UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Daniela Mônego Medina

Construção de um Ambiente de Programação Visual Orientada por Comportamentos

Dissertação submetida à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Mestre em Ciência da Computação

Raul Sidnei Wazlawick

Florianópolis, Julho de 2004.

CONSTRUÇÃO DE UM AMBIENTE DE PROGRAMAÇÃO VISUAL ORIENTADA POR COMPORTAMENTOS

Daniela Mônego Medina

Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Ciência da Computação – Sistemas de Conhecimento e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.

_____

Prof. Dr. Raul Sidnei Wazlawick
(Coordenador do Curso)

Banca Examinadora

_____

Prof. Dr. Raul Sidnei Wazlawick (orientador)

_____

Profª. Dra. Edla Maria Faust Ramos

_____

Prof. Dr. Leandro José Komosinski

_____

Prof. Dr. Flávio Bortolozzi (PUC/PR)

“Ninguém é tão grande que não possa aprender, nem
tão pequeno que não possa ensinar.”

(Voltaire)

Agradecimentos

Gostaria, neste momento que encontro-me em efusiva felicidade, de expressar meus sentimentos de gratificação, privilégio e intensa alegria. Sinto-me gratificada pela merecida conclusão deste objetivo que concretizou-se em conseqüência de árduo, constante e desprendido trabalho. Privilegiada por poder gozar de uma oportunidade concedida a poucos e que margeia a realidade da grande maioria da população brasileira, que tem cerceado este direito. E finalmente feliz por ser parte integrante e principal deste trabalho e vivenciar a expectativa futura de colher os frutos que esta conquista trará para a minha vida profissional. E dentro deste espírito comemorativo, gostaria de contagiar com minha alegria aqueles que direta ou indiretamente foram imprecindíveis para a realização deste sonho, aos quais respectivamente, presto minhas homenagens e agradecimentos:

Primeiramente a Deus, fundamento de minha religiosidade e alicerce espiritual frente aos contínuos obstáculos que permearam esta jornada.

Aos meus familiares, especialmente meus pais Carlos e Marli e minha irmã Patrícia, pela motivação, estímulo e incentivo que estiveram presentes da origem do projeto até esta fase conclusiva, onde mesmo a distância sempre me pareceram tão perto pela valiosa sustentação emocional que nossa sólida estrutura familiar arquitetada pelos meus pais propicia. Justificando e celebrizando a frase “para estar junto não é preciso estar próximo e sim dentro do peito”.

Ao meu namorado Jair, amigo, companheiro e conselheiro, que dividiu comigo os momentos de angústia e cansaço, sempre com palavras de incentivo que me fortaleceram e me deram a certeza de que vale à pena lutar por um ideal, a minha gratidão pelo apoio incondicional e irrestrito.

A todos os mestres deste Curso, em especial ao meu orientador Prof. Dr. Raul Sidnei Wazlawick, que prestou-me abnegada e desprendida assistência, sempre pautada no respeito aos princípios éticos, morais e científicos. Meus sinceros agradecimentos e parabéns pelo exemplo de profissionalismo.

Gostaria também de prestar especial agradecimento aos colegas André e Renato que, desprendendo-se de seus afazeres cotidianos, voluntariamente prestaram efetiva contribuição técnica para que este trabalho se concretizasse.

E, finalmente, aos contemporâneos de mestrado, funcionários da UFSC, colegas do LSC e amigos em geral, obrigada por contribuírem para esta conquista, que junto aos demais homenageados exerceram participação decisiva para que este sonho se tornasse realidade.

Sumário

Índice de Figuras

Índice de Tabelas

Siglas

CASE: Computer-Aided Software Engineering

POO: Programação Orientada a Objetos

VPE: Visual Programming Environment

VPL: Visual Programming Language.

Resumo

MEDINA, Daniela Mônego; WAZLAWICK, Raul Sidnei. Construção de um Ambiente de Programação Visual Orientada por Comportamentos. Florianópolis, 2004. 83p. Dissertação (mestrado) – Universidade Federal de Santa Catarina.

Na linguagem de programação de computadores deve-se obedecer a uma sintaxe textual durante o processo de programação e, em muitos casos, uma lógica imperativa, criando, desta forma, uma certa barreira para a construção de programas, principalmente por pessoas não especializadas, devido às dificuldades encontradas em aprender uma linguagem bem como as dificuldades em utilizá-la. O presente trabalho explora uma opção para amenizar os problemas enfrentados no processo de aprendizagem de programação buscando uma definição de um ambiente de programação visual baseado na noção de "comportamentos". Uma linguagem baseada em comportamentos é declarativa, o que a diferencia da maioria das linguagens visuais encontradas na literatura, que são do tipo imperativas, as quais possuem uma seqüência de instruções. Com isso, aqui foi abordado o tipo de linguagem de programação visual declarativa, onde se encontra uma relação entre os dados ou uma lista de declarações. Com as linguagens de programação visuais, não se elimina totalmente o texto, porém, por ser gráfica, este tipo de linguagem acaba proporcionando um maior conforto aos programadores em programar, tornando-se uma alternativa atrativa de programação. A definição proposta disponibiliza aos programadores um conjunto de comportamentos que podem ser atribuídos aos objetos para que estes exerçam suas funções no ambiente. Este ambiente tem sua construção feita sobre o Mundo dos Atores, uma ferramenta indicada para ser utilizada em disciplinas introdutórias de programação.

Palavras-chave: Linguagem de Programação Visual, Programação Orientada a Comportamentos, Mundo dos Atores.

Abstract

In the programming language of computers it must be obeyed a literal syntax during the programming process and, in many cases, an imperative logic, creating, of this form, a certain barrier for the construction of programs, mainly for people not specialized, inspite of the difficulties founded in learning a language as well in using it. The present work explores an option to brighten up the problems faced in the process of programming learning being searched a definition of a based environment of visual programming in the notion of "behaviors". Language based in behaviors is declarative, what differentiates it of the majority of the found visual languages in literature, that are of the type imperative, which possess a sequence of instructions. With this, the type of declarative visual programming language was boarded, which if finds a relation between data or a list of declaration. With the visual programming languages, the text is not eliminated totaly, however, for being graphical, this type of language finishes providing a bigger comfort to programmers in programming, and becomes an attractive alternative of programming. The proposed definition gives to programmers a set of behaviors which can be attributed to objects and so these will exert its functions in the enviroment. This environment has it construction made on the World of the Actors, an indicated tool to be used in disciplines introductory of programming.

Key words: Visual Programming Language, Guided programming for behaviors, Actors World.

Capítulo 1 – Introdução

A programação visual, segundo Rasia (2002), dispõe de elementos visuais como partes chaves no processo de desenvolvimento de *software*. Na programação visual, assim como na programação textual, o programador deve obedecer a uma sintaxe textual durante a programação, com isso, como na programação visual não se elimina totalmente o texto, a diferença entre estes dois tipos de linguagem de programação está no fato de que a primeira é gráfica.

De acordo com Sebesta (2000), é difícil para as pessoas compreenderem estruturas que não podem ser descritas ou representadas de alguma maneira. Como muitas pessoas têm mais facilidade de compreender um diagrama do que um texto, e como, segundo Evangelista (2002), as linguagens visuais estão freqüentemente relacionadas a maior facilidade de interpretação oferecida pelas figuras e diagramas nelas utilizados, a programação visual acaba proporcionando um maior conforto aos programadores em programar, pois na programação textual, o programador se depara apenas com códigos, podendo, assim, surgir uma barreira inicial no processo de aprendizagem devido as dificuldades encontradas em aprender tal linguagem bem como as dificuldades em utilizá-la.

Uma das maneiras de diminuir as barreiras encontradas em relação às linguagens textuais, poderia ser a utilização de linguagens visuais, pois, segundo Evangelista (2002), elas podem ser definidas como "*um conjunto de diagramas e ícones que correspondem às sentenças válidas em uma determinada linguagem de programação*". Evangelista (2002) considera a programação visual uma alternativa atrativa às linguagens textuais, sendo que, para ele, uma das razões que justifica esta atração, é fato de que "*a representação visual de um problema está muito mais próxima com a forma pela qual a solução é obtida ou entendida se comparada à representação textual*".

De acordo com Boshernitsan & Downes (1997), os seres humanos vêm há tempos se comunicando por meio de imagens. Com isso, começou-se a pensar na possibilidade de iniciar as pesquisas em linguagem de programação visual na tentativa de comandar o computador com desenhos, tendo como motivação que este tipo de linguagem seria

acessível a uma faixa mais ampla de pessoas em comparação com a linguagem textual. Com os avanços da capacidade de visualização gráfica e velocidade de processamento dos computadores, houve um grande crescimento nas pesquisas e experimentos nesta área.

Atualmente, grande parte das linguagens de programação visual encontradas são do tipo imperativas, sendo orientadas a objetos ou não. Neste trabalho pretende-se explorar a possibilidade de se ter uma linguagem de programação visual declarativa baseada em comportamentos. Com isso, é apresentada uma forma de programação, além de visual, baseada em comportamento. Comportamento pode ser definido como a forma de um elemento se relacionar com o ambiente. Esse relacionamento pode incluir a realização de tarefas, movimentação, resposta a eventos, entre outros, sendo que o elemento irá executar os comportamentos a ele atribuídos durante toda sua existência no ambiente.

Este trabalho consiste na busca de uma definição de um ambiente de programação visual baseado em comportamentos. Esta definição foi construída em cima do Mundo dos Atores que tem como base a linguagem/ambiente Smalltalk. O Mundo dos Atores é uma ferramenta construída por Mariani (1998), que se destina a ensinar programação de situações relativamente complexas com pouca sobrecarga cognitiva. Esta ferramenta auxilia no processo de aprendizado de Programação Orientada a Objetos (POO) e suporta a aprendizagem pelo método de ensino não seqüencial, sendo indicada para ser utilizada em disciplinas introdutórias de programação de computadores.

Como se trata de programação visual, a forma de apresentação do ambiente visual de comportamentos é proposto como sendo uma interface baseada no conceito de peças do tipo quebra-cabeça, sendo que, a partir do encaixe das mesmas, o programador irá compor os comportamentos para os elementos presentes no ambiente.

Espera-se, com este trabalho, minimizar os problemas enfrentados pelos programadores no momento da programação, tentando diminuir a barreira existente entre o programador e o aprendizado de uma linguagem de programação de computador.

1.1 Objetivos

1.1.1 Objetivo Geral

Com base no estudo referente à linguagem de programação visual, o principal objetivo deste trabalho consiste em construir um ambiente de programação visual baseada na atribuição de comportamentos para objetos dinâmicos.

1.1.2 Objetivos Específicos

A seguir, são listados os objetivos específicos deste trabalho:

a) Construção de um ambiente gráfico baseado no modelo do Mundo dos Atores.

b) Utilização das especificações de comportamentos apresentados no trabalho intitulado *Programação Orientada a Comportamentos como uma Extensão ao Modelo de Atores*, feito por Schütz (2003).

c) A partir da definição do ambiente de programação visual baseada em comportamentos proposto, desenvolver a implementação dos comportamentos sugeridos em Schütz (2003).

1.2 Justificativa

Segundo Sebesta (2000), a dimensão da capacidade intelectual do ser humano é influenciada pelo poder expressivo da linguagem utilizada para a comunicação dos pensamentos de cada um, sendo assim, pode-se dizer que os que possuem uma compreensão limitada da linguagem natural são limitados na complexidade de expressar seus pensamentos, especialmente em termos de profundidade de abstração.

Partindo-se do fato de que as pessoas têm mais facilidade de compreender diagramas do que textos surgiu a motivação de proporcionar uma linguagem de programação que não seja orientada a comandos e sim a comportamentos atribuídos aos objetos em um ambiente gráfico. Para Boshernitsan & Downes (1997), se houver, ao menos, uma redução quanto a necessidade da tradução de idéias visuais em algum tipo de representação textual, pode-se ajudar a atenuar o problema na curva de aprendizado de linguagens de programação.

Apesar da programação visual não eliminar totalmente o texto e nela, assim como na linguagem textual, também se deve obedecer a uma sintaxe, os objetos gráficos pertencentes na mesma, acabam proporcionando aos programadores, até mesmo aos iniciantes, uma maior capacidade no momento da programação, pois, com estes objetos gráficos, o programador pode, por exemplo, apenas informar alguns valores e com isso atribuir um comportamento a um objeto, sem se preocupar com o código existente por trás disso, ou seja, sem precisar entender, necessariamente, como isto foi feito ou como funciona. Segundo Knight (1998), os objetos visuais são os símbolos que representam algum código que de outra maneira teria que ser escrito.

A ferramenta Mundo dos Atores, utilizada como base para a realização deste trabalho, possui, segundo Schütz (2003), um ótimo potencial referente à capacidade de adequação em diferentes áreas de atuação. Nesta ferramenta, os programadores podem praticar sua criatividade manipulando elementos, como, por exemplo, inserir no ambiente objetos reativos e pró-ativos. Na aplicação desta ferramenta, apesar de se mostrar satisfatória no uso em disciplinas introdutórias de programação orientada a objetos, segundo Wazlawick et al (2001), foram encontradas algumas dificuldades, entre elas, a necessidade do uso de uma linguagem de programação, que apesar de simples e flexível, produz uma sobrecarga cognitiva.

1.3 Metodologia

Para a construção deste trabalho, inicialmente foi realizada uma revisão bibliográfica a partir de livros, artigos e outros materiais disponíveis referentes ao

assunto em questão, fundamentalmente sobre as linguagens de programação visual, comportamentos e o modelo do Mundo dos Atores.

O trabalho feito por Schütz (2003), intitulado *Programação Orientada a Comportamentos como uma Extensão ao Modelo de Atores*, foi abordado como principal fonte desta dissertação. Este trabalho desenvolvido por Schütz (2003), procurou detectar, por meio de avaliações, comportamentos em jogos do tipo *Video Game* para verificar a potencialidade destes comportamentos como elementos de uma linguagem de programação, consistindo na apresentação de um modelo de programação baseado na atribuição de comportamentos a objetos existentes em um ambiente, utilizando o modelo de eventos discretos do Mundo dos Atores.

Foi definida uma proposta de ambiente visual na ferramenta Mundo dos Atores, sendo que esta ferramenta tem se mostrado adequada no processo de aprendizado de programação. Na interface gráfica, foi disponibilizado um conjunto de comportamentos e condições para serem usados na definição dos papéis a serem executados pelos objetos presentes no ambiente. Os comportamentos e condições disponibilizados na interface seguiram as especificações contidas em Schütz (2003), sendo que foram a base para a implementação da definição do ambiente visual de programação proposto.

1.4 Resultados Esperados

Com a interface gráfica construída para a definição da ambiente visual proposto, espera-se que o uso da mesma satisfaça o programador quanto à facilidade de uso e entendimento da linguagem. A possibilidade de criação de novos comportamentos a partir das condições e comportamentos pré-definidos, além de reduzir o tempo de dedicação para atribuição de papéis para os atores, pois cada comportamento criado poderá ser usado em qualquer ator presente no ambiente, faz com que se tenha vários comportamentos diferentes a serem atribuídos.

Como, por meio da interface gráfica, os comportamentos são atribuídos a partir do encaixe de peças, a carga cognitiva envolvida não é um fator preocupante, pois apenas

são atribuídos alguns valores para a complementação dos papéis dos atores, sem a preocupação de entender como deve ser feito para que funcione.

1.5 Limitações do Trabalho

Pode ser considerado como uma das limitações deste trabalho, a pouca variedade de comportamentos básicos pré-definidos para a construção de aplicações e até mesmo para a criação de novos comportamentos. Também, o estudo e implementação das interferências, entre os comportamentos, definidas em Pianesso (2002), as quais podem acontecer a partir da combinação de certos comportamentos podendo até impossibilitar a ocorrência dos mesmos, não constam neste trabalho.

O trabalho atual limita-se a apresentar as definições contidas em Schütz (2003) em uma interface gráfica contribuindo para a diminuição da sobrecarga cognitiva.

1.6 Estrutura da Dissertação

O capítulo 2, a seguir, aborda algumas definições da linguagem de programação visual, mostrando a história da existência da mesma mostrando, também, de acordo com alguns autores, o por que usar tal linguagem. Ainda neste capítulo, são apresentadas algumas das utilidades cognitivas das linguagens visuais e são colocadas as estratégias usadas na construção das mesmas. É exposta, também, a classificação definida para as linguagens visuais mostrando, com isso, alguns exemplos de acordo com a classificação apresentada.

No capítulo 3, é feita uma apresentação do modelo do Mundo dos Atores, mostrando, inicialmente, o Smalltalk, ambiente onde foi construído o Mundo dos Atores. Neste capítulo são apresentadas as características do modelo, dando-se uma idéia geral do funcionamento do mesmo apresentando alguns exemplos.

No próximo capítulo (capítulo 4), mostra-se uma idéia sobre programação por comportamentos, mostrando a classificação dos mesmos e a descrição dos

comportamentos utilizados na definição do ambiente visual aqui proposto. Ainda neste capítulo, é exposto que os comportamentos podem possuir algumas condições, com isso, as condições utilizadas neste trabalho foram descritas.

O capítulo 5 trata da definição do ambiente de programação visual proposto em si, fazendo uma apresentação e descrição do mesmo.

No capítulo 6 são colocados como exemplos, a atribuição de alguns comportamentos definidos em Schütz (2003) em alguns dos atores presentes no palco mostrado no capítulo 5.

No capítulo 7, são apresentadas as conclusões referentes ao trabalho bem como sugestões para a continuidade do mesmo.

Capítulo 2 – Linguagens de Programação Visual

Serão mostradas neste capítulo algumas definições da linguagem de programação visual, mostrando a história da existência da mesma e também, de acordo com alguns autores, descreve o por que de usar tal linguagem. Ainda neste capítulo, são apresentadas algumas das utilidades cognitivas usadas nas linguagens visuais com a intenção de tornar a programação mais fácil, são colocadas algumas das estratégias utilizadas na construção das mesmas e é mostrado, também, que a interface de um sistema deve ser levada em consideração, pois para alguns programadores, a interface é o sistema. Ainda é exposta a classificação definida para as linguagens visuais mostrando, com isso, alguns exemplos de acordo com a classificação apresentada.

2.1. Definição de Linguagens de Programação Visual

A linguagem de programação consiste de símbolos e expressões com os quais programas de computador são desenvolvidos. A linguagem de programação visual, segundo Burnett (1999), é um ambiente onde são colocados em uso mais de uma dimensão para transmitir semântica, como por exemplo, o uso de objetos multidimensionais, relações espaciais e o uso da dimensão do tempo para especificar as relações semânticas antes e depois, sendo que, para o autor, a multidimensionalidade é a diferença essencial entre a linguagem de programação visual e a linguagem de programação textual.

Para Canning et al. (1999), a programação visual pode, também, ser definida como um paradigma que permite que os conceitos mentais possam ser representados de forma fotográfica. Pode-se dizer que a programação visual é um tipo de programação que possui elementos visuais para a especificação dos programas como peças chaves do desenvolvimento de um *software*.

A construção de um programa em um ambiente de programação visual, se dá por meio da manipulação dos elementos gráficos fornecidos por tal ambiente, com isso, conforme Young et al (2000), em um programa visual pode-se selecionar ícones

predefinidos e estes podem ser conectados para modelar os dados em um programa visual, sendo assim, os ícones podem definir a linguagem de programação. Com isso, segundo Canning et al. (1991), a programação visual possui um potencial para transformar programadores de nível menos elevado em programadores visuais. Isso se dá pelo fato de muitas pessoas possuírem maior facilidade de compreensão quando estão usando ambientes que proporcionam componentes gráficos para o desenvolvimento de alguma tarefa, neste caso, de um programa.

A programação visual, segundo Canning et al. (1999), emergiu como uma técnica para realçar a utilização do computador. Burnett (1999) coloca que, assim como cada palavra é um símbolo nas linguagens textuais tradicionais, na linguagem visual, cada objeto ou relação multidimensional é um símbolo. A coleção de um ou mais símbolos na linguagem visual é considerada como uma expressão visual, tais como diagramas, esboço a mão livre, ícones, ou demonstração de ações executadas por objetos gráficos. Contudo, quando a sintaxe de uma linguagem de programação inclui expressões visuais, esta linguagem é considerada como sendo uma linguagem de programação visual (*Visual Programming Language* - VPL).

De acordo com Burnett (1999), quando expressões visuais são usadas em um ambiente de programação, este ambiente é chamado de ambiente de programação visual (V*isual Programming Environment* - VPE). Um VPE é um ambiente que permite o uso de expressões visuais (tais como gráficos, desenhos, animações ou ícones) no processo da programação. Estas expressões visuais, como consta em FOLDOC (1995), podem ser usadas como relações gráficas para as linguagens de programação textual, podem ser usadas para dar forma à sintaxe das novas linguagens de programação visual que conduzem aos novos paradigmas, como a programação pela demonstração, ou podem ser usadas em apresentações gráficas de comportamento ou estrutura de um programa

Em Narayanan (1997), encontram-se algumas definições:

Linguagens visuais - Linguagens que fazem uso de representações visuais utilizadas para comunicação homem-homem ou homem-máquina, ou seja, linguagens de programação cuja sintaxe é baseada em representações visuais.

Programação visual - O uso de representações visuais para que se comuniquem dados e operações a um computador. A linguagem por trás pode ou não ser visual.

Linguagem de programação visual – Uma linguagem de programação com um alfabeto feito de representações visuais.

2.2. História das Linguagens de Programação Visual

De acordo com Boshernitsan & Downes (1997), os seres humanos vêm a tempos se comunicando usando imagens. Com isso, o campo da linguagem de programação visual faz o seguinte questionamento: "*porque devemos persistir em tentar nos comunicar com computadores usando linguagens de programação textual? Nós não seriamos mais produtivos e o poder de computadores modernos não estaria acessível a uma faixa mais ampla de pessoas se nós fôssemos capazes de comandar um computador simplesmente desenhando para ele as imagens que vemos em nosso olho mental quando consideramos soluções para problemas particulares?*" Boshernitsan & Downes (1997). Os fundamentalistas de linguagens de programação visual argumentam que a resposta para estas questões é afirmativa, sendo que as quais destacam as primeiras motivações para a maioria das pesquisas em linguagens de programação visual.

A idéia de transformar diagramas em programas computacionais, conforme Evangelista (2002), não é recente, contudo o interesse a programação visual despertou somente após a chegada dos computadores pessoais. Há alguns anos o constante crescimento das capacidades de visualização gráfica e velocidade de processamento de computadores, de acordo com Boshernitsan & Downes (1997), tornou possível um grande avanço na pesquisa e experimentos na área de estudo em questão.

Burnett (1999) relata que o primeiro trabalho em programação visual seguiu duas direções, uma referente à abordagem visual para linguagens de programação tradicionais, como fluxogramas executáveis, e novas abordagens visuais para aquelas consideradas significantemente divergentes das abordagens tradicionais, como programação onde a ação desejada era representada na tela. Muitos destes primeiros

sistemas, segundo Burnett (1999), tinham a vantagem de parecerem animadores e intuitivos quando eram demonstrados por meio de programas "brinquedo", mas encontravam dificuldades quando as tentativas eram feitas para desenvolver os mesmos em problemas mais realísticos. Estes problemas resultaram em um primeiro desencantamento com a programação visual, fazendo com que muitos acreditassem que a mesma era inapropriada para o trabalho real, sendo indicada apenas para um exercício acadêmico.

Para tentar superar este problema, segundo Burnett (1999), alguns pesquisadores de programação visual começaram a desenvolver meios para usar a programação visual somente em algumas partes do desenvolvimento do *software*, aumentando, assim, o número de projetos nos quais a programação visual poderia ajudar. Logo em seguida, ambientes de programação visual comerciais bem sucedidos começaram a aparecer. Entre os primeiros exemplos estava o Microsoft Visual Basic (para o Basic) e ParcPlace Systems' VisualWorks (para o Smalltalk).

Outro grupo de ambientes de programação visual, focados primeiramente em programação de grande granularidade eram as ferramentas da Computer-Aided Software Engineering (CASE) que davam suporte a especificação visual (por exemplo, usando diagramas) de relações entre os módulos do programa, atingindo um nível mais elevado na geração automática de código do código de composição.

Em contrapartida, outros pesquisadores de programação visual tiveram uma abordagem diferente, tentando aumentar os tipos de projetos adequados para programação visual através do desenvolvimento de sistemas de domínio específico. De acordo com Burnett (1999), esses pesquisadores descobriram que dispondo de meios para escrever programas para um problema de domínio particular eliminava muitas das desvantagens encontradas nas primeiras abordagens, porque eles davam suporte ao trabalho direto no estilo de comunicação do problema de domínio em questão, usando artefatos visuais (ícones e menus) refletindo as necessidades particulares, diagramas para solução de problemas e vocabulário específico para aquele domínio, não forçando o programador abandonar aquele estilo de comunicação. Um benefício adicional para essa abordagem foi a melhoria na facilidade de acesso, pois usuários finais começaram a

ser capazes de usar estes novos sistemas. Sendo assim, esta abordagem produziu, em pouco tempo, um número de sucessos tanto no mercado como na pesquisa.

O campo da programação visual, para Boshernitsan & Downes (1997) cresceu por meio da união entre trabalhos de computação gráfica, linguagens de programação e interação homem-máquina. Na década de 60, o grande passo foi o desenvolvimento do sistema Sketchpad (caderno de desenho) de Ivan Sutherland, desenvolvido em 1963 no computador TX-2 no MIT, se destaca como o melhor exemplo desta tendência. Sketchpad foi considerado o primeiro aplicativo de computação gráfica, tornando possível a criação de desenhos e alterações de objetos de maneira interativa. O sistema permitia ao projetista trabalhar com uma caneta de luz e teclado para manipular os objetos gráficos que apareciam na tela, sendo que sua interface gráfica foi de grande contribuição para as linguagens de programação visual.

Najork (1995) aponta que o irmão de Ivan Stutherland, Willian, também fez uma importante contribuição para a programação visual no ano de 1965, quando usou um computador TX-2 para desenvolver uma linguagem visual simples *dataflow*. O sistema permitia aos programadores, criar, depurar e executar diagramas *dataflow* em um ambiente visual unificado. O próximo marco das linguagens de programação visual, segundo Boshernitsan & Downes (1997), aconteceu no ano de 1975 com a publicação da dissertação de PhD de David Canfield Smith intitulada como “Pygmalion: Um ambiente criativo de programação”.

Burnett (1999) aponta que atualmente já existem linguagens de programação visuais e ambientes de programação visual disponíveis em vários domínios. Uma grande parte da literatura referente as linguagens visuais é sobre linguagens visuais que são visualizações de linguagens que, originalmente, eram linguagens textuais. Embora formalizar estas linguagens possa ser útil e suas formas visuais são vantajosas em relação as textuais, para Narayanan (1997) estas linguagens podem não ajudar-nos a entender estas propriedades que fazem as linguagens visuais humanas tão potentes.

2.3. Por que usar as Linguagens de Programação Visual

O modo visual, conforme Andrade (1998), é propício para pessoas que não possuem conhecimento sobre programação, facilitando a definição dos comportamentos para os objetos sem a necessidade de aprender a sintaxe de uma linguagem de programação, pois quando a programação envolve uma sintaxe formal, a dificuldade e a utilização da mesma torna-se mais difícil, pois os programadores devem aprender tal sintaxe, a qual possui conceitos abstratos como variáveis, procedimentos, fluxo de controle, estruturas de dados, entre outros.

Boshernitsan & Downes (1997) apontam que, se houver, ao menos, uma redução quanto à necessidade da tradução de idéias visuais em algum tipo de representação textual pode-se ajudar a atenuar o problema na curva de aprendizado, possibilitando pessoas, segundo Pianesso (2002), sem conhecimento em computação, serem capazes de programar sem a preocupação quanto a aprendizagem da sintaxe de uma linguagem textual formal, sendo que, de acordo com Repenning (1997), para um número acentuado de usuários finais do computador, a programação é algo "*assombrado*" e é exatamente para este tipo de programador, o que não quer programar, que a programação visual pode fazer a diferença.

Para Repenning (1997), a abordagem da programação visual atende às necessidades de programação para uma escala de programadores, incluindo usuários finais ocasionais do computador e programadores profissionais. Evangelista (2002) acredita que o uso da linguagem visual pode permitir ao programador fazer uma demonstração dos relacionamentos entre os dados em vez de ter que expressá-los por meio de comandos textuais. Sendo assim, o uso deste tipo de linguagem torna-se mais simples.

Pelo fato da programação visual dispor de elementos visuais como partes chaves no processo de desenvolvimento de *software*, Rasia (2002) acredita que ela torna-se útil devido os gráficos representarem de forma mais adequada estruturas bidimensionais. Porém, as linguagens de programação visuais podem vir a ter sua utilidade limitada se não for definido um perfil dos programadores, das suas motivações, necessidades e que tipo de problema estão querendo resolver. Os pesquisadores em linguagens de

programação visual, conforme Repenning (1997), estão fazendo testes dos sucessos e falhas de tal linguagem, a fim de desenvolver novos rumos na pesquisa, aumentando a eficácia da mesma.

As linguagens de programação visual se relacionam com o mundo por meio de um modo gráfico e usam imagens como um componente primário de pensamentos criativos. De acordo com Boshernitsan & Downes (1997), foi constatado que as linguagens de programação textuais são mais difíceis de aprender e de serem utilizadas eficientemente até mesmo por pessoas criativas e inteligentes.

Para Narayanan (1997), o pensamento visual está profundamente embutido no psicológico de cada cultura e as linguagens visuais antecedem o desenvolvimento das linguagens textuais. Segundo Cybis (2003), o sistema cognitivo humano é caracterizado pelo tratamento de informações simbólicas, ou seja, as pessoas elaboram e trabalham sobre a realidade através de modelos mentais ou representações montadas a partir de uma realidade.

Com isso, Narayanan (1997) aponta que o cérebro humano é acostumado a converter pensamentos em representações visuais. A facilidade com que as representações visuais são criadas e compreendidas, é certamente um fator significante para a motivação de pesquisa na área de linguagens visuais. O conhecimento do sistema visual humano oculta a complexidade computacional de analisar, interpretar e operar as semânticas existentes em representações visuais, do mais simples diagrama até as cenas complexas do mundo real. Os computadores ainda não são capazes de uma visão na mesma escala que a do homem e também não são capazes de gerar representações visuais que descrevam idéias e pensamentos abstratos, exceto apenas quando conduzidos por humanos.

Narayanan (1997) acredita que as principais razões para as pesquisas em linguagens de programação visuais são os benefícios que tais linguagens causam ao programador, permitindo homens e computadores se comunicarem e interagirem por meio de representações visuais exibidas na tela de um computador, tornando-as mais adequadas para o programador do que linguagens tradicionais de texto. As linguagens de programação são primeiramente destinadas para computadores, enquanto linguagens

visuais são destinadas primeiramente para humanos, sendo que o papel central das linguagens visuais utilizadas tanto por humanos quanto por computadores é de se tornarem ferramentas para comunicação e interação.

Uma vez familiarizado com tal linguagem, ela se torna uma ferramenta extremamente eficiente para comunicação e raciocínio. Muitas áreas de pesquisa desenvolveram suas próprias linguagens visuais, por exemplo, matemáticos usam diagramas comunicativos, físicos usam os diagramas de Feinman, cientistas de computação descrevem estruturas de dados através de caixas e setas. De acordo com Narayanan (1997), enquanto as linguagens visuais são extremamente úteis para pessoas em algumas tarefas, e em algum grau, são também usadas por computadores, porém é importante notar que elas não substituem as linguagens textuais. Nem mesmo charges de quadrinhos ficam completamente sem texto. Contudo, segundo Burnett (1999), a meta da programação visual é facilitar a programação para os humanos, e a meta das linguagens de programação visual é um melhor *design* da linguagem de programação.

Evangelista (2002) afirma que relatos mostram que as linguagens visuais, se forem bem projetadas, possuem inúmeras vantagens sobre as linguagens textuais, como: acesso aleatório à informação, maior capacidade de transferência de informação, expressividade da linguagem e descrição de um problema de forma concreta ou abstrata.

2.4. Utilidade Cognitiva das Linguagens Visuais

Para Burnett (1999), o fato de que as linguagens de programação visual são usadas com a intenção de tornar a programação mais fácil, leva à necessidade de mais pesquisas sobre como as habilidades cognitivas humanas são satisfeitas por inovação no design de linguagens de programação. Narayanan (1997) expõe algumas maneiras típicas que os computadores e os humanos utilizam para a comunicação, interpretação e raciocínio, sendo que assim possa haver maior entendimento em relação às linguagens visuais:

- Manipulação direta: permite com que os programadores executem ações interagindo diretamente com objetos exibidos na tela do computador ao invés de ter que descrever a ação, sendo que, dessa maneira, os programadores podem realizar as operações que desejam sem ter que descrever e conhecer o comando para um interpretador, o qual teria a função de traduzir este comando em uma ação real, ou seja, selecionando um arquivo para ser deletado em uma interface gráfica, por exemplo, pode ser feito arrastando o ícone do arquivo para um ícone que representa a lixeira. Isto executa o comando (move <file> trash-folder). Deste modo, os benefícios cognitivos da manipulação direta vêm da não necessidade de se mover entre duas representações muito diferentes, que seriam a representação visual natural, que permite atos interativos, e o programa textual, que especifica objetos e comportamentos na tela. Com isso, a interface gráfica para o programador na manipulação direta pode ser vista como uma linguagem visual que os humanos usam para interagir com os computadores.

- Visualização de informação: as representações visuais são freqüentemente usadas para fazer com que relações de maior importância tornem-se mais acessíveis à intuição humana. Um exemplo disto seria o gráfico de uma função, o qual faz com que suas características se tornem mais explícitas que uma tabela com valores, mesmo sabendo que ambos os métodos contém informações equivalentes.

- Visualização de *software*: a visualização das características estáticas e dinâmicas de programas pode facilitar a compreensão e a análise de erros para o programador. Um exemplo disso é quando a estrutura de dados e o controle de fluxo de um programa são visualizados através do uso de gráficos e diagramas de fluxo, facilitando o entendimento, assim como a estrutura hierárquica de arquivos de um sistema operacional é facilmente visualizada com árvores de diretórios ou pastas. A visualização de um programa gera uma linguagem visual que descreve a mesma informação contida na especificação sintática e semântica de um programa, mas de uma maneira que facilita a compreensão da observação humana.

2.5. Estratégias das Linguagens de Programação Visual

De acordo com Burnett (1999), é comum que se pense que o objetivo das pesquisas em linguagem de programação visual é a eliminação do texto. Segundo o autor isto é um equívoco, pois "*a maioria das linguagens de programação visual inclui texto, de pelo menos algum tamanho, em um contexto multidimensional*". Burnett (1999) expõe que o objetivo geral das linguagens de programação visual é a tentativa de se aprimorar o *design* das linguagens de programação, tendo, esta, poucas restrições sintáticas, proporcionando uma certa liberdade para a exploração de mecanismos de programas. Para Kojima (1989), as linguagens de programação visuais e textuais não são rivais, mas sim, uma complementa a outra.

Alguns objetivos com as pesquisas em programação visual eram esperados, tais como tornar a programação mais acessível, melhorar a precisão da execução dos programas e também melhorar a velocidade das tarefas. Deste modo, foram utilizadas algumas estratégias de programação visual a fim de que os objetivos das pesquisas fossem alcançados, conforme mostrado em Burnett (1999), tais como:

- Concretização (*Concreteness*): expressão de um programa usando exemplos particulares do domínio de interesse.

- Imediatismo (*Directness*): no contexto de manipulação direta, para Evangelista (2002), esta estratégia pode ser definida como a capacidade de exploração de um programa por tempo de desenvolvimento, sendo que o programador, segundo Burnett (1999), tem a liberdade para manipular valores e objetos.

- Clareza (*Explicitness*): relacionamentos explícitos no ambiente quando forem declarados textualmente ou visualmente. Burnett (1999) coloca que um exemplo disso em linguagem de programação visual seria o sistema de descrever claramente suas relações. De acordo com o exposto em Evangelista (2002), "*o efeito colateral é possível em muitas linguagens de programação devido ao fato das relações entre as variáveis não serem explícitas ou não estarem presentes formalmente no processo de comunicação entre os procedimentos*".

- Visualização Imediata (Feedback): esta estratégia, conforme Evangelista (2002), proporciona um retorno imediato de qualquer modificação feita no programa, permitindo uma localização mais eficiente de eventuais erros. Burnett (2001) acredita que esta característica fornece mais funcionalidades ao programa, eliminando os defeitos de programação.

2.6. Classificação da Linguagem de Programação Visual

Conforme exposto em Golin (1989), o termo "linguagem visual" é usado para descrever diversos tipos de linguagem, como: linguagens que manipulam a informação; linguagens que suportam interações visuais e linguagens para programar com expressões visuais. As linguagens de programação visual podem ser classificadas em três grupos, de acordo com a extensão da expressão visual utilizada, tais como: linguagens baseadas em ícones, linguagens baseadas em formulários e linguagens de diagramas.

Burnett & Baker (1994) criaram uma classificação das linguagens visuais para que os pesquisadores pudessem classificar seus trabalhos nos termos de suas áreas de contribuição bem como procurar trabalhos relacionados.

- Linguagens puramente visuais: para Boshernitsan & Downes (1997) este tipo de linguagem é caracterizado por sua confiabilidade em técnicas visuais ao longo do processo de programação. O programador manipula ícones ou outras representações gráficas para criar um programa, o qual é, em seguida, depurado e executado no mesmo ambiente visual. Para a representação visual desta classificação de linguagem, Evangelista (2002) coloca que são utilizados grafos, sendo que o paradigma usado é o de *dataflow* (fluxo de dados). Este paradigma é utilizado por linguagens de programação visual de domínio específico para composição de componentes de baixo nível que foram escritos de algum outro modo, por exemplo, Burnett (1999) cita os sistemas de visualização específica e sistemas de simulação, que freqüentemente fazem uso de programação visual *dataflow*.

- Programação por Exemplos: nestas linguagens, segundo Evangelista (2002), o programador pode criar e manipular amostras de dados ou representações visuais de suas estruturas para "ensinar" o sistema como executar certas operações, sendo que a sua representação visual é usualmente ligada a domínios específicos de aplicação.

- Linguagens visuais baseadas em regras: as linguagens pertencentes a esta classificação, segundo Evangelista (2002), possuem um conjunto de estados e regras, sendo que cada regra possui uma pré-condição e uma especificação de transformação, com isso, se a pré-condição for satisfeita, a especificação de transformação é executada. Todas as regras devem ser exclusivas, porém se o número de regras for muito grande, o sistema não garante a sua projeção.

- Sistemas híbridos (textual e visual): aqui tenta-se combinar elementos textuais e visuais. Conforme exposto em Boshernitsan & Downes (1997), nesta classificação são incluídos os programas que são criados visualmente e depois traduzidos em uma linguagem textual de alto nível e também aqueles que envolvem o uso de elementos gráficos em caso contrário a uma linguagem textual.

- Linguagens visuais baseadas em Restrições: nos ambientes de programação desenvolvidos segundo esta classificação, de acordo com Boshernitsan & Downes (1997), o programador modela objetos como objetos no ambiente visual que estão sujeitos a restrições desenvolvidas para imitar o comportamento de leis naturais, como a gravidade, permitindo que o programador especifique visualmente as propriedades destes objetos como também os seus inter-relacionamentos, neste caso devem-se assinalar valores para as variáveis de maneira que todas as restrições sejam verdadeiras. As linguagens baseadas em restrições são normalmente representadas por caixas e linhas.

- Linguagens visuais baseadas em formulários: este tipo de linguagem pode ser considerado como uma generalização da programação em planilhas eletrônicas, o que segundo Evangelista (2002), faz com que suas representações visuais se assemelham às interfaces das planilhas eletrônicas.

2.7. Interface

A interface de um sistema, de qualquer espécie, deve ser levada em consideração na construção do mesmo. Um fator que merece atenção no desenvolvimento de um *software*, é de que o mesmo seja interativo, isto é, o usuário deve estar em plena comunicação com o sistema e vice-versa. Segundo Ramos (1995), o usuário pode interagir com o sistema por diversos meios, através de resolução de problemas, da análise de representações gráficas, da simulação e da participação ativa no próprio ambiente, como um agente ativo do sistema, porém, para isto, o sistema deve contar com uma interface amigável. Ramos (1995) afirma que o ambiente em si também deve estimular o raciocínio, permitindo autonomia ao usuário, considerando que este possa explorar o ambiente livremente, traçando um percurso conforme seu interesse e desenvolvimento no assunto.

Para Ramos (1995), a interface em si define-se em ser a própria conexão do *software* com o sistema sensório-motor do ser humano. Presmann (1995), considera a interface como uma porta de entrada para o sistema, acreditando que a mesma "*... é, de muitas maneiras, a embalagem do software de computador*". Nascimento (2000) coloca que, para o usuário, a interface é o sistema, portanto, uma interface mal projetada pode sacrificar todas as funcionalidades do mesmo.

Segundo Lima (2001), o uso dos computadores não tem se limitado apenas às atividades produtivas, sendo que o mesmo já está fazendo parte do cotidiano de uma boa parte da população. Com isso, o autor aponta que a ergonomia (adaptação das ferramentas às necessidades do usuário), parte do princípio de que é a tecnologia que deve ser adaptada ao homem e não o contrário, estabelecendo, assim, princípios básicos que devem ser seguidos no *design* das interfaces, com o intuito de facilitar a interação humano-computador. A ergonomia, conforme consta em SIMPLES Consultoria (2004), tenta facilitar, dentre outras coisas, a clareza, organização, acessibilidade e também, procura fazer com que o usuário execute suas tarefas de maneira mais rápida e eficaz.

Como parte da ergonomia, a usabilidade se preocupa com o usuário acima de tudo, exigindo que a interface gráfica seja confortável, prática e funcional. A usabilidade refere-se à capacidade de um software ser compreendido, aprendido,

utilizado e ser atrativo para o utilizador, proporcionando eficiência e satisfação, permitindo o usuário atingir seus objetivos em um contexto de utilização específico, pois a construção de cenários, de acordo com Moraes (2000), envolve o contexto no qual o usuário está inserido. Uma boa usabilidade tem grande importância para o o sucesso e qualidade de um sistema. Para Terra (2001), a usabilidade pode também ser vista como uma medida de qualidade das experiências dos usuários no momento em que interagem com algum produto ou sistema.

Cybis (2003) acredita que as interfaces pouco amigáveis contribuíram para a famosa "*barreira da informática*". Para uma melhor aproximação humano-computador há a necessidade de uma comunicação eficaz. O conhecimento do destinatário e da mensagem é fundamental para o sucesso da comunicação humano-computador. De acordo com Netto (2001), é necessário que estas relações sejam agradáveis e eficientes, enfatizando que a agradabilidade é mais importante do que a eficiência.

2.8. Exemplos de Linguagens de Programação Visual

Prograph

Prograph (Figura 1) é um exemplo da programação visual cujo paradigma utilizado é o fluxo de dados. No *Prograph*, conforme Evangelista (2002), as classes são representadas por ícones e podem ser visualizadas a partir de um *browser* específico, sendo que o mesmo possui um editor para criação das classes e programas e um suporte de depuração para o uso de técnicas de visualização dinâmica. Para Burnett (1999), o paradigma *dataflow* se diferencia por ter como característica a clareza nos relacionamentos do programa. O *Feedback* está presente no ambiente do *Prograph*.

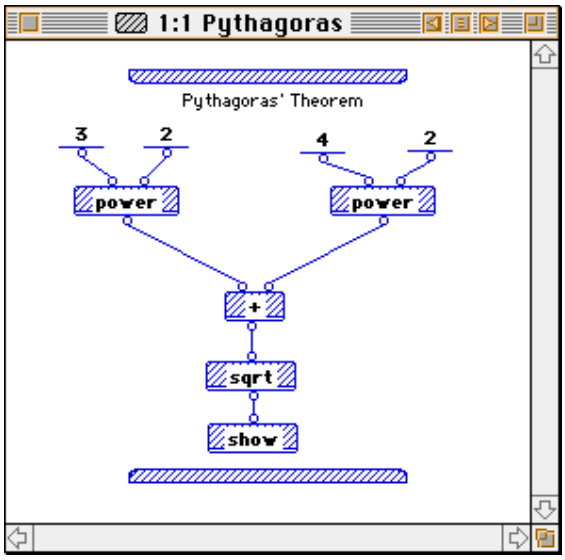

Figura 1 – Exemplo de aplicação *dataflow* implementado em *Prograph*

Fonte: Evangelista, 2002.

De acordo com Meyer (2000), o *Prograph* usa dois tipos de controles, um permite com que os dados passem por um teste bem sucedido (representado por uma caixa com um *checkmark*), e o outro em uma falha (representado por uma caixa com um "x"). Cada um dos dois tipos de controles possui casos que usam linhas na parte superior e inferior das caixas agindo como qualificadores da ação.

Conforme Figura 2, os controles possíveis são: sucesso sem qualificação adicional (*checkmark*), falha sem qualificação adicional ("x"), falha com uma linha acima, falha com uma linha abaixo, falha com linhas acima e abaixo, sucesso com uma linha acima, e sucesso com uma linha abaixo. Os controles sem qualificações adicionais permitem que o programa prossiga para o próximo método baseado em seu sucesso ou falha. Os controles com as linhas na parte superior fazem com que o laço deixe de prosseguir. Os controles com *checkmarks* ou com x com linhas na parte inferior, permitem que o programa termine de executar o código no caso especificado. O controle com linhas na parte superior e inferior, permite com que a execução do laço termine independente do valor.

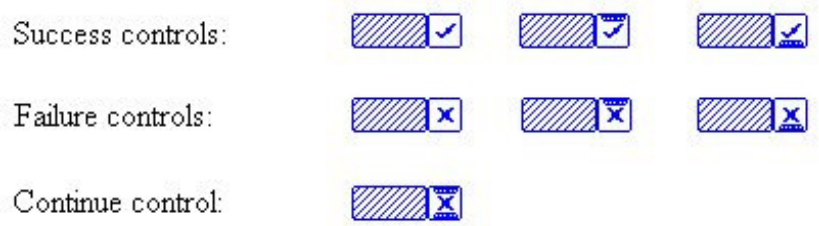

Figura 2: Controles do *Prograph*.

Fonte: Meyer, 2000.

Rehearsal World

Rehearsal World, representado na Figura 3, se enquadra nas linguagens de programação por exemplos. De acordo com Evangelista (2002), esta linguagem foi projetada para auxiliar os professores, sem experiência em programação, em suas atividades.

Assim como o Mundo dos Atores, *Rehearsal World* apresenta a metáfora de um palco, sendo assim, sua programação consiste em movimentar os atores pelo palco nos estágios apresentados, ensinando-lhes a interagir no ambiente com os outros atores por meio da emissão de mensagens, ou sugestões. Segundo Gould (1993), esta programação, que é executada no ambiente de programação Smalltalk-80 (Figura 4), consiste em um processo rápido, fácil e agradável, pois esta ferramenta apresenta um conjunto de dezoito atores pré-definidos, cada um com, aproximadamente, setenta mensagens com sugestões, sendo que estes atores, conforme exposto em Evangelista (2002), podem ser utilizados para montar um cenário e serem programados a responder novas mensagens.

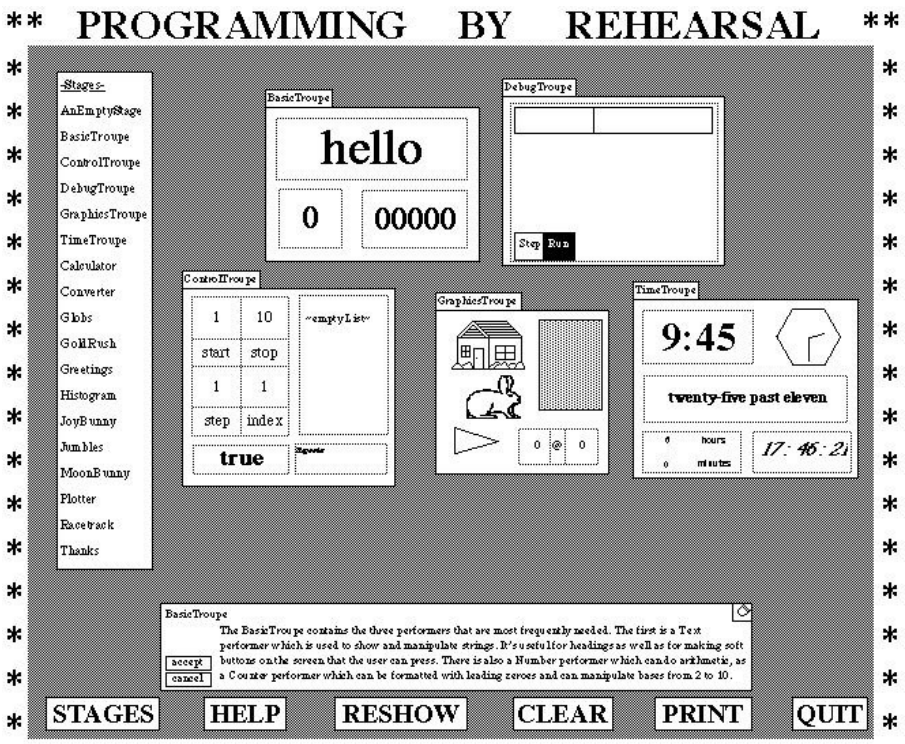

Figura 3 – Exemplo da programação em *Rehearsal* mostrando o mundo, o menu com os estágios disponíveis (a esquerda), os grupos e uma mensagem descritiva de ajuda.
Fonte: Goulde, 1993.

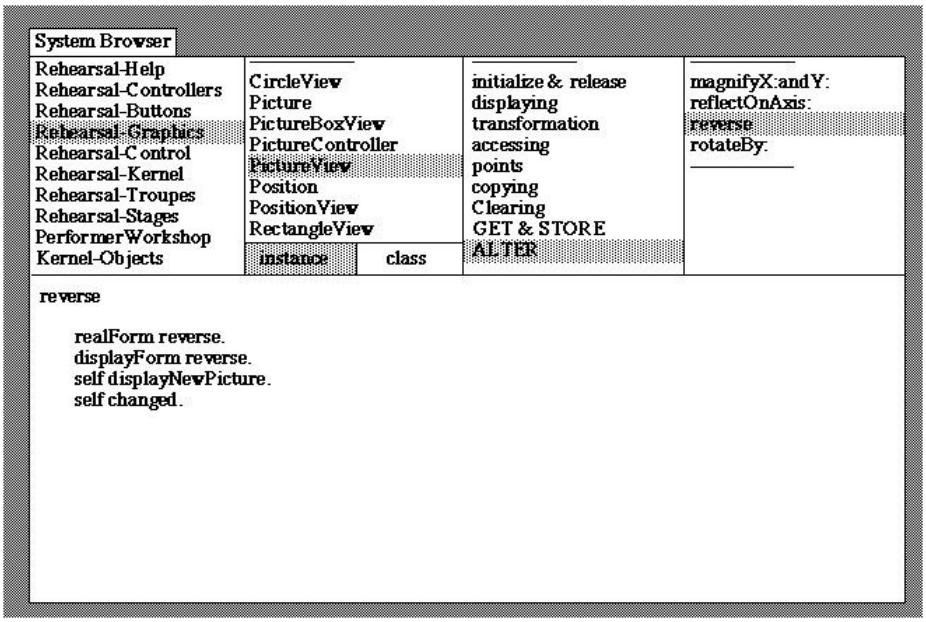

Figura 4: Ambiente Smalltalk-80 onde é executado o *Rehearsal World*.

Fonte: Gould, 1993.

Cocoa

Cocoa (Figura 5) é um tipo de linguagem de programação visual baseada na determinação de regras. Esta linguagem, conforme Burnett (1999), não possui as características para se fazer uma programação de uso geral, mas sim para tornar acessível a habilidade de crianças programarem suas próprias simulações. No *Cocoa* são encontradas as seguintes estratégias de programação visual: *Feedback*, Concretização e Imediatismo.

Figura 5: Um personagem do *Cocoa* seguindo as regras determinadas para ele.

Fonte: Burnett, 1999.

As regras, segundo Evangelista (2002), são divididas em duas partes, sendo que a da esquerda especifica a pré-condição e a da direita, graficamente, especifica a ação a ser empregada, conforme mostrado na Figura 6. Dando um duplo clique em qualquer personagem, irá abrir uma janela contendo todas as regras que dirigem os comportamentos do personagem em questão. Em cada ciclo da execução, cada uma das regras do personagem são consideradas de cima para baixo na lista de personagens. De acordo com Burnett (1999), a cada regra executada por um personagem, o sistema passa para o próximo personagem e nenhuma regra a mais para aquele personagem é checada até o próximo ciclo.

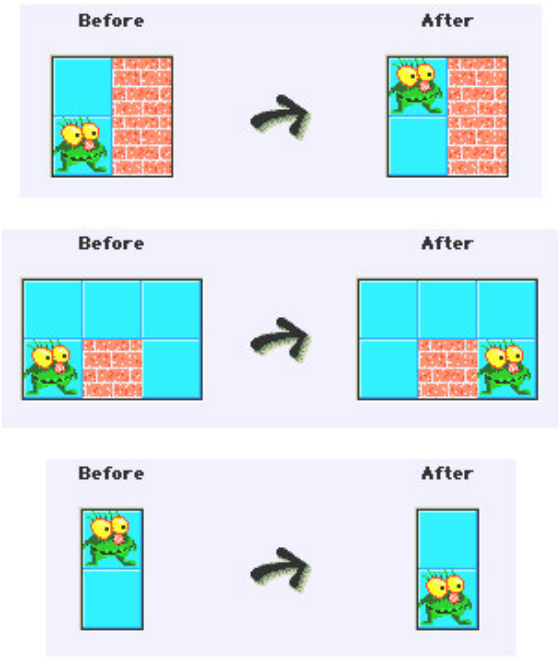

Figura 6: Exemplo de um conjunto de regras ensinando um personagem do *Cocoa* a pular o muro.

Fonte: Evangelista, 2002.

ThingLab

Evangelista (2002) aponta que *ThingLab* é um ambiente de programação que foi desenvolvido originalmente para a simulação gráfica de experimentos em Física e Geometria, como um laboratório da simulação para o uso educacional, porém, a grande contribuição proporcionada por este ambiente na computação, foi a inclusão de uma linguagem de programação visual baseada em restrições.

Os objetos em *ThingLab* são representados por caixas e linhas (Figura 7), os quais enviam e recebem mensagens. Os comportamentos executados pelos objetos contidos no ambiente, segundo Verwoerd (1999), são descritos pelas restrições ligadas a eles. As restrições são representadas como regras associadas a métodos, sendo que as regras são utilizadas em testes de verificação das restrições e os métodos descrevem alternativas que satisfaçam as restrições. Evangelista (2002) aponta que o *ThingLab* deve satisfazer as restrições definidas pelo programador para o objeto.

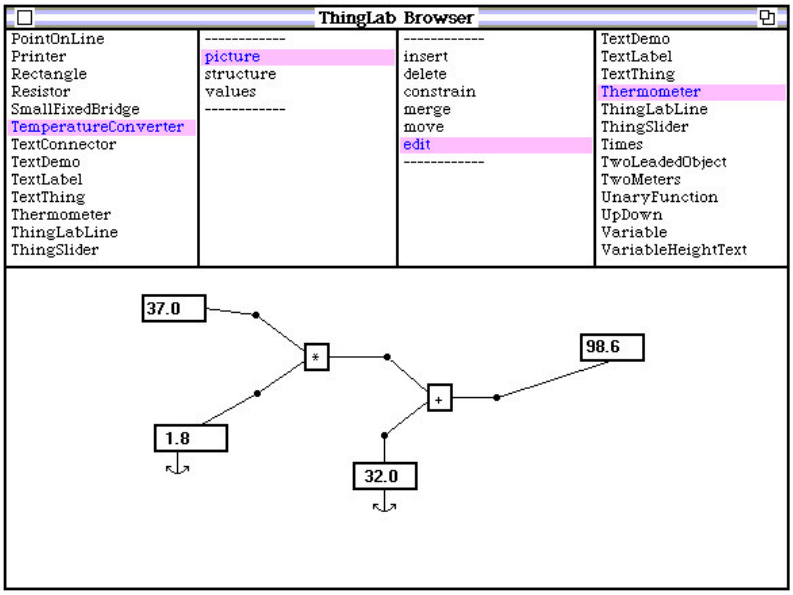

Figura 7: Exemplo de uma aplicação em *ThingLab*. Neste exemplo, as caixas conectadas a um ícone simbolizando uma âncora são constantes e as demais são variáveis.

Fonte: Evangelista, 2002.

A linguagem de programação visual *ThingLab* foi escrita em Smalltalk. Verwoerd (1999) acredita que *ThingLab* é um ambiente altamente gráfico e de fácil manuseio, porém possui uma certa dificuldade em representar sistemas complexos, no entanto, pode ser considerada como uma excelente ferramenta educacional.

Forms/3

Forms/3 é um ambiente de programação que foi fundamentado com o paradigma de formulários, sendo uma linguagem de programação visual baseada em formulários, proporcionando uma aparência de planilhas eletrônicas. Com isso, como colocado em Evangelista (2002), um dos aspectos abordados em *Forms/3*, é sua estrutura de células que se encontram arranjadas no formulário, como mostra a Figura 8.

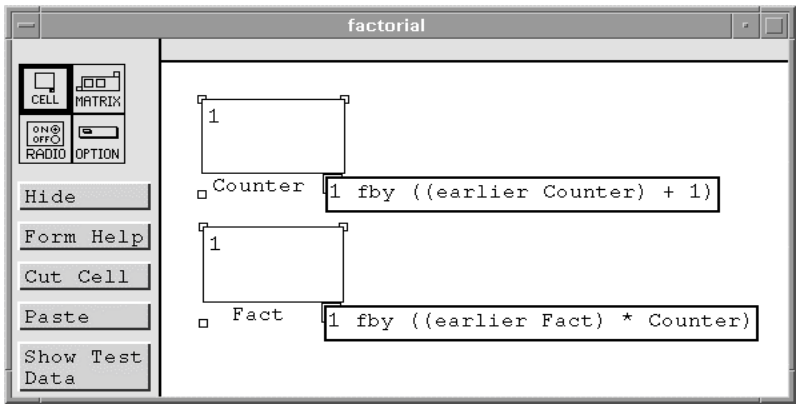

Figura 8: Exemplo de uma implementação para o cálculo fatorial realizada em *Forms/3*.

Fonte: Evangelista, 2002.

Segundo Burnett et al (2001), cada célula possui uma fórmula e alguns atributos visuais para o controle de sua aparência, sendo que o resultado do programa é determinado pela combinação destas fórmulas e atributos. O valor da célula é o resultado da execução da fórmula (Figura 9). Conforme Burnett (1999), as fórmulas se combinam em uma cadeia de restrições e o sistema garante que todos os valores exibidos na tela satisfaçam estas restrições.

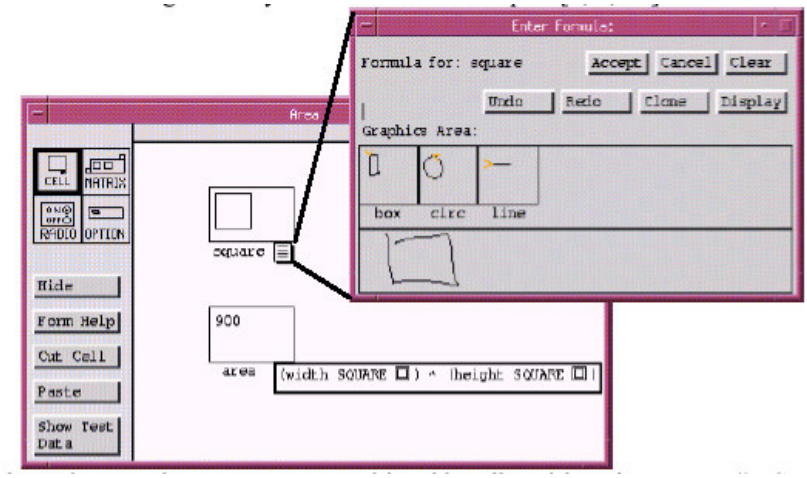

Figura 9: Exemplo da definição da área de um quadrado usando células e fórmulas no *Forms/3*.

Fonte: Burnett, 1999.

O público intencional de *Forms/3* são os programadores cujo trabalho é o de criar aplicações. Uma das metas do *Forms/3*, segundo Burnett (1999), tem sido reduzir o número e a complexidade de mecanismos necessários para fazer uma programação de aplicativos, com a intenção de aumentar a facilidade de uso por programadores.

No *Forms/3* estão presentes algumas das estratégias das linguagens de programação visual, tais como as descritas em Burnett (1999): *Feedback* que é fornecido em todos os casos, Concretização que está presente pelo fato da caixa resultante ser imediatamente vista quando as fórmulas são fornecidas e Imediatismo, presente na manipulação direta dos dados.

No capítulo a seguir, será feita uma apresentação da ferramenta Mundo dos Atores proposta por Mariani (1998), sendo que a implementação do ambiente proposto neste trabalho, foi feita sobre esta ferramenta.

Capítulo 3 – Mundo dos Atores

A seguir é feita uma apresentação da ferramenta Mundo dos Atores, mostrando, inicialmente, a linguagem/ambiente de programação Smalltalk, onde foi construído o Mundo dos Atores. Neste capítulo são apresentadas as características deste modelo, dando uma idéia geral do funcionamento do mesmo e apresentando alguns exemplos.

3.1. Ambiente Smalltalk

Um ambiente de programação, de acordo com Sebesta (2000), pode ser definido como um conjunto de ferramentas utilizadas para o desenvolvimento de um *software*. Smalltalk é uma linguagem e um ambiente de programação integrados e é o pioneiro no uso de janelas, sendo que sua interface é altamente gráfica, fazendo uso de janelas sobrepostas, menus suspensos e de um dispositivo de indicação de mouse para entrada dos dados.

Segundo Jonathan (1994), Simula, idealizada em 1966 na Noruega, foi a primeira linguagem a fazer uso da definição de classes de objetos na forma hierárquica de classes e subclasses. Smalltalk, desenvolvida no Centro de Pesquisas da *Xerox* na década de 70, incorporou as idéias de Simula e também o princípio de objetos ativos prontos para reagir a mensagens que ativam comportamentos específicos do objeto. Em Smalltalk não se "*chama uma função ou procedure, mas realmente envia uma mensagem a um objeto*" Abdala (2002).

Um programa em Smalltalk é todo composto por objetos. Os objetos, conforme Jonathan (1994), são encarados como "*processadores idealizados*" individuais e independentes, pois a eles podem ser transmitidos comandos na forma de mensagens. Após o reconhecimento de uma mensagem, o objeto ativa a rotina pertencente ao seu repertório particular e referente ao comportamento associado a mensagem recebida. Segundo Aguirre (2003), Smalltalk foi projetado com o intuito de ser de fácil aprendizado e utilização, sendo uma linguagem expressiva que usa um sub-conjunto da

linguagem humana, podendo ser usadas expressões claras, de acordo com o pensamento humano.

Para Abdala (2002), o conceito de polimorfismo em Smalltalk é muito importante, pois ele permite a criação de vários objetos diferentes possuindo métodos com o mesmo nome, sendo que isso não altera o comportamento do objeto, o qual se comporta de acordo com a mensagem recebida.

Um método, de acordo com Sebesta (2000), especifica a reação do objeto no momento em que o mesmo recebe uma mensagem correspondente àquele método. Contudo, a computação em Smalltalk é feita por meio de envio de mensagens, ou seja, a mensagem é enviada para um objeto, a partir daí, ele chama seus métodos e retorna com uma resposta à mensagem recebida.

A Figura 10 exibe a interface de Smalltalk.

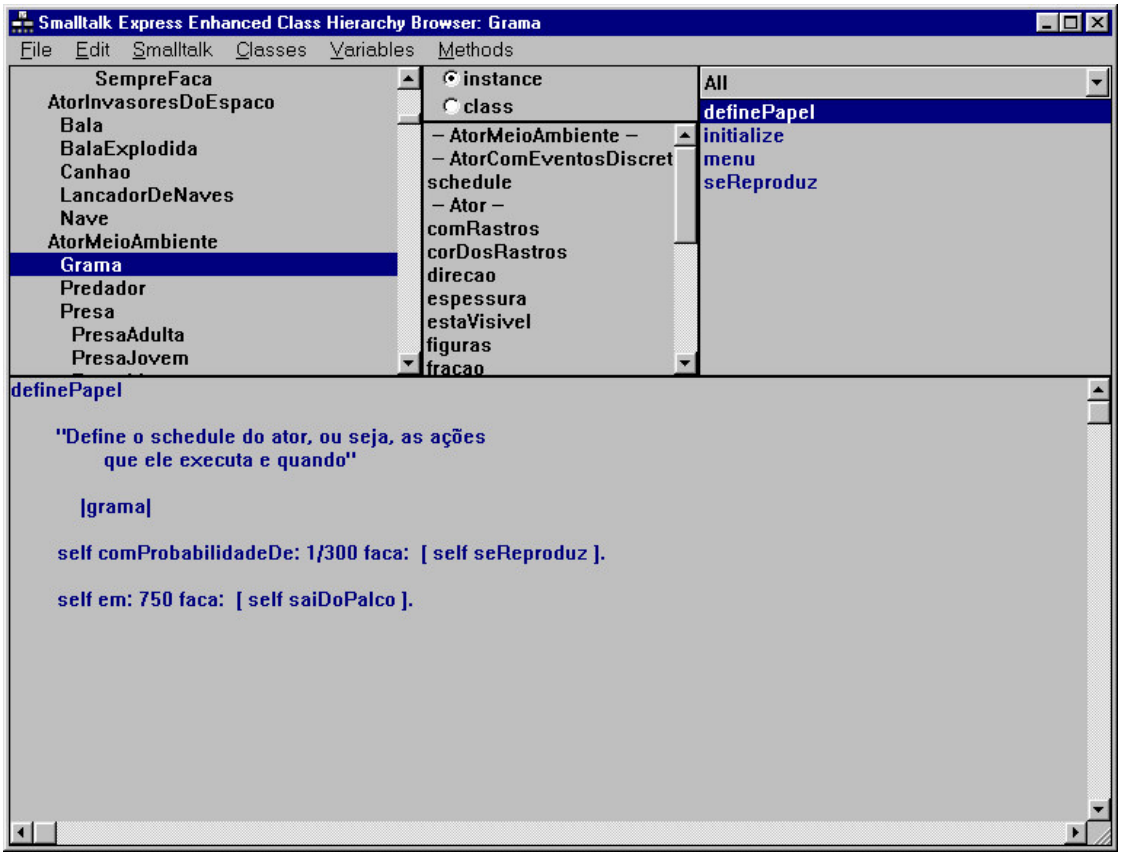

Figura 10: Interface gráfica do ambiente Smalltalk.

3.2. Modelo Mundo dos Atores

A ferramenta Mundo dos Atores, implementada e integrada ao ambiente/linguagem Smalltalk, foi proposta por Mariani (1998) com a intenção de que a mesma serviria como uma disciplina prática de introdução a POO em cursos de computação, porém ela superou as expectativas passando a oferecer um ambiente muito bom para implementações que apresentam um grau razoável de complexidade.

Esta ferramenta proporciona, gradativamente, o aprendizado de POO e suporta a aprendizagem pelo método não seqüencial. Por estar vinculada ao ambiente/linguagem Smalltalk, de acordo com Pianesso (2002), algumas dificuldades quanto ao uso da sintaxe foram observadas, sendo que isto não invalida o uso da ferramenta, pois seu objetivo é justamente o de permitir a aprendizagem de uma linguagem de programação orientada a objetos.

O Mundo dos Atores, como mostra a Figura 11, é formado por uma área chamada palco e uma área específica para a entrada dos comandos. Com isso, o programador trabalha com uma metáfora de palco, que, segundo Pianesso (2002), fazendo uma analogia com um teatro, o palco seria o espaço onde os atores (objetos) desempenham seus papéis, que no caso do presente trabalho, seriam os comportamentos.

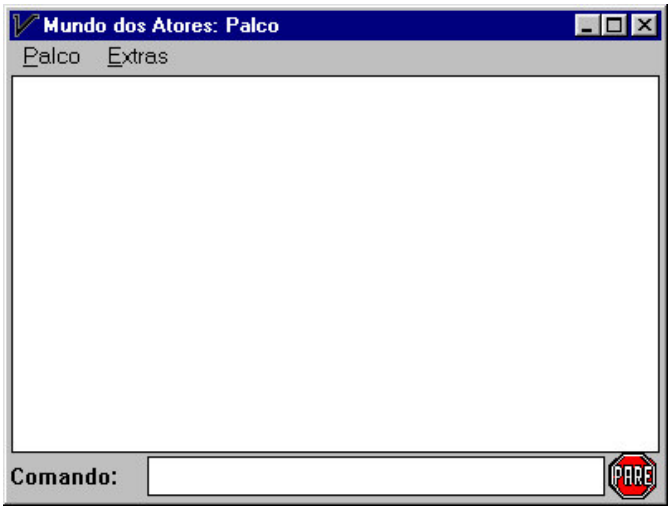

Figura 11: Interface gráfica do Mundo dos Atores.

De acordo com Schütz (2003), cada ator mantém uma referência com o palco e o palco com os atores nele incluídos, assim como, cada palco e cada ator possuem um conjunto de ações básicas que podem ser identificadas pelo programador, designadas como uma espécie de roteiro, indicando os papéis que o ator deverá desempenhar no palco. Segundo Pianesso (2002), a definição dos papéis possibilita comportamentos diferenciados para os atores, principalmente referente à movimentação no palco e à interação com os demais atores presentes no palco. Durante a animação, conforme Mariani (1998), cada ator desempenha seu papel de forma independente, sendo que o comportamento geral do sistema resulta dos comportamentos individuais de cada ator.

O Mundo dos Atores controla a execução dos papéis definidos para os atores através do mecanismo de paralelismo por eventos discretos, ou seja, utiliza o paradigma condição $\rightarrow$ ação. No modelo de eventos discretos, conforme exposto em Mariani (1998), a cada unidade de tempo discreto os atores executam suas ações trocando mensagens entre si, gerando uma ilusão de paralelismo. As ações dos atores estão descritas em um roteiro, indicando seus papéis no palco, os quais possibilitam o estabelecimento de comportamentos para os atores. De acordo com Aguirre (2003), os atores com eventos discretos que estão associados a um papel, deixam de ser apenas objetos reativos (estáticos) e passam a ser objetos pró-ativos (dinâmicos), pois, além de reagirem a comandos, passam a desempenhar seus comportamentos conforme a descrição do seu papel.

A construção do Mundo dos Atores foi baseada na linguagem LOGO. LOGO, segundo Ferruzzi (2001), é uma linguagem procedural sendo fácil criar novos termos ou procedimentos e os comandos básicos são termos do cotidiano. A linguagem LOGO possui um objeto chamado tartaruga, que é um pequeno cursor que aparece na tela do computador que obedece aos comandos da linguagem. O LOGO é formado por comandos chamados primitivos, os quais constituem a base de todos os procedimentos, tais como: para frente (PF), para direita (PD), para esquerda (PE), para trás (PT).

A Figura 12 mostra alguns dos comandos primitivos de LOGO. A partir desses comandos primitivos, podem-se criar outros comandos que são denominados

procedimentos, os quais, a partir do momento que estiveram na memória do programa, podem ser executados como os comandos primitivos.

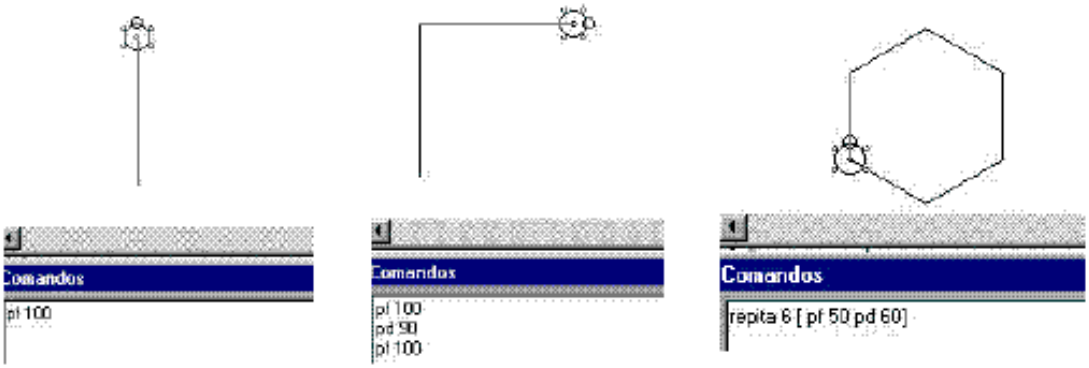

Figura 12: Comandos primitivos da Linguagem LOGO.

Fonte: Ferruzzi, 2001.

O LOGO, conforme Medeiros (2003), é uma linguagem de programação desenvolvida especialmente para o meio educacional, buscando auxiliar na formação de indivíduos capazes de investigar situações e trabalhar na solução de problemas. O processo de aprendizagem do LOGO se destina a resgatar um ambiente onde o conhecimento não é passado para o aluno, mas onde a aluno, interagindo com os objetos do ambiente, possa desenvolver alguns conceitos, como conceitos geométricos.

Para LOGO (2003), a linguagem LOGO é uma linguagem de programação interativa para todas as idades com recursos poderosos tais como métodos, recursão e a manipulação de programas como dados, fornecendo um ambiente de programação interativo. A linguagem LOGO consiste em aprender através do processo de "ensinar" o computador, o qual executa fielmente a descrição fornecida. De acordo com LOGO (2004), O *feedback* fiel e imediato é desprovido de qualquer animosidade ou afetividade que possa haver entre o programador e a máquina.

O Mundo dos Atores possui seu ambiente similar ao ambiente oferecido em LOGO. Como exemplo do ambiente do Mundo dos Atores, pode ser citado o Palco da

Caneta, no qual o objeto caneta pode ser controlado da mesma forma que o objeto tartaruga em LOGO.

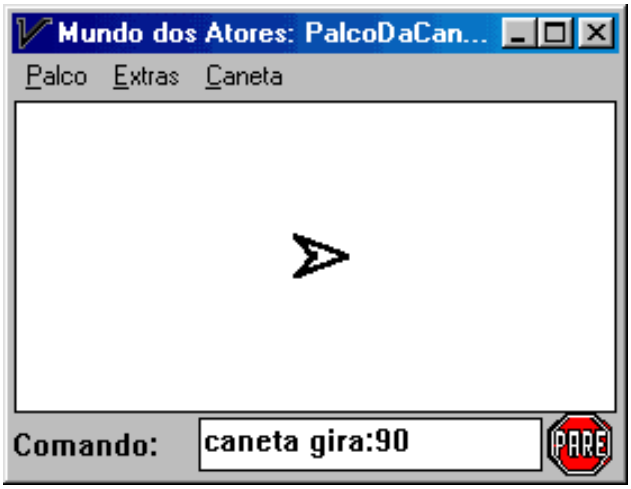

Figura 13: Palco da Caneta no Mundo dos Atores.

Fonte: Mariani, 1998.

Como mostrado na Figura 13, os papéis que cada ator irá desempenhar no palco são descritos na própria interface, na parte específica para a entrada dos comandos. Para isso, deve-se informar o ator que irá receber o comando seguido pelo nome do comportamento e dos valores, se for o caso. Por exemplo:

"*caneta anda: 50*"

"*caneta gira: 90*"

O comando “caneta anda: 50”, descrito acima, faz com que o ator caneta se movimente 50 unidades para a direção em que ele está apontando. Já o comando “caneta gira: 90” faz com que o ator caneta dê um giro de 90° no sentido horário. Mariani (1998) aponta que as ações *anda:* e *gira:* mencionadas acima, já fazem parte do repertório das ações básicas do ator caneta. Associado à área de comandos há um menu que apresenta algumas mensagens que também podem ser enviadas para o ator caneta, como mostrado na Figura 14 a seguir:

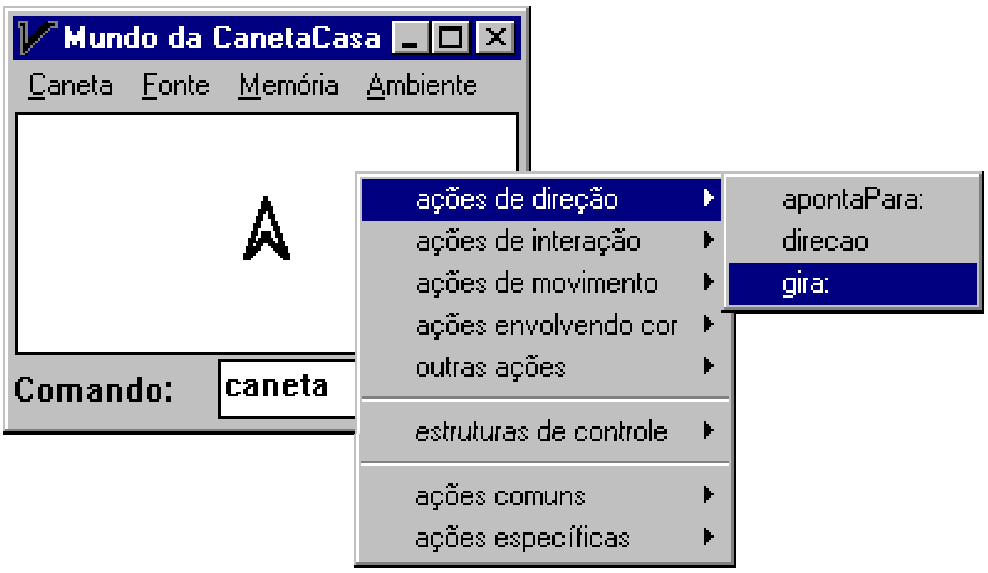

Figura 14: Mensagens que podem ser atribuídas para o ator caneta pertencente ao Palco da Caneta no Mundo dos Atores.

Fonte: Mariani, 1998.

Além das ações básicas, segundo Schütz (2003), novas ações podem ser adicionadas ao repertório do palco como também dos atores, como exemplo, as ações apresentadas nas Figuras 15 e 16, citadas em Mariani (1998), as quais fazem parte do trabalho de desenhar um palhaço.

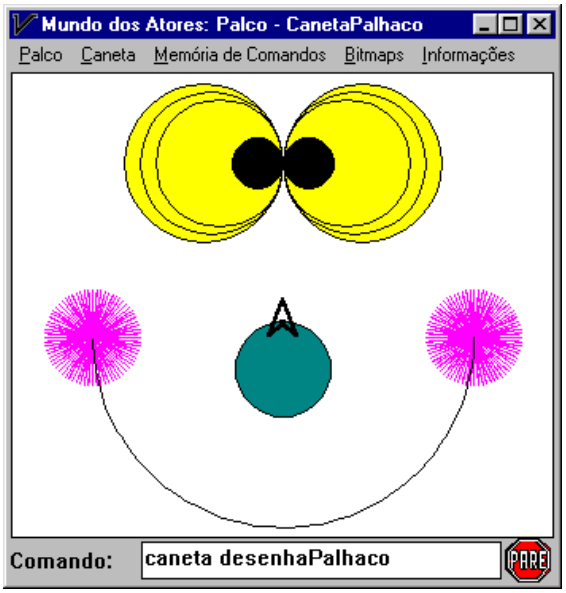

Figura 15: Palco da Caneta – desenho de um palhaço.

Fonte: Mariani, 1998.

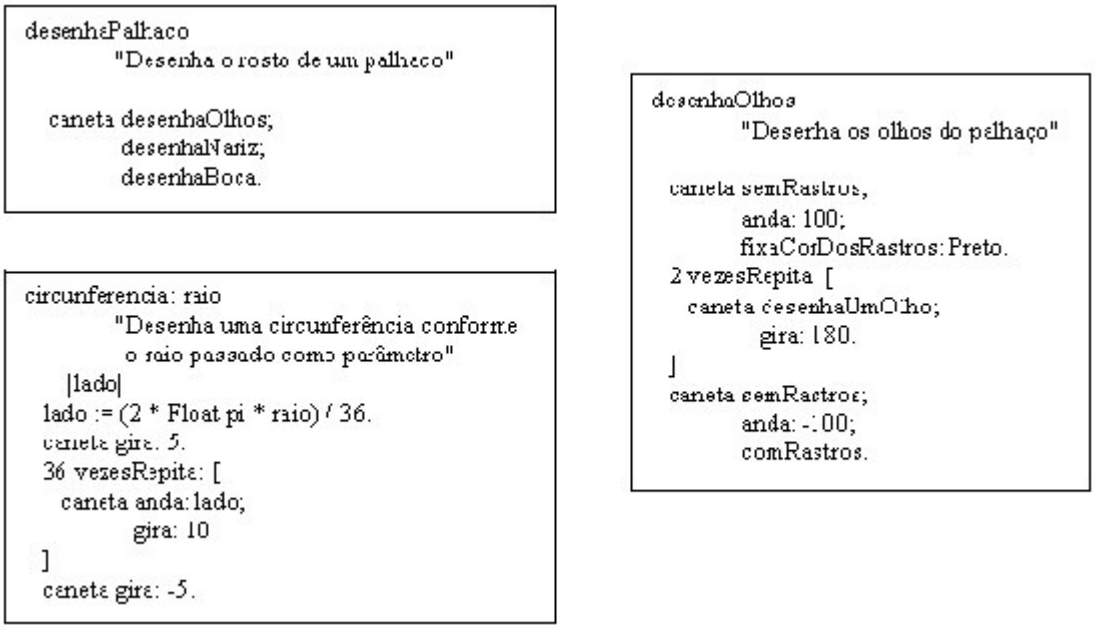

Figura 16: Palco da Caneta – comandos utilizados para o desenho do palhaço.

Fonte: Mariani, 1998.

Segundo Mariani (1998), algumas características do Mundo dos Atores fazem com que o mesmo se torne um ambiente rico para a exploração dos conceitos de POO, tais como:

- Metáfora de Palco: como, no Mundo dos Atores, o aprendiz trabalha com elementos virtuais (como a caneta), com atores e com o palco, tendo estes uma correspondência direta com elementos concretos equivalentes, desde o início ocorre um contato com objetos, mesmo que esta noção ainda não tenha sido formalmente apresentada para o aprendiz.

- Aberto e expansível: o palco mais simples do Mundo dos Atores, representado pelo Palco da Caneta (Figura 13), oferece uma funcionalidade similar a oferecida nas implementações feitas na linguagem *Logo*, onde é apresentada uma área gráfica (equivalente ao palco) e uma tartaruga (equivalente a caneta). O Mundo dos Atores permite, também, a definição de outros atores e a criação de novos palcos, sendo que

esta possibilidade de construção progressiva de modelos cada vez mais complexos é o ponto base para a exploração sistemática dos conceitos de POO.

- Integração com um ambiente de POO: como o Mundo dos Atores está implementado na linguagem/ambiente Smalltalk, ele proporciona uma total integração das aplicações e ferramentas oferecidas pelo ambiente, com isso, o Mundo dos Atores pode ser facilmente integrado com qualquer outra aplicação disponível. Isto significa que o programador não está restrito aos recursos oferecidos pelo Mundo dos Atores e sim livre para utilizar os recursos oferecidos pelo Smalltalk. Com esta característica, pode ocorrer a passagem gradativa e incremental para o desenvolvimento de sistemas computacionais independentes do Mundo dos Atores, como as aplicações comerciais.

Como exposto no início do item 3.2, a ferramenta Mundo dos Atores foi criada com o intuito de ser utilizada em disciplinas introdutórias de POO em cursos de computação devido a mesma proporcionar a manipulação de objetos. Com isso e com a utilização do modelo de eventos discretos, para ilustrar o potencial pedagógico da ferramenta, assim como em Mariani (1998), na Figura 17 será mostrado, como exemplo, o ecossistema chamado *PalcoMeioAmbiente*:

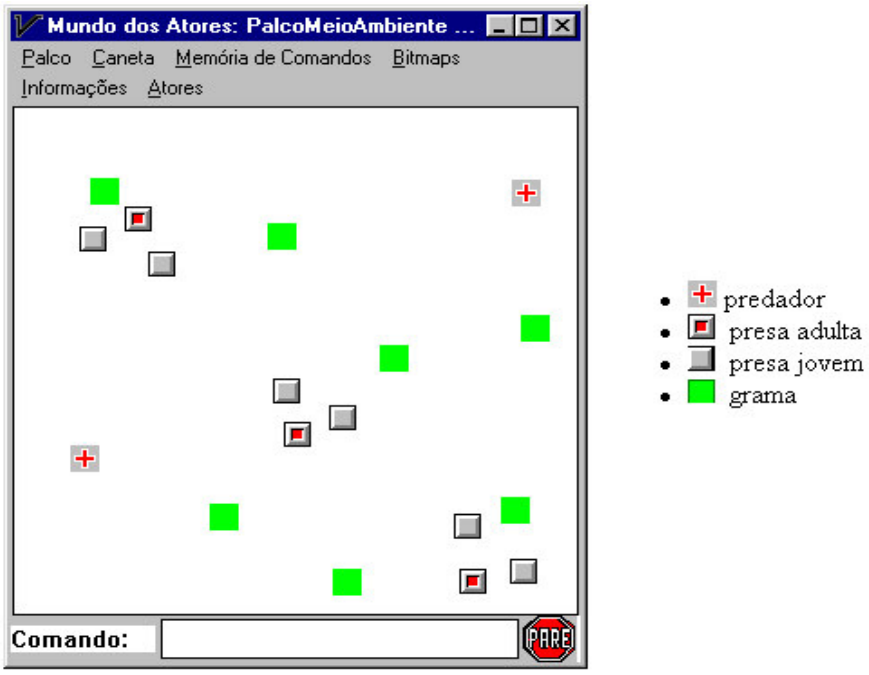

Figura 17: Palco Meio Ambiente.

Fonte: Mariani, 1998.

O *PalcoMeioAmbiente* possui como atores o predador, presa adulta, presa jovem e grama. No momento da animação, cada um desses atores irá executar o seu papel de forma independente. Por exemplo, o papel do ator *Grama* (ilustrado na Figura 18), consiste em reproduzir-se a cada 300 unidades de tempo e morrer quando atingir 750 unidades de tempo de vida.

```
DefinePapel
  "Define o papel da grama"
    |grama|

  self cada: 300 faca:  [
    grama :=  Grama new.
    self palco adiciona: grama.
    grama
          pulaPara: self posicao;
          apontaPara:
                (self aleatorio: 360);
          anda: self tamanho x.
  ].

  self em: 750 faca:  [self destroi].
```

Figura 18: Palco Meio Ambiente: papel do ator *Grama*.

Fonte: Mariani, 1998.

O papel do ator *PresaJovem* (Figura 19) é o de adotar uma *PresaAdulta* como mãe e seguí-la para onde ela for. Se a *PresaJovem* não tiver mãe, ela anda aleatoriamente (sem rumo) pelo ambiente. Após 1000 unidades de tempo a *PresaJovem* passa a ser uma *PresaAdulta*.

```
definePapel
    "Define o papel de um presa jovem"
       | distancia|
    self sempreQue: [mae isNil or: [mae palco isNil]] faca: [mae := self adultoMaisProximo].
    self sempreFaca: [
          mae isNil ifTrue: [
                self gira: (self aleatorio: 90) - 45; anda: 5.
          ] ifFalse: [
                (distancia := self distanciaPara: mae) > 15 ifTrue: [
                   self apontaPara: mae; anda: (distancia - 15  min: 13)
                ].
          ].
     ].
    self em: 1000 faca: [self tornase: PresaAdulta new].
```

Figura 19: Palco Meio Ambiente: papel do ator *PresaJovem*.

Fonte: Mariani, 1998.

O papel do ator *Predador* já é um pouco mais complexo. O ator inicia com um certo estoque de energia e a cada instante de tempo o nível da energia vai reduzindo uma unidade. O ator morre após 3000 unidades de tempo ou quando seu nível de energia se encontrar zerado. O Predador sai a procura de caça somente quando seu nível de energia for inferior a 700, sendo que a cada presa caçada, o nível de energia do *Predador* aumenta. Ainda, se o ator estiver com o seu nível de energia superior a 400, de tempos em tempos o mesmo se reproduz. A definição do papel do ator *Predador*, se encontra na Figura 20.

```
DefinePapel
    "Define o papel de um predador "
            |jovem|
    "Inicialmente o predador tem um certo estoque de energia (nivelDeAlimentação). A cada unidade
    de tempo discreto este nível é reduzido em uma unidade"
 self sempreFaca: [ nivelDeAlimentacao :=  nivelDeAlimentacao - 1].

    "O nivelDeAlimentação chegou a zero; ele morre"
 self sempreQue: [nivelDeAlimentacao = 0] faca: [self destroi].

    "O nivelDeAlimentação é menor que 700; ele procura caça. Cada presa que ele caça aumenta o
    seu nível de alimentação"
 self sempreQue: [nivelDeAlimentacao < 700] faca: [self caca].

    "Após 3000 unidades de tempo ele e morre"
 self em: 3000 faca: [self destroi ].

    "De tempos em tempos, se reproduz, somente se estiver alimentado"
 self cada: 1050 faca:  [
        nivelDeAlimentacao > 400 ifTrue: [
          jovem := Predador new.
          self palco adiciona: jovem.
          jovem pulaPara: self posicao.
        ].
].
```

Figura 20: Palco Meio Ambiente: papel do ator *Predador*.

Fonte: Mariani, 1998.

Um outro exemplo de implementação feita no Mundo dos Atores usando o modelo de eventos discretos que também pode ser citada, é o trabalho feito por um

aluno que teve participação no projeto Museu Virtual[1]. O projeto Museu Virtual tem como objetivo o desenvolvimento de uma ferramenta de autoria para a construção colaborativa de museus em realidade virtual onde imagens e objetos 3D possam ser expostos. Neste projeto, segundo Schütz (2003), propõe-se que estudantes em locais diferentes possam, simultaneamente, construir objetos em ambientes de realidade virtual e definir comportamentos para estes objetos.

Neste trabalho, descrito em Pinter (2001), o aluno desenvolveu a implementação de um protótipo de interface para a edição de comportamentos de atores em um ambiente de eventos discretos utilizando programação visual.

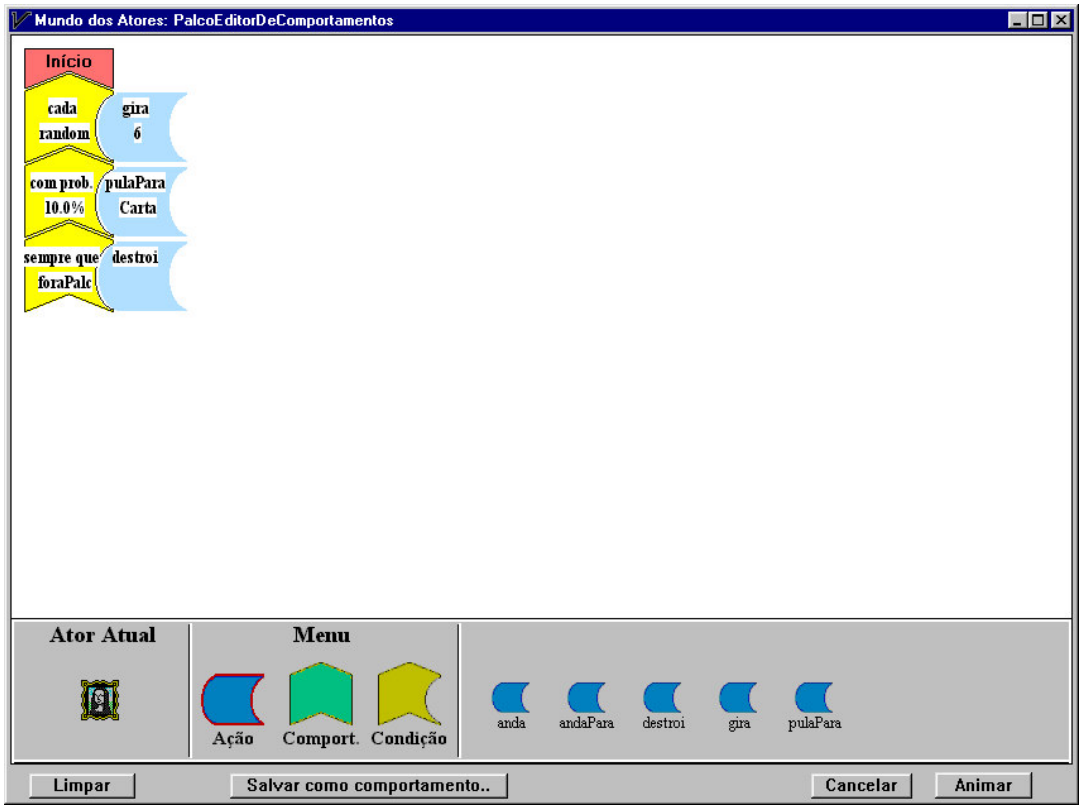

Figura 21: Interface do trabalho feito por Pinter (2001).

O projeto Museu Virtual, é coordenado pelo Prof. Dr. Raul Sidnei Wazlawick e possui como Instituição responsável, a Universidade Federal de Santa Catarina – UFSC.[1]

O objetivo do trabalho criado por Pinter (2001) era o de criar uma nova forma de programação, mais intuitiva, visual e que conduzisse o programador de tal forma que o mesmo não necessitasse de conhecimentos em programação de computadores. Com isso, foi criada uma interface gráfica (Figura 21) que permite a criação de comportamentos através de peças do tipo quebra-cabeça com grupos de ações e condições pré-definidas, representadas graficamente. A cada peça encaixada, é aberto um editor de valores para uma fácil modificação dos atributos da mesma. Com essas peças, o programador também tem condições de definir comportamentos mais complexos aos atores a partir de conjuntos de peças encaixadas.

Este protótipo, apesar de usar o paradigma condição → ação, que não é o caso do presente trabalho, foi usado como inspiração para a definição do ambiente visual de programação aqui proposto, sendo que este ambiente disponibiliza em sua interface, as definições de comportamento contidas em Schütz (2003), as quais serão descritas na próxima seção deste trabalho.

Capítulo 4 – Programação por Comportamentos

Este capítulo expõe uma noção sobre programação por composição de comportamentos, mostrando a classificação dos mesmos e a descrição dos comportamentos utilizados na definição do ambiente visual aqui proposto. Ainda neste capítulo, é exposto que os comportamentos podem ser condicionados, possuindo limitações para que sejam executados, com isso, as condições utilizadas neste trabalho são descritas.

4.1. Comportamento

Comportamento, segundo Schütz (2003), pode ser definido como uma ação contínua executada durante toda a vida de um ator (objeto), podendo ser modificada. O comportamento é a definição do papel a ser executado pelo ator durante sua existência (aparecendo ou não no ambiente de programação). Os comportamentos não precisam, necessariamente, estar visíveis para o programador.

Para Pianesso (2002), o comportamento é um conjunto de ações condicionadas que devem ser executadas a cada instante de tempo combinadas com o princípio de ocultação de informações.

4.2. Classificação dos Comportamentos

Os comportamentos, de acordo com Schütz (2003), são classificados como comportamentos evidentes, comportamentos latentes, comportamentos simples, comportamentos compostos e comportamentos terminais. A seguir, uma breve descrição das classificações citadas acima:

- Comportamentos Evidentes: são os comportamentos visíveis ao programador, ou seja, enquanto estiverem ativos, sua execução é perceptível todo o tempo, independente de qualquer condição.

- Comportamentos Latentes: são comportamentos que, apesar de estarem ativos, não são visíveis ao programador a todo instante, pois estes necessitam de uma ou mais condições para que sua execução seja realizada.

- Comportamentos Básicos: são comportamentos simples, ou seja, não são compostos por nenhum outro comportamento, sendo o elemento básico para a construção de outros comportamentos complexos com finalidades mais específicas. Não necessitam de outros comportamentos para completar sua ação. Conforme exposto em Pianesso (2002), servem como ponto de partida para a geração de outros comportamentos, sendo que a combinação dos comportamentos base favorece a criação de novos comportamentos. Os comportamentos básicos podem ser evidentes ou latentes.

- Comportamentos Compostos (Complexos): são comportamentos formados pela união de dois ou mais comportamentos básicos para atingir um determinado objetivo, ou seja, são comportamentos que possuem outros comportamentos na sua formação, podendo ser gerados utilizando-se comportamentos base. Um comportamento, para Pianesso (2002), independente de ser complexo ou básico, pode participar da composição de outros comportamentos.

- Comportamentos Terminais: são comportamentos executados uma única vez durante a existência do objeto no ambiente.

4.3. Atributos

Os comportamentos podem conter atributos em sua definição. Para Schütz (2003), os atributos são os elementos que irão diferenciar a forma de como o ator irá desempenhar os comportamentos a ele designados conforme os valores atribuídos. Os valores podem ser alterados durante a existência do ator no ambiente.

Os tipos que poderão ser assumidos pelos atributos, são apresentados na tabela abaixo.

Tabela 1: Definição dos tipos dos atributos.

Tipo	Valores possíveis
Inteiro	Conjunto dos valores inteiros, positivos e negativos.
Real	Conjunto dos números Reais.
Caractere	Caracteres alfanuméricos e símbolos.
Lógico	Variável com conteúdo binário, representando verdadeiro ou falso.
Classe	Referencia uma classe de objetos, podendo ser definida na descrição do programa desenvolvido ou estar encapsulada no próprio modelo. Representa um conjunto de objetos.
Objeto	Pode ser definido como a instância de uma classe, sendo assim um exemplar de um conjunto de objetos definidos através de uma classe.

Fonte: Schütz, 2003.

A importância de se usar classes como atributos, é a possibilidade de definir interação com mais de um ator presente no ambiente ao mesmo tempo, pois quando se atribui um comportamento a um ator pertencente a uma classe x, todos os atores dessa classe possuirão o mesmo comportamento. Por exemplo, ao atribuir o comportamento perseguir rato a um ator do tipo gato, todos os gatos pertencentes a este grupo possuirão o comportamento perseguir rato, assim, basta definir uma classe Gato, onde qualquer gato terá o mesmo comportamento de perseguir um ator do tipo rato.

4.4. Descrição dos Comportamentos

A seguir, serão listados os comportamentos utilizados na definição da linguagem visual proposta, segundo a sintaxe definida por Schütz (2003):

nome_do_comportamento(nome_do_parâmetro$_1$:tipo_do_parâmetro$_1$; ...;
nome_do_parâmetro$_n$: tipo_do_parâmetro$_n$);

Onde,

- **nome_do_comportamento:** identificação do comportamento.

- **nome_do_parâmetro$_n$**: identificação do parâmetro.

- **tipo_do_parâmetro$_n$**: valorização do parâmetro (inteiro, real, lógico e funções definidas pelo programador no caso de parâmetros serem variáveis)

4.4.1. Comportamentos Básicos Evidentes

A partir da descrição da sintaxe dos comportamentos mostrada anteriormente, na tabela 2, foram colocados os comportamentos identificados como básicos e evidentes e que estão presentes na definição de linguagem visual proposta.

Tabela 2: Comportamentos básicos e evidentes presentes na definição proposta.

Comportamento	Descrição
`andar(velocidade : Numero);`	O ator movimenta-se sempre para a mesma direção obedecendo a velocidade repassada pelo parâmetro. Esta velocidade pode ser informada através de um valor constante ou aleatório.
`Girar(angulo : Numero);`	O ator permanece girando conforme o valor do ângulo passado como parâmetro, podendo ser um valor constante ou aleatório.

Fonte: Schütz, 2003.

4.4.2. Comportamentos Básicos Latentes

Como citado no item 4.2, a característica de um comportamento latente é que este necessita de pelo menos uma condição para que sua execução torne-se ativa e visível ao programador. Na tabela 3, foram colocados os comportamentos identificados como básicos e latentes e que estão presentes na definição de linguagem visual proposta, bem como o que deve acontecer para que o mesmo passe a ser visível no ambiente.

Tabela 3: Comportamentos básicos e latentes presentes na definição proposta.

Comportamento	Descrição	Quando se torna visível
bater(classe : Classe)	Se o ator em movimento tocar em outro ator do ambiente que for especificado como parâmetro, o ator que possuir este comportamento, irá mudar a direção de sua trajetória.	Este comportamento torna-se ativo e visível no momento em que acontecer o toque do ator possuidor do comportamento e do ator indicado como parâmetro para o toque.
fugir(classe : Classe; velocidade : Numero)	Este comportamento consiste em fugir do ator indicado como parâmetro presente no palco a uma velocidade a ele indicada.	O comportamento é percebido quando o ator passado como parâmetro encontra-se no palco.
perseguir(classe : Classe; velocidade : Numero)	Este comportamento consiste em perseguir o ator indicado como parâmetro presente no palco a uma velocidade a ele indicada.	O comportamento é percebido quando o ator passado como parâmetro encontra-se no palco.
andarPara(classe : Classe; velocidade : Numero; ponto : Coordenadas)	O ator possuidor deste comportamento irá andar a uma velocidade conforme indicada como parâmetro, a um determinado ator ou a um determinado ponto (x,y), sendo que ambos serão definidos como parâmetro.	O comportamento é percebido quando o ator passado como parâmetro encontra-se no palco ou se o ator possuidor do comportamento atingir os limites do palco.
pularPara(classe : Classe; velocidade : Numero; ponto : Coordenadas)	Da mesma forma que foi descrito anteriormente, o ator possuidor deste comportamento irá pular a uma velocidade conforme indicada como parâmetro, em direção a um determinado ator ou a um determinado	O comportamento é percebido quando o ator passado como parâmetro encontra-se no palco ou se o ator possuidor do comportamento atingir os limites do palco.

	ponto (x,y), sendo que ambos serão definidos como parâmetro.	

Fonte: Schütz, 2003.

4.4.3. Comportamentos Básicos Terminais

Conforme descrito no item 4.2, estes tipos de comportamentos ocorrem uma única vez durante a existência do ator no ambiente, pois, como conseqüência destes comportamentos, o ator deixará de existir. A seguir é descrito o comportamento básico terminal presente na definição de linguagem visual proposta.

Tabela 4: Comportamento básico terminal presente na definição proposta.

Comportamento	Descrição
destruirSe()	O ator é destruído, sendo assim, deixa de fazer parte do ambiente.

Fonte: Schütz, 2003.

4.4.4. Comportamentos Compostos

Também chamados de comportamentos complexos, são os comportamentos gerados a partir dos comportamentos básicos ou até mesmo da junção de outros comportamentos compostos. Estes comportamentos somente poderão ser criados a partir dos comportamentos básicos já definidos no ambiente.

Para estes comportamentos não haverá uma tabela com a descrição dos mesmos, pois eles não são pré-definidos e sim criados a partir de outros, mas aqui se pode dar um exemplo de como seria um comportamento complexo, juntando os comportamentos básicos *andar(velocidade)* e *girar(ângulo),* podendo, assim, compor o comportamento composto *andaEmCírculos*, como segue:

COMP andaEmCirculos ← {anda(15); gira(45)}

Para a composição dos comportamentos compostos, pode-se usar alguns conceitos da Orientação a Objetos, como:

- Herança: para Pianesso (2002), a herança permite com que um novo comportamento seja criado a partir de outro já existente, podendo haver uma reutilização, sendo que o comportamento filho herda as características do comportamento pai. Assim, de acordo com Schütz (2003), um comportamento definido como filho (subclasse) do comportamento pai (superclasse) possui todos os comportamentos e atributos do pai, além das novas características que podem ser atribuídas a ele.

- Agregação (ou composição): a agregação permite a formação de novos comportamentos a partir do agrupamento de comportamentos diferentes, sendo que, conforme Pianesso (2002), cada comportamento componente deste agrupamento, pode ser tanto um comportamento básico como composto. Pode-se citar o exemplo exposto acima do comportamento *andaEmCirculos*, definido pela agregação dos comportamentos básicos *andar(velocidade)* e *girar(ângulo),* o qual resulta em um comportamento onde o objeto, ao mesmo tempo em que vai se movendo para frente a uma velocidade passada como parâmetro, vai mudando a sua direção conforme o ângulo também informado.

- Especialização de parâmetros: a especialização dos parâmetros é usada para especificar ao máximo o comportamento que o ator vai executar durante a sua existência no ambiente, definindo os valores para seus atributos. Um exemplo disto, citado por Schütz (2003), é o comportamento *AndarADezPorHora*(), definido como a*ndar*(10), o qual mantém o comportamento original, porém faz uma especialização de um parâmetro através da fixação do valor 10.

4.5. Condições

O comportamento pode possuir uma ou mais condições. As condições podem ser definidas como uma limitação para a ativação dos comportamentos, ou seja, definem o momento em que o comportamento será executado. De acordo com Schütz (2003), estas

condições podem ser o tempo, proximidade ou contato com outros objetos pertencentes no ambiente, presença ou não no palco, entre outros. Exemplos:

1. O comportamento *explodir* do ator A, somente será executado quando ocorrer a condição *aoTocar* no ator B presente no comportamento do ator A.

2. O comportamento *bater* do ator C será executado somente se o ator próximo a ele for, por exemplo, um ator do tipo pedra, ou seja, se for um ator do tipo madeira, o ator C passará direto, sem ter seu comportamento executado.

Um ator não modifica outro ator, o que acontece é a execução de um comportamento a partir de outro, ou seja, se um ator do tipo pedra se transforma em um ator do tipo gato, quer dizer que o comportamento do ator pedra era de se transformar em gato a partir de um comportamento de outro ator.

Exemplo: o ator do tipo pedra executa o comportamento *transformarEmBola* quando ocorrer a condição *aoTocar* no ator x, pertencente ao comportamento do ator do tipo pedra.

As condições podem, aqui, ser definidas segundo a sintaxe apresentada em Schütz (2003):

```
Condição() ⇒ comportamento().
aoTocar(ator x) ⇒ pularPara(coord(x,y)).
```

No exemplo acima, a condição *aoTocar(ator x)* faz a restrição do comportamento *pularPara(coord (x,y)),* fazendo com que o ator possuidor deste comportamento dê um salto direto para as coordenadas passadas como parâmetro somente no momento em que ele tocar no ator do tipo x. Na tabela abaixo estão listadas as condições presentes na definição proposta:

Tabela 5: Condições presentes na definição proposta.

Condição	Descrição
aDistância(distancia : Numero; alvo : Ator)	Esta condição é verdadeira se um objeto da classe informada estiver a

	uma distância menor que o parâmetro "distância" do objeto dono do comportamento.
aoTocar(objeto : Ator)	Esta condição é verdadeira quando o objeto dono do comportamento estiver tocando um objeto da classe indicada.
sempreQue(valor1 : Número; comparação : Comparação; valor2 : Número);	Esta condição é verdadeira se valor1 for comparável a valor2 pelo elemento do tipo "Comparação" (>, <, <=, >=, =, <>).
aCada(tempo : Número)	Esta condição se torna verdadeira quando o tempo de vida do ator for múltiplo do parâmetro "tempo".
ComProbabilidade(percentual : Número)	Esta condição se torna verdadeira a cada instante com probabilidade dada por "percentual".

Fonte: Schütz, 2003.

As condições, assim como os comportamentos, podem ser simples ou compostas. As condições compostas são definidas pela união de condições simples e até mesmo compostas já definidas no ambiente. Para a ligação entre as condições, são usados os conectivos E, OU e NÃO. Segundo a sintaxe definida em Schütz (2003), uma condição composta pode ser descrita da seguinte maneira:

<condição composta> ::= <condição> | <condição composta> *E* <condição composta> | <condição composta> *OU* <condição composta> | *NAO* <condição composta>

Onde,

- **<condição>**: é uma condição simples;
- **conectivo *E***: a condição composta resultante da combinação de duas condições por *E* será verdadeira quando ambas forem verdadeiras;
- **conectivo *OU***: a condição composta resultante da combinação de duas condições por *OU* será verdadeira quando pelo menos uma delas for verdadeira;

- **operador *NÃO***: funciona como um inversor da condição, fazendo com que esta opere de forma contrária à sua definição, sendo que se torna verdadeira quando a condição a que foi imposta tal operador for falsa.

Com a descrição da ferramenta Mundo dos Atores feita no capítulo 3, sobre a qual foi construído o ambiente visual em questão e a apresentação, no capítulo anterior, das definições de comportamentos contidas em Schütz (2003), as quais são disponibilizadas na interface do ambiente, no próximo capítulo é feita a exposição do ambiente visual de programação proposto neste trabalho.

Capítulo 5 – Definição do Ambiente Visual Proposto

Nesta seção é feita uma apresentação da definição do ambiente de programação visual proposto, fazendo uma descrição do mesmo.

5.1. Interface do Modelo

Como se trata de programação visual, a interface escolhida para a definição proposta, foi baseada no conceito de peças de quebra-cabeça, ou seja, o ator, as condições, os comportamentos e os conectivos, são representados por peças do tipo quebra-cabeça. Com isso, acredita-se estar proporcionando uma metáfora conhecida por praticamente todos os tipos de usuários, conhecedores de programação ou não, pois, na interface apresentada, o programador pode atribuir os comportamentos para os atores, encaixando as peças umas nas outras e preenchendo alguns parâmetros.

Através do encaixe das peças, o sistema pode proporcionar ao programador uma manipulação direta, conforme as utilidades cognitivas da linguagem visual descritas no item 2.4, pois é permitido ao programador executar as ações interagindo diretamente com os objetos exibidos na tela do computador ao invés de ter que descrevê-las, fazendo com que as operações sejam realizadas sem a descrição e conhecimento dos comandos e sim apenas pela atribuição de alguns valores.

5.2. Definição do Ambiente

A definição do ambiente de programação visual proposto, tem sua implementação desenvolvida no Smalltalk Express e construída sobre o Mundo dos Atores. Os principais elementos do ambiente são o palco e os atores. O palco é o local onde os objetos (atores) irão executar os respectivos comportamentos a eles atribuídos e também onde haverá a interação com os demais atores existentes no ambiente.

Nesta definição, é considerado que o ator, a quem será atribuído um ou mais comportamentos, já existe no ambiente, ou seja, o mesmo será apenas selecionado e não criado dentro do mesmo.

Como exposto acima, na interface do ambiente foram usadas peças do tipo quebra-cabeça, como as que seguem na Figura 22:

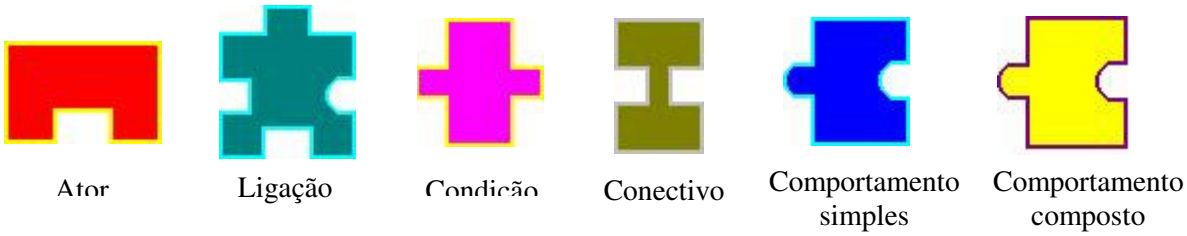

Figura 22: Peças utilizadas na interface do ambiente proposto.

A idéia dos encaixes das peças é mostrada na Figura 23 bem como a direção da varredura para a execução dos comportamentos, que se dá da esquerda para a direita e de cima para baixo:

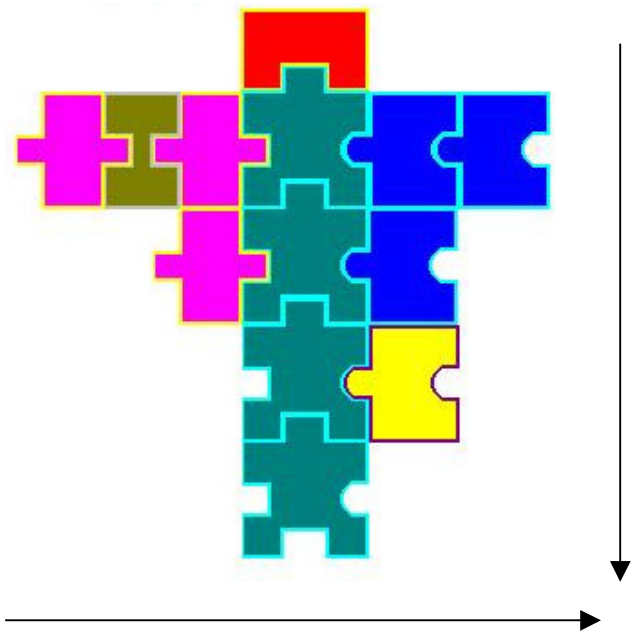

Figura 23: Exemplo do encaixe das peças contidas na definição proposta e direcionamento da varredura para a execução dos comportamentos.

Sempre quando for adicionado um comportamento, o sistema deixa em aberto a próxima ligação. Nem todo o comportamento precisa ter condição ligada a ele. Quando for atribuído um comportamento e não for atribuído ao mesmo nenhuma condição, subentende-se que o ator "sempre" deverá executar o comportamento, independente de qualquer condição.

Para abrir o ambiente de linguagem de programação visual proposto, em um palco específico construído no Mundo dos Atores (neste caso foi criado o *PalcoComportamentoPessoas*) deve-se fazer a seleção de um ator e, com o botão direito do mouse, selecionar a opção "Editar Comportamentos", conforme mostra a Figura 24, este comando irá abrir a tela mostrada na Figura 25, na qual são feitas as atribuições dos comportamentos por meio dos encaixes das peças.

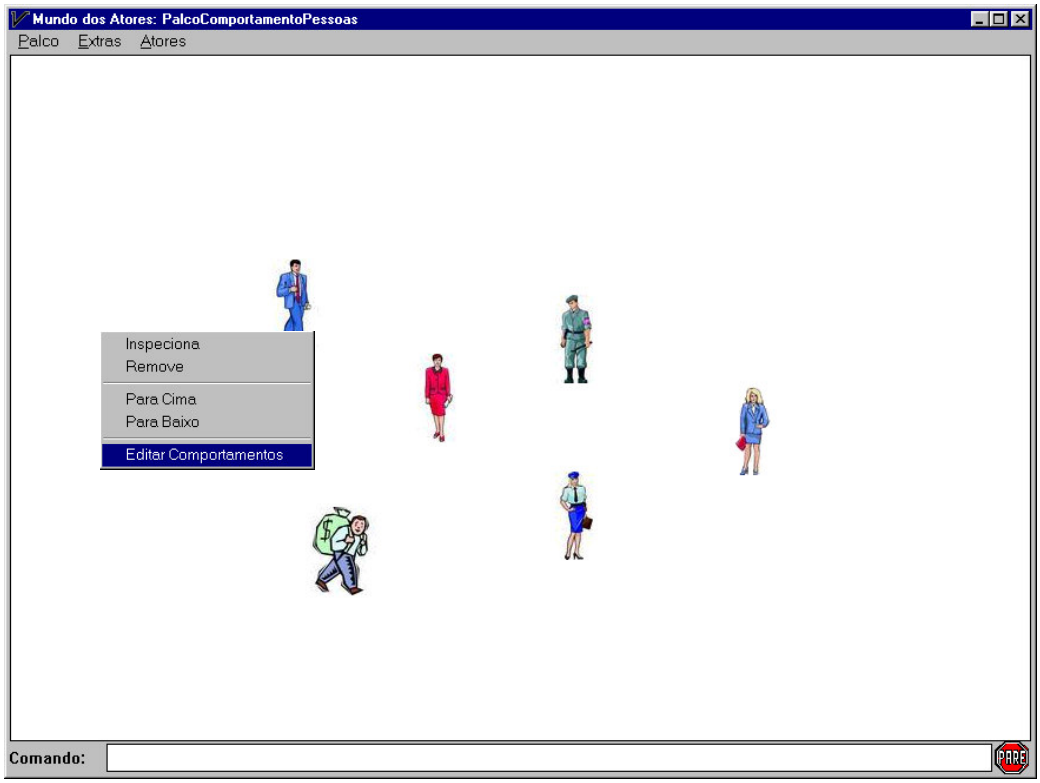

Figura 24: Seleção de um ator e da opção para abrir a tela de edição dos comportamentos.

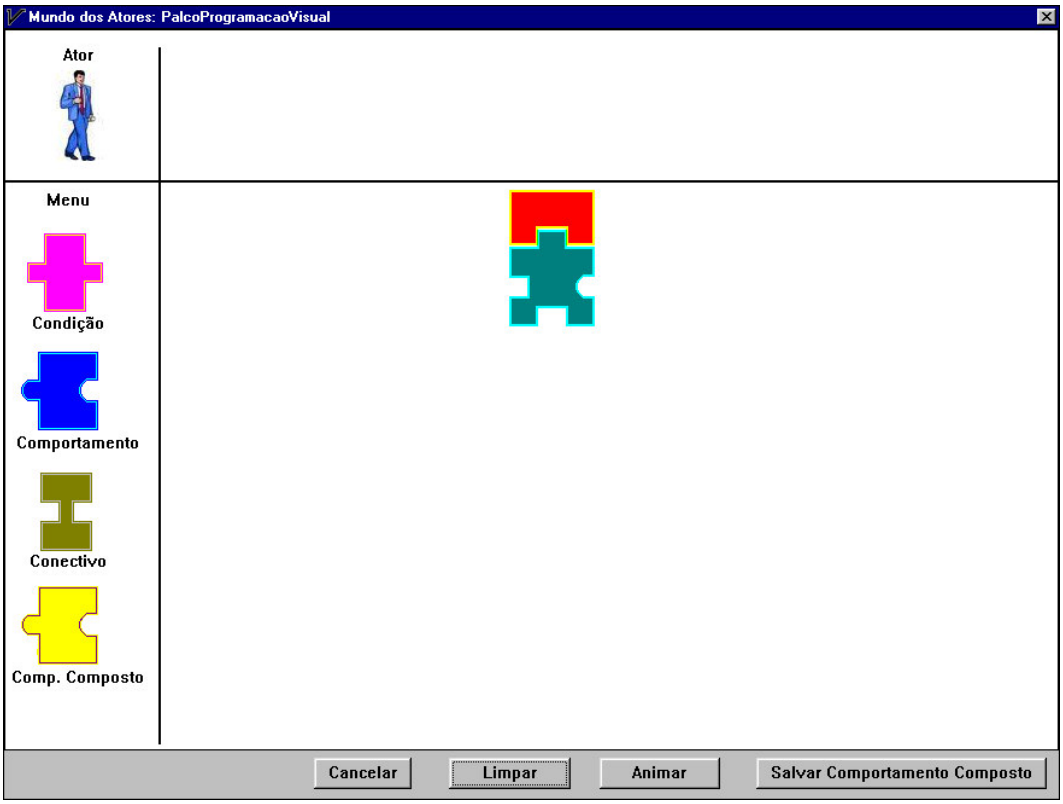

Figura 25: Tela inicial do ambiente proposto exibindo o ator selecionado e o menu com as peças principais.

Na tela inicial, na parte do palco onde serão colocadas as peças para a execução dos comportamentos, o sistema já apresenta duas peças como *default*, uma representando o ator como um ponto de início para o encaixe das peças e outra com função de ligação, a qual faz a ligação entre as condições e os comportamentos atribuídos.

No menu estão representadas as peças chaves, que também podem ser chamadas de matrizes, pois a partir delas, da seleção das mesmas, irão aparecer as peças relacionadas com as referidas opções. Por exemplo, se a peça do menu denominada "Condição", for selecionada, o sistema irá apresentar as opções das condições que foram pré-definidas no ambiente e que podem ser atribuídas aos comportamentos do ator. As Figuras 26, 27 e 28 demonstram como isso acontece.

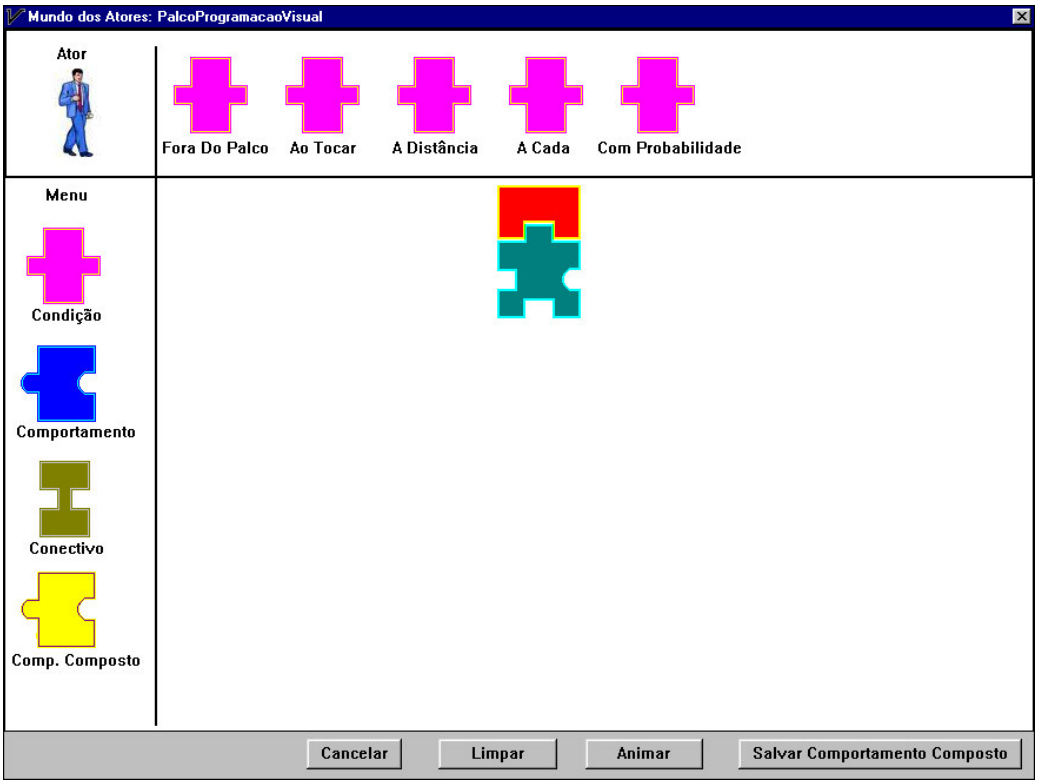

Figura 26: Apresentação das condições pré-definidas a partir da seleção da peça Condição presente no menu.

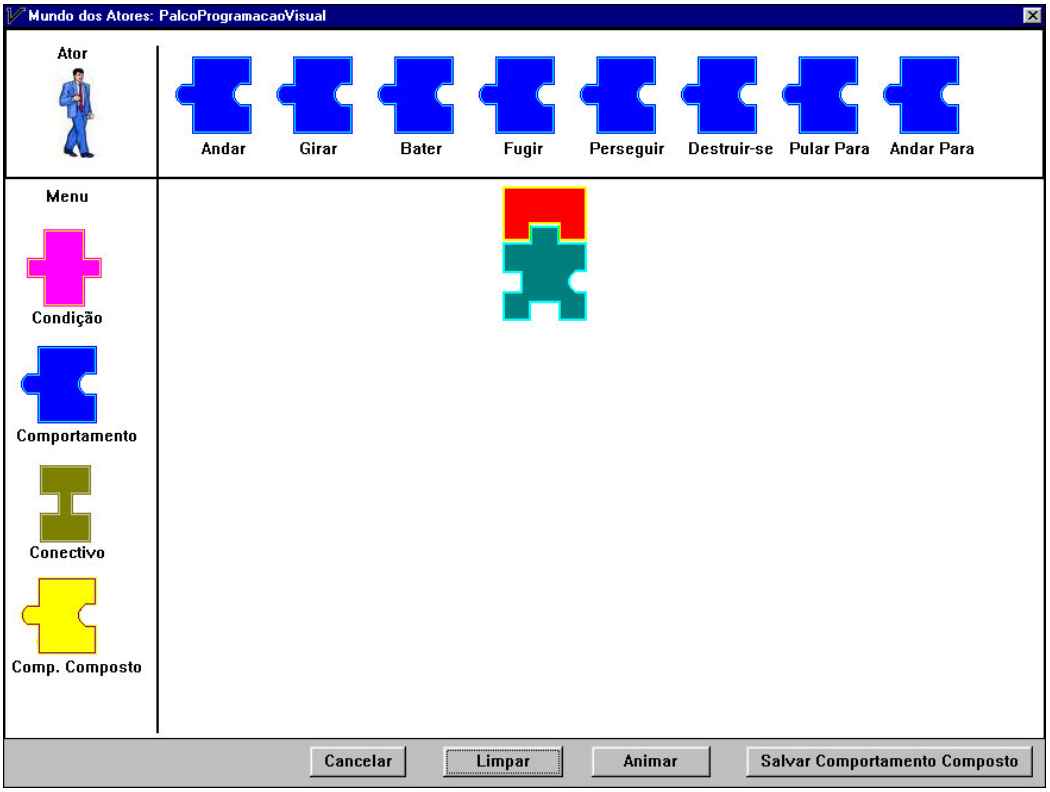

Figura 27: Apresentação dos comportamentos pré-definidos a partir da seleção da peça Comportamento presente no menu.

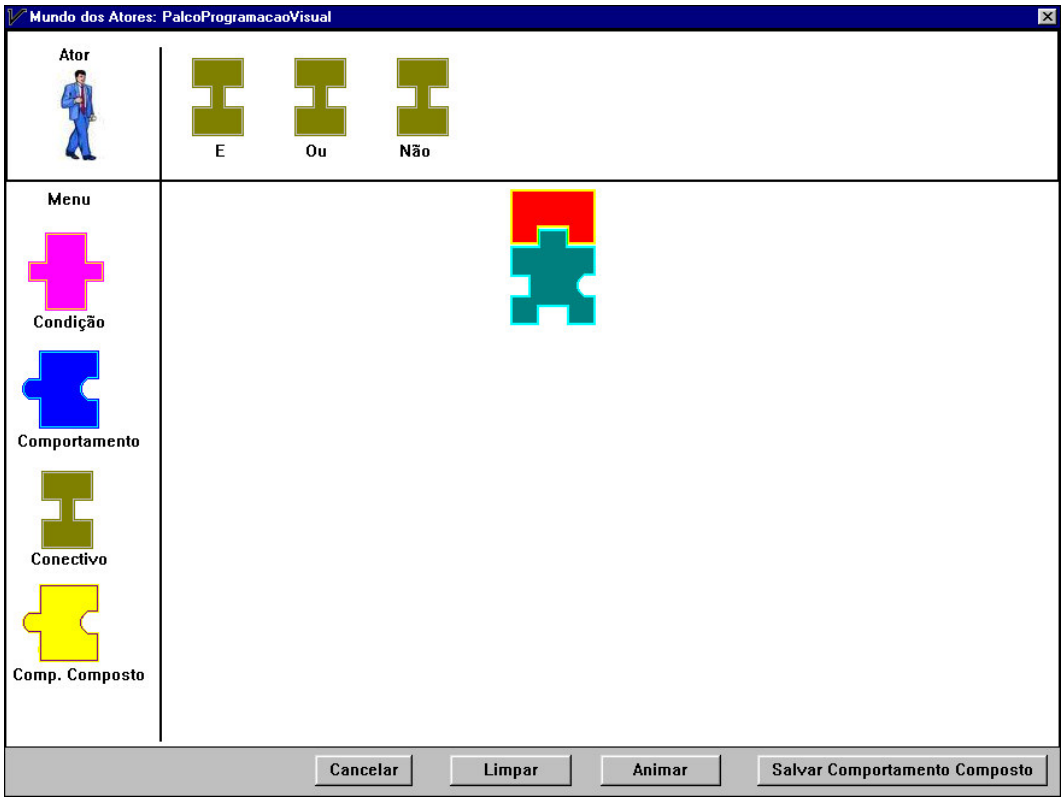

Figura 28: Apresentação dos conectivos pré-definidos a partir da seleção da peça Conectivo presente no menu.

Se já tivesse algum comportamento composto pré-definido, a mesma coisa aconteceria ao selecionar a peça "Comp. Composto" presente no menu, porém este não é pré-definido como as condições, comportamentos simples e conectivos, e sim, criado pelo programador durante o processo.

Com as condições, comportamentos e conectivos pré-definidos, o programador pode atribuir, ao ator selecionado, os comportamentos que o mesmo deverá executar e as condições necessárias, se for o caso. Para isso, basta selecionar a opção desejada, apresentada pelo sistema a partir da seleção de uma das peças do menu, e arrastar a peça com o mouse até o encaixe correto. Quando o encaixe for efetivado, o sistema apresentará uma janela com as opções de parâmetros a serem atribuídos para o funcionamento, tanto dos comportamentos como das condições (Figuras 29 e 30), isso fornece ao programador uma manipulação de valores.

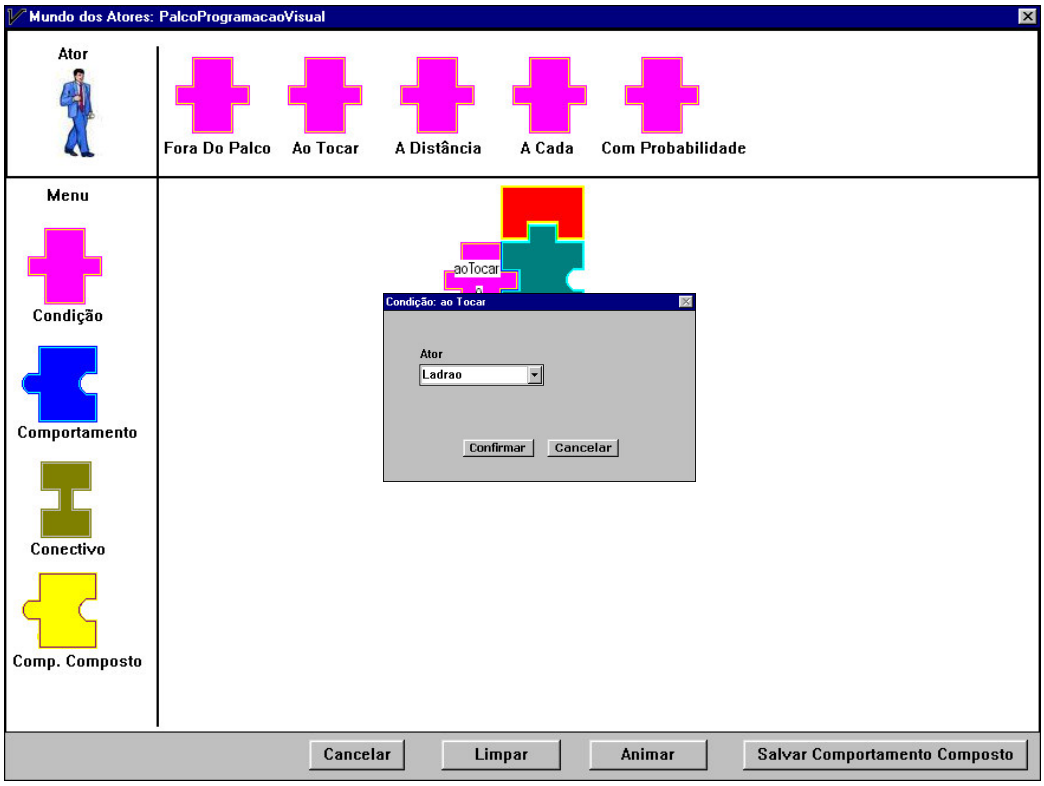

Figura 29: Apresentação da janela para a valoração dos atributos após o encaixe de uma peça, neste caso, de uma condição.

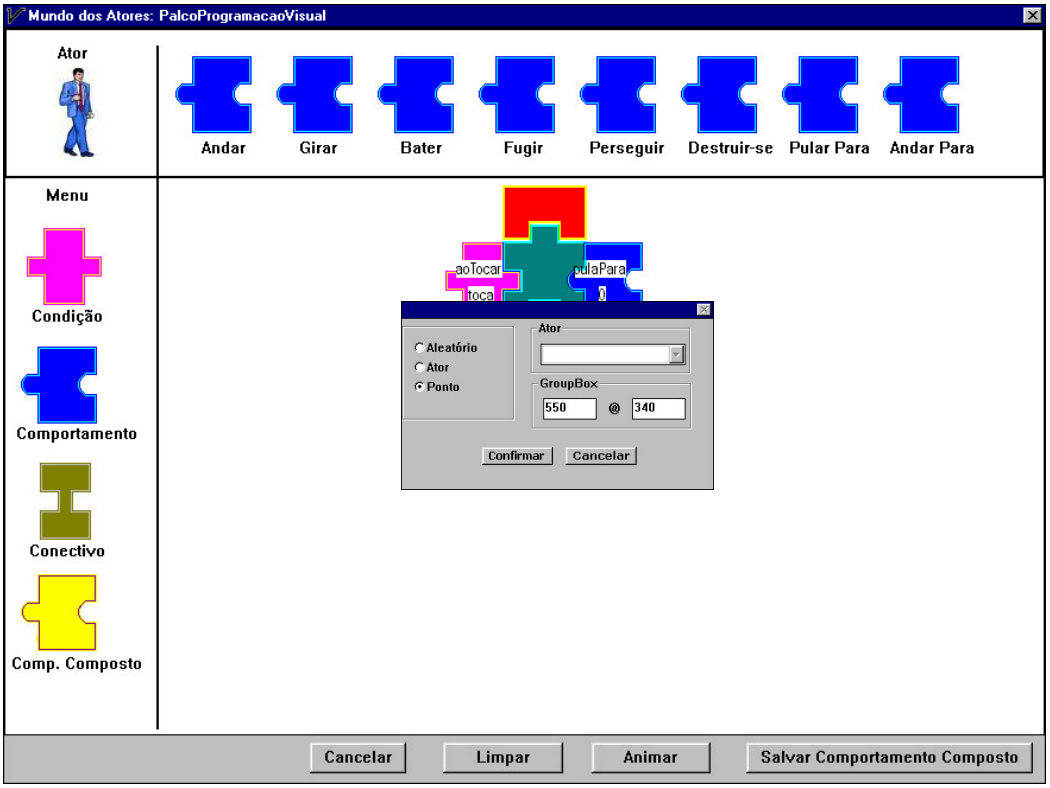

Figura 30: Apresentação da janela para a valoração dos atributos após o encaixe de uma peça, neste caso, de um comportamento.

Após a confirmação da valoração dos atributos e, conseqüentemente, do comportamento, mesmo que ainda tenha mais comportamentos a serem colocados ao lado do primeiro, o sistema já deixa em aberto a próxima peça de ligação, como mostra a Figura 31:

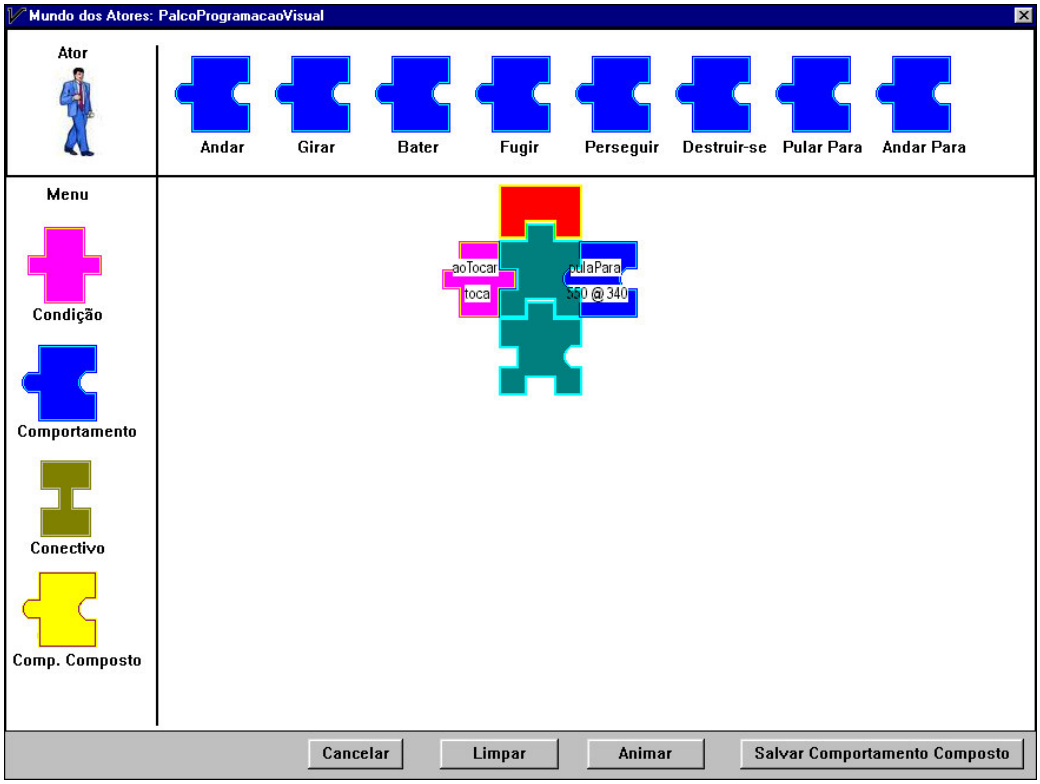

Figura 31: Apresentação da próxima peça de ligação em aberto.

Na figura acima, foi atribuído ao ator *HomemExec* (selecionado conforme consta na Figura 24) o comportamento *pularPara*(*550,340*), porém este comportamento somente será executado quando a condição *aoTocar(Ladrão)* for satisfeita. Ainda para o comportamento acima atribuído, poderia ser adicionada mais uma ou mais condições, neste caso bastaria selecionar um dos conectivos disponíveis e encaixar a outra condição, conforme mostra a Figura 32 a seguir:

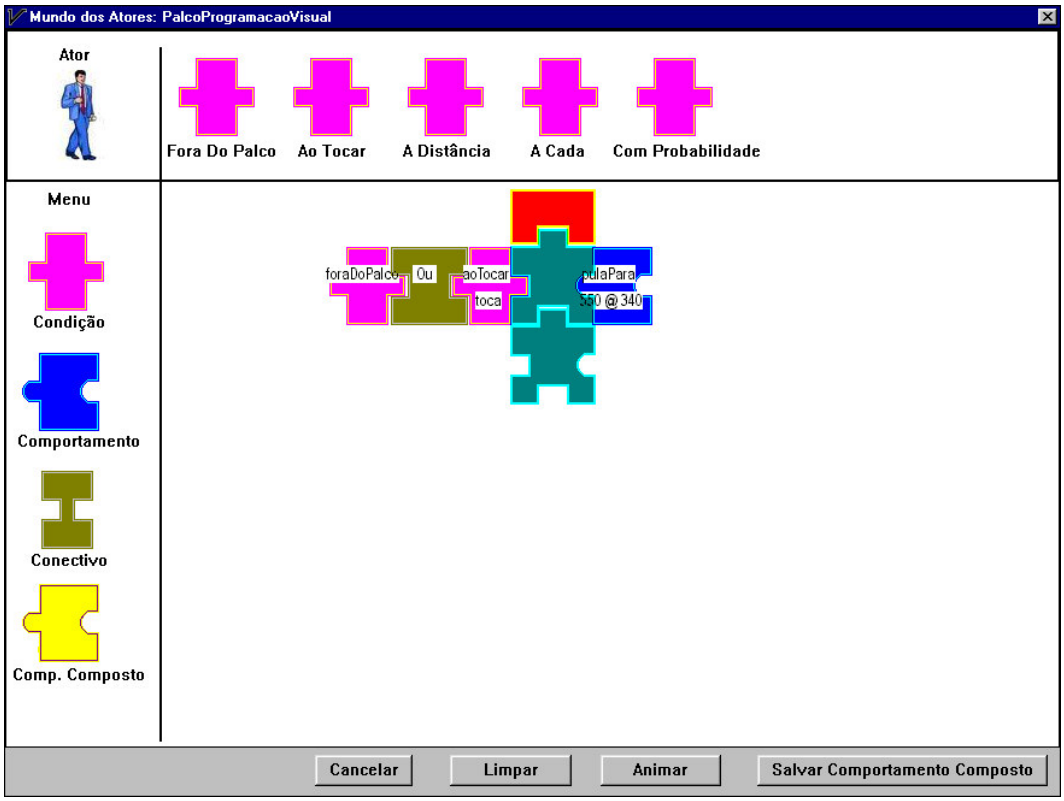

Figura 32: Apresentação de mais de uma condição para a execução de um comportamento.

Para os conectivos, não precisa nenhuma valoração de atributos, pois os mesmos já possuem o seu valor definido, que no caso é o "E", "Ou" ou "Não". Após a atribuição da outra condição, o comportamento pularPara(*550,340*) do ator selecionado somente será executado, neste caso, quando a condição *aoTocar(Ladrão)* ou a condição *foraDoPalco* for satisfeita.

Como exposto no item 4.3.4, os comportamentos compostos são gerados a partir da junção de dois ou mais comportamentos básicos ou pela junção de outros comportamentos compostos, sendo que os comportamentos compostos são criados a partir dos comportamentos básicos pré-definidos no ambiente. Neste caso, a formação dos comportamentos compostos se dá a partir das últimas peças encaixadas selecionando, após isso, o botão "Salvar Comportamento Composto". Por exemplo, se encaixar as peças referentes ao comportamento *pularPara(550,340)* e a condição *aTocar(Ladrão)* na primeira peça de ligação e encaixar as peças referentes aos comportamentos *andar(1)* e *girar(2)* e a condição *comProbabilidade(10)* na segunda

peça de ligação e, após isso, selecionar o botão “Salvar Comportamento Composto”, o comportamento composto será referente as últimas peças encaixadas, ou seja, as peças encaixadas na segunda peça de ligação. Quando o botão para salvar o comportamento composto é selecionado, deve-se escolher um nome para tal comportamento, conforme mostra a Figura 33, e, depois da confirmação do nome, o comportamento composto criado passa a fazer parte da coleção dos comportamentos compostos do ambiente.

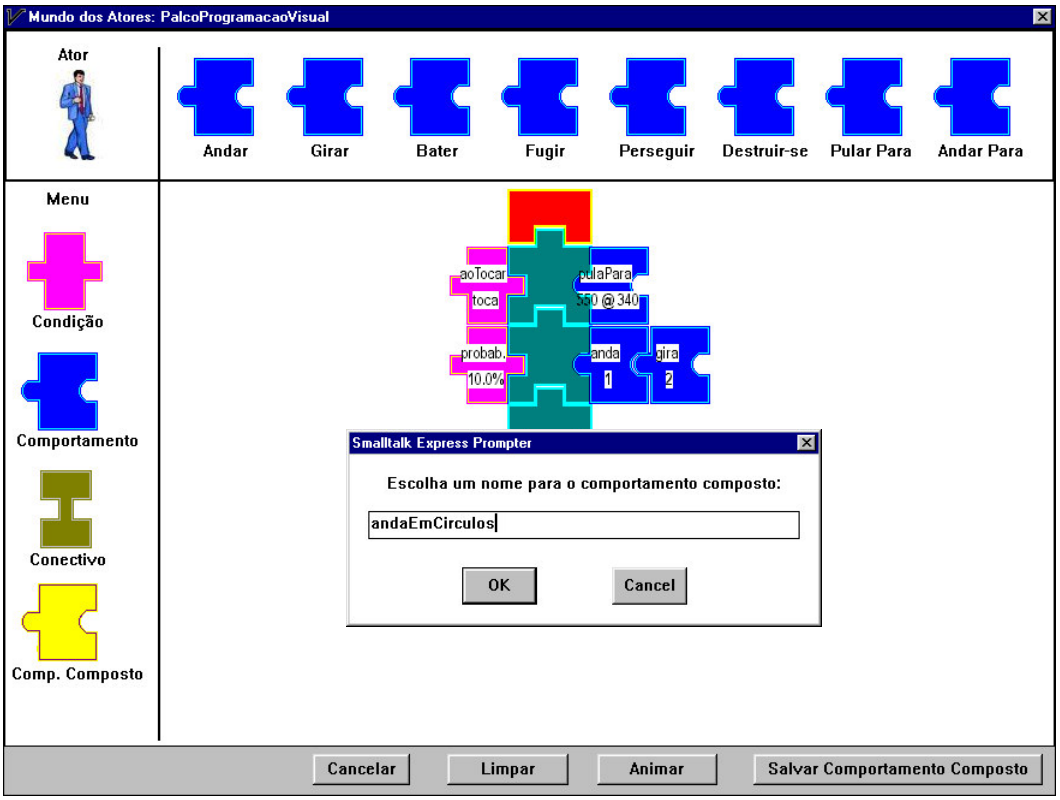

Figura 33: Criação de um comportamento composto.

Depois que um comportamento composto é criado, ele poderá ser usado nos demais atores do ambiente, sendo que ele estará presente como opção da peça “Comp. Composto” presente no menu (Figura 34).

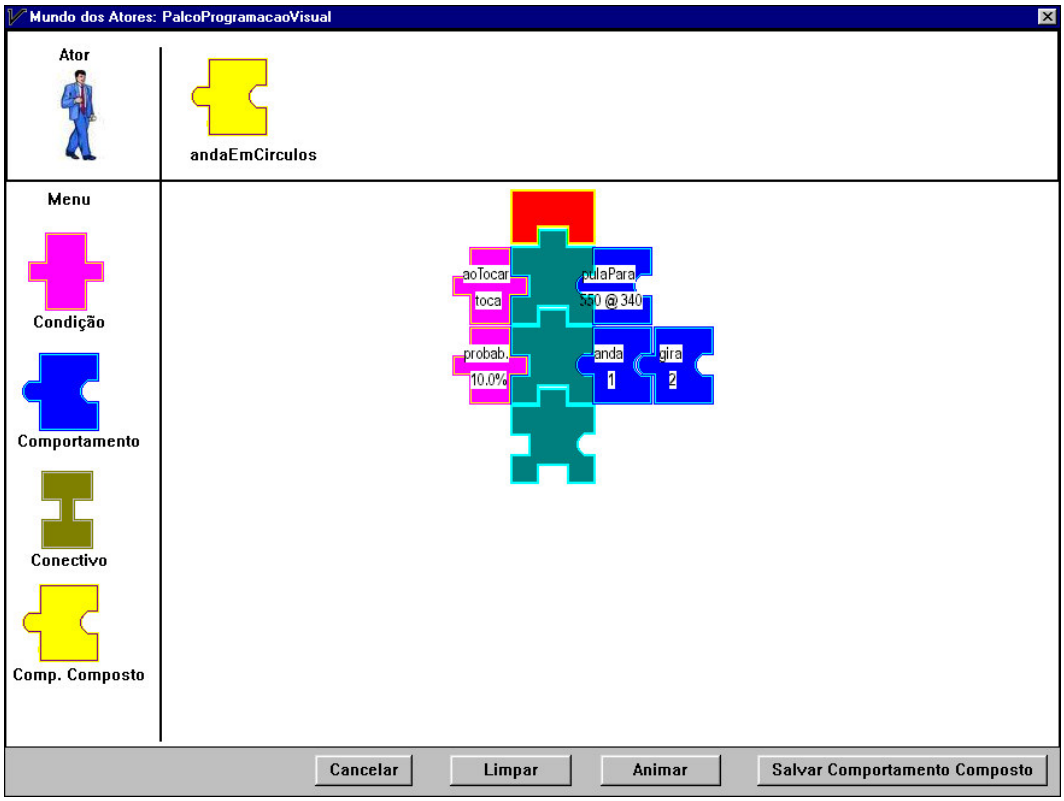

Figura 34: Apresentação do comportamento composto criado a partir da seleção da peça Comp. Composto presente no menu.

A partir do encaixe das peças o programador não precisa se preocupar com a carga cognitiva para o funcionamento do papel dos atores no ambiente, pois não é necessário entender como o código deve ser formado e sim, apenas atribuir alguns valores para que o mesmo seja formado. Para isso, conforme exposto na Figura 23, seguindo a idéia de que a execução dos comportamentos acontece da esquerda para a direita e de cima para baixo, o sistema faz a verificação da existência de peças do lado esquerdo da peça de ligação e logo após, do lado direito, gravando o conteúdo dos parâmetros atribuídos em cada peça, passando para baixo apenas quando nenhuma peça na linha em questão for encontrada, conforme mostrado abaixo:

```
cont := 0.
    [cont < atoresUteis size] whileTrue:[
        i := 0.
        condicaoExiste := false.
        [atoresEsq size > i] whileTrue:[


    "verificação das pecas do lado esquerdo da peça de ligação"

            iAtual := i.
            atoresEsq do:[:ator|
                (ator posicao = gridPosY1) ifTrue:[
                    condicaoExiste := true.
                    codigo addLast: ator texto.
                    gridPosY1 := gridPosY1 - (51 @ 0).
                    i := i + 1.
                    cont := cont + 1.
                ].
            ].

            (i = iAtual) ifTrue:[
                i := atoresEsq size.
            ].
        ].
        condicaoExiste ifTrue:[
            codigo addLast: 'faca: ['.
        ] ifFalse:[
            codigo addLast: 'self sempreFaca: ['.
        ].



    "verificação das pecas do lado direito da peça de ligação"

        j := 0.
        [atoresDir size > j] whileTrue:[
            jAtual := j.
            atoresDir do:[:ator|
                (ator posicao = gridPosY2) ifTrue:[
                   codigo addLast: ator texto.
                    gridPosY2 := gridPosY2 + (51 @ 0).
                    j := j + 1.
                    cont := cont + 1.
                ].
            ].
            (j = jAtual) ifTrue:[
                j := atoresDir size.
            ].
        ].
        gridPosY1 x: 402.
        gridPosY2 x: 504.
        gridPosY1 := gridPosY1 + (0 @ 66).
        gridPosY2 := gridPosY2 + (0 @ 66).
        codigo addLast: '].'.
    ].

    ^codigo asString
```

O programador pode encaixar quantas peças forem necessárias para a determinação do papel do ator no ambiente. Depois que o programador encaixar todas as peças, ou seja, depois que for determinado como o ator deverá agir no ambiente, ou em qualquer momento, o programador pode clicar no botão “Animar”, o qual executará os códigos formados a partir do encaixe das peças. Esta opção fornece um retorno ao programador do papel que foi ou que está sendo atribuído ao ator selecionado e, também, dos outros atores pertencentes ao ambiente.

Com a apresentação do ambiente feita acima, no capítulo a seguir serão feitas algumas demonstrações de atribuição de comportamentos para os atores pertencentes ao palco exposto na Figura 24, utilizando o ambiente visual de programação proposto neste trabalho.

Capítulo 6 – Validação da Definição Proposta por Meio de Exemplos

Neste trabalho não foi feita nenhuma avaliação de jogos do tipo *VideoGame* como foi feito por Schütz (2003) e sim a utilização dos comportamentos e condições identificados por ele. Neste capítulo são feitas algumas demonstrações com os atores presentes no palco exposto na Figura 24, a partir das condições e comportamentos apresentados no presente trabalho, a fim de demonstrar que a sobrecarga cognitiva pode ser diminuída em função de uma interface visual para a composição dos comportamentos a serem executados, conforme exposto em Schütz (2003), uma interface "*na qual as composições de comportamentos pudessem ser feitas apenas pelo arrastar e soltar de peças*".

Pegando como exemplo os atores contidos e denominados no palco *PalcoComportamentoPessoas*, conforme Figura 35, os comportamentos descritos em Schütz (2003) podem ser atribuídos para esses atores.

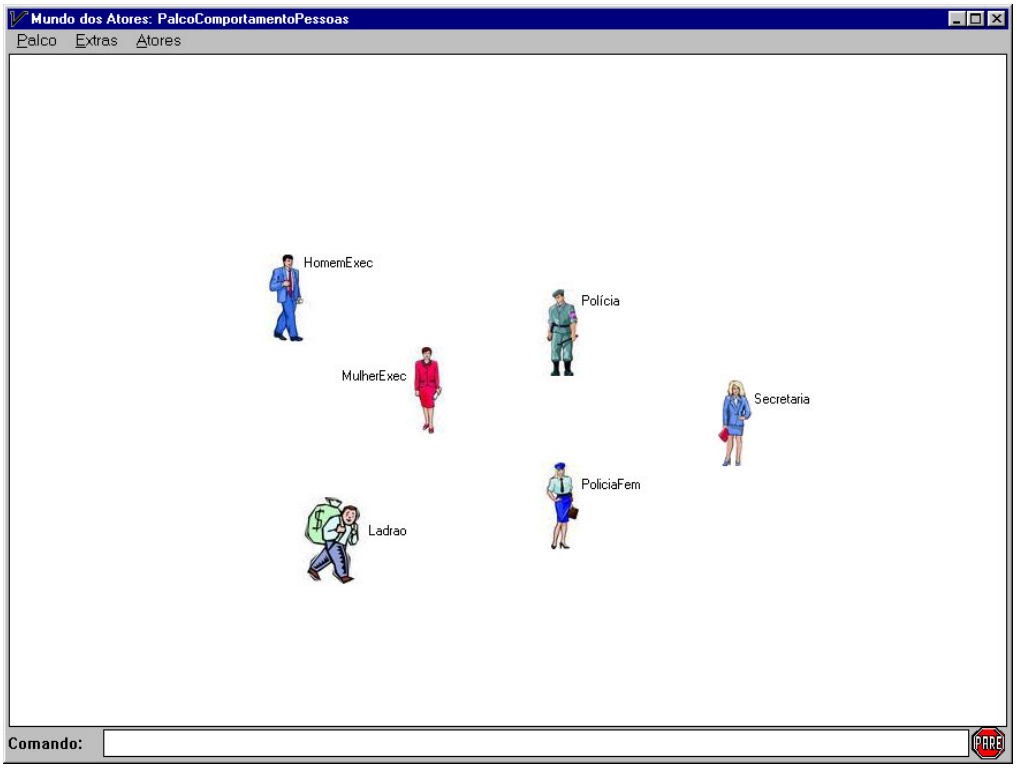

Figura 35: Identificação dos atores presentes no palco *PalcoComportamentoPessoas* usado como exemplo.

O palco utilizado como exemplo pelo ambiente proposto neste trabalho, conforme exposto na Figura 35, possui seis atores que se movem, cada um executando o seu papel. Para exemplificar a atribuição dos comportamentos com o encaixe das peças apresentadas no ambiente de programação visual proposto, tem-se o seguinte caso:

O ator *Policia* executa o comportamento (papel) de sempre perseguir o ator *Ladrao* com uma determinada velocidade, e o ator *Ladrao* executa o comportamento de sempre fugir do ator *Policia*, enquanto ambos estiverem na área visível do palco. Se o ator *Ladrao* ultrapassar o limite da área visível do palco, ele retorna ao palco em uma posição (x,y) especificada, por exemplo (280,400), continuando sua fuga a partir deste ponto, em uma velocidade diferente da velocidade do ator *Policia*. O ator *Policia*, por sua vez, quando ultrapassa o limite da área visível do palco, retorna em uma outra posição (x,y) especificada, por exemplo (480,240), recomeçando, a partir daí, sua perseguição ao ator *Ladrao*. Quando o ator *Policia* conseguir alcançar o ator *Ladrao*, este é destruído.

O ator *PoliciaFem* fica o tempo todo caminhando lentamente pelo ambiente, com uma velocidade especificada, circulando por todas as direções. Quando este ator se encontra com o ator *Ladrão*, este muda sua direção, girando para o lado oposto da direção atual e continua sua fuga, neste momento, o ator *PoliciaFem* começa sua perseguição, em uma certa velocidade, ao ator *Ladrão*.

O ator *Secretaria* fica no seu lugar de origem até que o ator *Ladrao* passe a uma determinada distância do mesmo. Neste momento, o ator *Secretaria* caminha rapidamente em direção ao limite da área visível do palco, e quando ultrapassa, retorna para um ponto aleatório.

Os atores *HomemExec* e *MulheExec* circulam pelo palco. Quando se encontram, o ator *HomemExec* inverte sua direção 90° em relação a sua direção atual e os dois continuam caminhando. Quando ambos ultrapassam o limite da área visível do palco, retornam para suas posições iniciais, isso pode acontecer com a *MulherExec*, também, quando ela se encontra com o ator *HomemExec*. Com uma probabilidade de 10%, o ator *HomemExec* caminha em direção ao ator *Secretaria*.

A partir do exposto acima, o papel (comportamento) do ator *Policia* no ambiente seria:

```
ATOR Policia {
      Persegue (Ladrão, 6).
      foraDoPalco() ⇒ pularPara(480,240).
    }
```

Na definição da programação visual proposta, os comportamentos acima expostos podem ser atribuídos ao ator *Policia* a partir do encaixe das peças relacionadas a tais comportamentos e pela valoração dos atributos dos mesmos, os quais irão diferenciar a forma de como o ator irá desempenhar os comportamentos a ele designados, como mostrado na Figura 36.

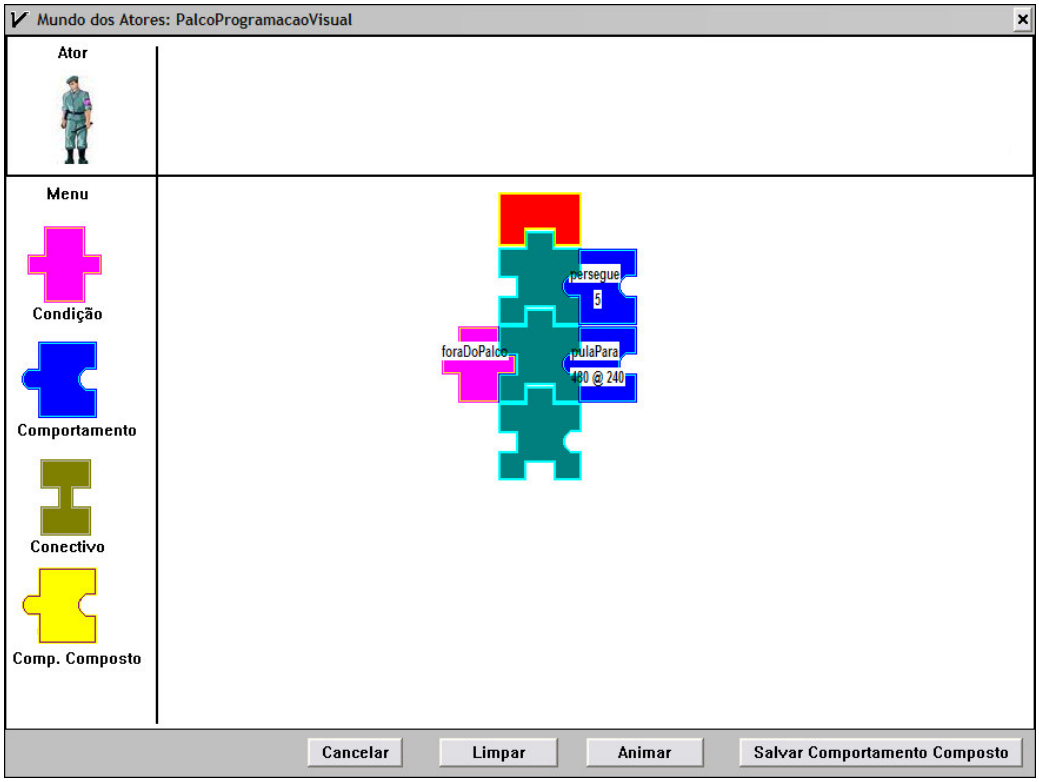

Figura 36: Composição do papel definido para o ator *Policia*.

Para o ator *Ladrao*, conforme colocado anteriormente, o comportamento do ator *Ladrao* no ambiente seria:

```
ATOR Ladrao {
      Foge(Policia,6).
      foraDoPalco() ⇒ pularPara(280,400).
      aoTocar (PocíciaFem) ⇒ gira(180),anda(4).
      aoTocar (Pocicia) ⇒ destroi().
    }
```

A valoração, tanto para os comportamentos *fugir*, *perseguir* e *andar*, se dá pela velocidade com que o ator deve se locomover pelo palco e também, no caso de *fugir* e *perseguir*, pela seleção do ator para quem deve ser direcionado.

Na definição da programação visual proposta, a composição dos comportamentos acima expostos ficariam conforme mostra a Figura 37:

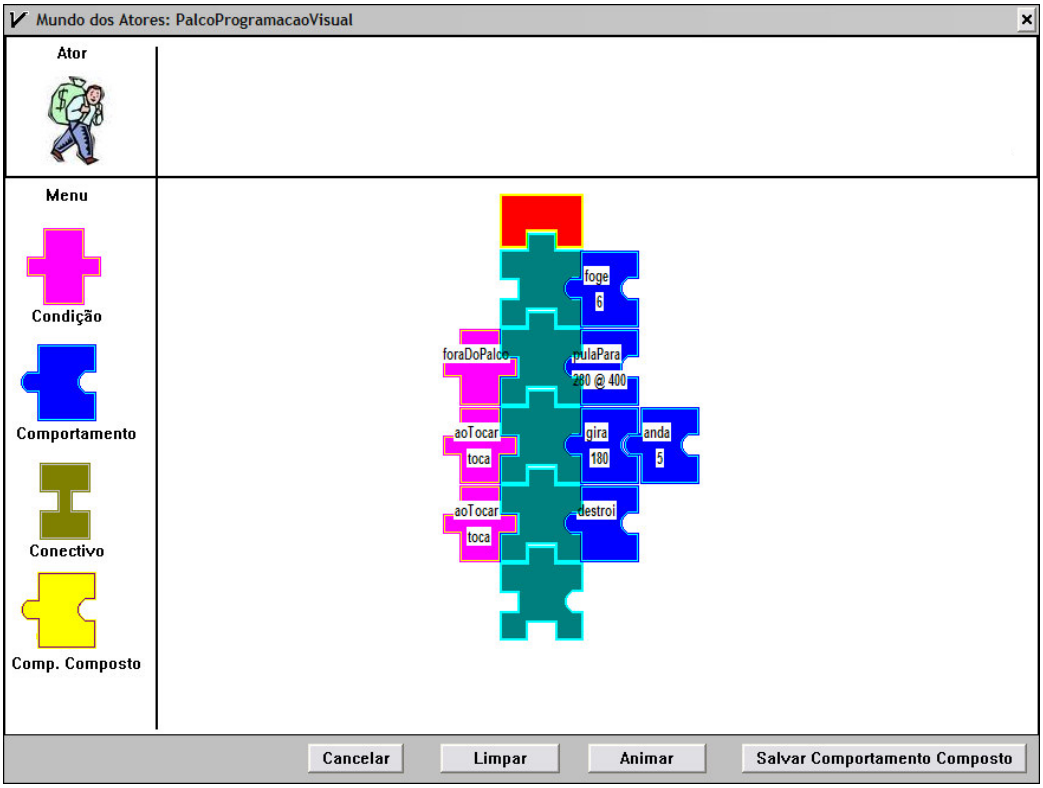

Figura 37: Composição do papel definido para o ator *Ladrão*.

A partir dos comportamentos atribuídos aos atores presentes no palco, cada um irá desempenhar seu papel diferenciado e independente, podendo ocorrer a interação entre os mesmos.

No caso do ator *PoliciaFem*, a descrição do comportamento para este ator seria:

```
COMP andaEmCirculos ← {
    anda(1); gira(2).
}


ATOR PoliciaFem {
    anda(1).
    andaEmCirculos().
    aoTocar(Ladrao) ⇒ Persegue(Ladrao, 5).
}
```

Na definição da programação visual proposta, a composição dos comportamentos acima expostos para o ator *PoliciaFem*, ficariam conforme mostra a Figura 38:

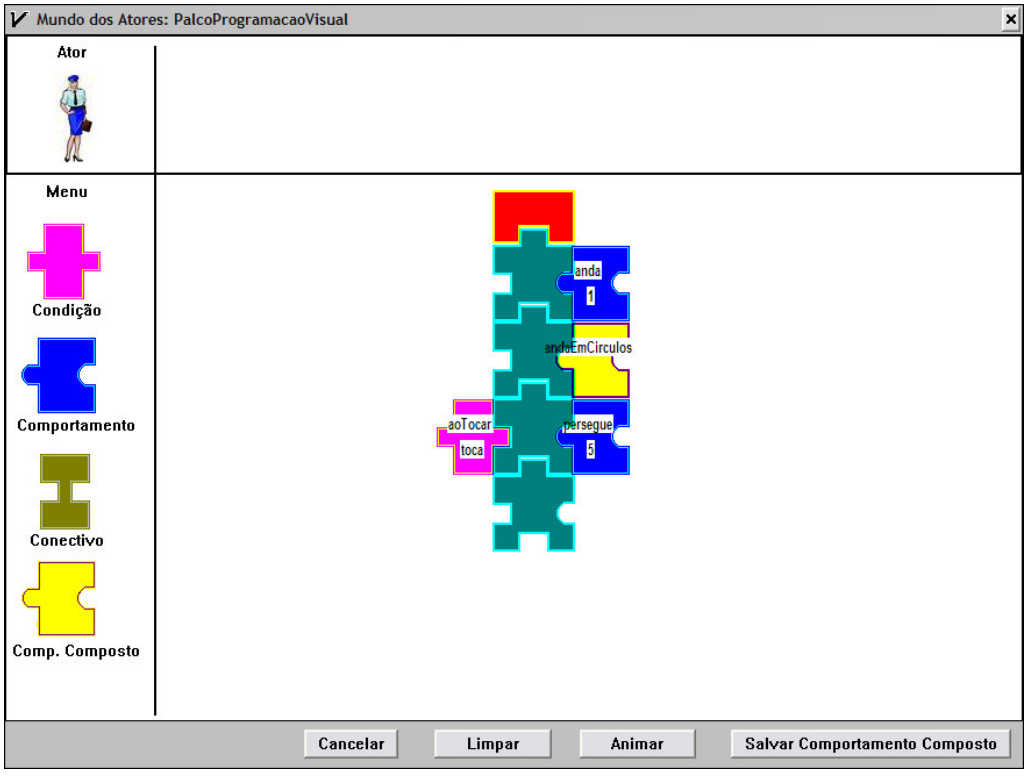

Figura 38: Composição do papel definido para o ator *PoliciaFem*.

A descrição do comportamento para o ator *Secretaria*, no caso descrito acima, seria:

```
ATOR Secretaria {
       aDistancia(3,Ladrao) ⇒ anda(2).
       foraDoPalco() ⇒ pularPara(random).
    }
```

Na definição da programação visual proposta a composição dos comportamentos acima expostos para o ator *Secretaria*, ficariam conforme mostra a Figura 39:

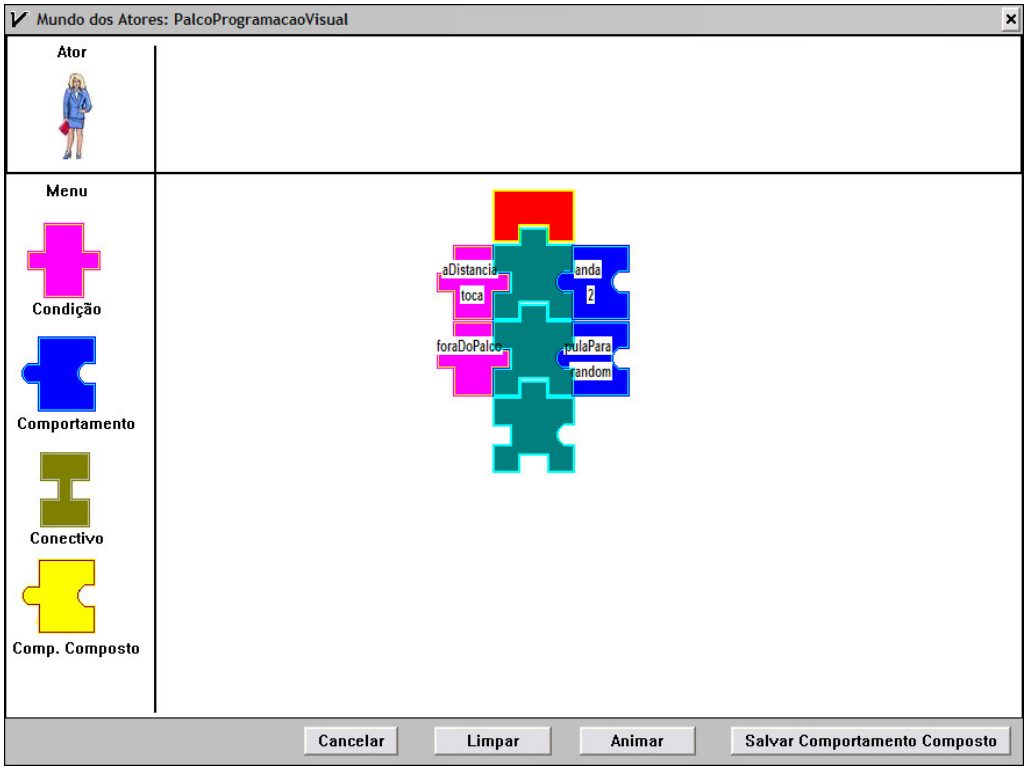

Figura 39: Composição do papel definido para o ator *Secretaria*.

No caso descrito acima, a descrição do comportamento para o ator *HomemExec*, seria:

```
COMP andaEmCirculos ← {
    anda(1); gira(2).
}


ATOR HomemExec {
    andaEmCirculos().
    foraDoPalco() ⇒ pularPara(240,210).
    aoTocar(MulherExec) ⇒ gira(90).
    comProbabilidade(10) ⇒ andaPara(Secretaria).
}
```

Na definição da programação visual proposta, a composição dos comportamentos acima expostos para o ator *HomemExec*, ficariam conforme mostra a Figura 40:

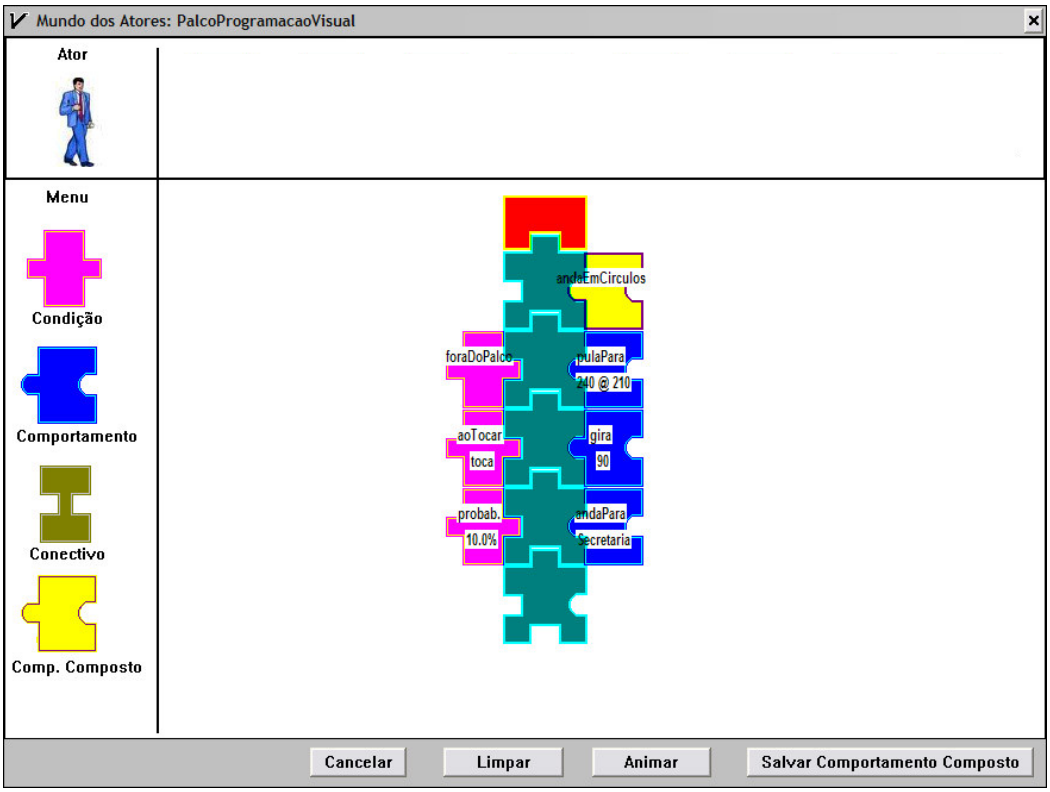

Figura 40: Composição do papel definido para o ator *HomemExec*.

E por fim, para o caso dos comportamentos do ator *MulherExec*, deve-se fazer uso de conectivo, pois este ator possui duas limitações para a execução de um mesmo comportamento. Com isso, a descrição do comportamento para o ator *MulherExec*, seria:

```
COMP andaEmCirculos ← {
    anda(1); gira(2).
}
```

```
ATOR MulherExec {
    andaEmCirculos().
    aCada(2) ⇒ anda(1)
    foraDoPalco() OU aoTocar(HomemExec) ⇒
            pularPara (360,290).
}
```

Na definição da programação visual proposta a composição dos comportamentos definidos para o ator *MulherExec*, seria como mostrado na Figura 41 a seguir:

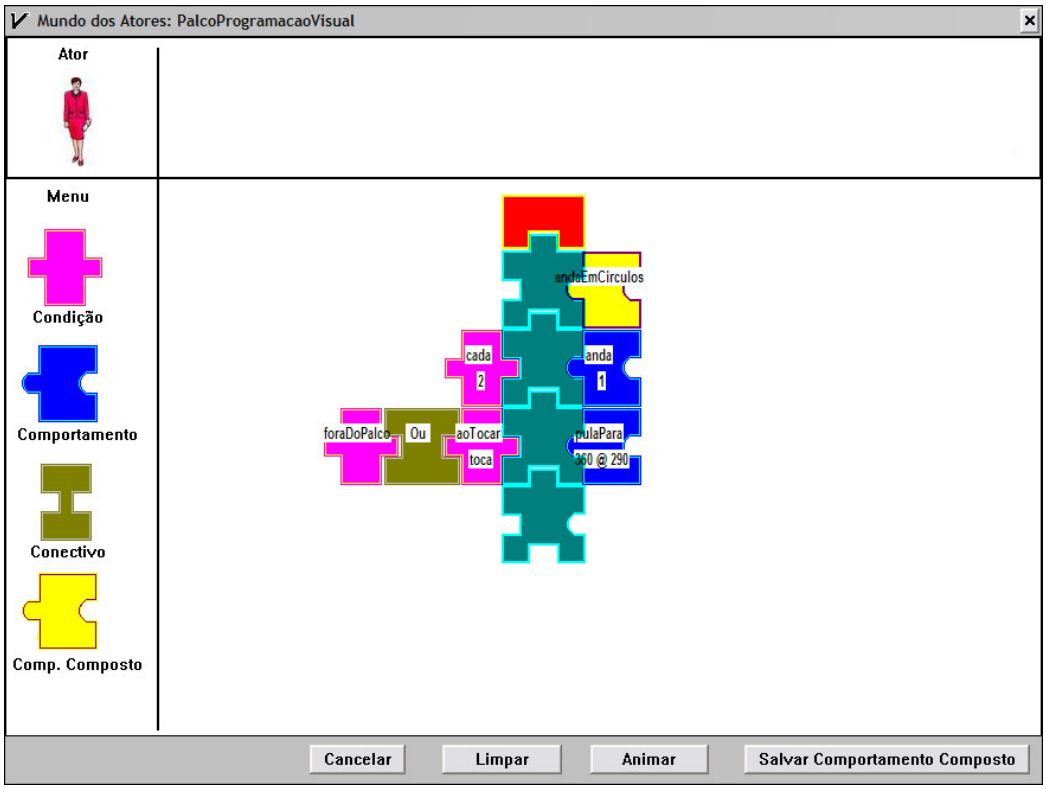

Figura 41: Composição do papel definido para o ator *MulherExec*.

A execução dos comportamentos é mostrada após a seleção do botão “Animar”, sendo que o sistema retorna ao palco onde se encontram os atores para os mesmos executarem seus papéis.

Capítulo 7 – Conclusão e Trabalhos Futuros

Neste trabalho foram apresentados alguns aspectos referentes à linguagens de programação visual, mostrando suas características e alguns exemplos utilizados com tais tipos de linguagem. Foi exibido, de forma sucinta, a ferramenta Mundo dos Atores, sendo que a definição aqui proposta foi construída sobre essa ferramenta. Também, como o ambiente de programação proposto é um ambiente baseado na atribuição de comportamentos, foi feito uma explanação do que é comportamento e os tipos de comportamento utilizados neste trabalho. Com isso, buscou-se uma proposta de ambiente de programação visual orientada por comportamentos com a demonstração de alguns exemplos.

As linguagens de programação visual possuem alguns aspectos positivos e negativos. Umas das vantagens deste tipo de linguagem seria o acesso espacial e gráfico às informações, enquanto que no texto, a leitura e entendimento são, na maioria das vezes, de natureza seqüencial. Outra vantagem seria a multidimensionalidade das expressões visuais, pois, como acredita-se que o ser humano possui a mente visualmente orientada, a figura faz com que a linguagem se torne mais rica, pois possui propriedades visuais como forma, cor, tamanho, entre outros. Além disso, as figuras permitem com que os objetos sejam representados tanto de forma concreta como abstrata, possibilitando, assim, a criação de diagramas ou ícones exclusivos para o domínio em questão.

São identificadas também, algumas restrições impostas por este tipo de linguagem, tais como o espaço utilizado pelos programas desenvolvidos em tais linguagens ser maior do que o ocupado pelas linguagens textuais. Também pode ser considerado como desvantagem o fato de que, muitas vezes, esse tipo de linguagem é desenvolvido para a utilização em domínios específicos e não para soluções de problemas de necessidades mais gerais, não podendo, assim, ser reaproveitado em outros trabalhos. Além disso, este tipo de linguagem requer alguns recursos gráficos para o seu funcionamento.

Tendo como referência as definições encontradas no trabalho feito por Schütz (2003), no ambiente proposto disponibilizou-se um conjunto de comportamentos e condições pré-definidos, sendo que a partir destes, novos comportamentos compostos

podem ser criados, gerando um número mais expressivo de comportamentos a serem utilizados. Estes novos comportamentos podem ser utilizados para os demais atores presentes no ambiente. As condições apresentadas fazem com que se tenha comportamentos condicionados, ou limitados, ou seja, comportamentos que ficam na dependência da ocorrência de algo para que os mesmos possam ser executados, sendo que se a condição não for declarada, indica que o comportamento sempre terá que ser desempenhado pelo ator.

No processo de definição da interface, houve a preocupação em apresentar um ambiente considerado fácil de ser entendido e manipulado. Com isso, a interface teve como base o conceito de peças do tipo quebra-cabeça, ou seja, todos os objetos do ambiente são representados por peças do tipo quebra-cabeça. Deste modo, acredita-se estar proporcionando uma ferramenta simples, com uma metáfora conhecida pela maioria dos usuários, independente do nível dos mesmos, conhecedores de programação ou não, pois com este ambiente, o programador pode definir os comportamentos a serem executados pelos objetos, a partir do encaixe das peças e atribuição de alguns valores, contribuindo, desta maneira, para a diminuição da sobrecarga cognitiva.

O ambiente de programação visual apresentado neste trabalho, é uma proposta de definição e dispõe de algumas limitações, dentre elas, a pouca variedade de comportamentos básicos pré-definidos, que o impedem de ser utilizado em aplicações de nível mais complexo, como por exemplo, em jogos. Além de expandir as opções de comportamentos básicos, até mesmo para uma maior possibilidade de criação de novos comportamentos compostos, o que contribuiria para o ambiente se tornar mais propício para o uso em aplicações mais complexas, a interface em si mereceria um tratamento futuro.

Esta definição teve sua implementação desenvolvida em Smalltalk e construída sobre o Mundo dos Atores. A definição de uma interface visual no Mundo dos Atores se deu devido ao fato de que a mesma tem se mostrado um recurso adequado para o processo de aprendizado em programação, podendo, esta ferramenta, com a interface gráfica, obter ainda mais sucesso neste processo, mesmo que, para a interação com a mesma, o programador deva possuir algum tipo e conhecimento computacional.

7.1. Trabalhos Futuros

De acordo com as limitações impostas neste trabalho, pode-se ter como propostas para trabalhos futuros os itens a seguir:

- Estudo, identificação e implementação das interferências que podem ocorrer entre os comportamentos atribuídos aos atores que se encontram no ambiente, pois, dependendo da combinação entre os comportamentos, podem ocorrer algumas interferências que influenciam no desempenho dos mesmos, podendo, a partir delas, ocorrer a criação de um comportamento diferente, a anulação de um dos comportamentos ou até mesmo a anulação de ambos.

- Definição e implementação de uma maior quantidade de comportamentos básicos para uma criação mais expressiva de novos comportamentos compostos e também para a atribuição de um número maior de comportamentos aos atores, pois a partir dos comportamentos pré-definidos e disponibilizados na definição proposta, não se consegue construir uma aplicação com a execução de comportamentos muito diferenciados e complexos.

Referências Bibliográficas

ABDALA, Daniel Duarte; WANGENHEIN, Aldo Von. Conhecendo o Smalltalk. 1ª edição. Florianópolis: Visual Books, 2002.

ANDRADE, Adja Ferreira de; HOFFMAN, Augusto Bohner; WAZLAWICK, Raul Sidnei. **Aprendizagem Cooperativa em Mundos Virtuais**. *In:* Taller Internacional de Software Educativo - TISE 98. 1998.

AGUIRRE, Fabrício Barcellos. Implementação de Sensores no Mundo dos Atores. Trabalho de conclusão do Curso de Ciências da Computação da Universidade Federal de Santa Catarina. 2003.

BOSHERNITSAN, Marat; DOWNES, Michael. **Visual Programming Languages: A Survey**. CS 263 Final Project. Computer Science Division. University of California, Berkeley, 1997. Disponível em: http://www.cs.berkeley.edu/~maratb/cs263/paper.pdf. Acesso em: 04 de junho de 2003.

BURNETT, Margaret M. & BAKER, Marla J. A Classification System for Visual Programming Languages. Department of Computer Science. Oregon State University, Corvallis. 1994.

BURNETT, Margaret M. **Visual Programming**. *In*: Encyclopedia of Electrical and Electronics Engineering (John G. Webster, ed.), John Wiley & Sons Inc., New York, (to appear approx. 1999). Oregon State University.

BURNETT, Margaret M**. Software Engineering for Visual Programming Languages**. Handbook of Software Engineering and Knowledge Engineering. Vol. 2. © World Scientific Publishing Company. Department of Computer Science. Oregon State University, 2001. Disponível em: http://citeseer.nj.nec.com/burnett01software.html. Acesso em: 04 de setembro de 2003.

BURNETT, Margaret; ATWOOD, John; DJANG, Rebecca Walpole; GOTTFRIED, Herkimer; REICHWEIN, James and YANG, Sherry**. Forms/3: a first-**

order visual language to explore the boundaries of the spreadsheet paradigm. Journal Functional Programming, 155-206, Volume 11, Number 2. March 2001.

CANNING, James; PELLAND, Dave; SLIGER, Sheron. **Visual programming in a X Windows workstation environment.** Symposium on Small Systems. Toronto, Ontario, Canada, 1991. p: 4 – 10. Disponível em: http://portal.acm.org/citation.cfm?id=111049&coll=Portal&dl=GUIDE&CFID=13828769&CFTOKEN=73866742. Acesso em: 06 de novembro de 2003.

CANNING, James; PELLAND, Dave; SLIGER, Sheron. **Eyes: A Visual Programming Environment.** Proceedings of the 19th annual conference on Computer Science. San Antonio, Texas, United States, 1999. p: 639 – 640. Disponível em: http://portal.acm.org/citation.cfm?id=328791&coll=Portal&dl=GUIDE&CFID=13828769&CFTOKEN=73866742. Acesso em: 06 de novembro de 2003.

CYBIS, Walter de Abreu. **Engenharia de Usabilidade: Uma Abordagem Ergonômica**. Laboratório de Utilizabilidade de Informática. Universidade Federal de Santa Catarina, UFSC, 2003. Disponível em. http://www.labiutil.inf.ufsc.br/apostila.htm. Acesso em: 22 de outubro de 2003.

EVANGELISTA, Silvio Roberto Medeiros. ProVisual - Um Modelo para Programação Visual de Matrizes. Tese de Doutorado em Engenharia Elétrica. Universidade Estadual de Campinas, UNICAMP. Campinas, SP, 2002.

FERRUZZI, Elaine Cristina. **Considerações sobre s Linguagem de Programação Logo.** Seminário Apresentado no GEIAAM – Grupo de Estudos de Inteligência Artificial Aplicada à Matemática – UFSC. Setembro, 2001.

FOLDOC – Free On-Line Dictionary of Computing. **Visual programming environment,** 1995. Disponível em: http://wombat.doc.ic.ac.uk/foldoc/foldoc.cgi?visual+programming+environment. Acesso em 17 de abril de 2004.

GOLIN, Eric J, REISS, Steven P. **The Specification of Visual Language Syntax**. Departament os Computer Science, Brown University, Providence, RI 02912. IEEE Workshop on , 4-6 Oct., pg. 105 a 110, 1989.

GOULD, Laura; FINZER, William F. **Rehearsal World: Programming by Rehearsal.** Páginas 79 – 100 do livro **Watch what I do:** programming by demonstration, Allen Cypher. 1993. Disponível em: http://www.acypher.com/wwid/Chapters/04Rehearsal.html. Acesso em: 27 de abril de 2004.

JONATHAN, Miguel. **Introdução à Programação Orientada a Objetos com Smalltalk**. *In*: Jornadas de Atualização em Informática – Congresso SBC, Rio de Janeiro,1994.

KNIGHT, Claire. **Visualisation for Program Comprehension: Information and Issues**. Visualisation Research Group, Centre for Software Maintenance. Department of Computer Science, University of Durham. October, 1998

KOJIMA, Keiji; MATSUDA, Yoshiki; FUTATSUGI, Seiji. **LIVE - Integrating Visual and Textual Programming Paradigms**. Central Research Lab, Hitachi Ltd. IEEE Workshop on , 4-6 Oct., pg.80 a 85, 1989.

LIMA, Maria Esther Russo; SARAIVA, Sérgio Dagoberto Oliveira. **Princípios ergonômicos aplicados à garantia da qualidade de interfaces para Sistemas de Recuperação da Informação (SRI).** *In* Anais 1º Congresso Nordeste de Qualidade e Produtividade, Natal, RN. 2001.

LOGO em uma única linha. Trabalho dos alunos do Curso de Licenciatura em Ciências Exatas. Disciplina: Instrumentação para o Ensino IFSC, USP, São Carlos. 2003 Disponível em: http://educar.sc.usp.br/licenciatura/2003/trabalho_logo.html. Disponível em 18 de junho de 2004.

LOGO, **Ensinando o Computador**. Disponível em: http://www.ars.com.br/arshome/logo.htm. Acesso em: 10 de junho de 2004.

MARIANI, Antonio Carlos. **O Mundo dos Atores: uma perspectiva de introdução à programação orientada a objetos.** *In* Anais do Simpósio Brasileiro de Informática (SBIE), Fortaleza, CE. 1998.

MEDEIROS, Zulmira. **Informática na Educação: A linguagem LOGO**. Auge Educacional, 2003. Disponível em: http://www.augeeducacional.com.br/zulmira_02.asp. Acesso em: 18 de junho de 2004.

MEYER, R. Mark; MASTERSON, Tim. **Towards a Better Visual Programming Language:** Critiquing Prograph's Control Structures. *In:* JCSC 15, pg. 183 – 196. Computer Science Department. Canisius College. Buffalo, NY. 2000.

MORAES, Anamaria de, MONT´ALVÃO, Claudia. **Ergonomia: conceitos e aplicações**. Rio de Janeiro, 2AB - Série Oficina, 2000. 2a Ed. 132p.

NAJORK, M. **Visual programming in 3-d**. *Dr. Dobb's Journal*, 20(12):18-31, December 1995. Disponível em: http://www.csun.edu/~renzo/research/valgo.pdf. Acesso em: 07 de outubro de 2003.

NARAYANAN, N. Hari; HÜBSCHER, Roland. **Visual Language Theory: Towards a Human-Computer Interaction Perspective.** 1997. Disponível em: http://citeseer.nj.nec.com/narayanan97visual.html. Acesso em: 28 de abril de 2004.

NASCIMENTO JR., Wellington Barbosa do. 2000. **Modelagem do Conhecimento Ergonômico para Avaliação da Usabilidade de Objetos de Interação**. Projeto de Dissertação submetido ao Programa de Pós-Graduação em Ciência da Computação, UFSC, Florianópolis, SC.

NETTO, Alvim Antônio de Oliveira. **Software de Entretenimento e suas Contribuições no Processo de Ensino-Aprendizagem: O Erro como Tomada de Decisão**. Dissertação de Mestrado. Juiz de Fora, MG, 2001.

PIANESSO, Ana Cláudia Fiorin. **Identificação e Classificação de Comportamentos de Objetos Dinâmicos.** Dissertação de Mestrado. Florianópolis: UFSC, 2002.

PINTER, André Demboski. **MuseuVirtual: Ferramenta de Autoria para Criação de Museus em Realidade Virtual para Apoio à Aprendizagem Colaborativa via Internet**. Bola ITI. Universidade Federal de Santa Catarina. 2001.

PRESMANN, Roger S. **Engenharia de Software**. Tradução por José Carlos Barbosa. São Paulo: Makron Books, 1995.

RAMOS, Edla M. F. **Análise Ergonômica do Sistema HiperNet buscando o Aprendizado da Cooperação e da Autonomia**. Projeto de Tese submetido ao Programa de Pós-Graduação em Engenharia de Produção, UFSC, Florianópolis, SC. 1995.

RASIA, Marcos. **HetNOS: Heterogeneous Networking Operating System.** Projeto de Graduação, Curso de Ciência da Computação da Universidade Católica de Pelotas, UcPel, 2002. Disponível em: http://atlas.ucpel.tche.br/~rasia/so2/pesquisa.htm. Acesso em: 28 de outubro de 2003.

REPENNING, Alexander, LOANNIDOU, Andri. **Behavior Processors: Layers between End-Users and Java Virtual Machines** Department of Computer Science. University of Colorado, Boulder CO. 1997. Disponível em: http://www.cs.colorado.edu/~l3d/systems/agentsheets/Documentation/VL-97-Paper.pdf. Acesso em: 05 de setembro de 2003.

SEBESTA, Robert W. **Conceitos de Linguagens de Programação.** 4ª edição. Porto Alegre: Bookman, 2000.

SCHÜTZ, Fernando. **Programação Orientada a Comportamentos como uma Extensão ao Modelo de Atores**. Dissertação de Mestrado. Florianópolis: UFSC, 2003.

SIMPLES Consultoria. **Serviços / Ergonomia de Software**. Disponível em: http://www.simplesconsultoria.com.br/servicos_ergonomia.php. Acesso em: 10 de julho de 2004.

TERRA, José Cláudio et all. **Usabilidade: conceitos centrais**. 2001. Disponível em: http://www.terraforum.com.br/lib/pages/download.php?l_intDocCod=132&l_intView=1. Acesso em: 10 de junho de 2004.

VERWOERD, Theuns. **Visual Programming Languages - ThingLab**. 1999. Disponível em: www.cosc.canterbury.ac.nz/~wolfgang/NewHome/cosc414/projects/thinglabFolder/html/thinglab.html. Acesso em: 28 de abril de 2004.

YOUNG, Mark; ARGIRO, Danielle, KUBICA, Steven. **Cantata: The Visual Programming Environment for the Khoros System.** Khoral Research, Inc. 2000. Disponível em: http://www.khoral.com/downloads/papers/cantata.pdf. Acesso em: 29 de abril de 2004.

WAZLAWICK, Raul S. et all. **Providing More Interactivity to Virtual Museums:** A Proposal for a VR Authoring Tool. Presence Teleoperators And Virtual Environments, MIT Press, Cambridge, USA. 10(6):647-656. December. 2001.