Jan. 2008. Disponível em: <http://www.scielo.br/scielo.php?script=sci_arttext&pid=S0102-88392000000200002&lng=en&nrm=iso>.

GONZALEZ, Ana. *Remote experiments*. 2007. Acesso em: 7 Jun. 2007. Disponível em: <http://smb.slac.stanford.edu/%users_guide/manual/Remote_experiments.html>.

GOOGLE. *Industry Awards and Accolades for Google*. Disponível em: <<http://www.google.com/press/awards.html>>. Acesso em: 10 de janeiro de 2008.

GUIDETOONLINESCHOOLS *Online Degrees, Online Colleges and Online Universities*. Acesso em: 10 Ago 2007. Disponível em: <<http://www.guidetoonlineschools.com/>>.

HOWARD, C., BOETTCHER, J., JUSTICE, L, ROGERS, P., BERG, G.. *Encyclopedia of Distance Learning*. Editura Idea Group Publishing, 2005.

INCONTRI, Dora. *Multimídia na Educação*. Comunicação & Educação, v 3, n. 7, 1996. ISSN: 0104-6829.. Acesso em: 20 Jan. 2008. Disponível em: <<http://revcom2.portcom.intercom.org.br/index.php/Comedu/article/view/4308/4038/4038.pdf>>.

JOOST, Merten; BACKHAUS, Udo. *JAPI - Eine Bibliothek zur plattform- und sprachunabhängigen Entwicklung grafischer Benutzeroberflächen*. 2001. Acesso em: 12 Fev. 2008. Disponível em: <http://www.fachportal-paedagogik.de/fis_bildung/suche/fis_set.html?FId=615358>.

KANSO, Mustafá Ali. *Projetando Ambientes de Aprendizagem Hipermídia com Recursos de Inteligência Artificial*. 2001. Dissertação de Mestrado em Engenharia de Produção - Universidade Federal de Santa Catarina, Florianópolis, 2001. Acesso em: 10 Jun. 2007. Disponível em: <<http://teses.eps.ufsc.br/defesa/pdf/7706.pdf>>

MATTEOLI, Lorenzo. *Complexity, Chaos and Knowledge Management*. 2004. Acesso em: 11 Ago 2007. Disponível em: <<http://members.iinet.net.au/~matteoli/html/Articles/complexity.html>>.

MORAN, J. M. *Para onde caminhamos na educação?* Microsoft, 2005. Acesso em: 10 Out. 2007. Disponível em: <http://www.microsoft.com/brasil/educacao/biblioteca/artigos/nov_05.msp>

NIED-UNICAMP. *SuperLogo 3.0*. Acesso em: 14 Nov. 2007. Disponível em: <http://nied.unicamp.br/publicacoes/pub.php?classe=software&cod_publicacao=70>.

NIELSEN, J. *Usability Engineering*. Boston: Academic Press Professional, 1993.

PAULI, Evaldo. *O Grande Aristóteles - Cap. 4 - Art. 2-o – A Ontologia de Aristóteles 0485y260*; 1997. Acesso em: 15 Abr. 2007. Disponível em: <<http://www.cfh.ufsc.br/~simposio/Megahist-filos/ARISTOTE/0485y260.html>>.

PEREIRA, A. E. C.; GRINGS, A.; SANTOS, M. V.. *Documentation Methods for Visual Languages*. In: FERRI, F. (Org.). *Visual Languages for Interactive Computing: Definitions and Formalizations*. Information Science Reference, 2007, p. 436-454.

POPPENDIECK, Mary & POPPENDIECK, Tom. *Lean Software Development: An Agile Toolkit*. Addison-Wesley, Upper Saddle River, NJ, 2003.

REDDY, Raj; WLADAWSKY-BERGER, Irving. *Digital Libraries: Universal Access To Human Knowledge - President's Information Technology Advisory Committee - Panel on digital libraries*. 2001. Acesso em: 30 Ago. 2007. Disponível em: <<http://en.scientificcommons.org/6837610>>.

RIVED. *Rede Interativa Virtual de Educação*. Acesso em: 12 Dez. 2007. Disponível em: <<http://www.rived.mec.gov.br/>>.

RUDE, Brian D. *Tactics and Strategies of Classroom Discipline*. Exposition Phoenix Press, 1979. p. 127. ISBN-13:9780682493079.

_____. *The Rationale of Laboratory Exercises In The Teaching Of Science*. 2007. Acesso em: 22 Ago. 2007. Disponível em: <<http://www.brianrude.com/ratlab.htm>>.

SANTANA, Larissa. *Por dentro da empresa que dominou o mundo*. Exame, São Paulo: Edição 915, Ano 42, Nº 6 , p 20-29, 9 Abr. 2008.

SOON, Ong Teck. *Avionics Project Design (Problem-Based Learning.)* Acesso em: 3 Fev. 2008. Disponível em: <<http://www.learnerstogether.net/avionics-project-design-problem-based-learning/56>>.

SPOLSKY, Joel. *Projeto de Interface com Usuário para Programadores: Associações e Metáforas*. Acesso em: 14 Set. 2007. Disponível em: <<http://brazil.joelonsoftware.com/uibook/chapters/4.html>>. Traduzido por Rosana de Oliveira, 18 de abr. de 2000.

VIDE, Video Development Initiative. *Laboratórios Remotos*. 2004. Acesso em: 12 Nov. 2007. Disponível em: <http://www.videnet.gatech.edu/cookbook.pt/list_page.php?topic=2&url=remotelab.html&level=2&sequence=2.5&name=Laborat%C3%B3rios%20remotos>.

WIERING, Mike. *Clean Game Library*. Acessado em: 11 Jan. 2008. Disponível em: <<http://clean.cs.ru.nl/platformgameinfo.html>>.

GLOSSÁRIO

Callback - É um modo de enviar uma função como um parâmetro dentro de uma outra função. Quando a função de *callback* completa, o controle é passado de volta à função original.

Desktop - Expressão inglesa oriunda de *desktop publisher* (editor de textos de mesa). São os computadores de mesa, estes que têm tudo separado (rato ou mouse, teclado, CPU ou unidade central de processamento, monitor, impressora...), e são os de menor custo. *Desktop* é o tampo da mesa, onde se apóiam os objetos de trabalho. Esta idéia é metaforicamente usada no meio computacional para denominar a área de trabalho do usuário, como se a tela ou *ecran* do monitor de um computador pessoal representasse a área de trabalho de uma mesa real de um escritório.

Grafismo - Modo de escrever as palavras de determinada língua. Modo de escrever peculiar a um indivíduo.

LISAAC - é uma linguagem objeto pura, muito rápida (como código C). Maiores informações veja o sítio do projeto Issac em <http://isaacproject.u-strasbg.fr/li.html>.

Lorem Ipsum – termo utilizado em *design* gráfico e editoração para preencher o espaço de texto em publicações (jornais, revistas, e *websites*), com a finalidade de verificar a tipografia, o lay-out e formatação antes de utilizar conteúdo real. Muitas vezes este texto também é utilizado em catálogos de tipografia para demonstrar textos e títulos escritos com as fontes. Tira o foco do texto, possibilitando análises mais eficientes do *design*.

Metáfora - é a designação de um objeto ou qualidade mediante uma palavra que designa outro objeto ou qualidade que tem com o primeiro uma relação de semelhança (p.ex., ele tem uma vontade de ferro, para designar uma vontade forte como o ferro).

PC – O computador pessoal é um computador de pequeno porte e baixo custo. Eles se destinam ao uso pessoal ou para uso de um pequeno grupo de indivíduos. A expressão "computador pessoal" (sua abreviação em inglês PC vêm de *Personal Computer*) é utilizada para denominar computadores de mesa (*desktops*), *laptops* ou *Tablet PCs* executando vários sistemas operacionais em várias arquiteturas. Os sistemas operacionais predominantes são Microsoft Windows, Mac OS e Linux. As principais arquiteturas são as baseadas nos processadores x86, x64 e PowerPC. No entanto o termo PC tem sido utilizado apenas para computadores pessoais derivados do modelo da IBM, o IBM PC, devido à larga campanha publicitária feita em 1981. A atual

convergência entre PCs e *palmtops*, celulares e dispositivos móveis que possuem funções similares, sistemas operacionais e componentes leva a uma discussão sobre a extensão do termo "computador pessoal" a esses dispositivos.

Superlogo - O Superlogo é um *software* desenvolvido pelo Núcleo de Informática Aplicada à Educação da Universidade Estadual de Campinas - UNICAMP e é disponibilizado gratuitamente para *download*.

Toolbox - É constituída por um conjunto de informações, suficientes para que um produto seja gerado. Um produto é formado pelos códigos e documentações necessários para a utilização da *toolbox* em determinada plataforma de integração sob determinada plataforma computacional. Assim, por exemplo, uma *toolbox* de Processamento de imagens pode ser a base de um produto que permita sua utilização a partir do *software* **MATLAB**. A mesma *toolbox* poderia originar outro produto para ser utilizado para a linguagem de programação **Python**.

Widget - É um termo que designa os elementos gráficos da interface gráfica com o usuário (GUI). Qualquer item de uma interface gráfica é chamada de *widget*, por exemplo: janelas, botões, menus e item de menus, ícones, barras de rolagem, etc. Os *ícones* que ficam no *desktop* são também denominados como *Widgets*.