UNIVERSIDADE FEDERAL DE SANTA CATARINA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FRANCISCO ANTONIO FERNANDES REINALDO

DEFINIÇÃO E APLICAÇÃO DE UM FRAMEWORK PARA DESENVOLVIMENTO DE REDES NEURAIS MODULARES E HETEROGÊNEAS

Florianópolis - SC
2003

FRANCISCO ANTONIO FERNANDES REINALDO

DEFINIÇÃO E APLICAÇÃO DE UM FRAMEWORK PARA DESENVOLVIMENTO DE REDES NEURAIS MODULARES E HETEROGÊNEAS

Dissertação de Mestrado submetida à avaliação como requisito parcial para a obtenção do grau de Mestre em Ciências da Computação pela Universidade Federal de Santa Catarina (UFSC) no curso de Mestrado em Ciências da Computação com área de concentração em Sistemas de Conhecimento (Inteligência Artificial).

PROF. DR. EEL. MAURO ROISENBERG
(Orientador)

Florianópolis - SC
2003

CIP - CATALOGAÇÃO NA PUBLICAÇÃO

Reinaldo, Francisco Antonio Fernandes
Definição e aplicação de um framework para desenvolvimento de redes neurais modulares e heterogêneas / por Francisco Antonio Fernandes Reinaldo - Florianópolis: CPGCC da UFSC, 2003. 86f.:il.

Dissertação - Universidade Federal de Santa Catarina. Programa de Pós-Graduação em Ciências da Computação, Florianópolis, BR-SC, 2003. Orientador: Roisenberg, Mauro.

1. Rede Neural Artificial 2. PiramidNet 3. UML 4. Padrões de Projeto
5. Framework 6. Robótica. I. Roisenberg, Mauro. II. Título.

DEFINIÇÃO E APLICAÇÃO DE UM FRAMEWORK PARA DESENVOLVIMENTO DE REDES NEURAIS MODULARES E HETEROGÊNEAS

FRANCISCO ANTONIO FERNANDES REINALDO

Essa dissertação foi julgada adequada para a obtenção do título de Mestre em Ciência da Computação com Área de Concentração em Sistemas de Conhecimento e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.

Prof. Mauro Roisenberg
Orientador

Prof. Fernando Álvaro Ostuni Gauthier
Coordenador

Prof. Mauro Roisenberg
Presidente da Banca

Prof. Jorge Muniz Barreto
Membro da Banca

Prof. Ricardo Pereira e Silva
Membro da Banca

Prof. Marcelo Soares Pimenta
Membro da Banca

EPÍGRAFE

"Na construção de uma frase, cada palavra é um parto."
Jovelino Falqueto

"Há homens que lutam um dia e são bons.
Há outros que lutam um ano e são melhores.
Há aqueles que lutam muitos anos e são muitos bons.
Mas há aqueles que lutam toda a vida.
Esses são os imprescindíveis."
Bertolt Brecht

"Não é o desafio que nos deparamos que determina
quem somos e o que estamos nos tornando, mas
a maneira com que respondemos ao desafio."
Henfil

DEDICATÓRIA

Essa obra é dedicada aos meus pais, Alcebíades e Emília, que foram muito compreensíveis em entender, apoiar e acreditar em todas minhas as decisões. A minha irmã, Franci Dely (*in memorian*), que me ensinou que a vida deve ser vivida . A minha namorada, Adriana Salvador, que agüentou firme a distância e sempre acreditou em meu trabalho.

AGRADECIMENTOS

Agradeço a Deus por ter me dado saúde, força e fé. Ao meu amigo e professor orientador, por ter me acolhido, tido a paciência que teve em todo o caminho, ter sido compreensivo, um excelente orientador e ter me ensinado a amadurecer na pesquisa científica, ajudando a ampliar minhas idéias de pesquisa.
Ao meu "grande" amigo Barreto, por sempre ter me ajudado, ter confiado em mim e passado suas experiências de vida. Ao professor Paulo S. S. Borges, por ter sido pertinente na pré-banca. Ao prof. Ricardo P. Silva, que me ajudou a colocar as idéias em ordem. Ao M. Pimenta, que deu uma "passada de olho". Ao meu amigo Leonardo F., que sempre me ajudou e me deu forças para resolver os problemas que encontrava. A Cíntia Schoeninger, minha excelente amiga 24h, por não medir esforços em me ajudar em qualquer situação. Ao meu companheiro de estrada Flavinho, por ter me apresentado esta oportunidade Inteligente. Aos meus companheiros de laboratório e amigos que fiz aqui em Florianópolis, Lú, Eli, Anderson, Renato, Polyana, Fauze, Boi, Urso, Ken, Fátima, Cláudia, Romualdo, Pavan e EdsonCordeiro, por terem me passado algumas dicas para que eu pudesse incrementar o trabalho. Ao marido da Cíntia, Rafael Signori pela restauração doc. A BethMG pelas dicas portuguesas. Ao marido da Fátima, Rogério Schmitt, por ter resolvido um problema *template* e ao pessoal da secretaria que me sempre me ajudou.

RESUMO

Uma das alternativas mais promissoras atualmente para programação de robôs móveis inteligentes é a Robótica Baseada em Comportamento (RBC). Em linhas gerais, esta corrente agrega uma forte inspiração biológica e defende que através da observação de mecanismos biológicos e a solução da sua essência em sistemas de computação, é possível fazer emergir comportamentos complexos e inteligentes em sistemas robóticos. Baseado nessas idéias, criou-se a Arquitetura PiramidNet. Nessa arquitetura, é possível gerar comportamentos complexos e inteligentes em um robô móvel. Para que isto aconteça, necessita-se construir uma hierarquia de modelos heterogêneos de redes neurais artificiais (RNAs) e que emule em um sistema de computação. Após pesquisas de mercado, constatou-se que as ferramentas oferecidas para o desenvolvimento de RNAs, não ofereciam o recurso de conexão entre redes neurais heterogêneas. Usando a UML e Padrões de Projeto, o Framework PiramidNet contém uma estrutura de classes necessária para gerar uma ferramenta gráfica de nome Ferramenta PiramidNet. Esta Ferramenta inclui características que permitem ao usuário: interconectar módulos heterogêneos de RNAs, treiná-los e obter um código-fonte. Este código-fonte, código ANSI C obtido através da codificação do modelo gráfico, gerado automaticamente pela Ferramenta, pode ser facilmente compilado e transportado para o sistema alvo escolhido à execução de comportamentos. A Ferramenta PiramidNet capacita o usuário a projetos gráficos com modelos heterogêneos de RNAs, como também gerar um código-fonte sem conhecer uma linguagem de programação. Com esta Ferramenta, o usuário gerar um sistema nervoso para o robô móvel.

Palavras-chave: Redes Neurais Artificiais, PiramidNet, UML, Padrões de Projeto, Framework e Robótica.

ABSTRACT

TITLE: PROJECTING A FRAMEWORK AND PROGRAMMING A SYSTEM FOR DEVELOPMENT OF MODULATE AND HETEROGENEOUS NEURAL NETS.

One of more ones promising alternatives to program intelligent mobile robots is the Behavior-based approach. In general lines, this research follows a strong biological inspiration and it defends that through the observation of biological mechanisms and the programming of them - essence - in computacional systems. This line is capable to do to emerge complex and intelligent behaviors in robotic systems. The PiramidNet Architecture is based on these ideas. In this architecture, it is possible to generate complex and intelligent behaviors in a mobile robot. To do so, it is necessary to build a hierarchy of heterogeneous architectures of artificial neural nets(ANN) emulating in a computacional system. Looking forward to solve this type of problem, it was verified that the tools offered didn't offer the possibility of internal connection among neural nets heterogeneous. Using UML and Design Patterns, Framework PiramidNet contains a necessary structure of classes to generate a graphic tool of name Ferramenta PiramidNet. This tool includes characteristics that allow to the user: to interconnect heterogeneous modules of ANN, to train them and to obtain a final source code - code ANSI C obtained through the code of the graphic model. This code was generated automatically by the Tool and can be compiled easily and transported for the white system chosen for the execution of behaviors., the tool leaves the user capable to build graphic ANN projects and to generate a source code without to know of programming language. Therefore user could get to generate a nervous system for the mobile robot.

Keywords: Artificial Neural Nets, PiramidNet, UML, Framework and Robotic.

SUMÁRIO

1 INTRODUÇÃO 14
1.1 Motivação 15
1.2 Objetivo Geral 17
1.2.1 Objetivo Específico 17
1.3 Organização do Texto 18

2 PROJETO PIRAMIDNET – REDES NEURAIS MODULARES E HIERÁRQUICAS 20
2.1 Considerações Iniciais 20
2.2 Definição 20
2.3 Inspiração 21
2.4 O Projeto 24
2.5 Modularização 25
2.6 Classes de Comportamentos 27
2.6.1 Estereotipados 27
2.6.1.1 Reflexos 27
2.6.1.2 Taxias 28
2.6.2 Reativos ou Padrão Fixo de Ação (PFA) 28
2.6.3 Instintivos 29
2.6.4 Deliberativos 30
2.7 Arquitetura PiramidNet e as Classes de Comportamento 30
2.8 Considerações Finais 32

3 FRAMEWORKS ORIENTADOS A OBJETOS 33
3.1 Considerações Iniciais 33
3.2 Fundamentos 33
3.3 Definição 34
3.4 Projeto 35
3.4.1 Elementos do Projeto 36
3.5 Metodologias de Desenvolvimento de Frameworks 37
3.6 Desenvolvimento 40

3.6.1 Estrutura 41
3.6.2 Padrões de Projetos (*Design Patterns*) 41
3.6.3 Tipos de Frameworks 45
3.7 Framework de Aplicação 47
3.7.1 Vantagens 48
3.8 Considerações Finais 48

4 FRAMEWORK PIRAMIDNET E FERRAMENTA PIRAMIDNET 49
4.1 Considerações Iniciais 49
4.2 Definições 49
4.3 Recursos e Linguagem 50
4.4 Modelagem Visual por Diagramas 50
4.4.1 Diagramas de Casos de Uso 51
4.4.2 Diagrama de Classes 53
4.4.3 Diagramas de Interação - Seqüência 60
4.4.4 Diagramas de Comportamento - Atividade 62
4.4.5 Diagramas de Implementação - Componentes 63
4.5 Interface Gráfica 64
4.6 Utilizando a Ferramenta 66
4.7 Considerações Finais 70

5 POTENCIALIDADE DA FERRAMENTA PIRAMIDNET 71
5.1 Considerações Iniciais 71
5.2 Projeto Manual de RNAs: Seguir Paredes 71
5.3 Projeto Automatizado de RNAs: Seguir Paredes 75
5.3.1 Gerando o Código ANSI C 75
5.4 O Potencial da Ferramenta PiramidNet 76
5.5 Considerações Finais 77

6 CONCLUSÕES 78
6.1 Limitações 80
6.2 Sugestões para Trabalhos Futuros 80

REFERÊNCIAS BIBLIOGRÁFICAS 81

LISTA DE ILUSTRAÇÕES

Figura 1 – Pirâmide com RNAs Hierárquicas, reproduzido de Roisenberg (2000)..... 21
Figura 2 - Reproduzido de Eibl-Eibesfeldt (1979) (PATO.JPG, 2002)..... 29
Figura 3 - Reproduzido de Eibl-Eibesfeldt (1979) (GARRAS.JPG, 2002)..... 29
Figura 4 - Ciclo de ação-percepção, Arkin (1999). 31
Figura 5 - Aplicação desenvolvida reutilizando um framework – exemplo genérico,
extraído de (SILVA, 2000) 47
Figura 6 – Montando o Projeto - Diagrama de casos de uso da Ferramenta PiramidNet..... 52
Figura 7 - CRNA – Classe abstrata do framework com seus relacionamentos. 54
Figura 8 - Diagrama de classes do Framework. 56
Figura 9 –Treinamento de RNAs - Exemplo de diagrama de seqüência da Ferramenta. 61
Figura 10 – Inserir Sensor – Exemplo de diagrama de atividades da Ferramenta. 62
Figura 11 – Classe CAtuador - Exemplo de diagrama de componentes da Ferramenta. 63
Figura 12 - Tela principal da Ferramenta PiramidNet. 65
Figura 13 - Barra de Botões da Ferramenta PiramidNet. 66
Figura 14 - Formulário Sensor da Ferramenta PiramidNet 66
Figura 15 - Formulário Atuador da Ferramenta PiramidNet..... 67
Figura 16 - Formulário [+] Direta da Ferramenta PiramidNet. 68
Figura 17 - Menu popup Rede/Link da Ferramenta PiramidNet 69
Figura 18 - Menu popup Sensor da Ferramenta PiramidNet 69
Figura 19 - Menu popup Atuador da Ferramenta PiramidNet..... 69
Figura 20 - Tela de experimento da Ferramenta PiramidNet. 70
Figura 21 – Hierarquia dos AEFs, extraído de Silva (2001). 72
Figura 22 - Diagrama feito manualmente, Projeto de RNAs, extraído de Silva (2001). 74
Figura 23 - Diagrama proposto pela Ferramenta PiramidNet, Projeto de RNAs. 75

LISTA DE ABREVIATURAS

AEFs – Autômatos de Estados Finitos

CORBA - Common ORB Architecture (Arquitetura ORB Comum)

IAC - Inteligência Artificial Conexionista

IAS - Inteligência Artificial Simbólica

MK - Mapas Auto-organizáveis de Kohonen

ORB - Object Request Broker (Corretor de Pedido de Objeto)

PFA - Padrão Fixo de Ação

RMI - Remote Method Invocation (Pedido de Método Remoto)

RNAs - Redes Neurais Artificiais

SA - Subsumption Arquitecture (Arquitetura de Assunção)

SE - Sistema Especialista

SNC - Sistema Nervoso Central

UML - Unified Modeling Language (Linguagem de Modelagem Unificada)

1 INTRODUÇÃO

A implementação da robótica baseada em comportamentos usando técnicas inspiradas na Natureza, em especial as Redes Neurais Artificiais (RNAs), tem sido uma área de pesquisa extremamente promissora. Projetos como o PiramidNet, coordenado pelo Prof. Dr. Mauro Roisenberg (ROISENBERG, 2000), tem feito uso da inspiração biológica para gerar comportamentos inteligentes em robôs moveis. Este projeto usa modelos heterogêneos de RNAs interconectadas para gerar comportamentos inteligentes. Outros trabalhos, sob a supervisão do prof. Roisenberg, mostraram a possibilidade de implementação de diferentes classes de comportamentos, utilizando diferentes modelos de estruturas de RNAs (SILVA, 2001) (OLIVEIRA, 2001). Para o usuário, as vantagens advindas da utilização de RNAs incluem a facilidade em gerar e coordenar os comportamentos do robô, além daquelas vantagens próprias das RNAs, tais como: imunidade a ruídos, tolerância a falhas, programação através de exemplos, processamento paralelo e distribuído etc.

Para que os pesquisadores de RNAs possam se utilizar desta abordagem biologicamente inspirada e assim gerar uma combinação de diferentes comportamentos, é necessário o uso de diferentes modelos de redes neurais. O enfoque deste trabalho é construir um framework (Framework PiramidNet) e modelar uma ferramenta gráfica (Ferramenta PiramidNet) que trabalhe com modelos de RNAs heterogêneas e interconectadas. Valendo-se da inspiração da Natureza para gerar modelos hierárquicos e heterogêneos de redes neurais artificiais, estes modelos serão responsáveis por emergir comportamentos no robô. A Natureza, através dos mecanismos de evolução e seleção natural, criou sistemas nervosos para resolver o problema da sobrevivência do organismo em uma diversidade de ambientes. Com esta Ferramenta, o usuário poderá também construir um “sistema nervoso” para o robô, fazendo inserções e exclusões de conexões “nervosas” nas estruturas neurais.

Na literatura consultada, encontrou-se apenas uma obra que aborda o uso de RNAs em frameworks. Infelizmente, o livro de Adam Blum (1992) faz uma classificação pouco adequada para frameworks e o real uso de RNAs. Este trabalho não se baseou no livro de Blum e defende que para resolver problemas em RNAs, deve-se existir uma topologia de redes. A seleção e construção topológica de RNAs estão ligadas a classes-problema que a mesma é capaz de resolver. Como se tem classes com comportamentos, existem alguns que são reflexivos, que usam a rede direta. Já certos comportamentos que não são reflexivos utilizam a rede com ciclos e isto é algo importante. Este conhecimento permite associar um tipo de problema à topologia de uma rede neural. Logo, parece-me adequado definir as classes

de comportamentos que o livro de Adam Blum não faz. Também convém anotar que o trabalho de Blum é de uma época em que a grande ênfase era nos algoritmos de aprendizagem das redes neurais, mas o conceito de complexidade neural só começou a ser despertado em 1994 com o trabalho de Parberry (1994) e a atual classificação de complexidade sugerida em 1994 na tese e concurso para titular de Jorge Muniz Barreto. Os principais resultados de J. M. Barreto são apresentados em sua obra no capítulo 8 (BARRETO, 2001).

Assim, este trabalho propõe:

- Descobrir comportamentos através da implementação de redes neurais artificiais heterogêneas independentes e interconectadas;
- Construir um framework que será usado como suporte à produção de uma ferramenta que apóie o uso de RNAs;
- Usar a interface visual para desenvolver projetos de RNAs;
- Converter os diagramas, desenvolvidos pelo usuário da ferramenta, em código-fonte - linguagem C;
- Desenvolver projetos gráficos usando RNAs modulares e heterogêneas;
- tornar amplamente conhecida a aplicabilidade de RNAs para resolução de problemas;
- disponibilizar diferentes modelos de RNAs para que sejam aplicadas em artefatos robóticos.

1.1 Motivação

Observando um inseto, mesmo que pouco evoluído, percebe-se que este tem diferentes comportamentos e diferentes estruturas nervosas (HEBB, 1949 *apud* BARRETO, 2001). A possibilidade de gerar diferentes comportamentos, sugere a necessidade de possuir diferentes modelos de RNAs. Para se alcançar comportamentos complexos é necessário a existência de redes heterogêneas independentes e interconectadas.

Comportamentos simples do tipo reflexivo que envolva basicamente mecanismos de ação e reação, podem ser obtidos através da utilização de RNAs Diretas. Comportamentos mais complexos que envolvam a necessidade de memorização de eventos passados, bem como a coordenação dos comportamentos, podem ser implementados por RNAs Recorrentes (ROISENBERG, 2000). A implementação de RNAs heterogêneas e interconectadas de forma

hierárquica contribui significativamente para a geração de comportamentos complexos. Diferentes tipos de comportamentos podem ser gerados para robôs móveis. Por exemplo, comportamentos reflexivos e reativos oferecem aos robôs a capacidade de se tornarem inteligentes, caso se considere inteligência como a capacidade de reagir adequadamente aos estímulos vindos do ambiente no intuito de realizar uma tarefa ou atingir um objetivo.

Uma vez estabelecido que a construção de RNAs heterogêneas e interconectadas – organizadas de forma hierárquica - possam ser utilizadas para gerar comportamento inteligente em robôs móveis, é necessário encontrar uma ferramenta de desenvolvimento de RNAs que tenha esta característica. A ferramenta deve permitir ao projetista: desenvolver um tipo de modelo de RNA heterogênea e interconectada; alcançar os comportamentos desejados; gerar código-fonte do diagrama, para posterior implementação no artefato robótico.

Ferramentas comerciais como Matlab, NeuroDimension/NeuroSolution e SNNS foram analisadas, através de um conjunto de perguntas, a fim de verificar sua eficácia em problemas com interconexão heterogênea de RNAs. As perguntas elaboradas para esta análise foram:

a) É possível trabalhar com duas ou mais RNAs em uma mesma área de trabalho?
b) É possível simular tipos heterogêneos de RNAs?
c) É possível conectar sensores, atuadores e RNAs?
d) a partir de uma interface visual, é possível gerar um programa-fonte que tenha uma estrutura de código com suporte a sensores e atuadores para artefatos robóticos?
e) É possível estender novas RNAs?
f) É possível alterar o código-fonte gerado?
g) É possível implementar especificidades na rede neural selecionada?

Depois de colhida cada resposta, percebeu-se que as ferramentas provaram ser insuficientes para resolver o problema proposto de conexão entre RNAs heterogêneas.

O estudo em questão se propõe a desenvolver um framework que possibilite montar uma ferramenta gráfica que dê suporte a RNAs com recursos de modelagem para o usuário. Estes recursos possibilitam ao desenvolvedor mais flexibilidade e modularidade na criação de novos modelos de redes. Por conseqüência da ligação de redes neurais artificiais dos mais diversos tipos e comportamentos complexos poderão ser capazes de emergir no artefato robótico. O framework inicia com dois modelos implementados: redes diretas e redes recorrentes.

1.2 Objetivo Geral

O objetivo geral desta dissertação é desenvolver um framework orientado a objetos que seja capaz de criar uma ferramenta gráfica. Esta ferramenta utilizará modelos de RNAs heterogêneas.. Basicamente, a ferramenta deve: disponibilizar um conjunto de modelos de RNAs; inserir e apagar graficamente estruturas de RNAs sem afetar estruturas anteriores; conectar objetos; gerar programas-fonte - proveniente da interface gráfica Estas estruturas modulares podem ser hierárquicas e estarem interconectadas com outras redes heterogêneas.

Visto que as ferramentas de mercado analisadas não apresentaram os recursos necessários para que o usuário possa conectar modelos heterogêneos de redes neurais artificiais de forma gráfica, esta nova proposta vem suprir a necessidade desta comunidade de pesquisadores da área de Inteligência Artificial Conexionista (IAC).

Além disso, tanto o framework como a ferramenta originada a partir da estrutura de classes pré-moldadas do framework, devem ser facilmente compreensíveis por novos usuários e expansíveis para novos modelos de redes neurais. Estas redes neurais artificiais podem ser incorporadas de maneira a não afetar o funcionamento do sistema. Para atingir este objetivo, técnicas de Engenharia de Software serão empregadas. Para se construir um framework e conseqüentemente uma ferramenta, é importante desenvolver um planejamento que alcance sempre as necessidades com relação à aplicabilidade.

1.2.1 Objetivo Específico

Para alcançar o objetivo geral, deve-se atingir os objetivos específicos abaixo:

a) Definir uma área de modelagem para o usuário trabalhar com modelos de RNAs;
b) Especificar modelos heterogêneos de RNAs;
c) Conectar diferentes objetos no ambiente;
d) Recuperar modelos salvos de projetos;
e) Adicionar modelos salvos de RNAs ao modelo de trabalho corrente;
f) Converter automaticamente o diagrama estruturado pelo usuário para um formato, ANSI C, que seja exportado para qualquer sistema alvo, inclusive para os robôs Khepera e Lego;
g) Imprimir o diagrama estruturado pelo usuário;

h) Descrever a utilização das RNAs hierárquicas modulares, envolvidas no Projeto PiramidNet, para a obtenção de classes de comportamentos observados na Natureza;
i) Exemplificar o uso de RNAs Diretas com o comportamento reflexivo e de RNAs Recorrentes com o comportamento reativo;
j) Utilizar conceitos de Engenharia de Software, Frameworks e UML, para construir uma ferramenta voltada ao domínio e aplicação.

1.3 Organização do Texto

No escopo de realizar os objetivos citados, este trabalho está organizado em 6 capítulos. Tendo em vista a diversidade de temas que estão sendo abordados nesta dissertação, para um bom entendimento dos assuntos, faz-se a necessária explicação de conceitos dos temas primários Inteligência Artificial (cap. 2) quanto de Engenharia de Software (cap 3).

No capítulo 2, Projeto PiramidNet - Redes Modulares e Hierárquicas, apresenta o estudo de técnicas para geração dos módulos estruturais; esquemas de aprendizado biologicamente compatíveis e a adaptação progressiva de um organismo; explana a conexão hierárquica e modular de arquiteturas neurais; expõe qualidades e diferenças com relação ao comportamento e a inteligência, enfatizando os níveis inferiores e superiores da estrutura piramidal.

No capítulo 3, Frameworks Orientado a Objetos, faz uma breve conceituação, destacando o projeto e enfocando as vantagens de usar frameworks. Suas características, aplicabilidades, formas de estruturação e metodologia serão usadas para fazer uma análise e projetar o Framework PiramidNet. Um estudo pormenorizado indicando as técnicas metodológicas que já existem, contemplam a meta e finalizam este capítulo.

No capítulo 4, Framework PiramidNet e Ferramenta PiramidNet, faz-se uma definição formal sobre o framework e a ferramenta gerada a partir da estrutura do framework. Comentam-se os principais diagramas UML, suas características e funcionalidades.

No capítulo 5, Exemplo de Utilização, trata da terminologia e utilização da ferramenta. Usando um modelo de problema envolvendo o uso de RNAs heterogêneas hierárquicas e interconectadas, mostrado ao usuário a procedência com a ferramenta para montar seu projeto.

No capítulo 6, Conclusões, os resultados obtidos são interpretados e analisados, onde se procura expor as dificuldades encontradas e o grau de satisfação alcançado com o trabalho. Este capítulo também sugere trabalhos futuros.

2 PROJETO PIRAMIDNET – REDES NEURAIS MODULARES E HIERÁRQUICAS

2.1 Considerações Iniciais

Os conjuntos de atitudes e reações dos seres vivos em face do meio social, constituem uma variedade de comportamentos. Estes comportamentos são alcançados graças a uma hierarquia modular e interconectada de sistemas independentes, no Sistema Nervoso Central (ROISENBERG, 1998b). Buscando na escala evolutiva por um "ótimo", a Natureza procura atingir um alto nível de inteligência e autonomia através das hierarquias de redes neurais biológicas (neuronais) (ROISENBERG, 1998b). Assim, este capítulo apresenta um estudo envolvendo modelos de RNAs com suas hierarquias e o resultado proveniente da interconexão destas redes, objetivo principal do Projeto PiramidNet.

2.2 Definição

A evolução do sistema nervoso nos organismos é caracterizada por uma série de transições sistêmicas, produzindo níveis imediatos de complexidade e/ou controle, conforme a classificação proposta por Turchin (1977), Heylighen (1995, p. 59-85) e Pallazo (2002). A etapa em que os sensores se ligam um-para-um aos efetores por um caminho nervoso ou arco reflexo, corresponde a reflexos. A medida que cresce a complexidade de conexões nervosas, surge uma estrutura. Esta estrutura pode ser reconhecida como um cérebro rudimentar.

O Projeto PiramidNet, desenvolvido no Laboratório de Conexionismo e Ciências Cognitivas (L3C) do Departamento de Informática e Estatística (INE) da UFSC e coordenado pelo Professor Mauro Roisenberg, utiliza-se de modelos de RNAs para gerar comportamentos inteligentes em robôs móveis. O Projeto PiramidNet tem como objetivos: a investigação; o desenvolvimento de técnicas e a criação de ferramentas de auxílio a pesquisadores, para a geração automática dos módulos de uma estrutura neural hierárquica e/ou heterogênea.

A proposta do projeto PiramidNet propõe-se a organizar de uma forma hierárquica e piramidal, as arquiteturas RNAs. Assim, as arquiteturas mais simples posicionam-se na base e as arquiteturas mais complexas posicionam-se no topo da pirâmide. Desta forma, na Figura 1, é possível arranjar os modelos heterogêneos de RNAs em diferentes níveis hierárquicos.

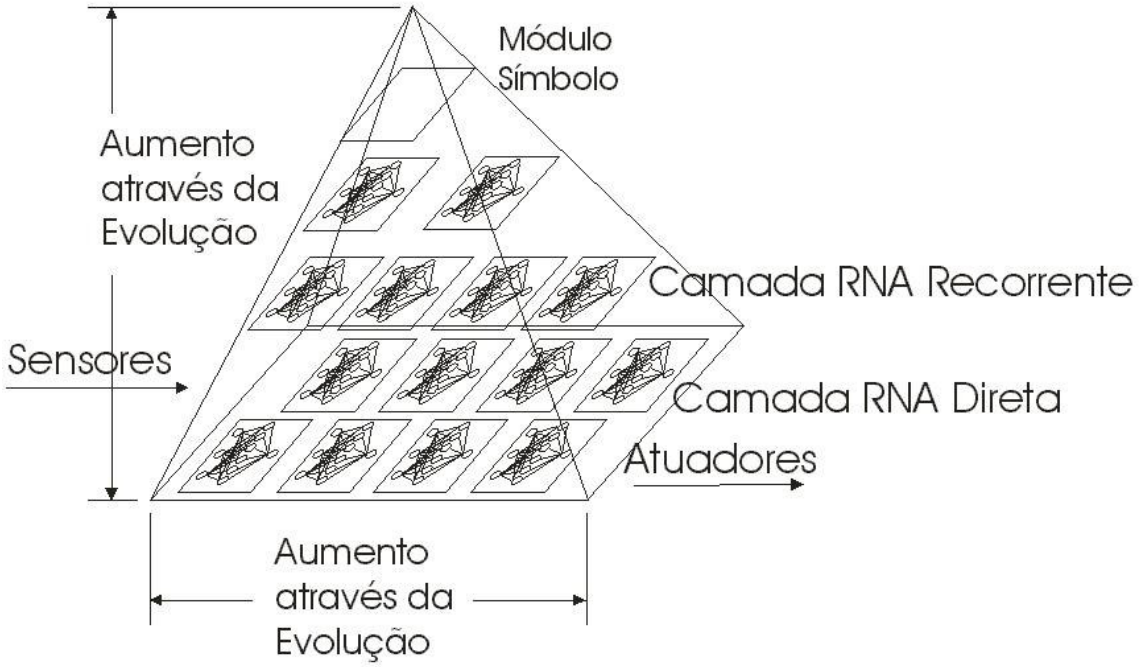

Figura 1 – Pirâmide com RNAs Hierárquicas, reproduzido de Roisenberg (2000).

No sentido horizontal, cada nível da hierarquia é composto por várias redes homogêneas, isto é, com a mesma topologia. No sentido vertical, níveis ou camadas superiores vão se formando com redes de topologias diferentes. Nesta organização, as RNAs estão interconectadas e as redes superiores, mais complexas, controlam as redes dos níveis inferiores, mais simples.

2.3 Inspiração

Utilizando-se das áreas de inteligência computacional e robótica, é possível viabilizar a criação de produtos/serviços na informática. Estes produtos/serviços surgem como programas com regras de produção e se estende a artefatos eficientes que possam substituir o homem em um determinado setor de trabalho onde envolvam riscos de saúde, atividades repetitivas, estressantes etc, segundo Anderson (1990, p. 145-168), Barto (1990, p. 5-58), Bourgine (1991, p. 11-17) e Brooks (1985, 1991. p. 3-10).

Na Inteligência Artificial Simbólica (IAS), deve-se conhecer claramente o domínio do problema e as formas de resolvê-lo. Esta solução é obtida manipulando conhecimentos básicos e imitando, ao menos até certo ponto, o modo de raciocínio usado por humanos na solução do problema. Para isto, a IAS deve ser usada quando o problema é bem definido, que

se tenha uma boa idéia de como ele seria resolvido e que seja explícito o modo de achar uma solução. Para isto, Barreto afirma que "Raciocínio impreciso, generalizações, raciocínio por padrão, aprendizado, tudo deve ser convenientemente previsto" (BARRETO, 2001).

Sistemas Especialistas Simbólicos (SE) utilizam-se de regras de produção que podem ser utilizadas para simular inteligência artificial no computador (BARRETO, 2001) (BITTENCOURT, 1998). Utilizando um conjunto de dados e regras de inferência, um computador pode ajudar no diagnóstico de uma doença comparando sintomas que estão armazenados em um banco de dados. Contudo, seria difícil obter o comportamento inteligente de um robô em situações onde o ambiente é dinâmico e seu repertório de ações já está pré-definido por um conjunto de regras.

Tendo em vista os fracassos e as dificuldades em implementar programas baseados em IAS na década de 80, Rodney A. Brooks (1985) deu início a projetos que procuravam sanar o problema da programação direta. Posteriormente em 1990, através de seu artigo intitulado "*Elephants Don't Play Chess"*, o autor critica a IAS e alega "*In this paper we argue that the it symbol system hypothesis upon which ´classical AI´ is based is fundamentally flawed, and as such imposes severe limitations on the fitness of this progeny*" (BROOKS, 1990). Em conseqüência disso, o autor desenvolveu um modelo de arquitetura que pudesse operar em ambientes dinâmicos, chamado Arquitetura de Assunção ("Subsumption Architecture"-SA). Conforme Oliveira (2001), este modelo de arquitetura propõe a operação de módulos dotados de comportamentos individuais, dispostos em camadas hierarquizadas. Estas camadas respondem a estímulos ambientais, não se baseando na representação simbólica (ALVES, 1993).

A abordagem baseada em comportamentos conseguiu fazer avançar a pesquisa envolvendo o paradigma de robôs de terceira geração - robôs inteligentes. Nesta abordagem, um controlador baseado em comportamentos foi particionado em camadas. Cada camada torna-se responsável por um tipo de comportamento ou tarefa a ser executada pelo sistema global. Cada camada usa apenas as informações do sensor que lhe é acoplado e a percepção de mundo que necessita para trabalhar com sua tarefa específica, isto é, cada camada é completa (no sentido de implementar um comportamento ou tarefa). Cada camada, resolve somente o problema que lhe é pertinente. Brooks construiu um robô móvel autônomo utilizando a SA, cuja primeira camada, a camada 0, evita obstáculos. Em seguida, adicionou a camada 1, que introduz a atividade de fazer com que o robô se dirija a um determinado lugar. Independentemente, a primeira camada livra o robô dos obstáculos que lhe interponham. A segunda camada monitora o progresso do robô e envia comandos atualizados aos atuadores,

sem estar consciente dos obstáculos que foram evitados pela camada inferior. Cada camada é composta por uma rede de topologia fixa de máquinas de estado finito simples que são combinados através de mecanismos chamados inibidor e supressor. Estas máquinas são ativadas através de mensagens e a mudança de estado ocorre quando chega uma determinada mensagem ou quando o tempo estipulado para este estado expira. Não há memória global compartilhada. As entradas de cada máquina de estado finito podem ser suprimidas e as saídas podem ser inibidas por outras máquinas (ALVES, 1993 *apud* SILVA, 2001).

Esta técnica de SA não necessita de memória ou decisões lógicas a serem feitas, somente respostas diretas a estímulos. Como exemplo, é possível construir um robô que siga uma fonte luminosa. Vários mecanismos de estímulo/resposta operando juntos em um robô podem criar comportamentos elaborados que podem parecer inteligentes.

Percebe-se que as propostas de Brooks (1990, p. 3-15) buscam diretamente no mundo físico a representação para o desenvolvimento dos chamados sistemas reativos ou comportamentais. Não existe, explicitamente, dentre destes sistemas um objetivo plano ou modelo do mundo, ocorrendo apenas uma reação à situação que eles têm em mãos.

Outra técnica utilizada em inteligência artificial é através de RNAs. Estas RNAs são modeladas conforme o sistema nervoso dos seres vivos, com a vantagem de que são capazes de lidar melhor com ambigüidades do que os sistemas baseados em regras. Modelos específicos de RNAs aprendem por exemplos codificados em valores. Estes valores são encaminhados a camada de entrada, gerando suas saídas esperadas. Uma RNA treinada pode responder a uma entrada com uma provável saída.

Da mesma maneira que os conjuntos de RNAs correspondem ao modelo computacional do sistema nervoso, é certo que manifeste inteligência e que determine o comportamento dos seres vivos (ROISENBERG, 1997, p. 34-45). Através das RNAs, é possível implementar a inteligência e o controle dos sistemas robóticos para atingir nos robôs móveis um nível de satisfação de autocontrole próximo ao requerido.

De acordo com Roisenberg (1996, p. 211-220, 1998b), o processo evolutivo dos seres vivos primeiramente desenvolveu uma estrutura nervosa simples capaz de responder a estímulos ambientais. A necessidade de sobreviver e se reproduzir em ambientes muitas vezes hostis e dinâmicos, forçaram a gerar estruturas modulares nos cérebros dos seres vivos capazes de realizar tarefas complexas. Sendo assim, é correto afirmarmos que nos organismos evoluídos, os nervos e o Sistema Nervoso Central (SNC) compõem uma complexa rede de interconexões onde conjuntos de sinais recebidos são integrados e processados pelo cérebro.

Em conseqüência da busca por uma expressão biológica mais forte, o Projeto PiramidNet utiliza-se da Inteligência Artificial Conexionista. A IAC estuda a arquitetura e os algoritmos inspirados nas redes neuronais. Os resultados gerados pela IAC têm sido de grande valor nas implementações de controles para agentes artificiais autônomos (OLIVEIRA, 2001). Usando a IAC, acredita-se na possibilidade de demonstrar inteligência construindo uma máquina que imite a estrutura do cérebro. Esta máquina seria capaz de obter o conhecimento de como realizar uma tarefa onde exija a necessidade de níveis maiores de inteligência no robô, tais comportamentos como: fugir de obstáculos, vagar no ambiente sem esbarrar em objetos, atingir uma posição determinada etc (ANDERSON, 1990) (BOURGINE, 1991).

2.4 O Projeto

Um robô móvel possui sensores, rodas ou outros aparatos que operam em um ciclo que envolve a aquisição de informações do "ambiente dinâmico" e a geração de dados de saída.

De acordo com Roisenberg (2000), defini-se "ambiente dinâmico" ou "ambiente aberto" como sendo aquele lugar que não é totalmente conhecido, podendo apresentar obstáculos e situações imprevisíveis. A operação dos robôs móveis nestes ambientes está associada a uma série de problemas. Em primeiro lugar, é impossível descrever completamente o ambiente e os obstáculos que o robô irá encontrar; o ambiente aberto apresenta constantemente eventos e situações imprevisíveis que podem levar a replanejamentos e novas ações. Em segundo lugar, desenvolver robôs móveis capazes de realizar suas tarefas operando em ambientes abertos, dinâmicos e desestruturados, de maneira autônoma, tem sido um desafio para muitos pesquisadores de áreas como Robótica e Inteligência Artificial (ANDERSON, 1990, p. 145-168) (BOURGINE, 1991, p. 11-17) (BROOKS, 1991, p. 3-10) (LANGTON, 1991) (LAPRADE, 1993, p. 166-174) (LEITE, 1993, p. 175-184).

Em decorrência da preservação e evolução das espécies, agentes biológicos sentiram a necessidade de resolver os problemas de adaptação e otimização de comportamentos - desde os mais simples até os mais complexos. Em conseqüência disso, a busca incessante por um nível de satisfação "ótimo" (BARTO, 1990, p. 5-58) é a resposta para o animal poder se sobressair a um aleatório ambiente proposto.

Assim, a Natureza utilizou de mecanismos de otimização para desenvolver sistemas que apresentassem uma extraordinária performance de operação e sobrevivência. O interesse em estudar e se utilizar destes mecanismos naturais, leva de encontro a desenvolver modelos que, assim como os animais operariam muito bem no mundo real. Pesquisadores como Rashevsky (1973, p. 143-175) e Rosen (ROSEN, 1973, v. 1, p. 361-393) (*apud* ROISENBERG, 2000) têm procurado mostrar que fatores passíveis de observações e conceitos de otimizações biológicas, obedecem a leis perfeitamente expressas de forma matemática, conseqüentemente podendo ser implementadas em um computador.

Estudando os feitos comportamentais de animais e utilizando-se da abordagem Conexionista, é possível implementar aspectos das soluções encontradas pela Natureza para a sobrevivência dos animais em robôs móveis (BEER, 1990). Através destes estudos, constata-se da existência de uma hierarquia no Sistema Nervoso Central e que esta hierarquia comportamental se dá através de sistemas modulares independentes e interconectados. Utilizando-se de um simulador de robô de domínio público chamado Khepera, foi possível recriar um comportamento animal em um ambiente computacional (SILVA, 2001).

Após a efetivação dos testes no simulador, usando a RNA Direta, Roisenberg (2000) e Silva (2001) perceberam que o robô gerou um comportamento limitado. Em decorrência desse resultado, fez-se a implementação de novas estratégias capazes de gerar comportamentos mais complexos. Passou-se então a uma segunda etapa, a de estudar a organização do sistema nervoso dos seres vivos.

Surgiu a idéia inicial de se utilizar uma nova estratégia de controle para gerar comportamentos, uma arquitetura de rede neural artificial modular e hierárquica – Projeto PiramidNet. Neste projeto, estabelece-se uma arquitetura hierárquica comportamental, onde as redes neurais dos níveis inferiores são responsáveis pela implementação do repertório de comportamentos básicos do robô móvel, enquanto que as redes dos níveis superiores fazem a tarefa de seleção e coordenação dos comportamentos básicos do nível inferior, fazendo emergir um comportamento mais complexo e inteligente.

2.5 Modularização

Para que um agente possa interagir com o mundo real, deve-se ter um conjunto de comportamentos (REINALDO, 2001). Cada comportamento ajusta-se ao estímulo ambiental, i.e., a cada ação detectada, uma reação pode ser gerada.

É possível implementar em um sistema de robô móvel uma grande diversidade de comportamentos, mas devido ao problema de escalabilidade a tarefa torna-se extremamente difícil. Inspirando-se na Natureza, toma-se como solução a aplicação da modularização e hierarquização. Esta solução já foi utilizada em outras técnicas de controle de robôs tais como: a SA proposta por Brooks (1989, 1991, p. 3-10). O desafio neste caso, é aplicar este princípio às Redes Neurais Artificiais.

A modularização divide-se em duas partes: a modularização repetitiva, que se refere ao uso do mesmo tipo de módulo em tempos distintos; modularização diferenciada que se refere ao uso de módulos distintos para a organização de um todo.

Esta característica modular e hierárquica é encontrada no cérebro sob a forma de vários processos sendo executados em paralelo e de maneira hierárquica, por exemplo, na ação de escrever poesias, o agente utiliza uma parte do cérebro para controlar os movimentos da mão para escrever, no entanto a idéia do que escrever está sendo gerada em outra parte do cérebro. Isto significa dois ou mais processos distintos sendo executados em paralelo.

A atuação da modularização é assistida de duas formas: em casos onde serão usados os mesmos módulos para executar processos diferentes em tempos distintos e em casos onde um mesmo processo estará sendo executado por módulos diferentes, em distintas regiões do cérebro ao mesmo tempo.

O sistema modular é classificado em duas estruturas: a horizontal e a vertical.

Uma estrutura horizontal é composta por estruturas modulares de neurônios na mesma camada hierárquica. Utilizando a interpretação de imagens visuais simples na estrutura do sistema, linhas e arcos seriam representados em neurônios que estariam na mesma camada. Caso a imagem fosse mais rica em detalhes, as características seriam combinadas e depois representadas através das camadas subseqüentes de neurônios que são interpretadas pelo córtex (ROISENBERG, 2000).

A estrutura vertical é encontrada no cérebro primata e compõe-se de múltiplos caminhos de processos paralelos. Esta estrutura permite distinguir os processos separado-os em diferentes tipos de informação (ROISENBERG, 2000).

Um bom exemplo pode, mais uma vez, ser achado no sistema visual. Diferentes aspectos de incentivos visuais como forma, colorido, movimento e posição, são processados em paralelo através dos sistemas neurais anatomicamente separados, organizados no caminho da zona magnocelular, de células grandes e uma zona parvocelular, de células pequenas (BARRETO, 2001). Estruturas convergentes integram estes processos de informação visual

separadamente no mais alto nível hierárquico para produzir uma percepção unitária (BOERS, 1992).

A organização modular foi um dos modos que a Natureza encontrou para dar inteligência aos seres vivos. O Projeto PiramidNet se baseia em estruturas horizontais e verticais da organização modular para o desenvolvimento de trabalhos computacionais onde existem neurônios organizados na forma horizontal e vertical. Também procura entender como esta estrutura interage na emergência de comportamentos.

2.6 Classes de Comportamentos

Para que seja possível lidar com os comportamentos, é necessário conhecê-los. A relação abaixo expõe algumas classes de comportamentos possíveis:

2.6.1 Estereotipados

Roisenberg (1999), detalha abaixo esta classe mais simples de comportamentos dividindo-a em reflexos e taxias.

> Estes comportamentos simples se caracterizam por apresentarem uma resposta praticamente instantânea a um dado estímulo disparador e quando o estímulo cessa, cessa também a ação correspondente. Esta classe de comportamentos simples pode ser modelada como funções booleanas das entradas e pode ser implementada através de redes neurais diretas, com conexões entre os sensores e os atuadores ou passando por uma camada intermediária de neurônios (ROISENBERG, 1996). Encontra-se na Natureza evidências abundantes de que tais comportamentos são efetivamente implementados através de conexões diretas entre neurônios, bastando observar o sistema nervoso e o repertório de comportamentos de animais existentes no início da escala da série filogenética, tais como os metazoários e os vermes.

2.6.1.1 Reflexos

Fronteira define o termo reflexo como "[...]sendo uma atividade involuntária de um órgão ou outro, como resposta a um determinado estímulo ambiental" (FRONTEIRA, 1999). Silva (2001) atenta na importância da duração do estímulo que disparou o comportamento. Nesta importância temporal, ressalta-se que o tempo decorrido entre a ocorrência do estímulo e o surgimento da resposta é bastante reduzido, pois envolve pouquíssimos neurônios. O conhecido exemplo flexor/extensor no qual se utiliza o reflexo patelar. Neste exemplo, um médico golpeia o tendão abaixo do joelho de um paciente com um pequeno martelo e recebe

um chute de volta. Outro exemplo é o reflexo tendinoso, onde a contração do músculo foi provocada por percussão de seu tendão. Estes são exemplos de atividades involuntárias (BARRETO, 2001).

2.6.1.2 Taxias

Direciona o animal a afastar-se ou aproximar-se da fonte emissora do estímulo (OLIVERIA, 2001). Um exemplo é a quimiotaxia, uma ação atrativa ou repulsiva é demonstrada por certas células vivas em relação a outras células ou substâncias que exercem sobre aquelas uma influência química; outro exemplo seria a fototaxia, que retrata o conjunto dos movimentos dos seres vivos induzidos pela luz.

2.6.2 Reativos ou Padrão Fixo de Ação (PFA)

Um comportamento reativo ou seqüencial define uma classe de comportamentos mais complexos e trabalha em um nível muito baixo de abstração sem conhecimento prévio do modelo ambiental. Este comportamento baseia-se no princípio da reatividade dos artefatos, i.e., na suposição de que comportamentos inteligentes possam ser gerados na ausência de uma representação simbólica.

Estes comportamentos são normalmente formados por uma sucessão de respostas estereotipadas como uma reação a um determinado estímulo. O comportamento reativo e o comportamento estereotipado parecem ser inatos aos animais e transmitidos através das gerações como um conjunto de conexões geneticamente pré-determinadas entre os neurônios. Estas conexões foram determinadas provavelmente através de processos evolucionários, não existindo aparentemente nenhum processo de aprendizado envolvido. Pode facilmente demonstrar que o comportamento reativo é modelado através de autômatos de estados finitos e que necessita uma rede neural recorrente, com conexões em ciclo entre os neurônios para ser implementada (ROISENBERG, 1996, 1999).

Intermediário entre a classe de comportamentos estereotipados e instintivos, o Padrão Fixo de Ação (PFA) é um nome originariamente etológico, identificando o estímulo vindo do ambiente, i.e., o ponto final de uma ação serve como ponto inicial para a partida seguinte. O etologista holandês Niko Tinbergen foi um dos primeiros a estudar os PFAs em vertebrados.

De acordo com Cardoso (2002), um exemplo clássico de PFA estudado por Tinbergen é o comportamento de catar ovos pela ganso fêmea. Ao ver o ovo fora do ninho (estímulo

desencadeante), a fêmea inicia um movimento repetido de arrastar o ovo com o bico e o pescoço, Figura 2. No entanto, caso o ovo escape ou se o experimentador retirá-lo, o ganso continua a efetuar os movimentos estereotipados mesmo na ausência do ovo, até chegar ao ninho. Assim, o estímulo interno desencadeou um processo e este se desenrolou até o final, mesmo que o estímulo externo que deu origem ao comportamento não esteja mais presente.

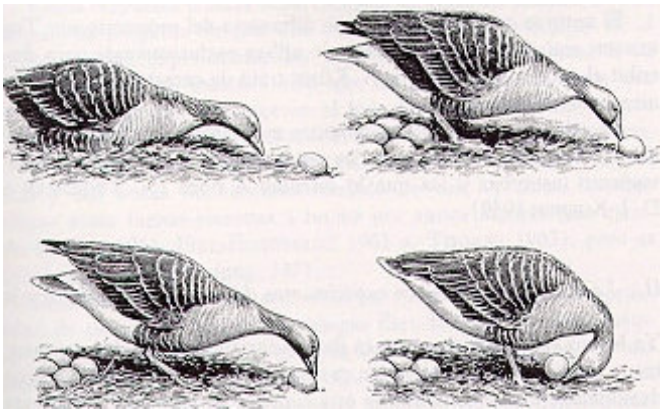

Figura 2 - Reproduzido de Eibl-Eibesfeldt (1979) (PATO.JPG, 2002)

Cardoso (2002) também cita que o ser humano também tem vários movimentos PFA, que se manifestam geralmente no pós-natal imediato. Denominados de Reflexos Comportamentais, um deles é o característico reflexo de prensão das mãos do recém-nascido, que se agarra fortemente ao redor do objeto que o toca. Experiências quantitativas mostram que eles reagem com maior força ao contato dos cabelos e pêlos. Esse PFA evolutivamente surgiu nos primatas para fazer com que o filhote se agarre no pêlo da mãe e não caia durante movimentos fortes. Colocado sobre um fio de varal, o recém-nascido se agarra nele fortemente com as mãos e os pés, ficando suspenso sem a ajuda de um adulto, Figura 3.

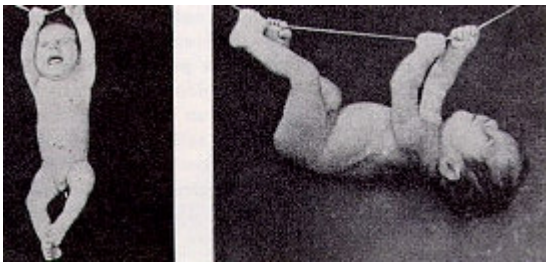

Figura 3 - Reproduzido de Eibl-Eibesfeldt (1979) (GARRAS.JPG, 2002)

2.6.3 Instintivos

Sheldrake afirma que “Os instintos são hábitos de comportamento estereotipados e inatos da espécie e dependem de uma memória coletiva inconsciente, onde se apresenta a

mesma característica holística e propositada da morfogênese" (SHELDRAKE, 1991). Complementa Cardoso que "o instinto é determinado geneticamente (DNA) e possue uma interação complexa com estímulos ambientais" (CARDOSO, 2002).

Para que se desencadeie o comportamento instintivo são necessários: um estímulo externo - ambiente que o cerca - e um estímulo interno - o nível hormonal. Como exemplo, tem-se o caso do cão que urina nas árvores e nos cantos. Este comportamento instintivo representa a delimitação do território pelos cães machos. Este instinto é tão forte que os cães machos tem dificuldade de aprender a urinar em locais específicos; sua tendência é querer "marcar" todo o território.

2.6.4 Deliberativos

Esta classe de comportamentos é definida como aquela que toma uma decisão baseada na presença de um modelo interno do mundo e na capacidade de manipular os objetos simbólicos da representação antes de uma ação efetiva (FRONTEIRA, 1999) (OLIVEIRA, 2001). Em um ambiente dinâmico, as decisões são tomadas via raciocínio lógico baseando-se em reconhecimento de padrões ou manipulação simbólica (WOOLDRIGE, 1994).

Em Oliveira (2001), um artefato robótico executa um esquema básico do controle deliberativo: a seqüência de sentir-modelar-planejar-agir. Informações são capturadas por sensores especializados que as enviam para um modelo do ambiente. Em seguida, um módulo de planejamento decide quais ações tomar (MCFARLAND, 1993).

2.7 Arquitetura PiramidNet e as Classes de Comportamento

Modelar sistemas comportamentais para robôs móveis com base na Natureza, faz o diferencial na resolução de problemas.

Hoje, as aplicações vão desde reconhecimento de padrões a aplicações financeiras. "O sucesso das RNAs faz crer que um computador usando estas redes, como bloco básico, possa resolver problemas que computadores que não usem esta tecnologia são incapazes, ou ao menos teriam muita dificuldade para resolver." (BARRETO, 2001).

A classe de comportamentos estereotipados, com seus reflexos e taxias, parece estar associada a um sistema nervoso simples, podendo ser implementada com RNA Direta. Hogg, Martin e Resnik (1991) utilizaram esse comportamento reflexivo baseando-se nas criaturas de

Braitenberg: Tímido, Indeciso, Paranóico, Obstinado, Inseguro e Dirigido. Eles geralmente têm só um sensor e "cérebro" eletrônico bastante limitado. Já a classe de comportamentos reativos, vem demonstrar o bom desempenho em ambientes mutáveis e imprevisíveis (MÖLLER et al., 1998).

Podendo ser implementada por uma RNA Recorrente, esta classe define-se por possuir várias características como: baixa exigência computacional; rápido tempo de resposta; desempenho robusto em ambientes estáveis; dispõe de um conjunto de comportamentos básicos que lhe permite responder a determinado estímulo e dependente em larga escala dos sensores na aquisição de informação a fim de decidir quais ações imediatas devem ser tomadas para satisfazer a tarefa corrente.

A abordagem deliberativa, próxima da IAS, faz uso de representações do mundo e planeja com base em deliberações sobre este modelo interno (ARKIN, 1999). Através da classe de comportamentos deliberativos, alguns aspectos como mapas topológicos mentais podem ser moldados com uma classe especial de RNA de aprendizagem não-supervisionada, o Mapa Auto-organizável de Kohonen (MK). Logo, MK é apresentado como instrumento para o mapeamento do ambiente robótico. Oliveira (2001) utilizou o MK para efetivar a construção de mapas ambientais do robô móvel pela simplicidade de representação, economia descritiva e plausibilidade biológica. O MK é capaz de representar a topologia do ambiente de navegação do robô móvel (HAFNER, 2000) (OWEN, 1996).

A fusão de classes de comportamentos estereotipados, reativos e deliberativos, possibilitam uma vasta gama de projetos de novas arquiteturas de controle de robôs móveis, ver Figura 4.

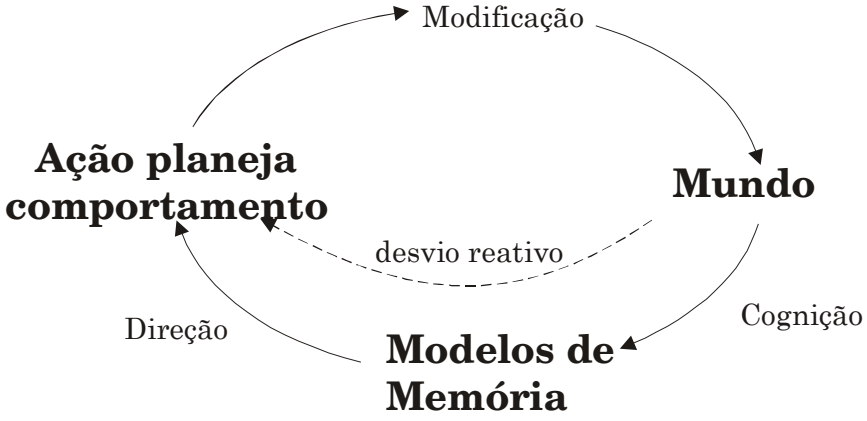

Figura 4 - Ciclo de ação-percepção, Arkin (1999).

A arquitetura hierárquica surge como alternativa ao emprego de abordagens isoladas. Buscando pelo objetivo de atingir alto grau de autonomia no robô móvel, itens imprescindíveis como a reação, a aprendizagem e a deliberação se complementam.

2.8 Considerações Finais

É importante enfatizar que a Natureza serve de inspiração para a área da robótica e inteligência computacional. As características tratadas neste capítulo são determinantes para o entendimento e prosseguimento desta dissertação. A intenção deste capítulo é mostrar que estruturas de redes neurais artificiais modulares heterogêneas e hierárquicas serão o ponto chave na construção do Framework PiramidNet. O que se aspira nos próximos capítulos vem de encontro com conceitos de ferramentas de Engenharia de Software para construir um framework orientado a objetos que supra a necessidade do usuário de construir um aplicativo capaz de interconectar arquiteturas neurais distintas.

3 FRAMEWORKS ORIENTADOS A OBJETOS

3.1 Considerações Iniciais

Este capítulo trata da utilização dos conceitos de Engenharia de Software para construir frameworks orientado a objetos. Utilizando diagramas da Linguagem de Modelagem Unificada (UML), Padrões de Projeto (*Design Patterns*) e da ferramenta de modelagem (Rational Rose 2002), torna-se possível ilustrar diferentes modelos de análise e projetos (LEE, 1997) (BOGGS, 2002). Este capítulo chama a atenção para itens como: a reutilização de códigos, tipos de frameworks e procedimentos para desenvolvimento. Estes itens compõem o conhecimento base na construção de Frameworks.

3.2 Fundamentos

A Engenharia de Software objetiva desenvolver sistemas com melhor qualidade, a custa de menor tempo e esforço. Estes requisitos são necessários na construção de qualquer programa. Usando Padrões de Projeto, constroem-se frameworks que favoreçam o usuário no reuso de análise, projeto e código. Utilizando programas já desenvolvidos e depurados, o tempo de desenvolvimento, de testes e as possibilidades de erros na nova produção tornam-se menores. A ampla reutilização de programas é um dos fatores a favor da abordagem de orientação a objetos.

Boggs aponta três benefícios primários da elaboração e utilização de frameworks :

> "The team does not need to spend unnecessary time modeling elements that exist. The focus of the modeling effort in on what's unique to a project, not reinventing existing components (although reusing then is fine!). A framework helps provide consistency across project. […]Using framework in different projects ensure that both teams are building from the same foundation" (BOGGS, 2002).

Em seu livro "*Design Patterns for Object-Oriented Software Development*", Pree (1995) enfatiza o uso de padrões para a construção de uma base forte. Neste livro, o autor cita que "*Design Patterns recently emerged as a glimmer of hope on the horizon for supporting the development and reuse of frameworks*".

Larman (2000) cita que se pode alcançar três estados: análise, projeto e código. A fase de análise dá ênfase a itens como: compreensão dos requisitos, dos conceitos e das operações.

Dependendo da similaridade com outros projetos, estes itens na fase de análise possam ser reaproveitados. A fase do projeto utiliza soluções baseadas no paradigma de orientação a objetos, diagramas que ajudam o usuário entender melhor o domínio e relacionamento das classes. Um trecho da tese de Silva (2000), citado abaixo, afirma a idéia de utilizar a reusabilidade de classes.

> Um programador que usa linguagem C por exemplo, utiliza-se de bibliotecas de funções. Isto se classifica como reutilização de rotinas [MEY 88]. A reutilização no nível de módulo corresponde a um nível de granularidade superior à reutilização de rotinas. A reutilização de classes em orientação a objetos, segundo Meyer, corresponde à reutilização de módulo. Quando uma classe é reutilizada, um conjunto de rotinas é reutilizado (métodos), bem como uma estrutura de dados (atributos). Reutilizar classes, portanto, tende a ser mais eficiente que reutilizar rotinas e de mais alto nível de granularidade (uma classe de objetos tende a ser um artefato de software mais complexo que uma rotina isolada) (SILVA, 2000).

Silva (2000) enfatiza o desenvolvimento e especificações de projeto baseadas no paradigma de orientação a objetos. Nota-se que o reuso de classes interligadas é 'apoiada' por frameworks. Quando frameworks é confrontado à reutilização de classes isoladas ou de bibliotecas, percebe-se um grande aumento de produtividade no desenvolvimento de sistemas.

3.3 Definição

Defini-se frameworks como um conjunto de classes-base (esqueleto) sobre o qual uma ferramenta é construída. Nesta dissertação, este esqueleto disponibilizará classes para construir uma ferramenta que permita desenvolver e trabalhar com vários modelos heterogêneos de RNAs interligadas. Toda a ferramenta criada a partir de um framework tem a responsabilidade de interpretar a necessidade do usuário que se faz presente através da administração de itens como: menus, formulários, segurança e outros.

Larman (2000) define frameworks como um subsistema extensível para um conjunto de serviços relacionados. Também é possível ser classificado como um conjunto coeso de classes que colaboram para fornecer serviços para o núcleo invariante de um sistema lógico. Contém classes concretas e abstratas que definem interfaces a serem seguidas, interações entre objetos das quais a aplicação deve participar como também outros invariantes.

Silva (2000) define frameworks como um conjunto de estruturas de classes que constituem implementações incompletas que quando estendidas, permitem produzir diferentes artefatos de um "sistema".

Outra definição é indicada por Wirfs-Brock (1991) *apud* Silva (2000) que diz:

> Um framework é um esqueleto de implementação de uma aplicação ou de um subsistema de aplicação, em um domínio de problema particular. É composto de classes abstratas, concretas e provê um modelo de interação ou colaboração entre as instâncias de classes definidas pelo framework. Um framework é utilizado através de configuração ou conexão de classes concretas e derivação de novas classes concretas a partir das classes abstratas do framework.

3.4 Projeto

A fase do projeto para construção de frameworks visa produzir uma arquitetura bem estruturada através da padronização de classes com fácil integração, portabilidade e consistência. De forma que:

a) A padronização refere-se ao uso de padrões de projetos na construção de classes, usando das idéias de outras aplicações que deram certo;
b) A integração e portabilidade devem ser tratadas como uma das grandes prioridades, devendo ser planejada já na concepção de suas classes;
c) A consistência refere-se construção homogênea e compatibilidade entre classes.

A adaptação de um desenvolvedor ao ambiente de frameworks, deve ser facilitada através da existência de uma estrutura orientada a objetos bem projetada. Pressman (1995), Yourdon (1991) e Silva (2000) citam abaixo alguns requisitos imprescindíveis para obtenção de frameworks bem projetados.

A modularidade - alta coesão e baixo acoplamento - garante-se pela definição de interfaces limpas e consistentes, como também o encapsulamento da implementação de métodos. Todos os objetos da mesma linhagem devem ter a mesma interface e que quando pública para as classes no framework, possam ser tão simples quanto possível, pois é importante mantê-la simples para alcançar a funcionalidade desejada. Assim, a qualidade é atingida através do encapsulamento dos aspectos comportamentais de frameworks.

Quanto a reusabilidade, frameworks devem fornecer um grau muito elevado de reutilização - muito mais do que classes individuais, porque além de herdar as características inerentes às classes-pai, herda-se também o comportamento e o relacionamento destas. Logo, frameworks é definido como uma arquitetura de um sistema reutilizável em termos de contratos de colaboração entre classes abstratas e um conjunto de pontos adaptáveis, ou *hot spots. Hot spots* consistem de pontos adaptáveis que precisam ser especializados em uma aplicação. Os contratos de colaboração definem as regras que tal especialização deve

obedecer. Um ponto adaptável é implementado a partir de padrões de projeto e tais padrões consistem na principal documentação de frameworks (GAMMA, 1994). O projeto dos pontos adaptáveis é a tarefa central na construção de frameworks. Partindo deste escopo, é possível gerar novas aplicações usando componentes genéricos. A utilização de um padrão permite explorar as similaridades para a obtenção de soluções a outras classes de problemas. Isto garante maior produtividade de programação, grande economia em um nível de qualidade satisfatória, performance e confiabilidade do programa, significando menos esforço e menor risco de ocorrência de erros.

A capacidade de estender funcionalidades a partir de comportamentos, é um recurso dos frameworks. Com este recurso, geram-se subclasses das classes existentes do framework. Isto alcança o comportamento especializado que o desenvolvedor poderia requerer em uma nova ferramenta.

Frameworks oferecem robustez para suas ferramentas. Robustez é a capacidade de responder a eventos externos, mantendo o controle da execução da aplicação. A principal característica de frameworks é a obtenção de um nível de funcionamento do sistema que suporte situações que não foram previstas na especificação dos requisitos. A ferramenta, na pior das hipóteses, deve continuar ativa em situações anormais.

Finalmente, frameworks oferecem suporte a compatibilidade. Isto corresponde à facilidade na combinação de produtos do sistema. A impossibilidade da interação com outros produtos resultam na não escolha ou no abandono de sua utilização.

Considerando os requisitos anteriormente citados, o desenvolvedor de frameworks adquire o conhecimento sobre o domínio do problema que irá trabalhar, no caso Redes Neurais Artificiais. Além disso, quando se desenvolve um sistema com padronização UML, deve-se permitir a reusabilidade de código e a programação baseada em classes com interfaces bem definidas que garantam uma integração modular.

3.4.1 Elementos do Projeto

Para ter um projeto de frameworks, necessita-se dos seguintes itens: desenvolvedor, definição do domínio e futuras aplicações.

Desenvolvedor**:** é a peça chave do projeto, sendo o encarregado de manter, organizar e escolher quais classes irão compor a estrutura do framework.

Definição do domínio**:** de acordo com Sauvé (1999) e Wirfs-Brock (1991), é de suma importância uma análise sobre que domínio irá trabalhar. Utilizando-se da reutilização, através da generalização das classes que deve ocorrer no início de sua concepção, tem-se a necessidade de gerar um conjunto de aplicações com bases parecidas. Isto leva o desenvolvedor a decidir que tipo de ferramenta é possível gerar com a estrutura do framework. O processo de generalização trabalha de forma a percorrer elementos diferentes, eliminando diferenças, selecionando e agrupando os elementos similares que restaram nas classes.

Futuras aplicações**:** serve para que diferentes ferramentas possam ser criadas enfocando uma mesma base de estrutura do framework. Utilizando uma estrutura com ligações entre classes pré-definidas, diminui em muito o tempo e esforço de criar uma nova ferramenta, pois aparenta ser uma produção em série. Vale ressaltar que, mesmo construindo uma estrutura de framework com inúmeras classes, ainda sim, não há possibilidade do framework conter todas as classes que são pertinentes a uma ferramenta específica. Algumas ferramentas reservam o direito a características únicas. Logo, é de suma importância definir quais são pontos de adaptação desta estrutura.

3.5 Metodologias de Desenvolvimento de Frameworks

Para que se desenvolva frameworks, é importante conhecer as principais características de algumas metodologias. A evolução no desenvolvimento de frameworks é feita através de constantes mudanças na sua estrutura de classes. Esta apuração cíclica ocasiona ajustes na estrutura de classes, tomando como parâmetros os aspectos de generalidade e extensibilidade, implicando no manuseio de uma grande quantidade de informações.

> Pode-se afirmar que o desenvolvimento de um framework é mais complexo que o desenvolvimento de aplicações específicas do mesmo domínio, devido: a necessidade de considerar os requisitos de um conjunto significativo de aplicações, de modo a dotar a estrutura de classes do framework de generalidade, em relação ao domínio tratado; a necessidade de ciclos de evolução voltados a dotar a estrutura de classes do framework de alterabilidade e extensibilidade (SILVA, 2000).

Silva (2000) cita brevemente as principais características de três metodologias voltadas ao desenvolvimento de frameworks: Projeto Dirigido por Exemplo (JOHNSON, 1993), Projeto Dirigido por Pontos Adaptáveis (PREE, 1994) e o Projeto da Empresa Taligent (IBM) (TALIGENT, 1994 *apud* Silva 2000). Para entendimento do leitor e fundamentação à construção do framework proposto nesta dissertação, usaram-se algumas características – descritas abaixo - pertencentes a cada uma das três metodologias citadas acima, pois elas se completam.

Projeto Dirigido por Exemplo

Para se construir frameworks, é necessário entender o que vai ser gerado no seu domínio através da análise adequada de exemplos particulares. Mesmo não usando técnicas de modelagem detalhadas como as metodologias de análise e projeto orientadas a objetos, é importante aprender a analisar os aspectos semelhantes de diferentes aplicações que já foram feitas e assimilar as abstrações já conhecidas. Deve-se extrair diretivas básicas (abstrações), através de uma forma *bottom-up* compondo assim o domínio de aplicação. Com o resultado colhido da generalização, criam-se classes abstratas que agrupam os aspectos semelhantes adquiridos, dando às classes concretas de nível hierárquico inferior a especialização pela flexibilização para satisfazer cada aplicação particular (JOHNSON, 1993). Nas classes abstratas são guardadas as características gerais do domínio de aplicação. No ambiente de frameworks, um método de uma classe abstrata pode ficar propositalmente incompleto para que sua definição seja acabada na geração de uma aplicação. Apenas os atributos a serem utilizados por todas as aplicações de um domínio são incluídos em classes abstratas (SILVA, 2000).

Projeto Dirigido por Pontos Adaptáveis

Pelo autor Pree (1994), conforme Silva (2000) e Schmid (1997), uma aplicação orientada a objetos é completamente definida. Frameworks, ao contrário possui partes propositalmente indefinidas - o que lhe dá a capacidade de ser flexível e se moldar a diferentes aplicações se implementado a partir de padrões de projeto (GAMMA, 1994). Os pontos adaptáveis, partes extensíveis, são a essência da metodologia em uma estrutura de classes flexíveis em um domínio e a partir disto, construir o framework.

Projeto da Empresa Taligent (IBM)

A metodologia proposta pela empresa Taligent (empresa já extinta) difere das anteriores pelo conjunto de princípios que norteiam o desenvolvimento de frameworks.

Primeiramente, a visão de desenvolver frameworks que cubra as características e necessidades de um domínio é substituída pela visão de produzir um conjunto de frameworks estruturalmente menores, mais simples e que usados conjuntamente, darão origem às ferramentas. A justificativa para isto é que pequenos frameworks são mais flexíveis e podem ser reutilizados mais freqüentemente. Assim, a ênfase passa a ser o desenvolvimento de frameworks pequenos e direcionados a aspectos específicos do domínio.

Um outro aspecto das linhas mestras do processo de desenvolvimento é tornar o uso de frameworks o mais simples possível, através da minimização da quantidade de código que o usuário deve:

a) Disponibilizar implementações (classes) concretas que possam ser usadas diretamente;
b) Minimizar o número de classes que devem ser criadas;
c) Minimizar o número de métodos que devem ser sobrepostos.

A metodologia propõe a seqüência de quatro passos, apresentados a seguir.

<u>Identificar e caracterizar o domínio do problema:</u>

a) Analisar o domínio e identificar os frameworks necessários (para o desenvolvimento de ferramentas), o que pode incluir frameworks desenvolvidos e a desenvolver;
b) Examinar soluções existentes;
c) Identificar as abstrações principais;
d) Identificar o limite de responsabilidades do framework em desenvolvimento (considerando que esteja no processo de desenvolvimento de um framework específico);
e) Validar estas informações com especialistas do domínio.

Definir a arquitetura e o projeto:

a) Refinar a estrutura de classes obtida no passo anterior, centrando atenção em como os usuários interagem com o framework,
   - que classes o usuário instancia ?
   - que classes o usuário produz (e que métodos sobrepõe) ?
   - que métodos o usuário chama ?

b) Aperfeiçoar o projeto com o uso de Padrões de Projeto;

c) Validar estas informações com especialistas do domínio.

Implementar frameworks:

a) Implementar as classes principais;

b) Testar o framework (gerando aplicações);

c) Solicitar o procedimento de teste a terceiros;

d) Iterar para refinar o projeto.

Desdobrar frameworks:

a) Providenciar documentação na forma de diagramas, receitas (como usar o framework para desenvolver determinada ferramenta) e exemplos de ferramentas simples (incluindo o código da implementação, que complementa o framework);

b) Manter e atualizar o framework, seguindo as regras abaixo,
   - corrigir erros imediatamente;
   - adicionar novas características ocasionalmente;
   - mudar interfaces tão infreqüentemente quanto possível, "é melhor adicionar novas classes que alterar a hierarquia de classes existente, bem como adicionar novos métodos que alterar ou remover métodos existentes".

3.6 Desenvolvimento

O processo de desenvolvimento de frameworks está assentado em bases sólidas que definem sua estrutura através de tipos de padrões de projeto. A seguir, documenta-se cada processo.

3.6.1 Estrutura

Johnson (1997) define o desenvolvimento de frameworks pela sua estrutura. A reutilização de uma estrutura de frameworks deve ser feita através de suas classes e suas instâncias. A aplicação de padrões que já foram testados e observados em outros sistemas, torna-se importante para um claro desenvolvimento da estrutura. A função de frameworks é permitir que através de uma idéia principal de uma determinada família de aplicações, o desenvolvedor tenha a possibilidade de começar a criar uma ferramenta a partir de um nível superior na implementação da arquitetura do framework.

Conforme Silva (2000) e Larman (2000), a repetitiva evolução da estrutura de classes, faz-se necessária para o desenvolvimento de frameworks. Usando-se da evolução iterativa, as adaptações na estrutura iniciam nas classes e se estendem até os pontos de generalização e flexibilização. Este desenvolvimento cíclico baseia-se no aumento e no refinamento sucessivo de um sistema, ou seja, através de vários ciclos de desenvolvimento de análise, de projeto, de implementação e de testes.

3.6.2 Padrões de Projetos (*Design Patterns*)

Padrão de Projeto refere-se a uma solução que foi desenvolvida e aperfeiçoada ao longo do tempo (GAMMA, 2000). Citando a frase de Christopher Alexander (arquiteto) *apud* Gamma (2000), ele afirma: "Cada padrão descreve um problema no nosso ambiente e o núcleo da sua solução, de tal forma que você possa usar esta solução mais de um milhão de vezes sem nunca fazê-lo da mesma maneira". Larman (2000) complementa: "Os padrões fornecem um conjunto de estilos explicáveis, através dos quais os sistemas orientados a objetos bem-projetados podem ser construídos". Pree (1994) diz: "*Instead of having to choose from an almost infinite number of possible combinations of actions, patterns allow the solution of problems by providing time-tested combinations that work*".

Cada padrão fornece soluções que atuam de forma simples onde existam problemas específicos. Utilizando-se de modelagens e recodificações, o padrão captura soluções simples e facilmente aplicáveis sem precisar exigir de recursos adicionais da linguagem. Gamma (2000) afirma: "Um padrão nomeia, abstrai e identifica os aspectos-chave de uma estrutura de projeto comum para torná-la útil à criação de um projeto orientado a objetos reutilizável". Larman (2000) também cita: "[...] os padrões comunicam os idiomas e as soluções extraídas 'do melhor da prática' que os projetistas experientes aplicam na criação de sistemas".

Usando os padrões documentados pelo GoF - Gangue dos Quatro (GAMMA, 2000), GRASP – Padrões Gerais para Atribuição de Responsabilidades (LARMAN, 2000) e profissionais que desenvolvem aplicações, pode-se construir um projeto tornando-o mais flexível, modular, reutilizável e compreensível.

Gamma esclarece que "[...] o padrão de projeto identifica as classes e instâncias participantes, seus papéis, colaborações e a distribuição de responsabilidades [...]" (GAMMA, 2000). Larman também conclui dizendo: "[...] a atribuição de responsabilidades, durante o projeto orientado a objetos, pode ser feita com base no uso de padrões." (LARMAN, 2000).

Nesta dissertação, alguns padrões foram usados para construir uma estrutura reutilizável. A idéia de usar padrões de projeto é reutilizar projetos bem-sucedidos ao basear novos projetos na experiência anterior, não tendo que resolver cada problema a partir do início.

Gamma (2000) apresenta cada padrão de projeto, resumidamente desta forma:

- O nome do padrão: contido em até duas palavras, faz uma explicação breve do problema proposto, suas soluções e conseqüências;
- O problema: descreve quando aplicar o padrão;
- A solução: descreve os elementos que compõe o projeto, seus relacionamentos, suas responsabilidades e colaborações, fornecendo uma descrição abstrata de um problema de projeto;
- As conseqüências: são os resultados e análises das vantagens e desvantagens da aplicação.

Os padrões de projeto são classificados segundo dois critérios, ver Tabela 1:

- Finalidade: reflete o que um padrão faz, podendo ter a finalidade de criação, estrutural ou comportamental;
- Escopo: especifica se o padrão poder ser aplicado primariamente a classes ou a objetos,
- Classe: lida com os relacionamentos entre classes e subclasses através de herança (estáticos);
- Objeto: lida com relacionamentos entre objetos que podem ser mudados em tempo de execução (dinâmicos).

Tabela 1 – A organização dos Padrões de Projeto

		Propósito		
		De criação	Estrutural	Comportamental
Escopo	Classe	Factory Method	Adapter (class)	Interpreter Template Method
	Objeto	Abstract Factory Builder Prototype Singleton	Adapter (object) Bridge Decorator Façade Flyweight Proxy	Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor

Fonte: Gamma (2000).

O uso de padrões é viável para solucionar diversos problemas. No texto abaixo, Gamma (2000) relata alguns problemas:

a) Procurando por objetos apropriados: as metodologias de projeto orientadas a objetos fornecem muitas abordagens diferentes de decomposição. Padrões como o Composite, o Strategy e o State são raramente encontrados durante a análise ou mesmo durante os passos iniciais do projeto; eles são posteriormente descobertos durante o processo de tornar um projeto mais flexível e reutilizável;

b) Determinando a granulidade dos objetos: refere-se ao seu tamanho e número. A principal solução deste problema é quebrar um grande objeto em vários objetos menores. A criação e o cuidado de um objeto também são muito importantes. Neste subtópico encaixam-se os padrões: Façade, Flyweight, Abstract Factory, Builder, Visitor e Command;

c) Especificando interface de objetos: o conjunto de todas as assinaturas definidas pelas operações de um objeto denomina-se interface. As interfaces são fundamentais em sistemas orientado a objetos, pois os objetos são conhecidos através de suas interfaces. Os padrões ajudam a definir as interfaces pela identificação de seus elementos-chave e pelos tipos de dados que são enviados – padrão Memento. Também especificam os relacionamentos entre interfaces – padrão Decorator, Proxy e Visitor;

d) Especificando implementações de objetos: uma implementação de objeto é definida por sua classe. Novas classes podem ser definidas em termos de classes existentes, usando-se a herança de classe. Uma classe abstrata, cuja característica é definir uma interface, não é instanciada. Subclasses podem refinar e redefinir os comportamentos das suas classes ancestrais;

e) Herança de classe *versus* herança de interface: A herança de classe define a implementação de um objeto em termos da implementação de outro objeto. A herança de interface (subtipificação) descreve quando um objeto é usado no lugar de outro. Embora algumas linguagens não apóiem a distinção, muitos programadores fazem isto na prática. Programadores em C++ manipulam objetos através de tipos definidos por classes abstratas. Esta distinção é importante, muitos padrões dependem desta distinção, cito: Chain of Responsability, Composite, Component, Command, Observer, State e Strategy;

f) Programando para uma interface, não para uma implementação: quando é feita uma herança, definem-se famílias de objetos com interfaces idênticas. Isto implica que uma subclasse meramente acrescenta ou substitui operações da classe-mãe e não oculta operações dela. Todas as subclasses podem responder a solicitações na interface desta classe abstrata. Seus benefícios incluem: 1) não há necessidade de conhecimento dos tipos específicos que eles usam; 2) não há necessidade de conhecimento das classes que implementam estes objetos. Estes benefícios reduzem as dependências de implementações entre subsistemas;

g) Herança *versus* composição: A reutilização por meio de subclasses é chamada reutilização de caixa branca (com visibilidade). A composição de um objeto é uma alternativa a herança de classes. Com a composição, a reutilização é chamada reutilização de caixa preta (sem visibilidade). Neste caso, a herança viola a encapsulação. Com a composição, os objetos são acessados através de suas interfaces. A implementação de um objeto será escrita em termos de interface (anteriormente citado) e ajuda a manter cada classe encapsulada;

h) Delegação: um objeto receptor delega operações para seu delegado. Isto torna a composição tão poderosa como a herança. A principal vantagem da delegação é que ela torna fácil compor vários comportamentos em tempo de execução e mudar a forma como são compostos. Diversos padrões usam a delegação: o State, o Strategy, o Visitor, o Mediador, o Chain of Responsability e o Bridge;

i) Tipos parametrizados: conhecido como templates em C++, esta técnica é usada para reutilizar funcionalidades;

j) Projetando para mudanças: faz-se necessária à antecipação de novos requisitos e mudanças nos requisitos existentes, do contrário está sujeito ao risco de grandes formulações no futuro.

A seleção de um padrão de projeto é difícil. Cada padrão oferece uma resposta a um projeto particular. Gamma (2000) apresenta diversas abordagens para descobrir o padrão correto:

a) Considerar como os padrões de projeto solucionam seus problemas de projetos;
b) Analisar os padrões relevantes ao seu problema;
c) Estudar como os padrões se inter-relacionam;
d) Estudar os padrões de finalidades semelhantes;
e) Examinar uma causa de reformulação de projeto;
f) Considerar o que deveria ser viável no seu projeto.

Depois de escolhido o padrão, é preciso saber como usá-lo:

a) Prestar atenção à aplicabilidade e conseqüência;
b) Compreender as classes, os objetos e seus relacionamentos;
c) Procurar por exemplos de códigos;
d) Escolher nomes com sentido para as classes, objetos e seus relacionamentos;
e) Definir toda a classe;
f) Implementar as operações para suportar as responsabilidades e colaborações presentes no padrão escolhido.

Por último e não menos importante, o surgimento do termo metapadrões. Introduzido por Pree (1994), metapadrões é um conceito que se refere a um conjunto de padrões de projetos que descrevem como construir frameworks independentes para um domínio específico. A idéia de metapadrões surge para complementar a usabilidade dos padrões de projeto.

3.6.3 Tipos de Frameworks

Para desenvolver frameworks, Sauvé (1999) indica dois critérios simples para um bom entendimento: deve-se primeiramente saber **como o framework é usado** e depois **onde o framework é usado**.

Indo pelo caminho de **como o framework é usado**, existem vários tipos de frameworks. Fayad (1997) classifica-os em caixa-branca, caixa-preta e Sauvé (1999) complementa a classificação com o tipo híbrido. A seguir, detalha-se os termos deste tipo de classificação.

Focado na Herança, Caixa Branca ou Direcionado a Arquitetura: os métodos são definidos em interfaces ou classes abstratas e devem ser implementados por uma aplicação. Através da modelagem orientada a objetos, a ligação dinâmica e a herança são mecanismos que trabalham juntos. Assim, a funcionalidade cuida da sobrecarga de métodos pré-definidos, herança de classes e subclasses.

Focado na Composição, Caixa Preta ou Direcionado a Dados: utiliza-se da funcionalidade já existente no framework, sem a possibilidade da arquitetura ser vista ou alterada. A extensão é feita a partir de interfaces fornecidas e definidas para os componentes. As composições e instanciações definem as particularidades desta aplicação. Através de padrões definidos no projeto, os recursos existentes são reutilizados e estendidos por meio da **definição** e **integração** de componentes que se ajustam a uma interface específica. No fim, o framework se torna orientado a componentes.

Híbrida: resultado originário da união de duas classificações: caixa-branca e caixa-preta. Na maioria dos frameworks, sua classificação tem sido Focada na Herança com algumas funcionalidades prontas Focadas na Composição.

Segundo Fayad (1997) e Sauvé (1999), outro item de classificação que deve ser entendido para conseguir diferenciar frameworks é sabendo seu argumento de utilização e que responde a pergunta: **onde o framework é usado?** Abaixo, apresentam-se três modelos de framework.

Framework de Suporte: dedica-se a assuntos que envolvam serviços em nível de sistema operacional e não de aplicação, por exemplo: comunicação, acesso a arquivos, computação distribuída, interfaces gráficas e linguagens de programação.

Framework de Aplicação: trata de assuntos que envolvam o projeto da construção de aplicações específicas. Encapsula conhecimento aplicável a uma vasta gama de aplicações, por exemplo: construção de interface gráfica para usuário, telecomunicações, finanças, produção e educação, geoprocessamento etc.

Framework de Integração: são responsáveis por integrar aplicações distribuídas e componentes em uma mesma arquitetura. Encapsula conhecimento aplicável a aplicações pertencendo a um domínio particular de problema, por exemplo: ORB, CORBA e o RMI.

Para a construção do Framework PiramidNet, escolheu-se a classificação do tipo híbrida focando um framework de aplicação.

3.7 Framework de Aplicação

De acordo com Meyer (1988), a criação de módulos em pequenas quantidades e o uso do princípio da ocultação de informação, destacam a vertente modularidade, já comentada anteriormente. Também é possível integrar os objetos que foram definidos na estrutura do framework, criando futuras ferramentas (WIRFS-BROCK, 1990, 1991) (JOHNSON, 1993) (SILVA, 2000). Valendo-se do modelo de colaboração pela união das classes e comunicação entre os objetos, geram-se soluções a problemas semelhantes de um nicho através da forma de reuso pela análise, projeto e código. Seu conjunto de classes é flexível e extensível para que seja permitida a construção de várias aplicações especificando apenas suas particularidades (LARMAN, 2000).

Com o Framework de Aplicação, Figura 5, verifica-se todo o aproveitamento do sistema promovendo uma diminuição do esforço necessário para produzir a ferramenta. Na área hachurada, existe uma estrutura de framework pré-definida com seus objetos e conexões. Abaixo da área hachurada, existe um complemento de classes que, quando unida à estrutura do framework, caracteriza uma nova ferramenta. Silva (2000) cita que através da infra-estrutura e do projeto que os frameworks oferecem, reduz e muito a quantidade de código a ser desenvolvida, testada e depurada. A infra-estrutura de projeto, disponibilizada ao desenvolvedor da aplicação somada com interconexões preestabelecidas, definem a arquitetura da aplicação. A flexibilidade em moldar o framework a uma necessidade específica, faz com que o desenvolvedor possa escrever um código de programação que possibilite estender ou particularizar o comportamento do framework.

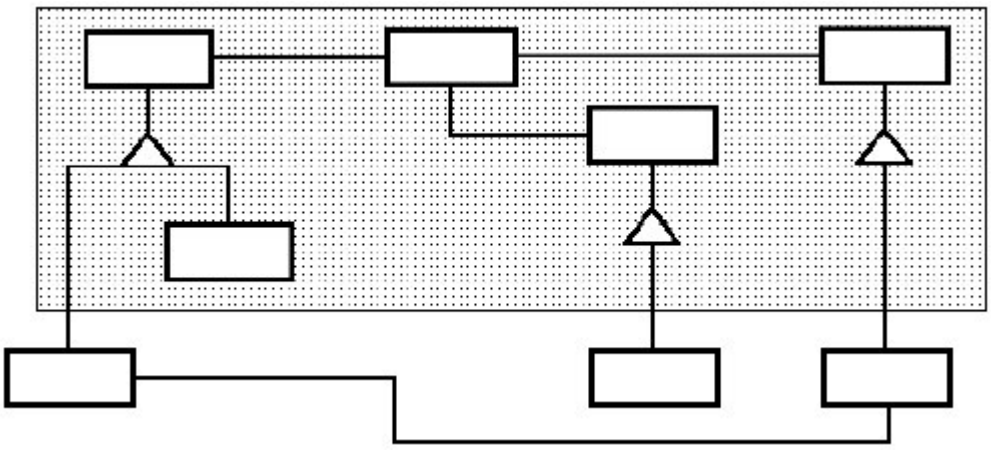

Figura 5 - Aplicação desenvolvida reutilizando um framework – exemplo genérico, extraído de (SILVA, 2000)

“[...] um framework é utilizado através de configuração ou conexão de classes concretas e derivação de novas classes concretas a partir das classes abstratas do framework" (WIRFS-BROCK, 1991 *apud* SILVA, 2000).

Os frameworks são estruturas de classes inter-relacionadas, que permitem não apenas a reutilização de classes, mas minimizam o esforço para o desenvolvimento de aplicações - por conterem o protocolo de controle da aplicação (arquitetura). Os frameworks invertem a óptica do reuso de classes, da abordagem *bottom-up* para a abordagem *top-down*: o desenvolvimento inicia com o entendimento do sistema contido no projeto do framework e segue no detalhamento das particularidades da aplicação específica, o que é definido pelo usuário do framework (TALIGENT, 1995) *apud* (SILVA, 2000).

3.7.1 Vantagens

Existem várias vantagens para usar um Framework de Aplicação, por exemplo:

a) Diminui o montante de linha de código comparado ao que o usuário fará a partir do zero e que depois será testada e depurada;
b) Permite uma infraestrutura com um alto nível de portabilidade para o projeto, disponibilizado ao desenvolvedor da aplicação;
c) Estabelece uma arquitetura da aplicação, já definida pelas interconexões;
d) Molda a aplicação em de áreas onde é possível estender um código escrito, a uma necessidade específica tornando-a então particular a outras aplicações;
e) Agrupam diferentes quantidades de classes, mais ou menos complexas;
f) Encapsula o projeto genérico completo para um domínio de aplicação.

3.8 Considerações Finais

As definições e características aqui abordadas são de extrema importância no entendimento e construção de frameworks. No capítulo que se segue, apresenta-se um protótipo para o usuário com algumas definições de classes para modelar as necessidades do desenvolvedor.

4 FRAMEWORK PIRAMIDNET E FERRAMENTA PIRAMIDNET

4.1 Considerações Iniciais

O capítulo 2 apresentou as idéias do Projeto PiramidNet, enfatizou a utilização de vários modelos de redes neurais artificiais e a influência destes no comportamento robótico. O capítulo 3 citou as características necessárias ao bom entendimento e construção de frameworks com a finalidade de trabalhar com modelos de RNAs. Neste capítulo, apresenta-se a estrutura do Framework e sua Ferramenta (estendida a partir do Framework).

Para validar o Framework proposto nesta dissertação, torna-se necessário por em prática os conceitos até aqui apresentados. Este capítulo abrange a estrutura do Framework PiramidNet e as telas da Ferramenta PiramidNet. O Framework e a Ferramenta já estão sendo utilizados no laboratório L3C para construção de estruturas "nervosas" que serão hospedadas no artefato robótico.

4.2 Definições

Framework PiramidNet é um tipo de framework de aplicação. Tem como objetivo principal dar suporte à criação de ferramentas de interface gráfica que utilizam estruturas de RNAs heterogêneas e interconectadas. Este framework foi criado objetivando auxiliar o trabalho de pesquisadores e alunos que estão envolvidos com o Projeto PiramidNet e com quaisquer projetos que usem RNAs. O Framework PiramidNet tem uma estrutura que possibilita o desenvolvimento continuado de várias classes e estruturas de RNAs.

Ferramenta PiramidNet é uma ferramenta que foi gerada a partir da estrutura pré-estabelecida do Framework PiramidNet. O objetivo principal desta Ferramenta é modelar e interconectar sensores, atuadores e outras estruturas de redes neurais heterogêneas em um ambiente gráfico. A Ferramenta suporta o recurso de transformação de modelagem gráfica em programa. Este programa gerado (código-fonte) poderá ser compilado e transferido para um robô móvel. Sendo extensíveis, o Framework e a Ferramenta contam primeiramente com duas arquiteturas iniciais: rede direta e rede recorrente. Esta Ferramenta favorece a construção de

inúmeros projetos de RNAs que podem ser úteis em diferentes áreas de pesquisa, cito: reconhecimento de padrões, reconhecimento de faces e segurança.

4.3 Recursos e Linguagem

Tanto o Framework como a Ferramenta, contam com os seguintes recursos:

a) Possui interface gráfica;
b) Possui área gráfica para inserção/exclusão de objetos (sensores, atuadores, RNAs e conexões);
c) Possui modelos heterogêneos de RNAs;
d) Recupera modelos salvos de projetos;
e) Adiciona modelos salvos ao modelo de trabalho corrente;
f) Converte automaticamente o modelo gráfico estrutural para um formato, ANSI C, que seja exportado para qualquer sistema alvo, inclusive para os robôs Khepera e Lego;
g) Gera arquivo ou impressão do modelo gráfico do projeto.

Este Framework foi escrito em linguagem de programação C++ por ser uma linguagem estável e orientada a objetos (DEITEL, 2001). A programação orientada a objetos faz uso do modelo de classes e elementos. O entendimento do problema proposto no projeto principal é facilitado através de vários itens como: a) a flexibilidade e abstração de classes para construir novos modelos de redes neurais artificiais; b) a facilidade de estabelecer ligações entre as classes; c) a possibilidade de gerar interfaces padronizadas e objetivas; d) o encapsulamento da implementação de métodos; e) a reusabilidade de código; f) a extensibilidade através da inserção de novas classes e a robustez no tratamento de exceções. Para a criação do ambiente e interface gráfica, optou-se pela ferramenta de desenvolvimento Borland C++ Builder versão 6.0 (DIAS, 2000) (MATEUS, 2000) (SCHILDT, 2001).

4.4 Modelagem Visual por Diagramas

O objetivo da UML é descrever, através de uma modelagem visual, qualquer tipo de sistema que possua uma estrutura estática e um comportamento dinâmico, em termos de diagramas orientado a objetos. A notação UML originou-se de esforços colaborativos de

Grady Booch, James Rumbaugh, Ivar Jacobson, Rebecca Wirfs-Brock, Peter Yourdon e outros. De acordo com Boggs (2002), a consolidação dos métodos que originaram a UML surgiu em 1993. Boggs cita que "*Visual modeling is the process of talking the information form the model and displaying it graphically using a standard set of graphical elements. A standard is vital to realizing one of the benefits of visual modeling: communication*".

A UML permite desenvolver diferentes tipos de diagramas visuais que representam aspectos ímpares de um sistema. Dentre os diagramas que compõem a UML, esta dissertação se resume a alguns que são necessários ao bom entendimento do framework. Os diagramas são: de casos de uso, de classe, de colaboração, de atividade e o de componente.

Com suporte a modelos estáticos, comportamentos dinâmicos e funcionais, a UML faz seu diferencial (RUMBAUGH, 1991). A modelagem estática é suportada pelo diagrama de classes e de objetos, que consiste nas classes e seus relacionamentos. Os relacionamentos podem ser de associações, heranças (generalizações), dependências ou refinamentos. Os comportamentos dinâmicos são suportados pelos diagramas de colaboração e atividade. A modelagem funcional é suportada pelo diagrama de componente.

4.4.1 Diagramas de Casos de Uso

Uma definição de diagrama de casos de uso foi extraída de Jacobson (1992) (*apud* LARMAN, 2000) que diz: "Um caso de uso é um documento narrativo que descreve a seqüência de eventos de um ator (um agente externo) que usa um sistema para completar um processo". Os atores representam o papel de uma entidade externa ao sistema como um usuário, um hardware ou outro sistema que possa interagir com o sistema modelado. Os atores iniciam a comunicação com o sistema através dos casos de uso. Um caso de uso representa uma seqüência de ações executadas pelo sistema.

Um ator é conectado a um ou mais casos de usos através de associações. Caracteristicamente, o ator instiga o sistema com interações, eventos de entrada ou recebe algo do mesmo. Pode-se existir relacionamentos de generalização com casos de uso e seus atores. Estes relacionamentos definem um comportamento comum de herança (superclasses especializadas em subclasses).

Ribeiro (2000) cita que quando um caso de uso é implementado, a responsabilidade de cada passo da execução deve ser associada às classes que participam da colaboração, tipicamente especificando as operações necessárias dentro destas classes juntamente com a

definição de como elas irão interagir. Utilizando-se de um cenário, é possível mostrar o caminho específico de cada ação. Um cenário é uma instância de um caso de uso ou de uma colaboração. Quando visto ao nível de um caso de uso, apenas a interação entre o ator externo e o caso de uso é percebida. No nível de uma colaboração, todas as interações e passos da execução que implementam o sistema serão descritos e especificados.

O diagrama de casos de uso, Figura 6, demonstra as funções de um ator externo instigando ações na Ferramenta PiramidNet gerada sob o Framework PiramidNet. O diagrama especifica quais funções este usuário poderá desempenhar. Percebe-se que não existe preocupação com a implementação de cada uma destas funções, já que este diagrama apenas resume quais funções deverão ser suportadas pelo sistema modelado.

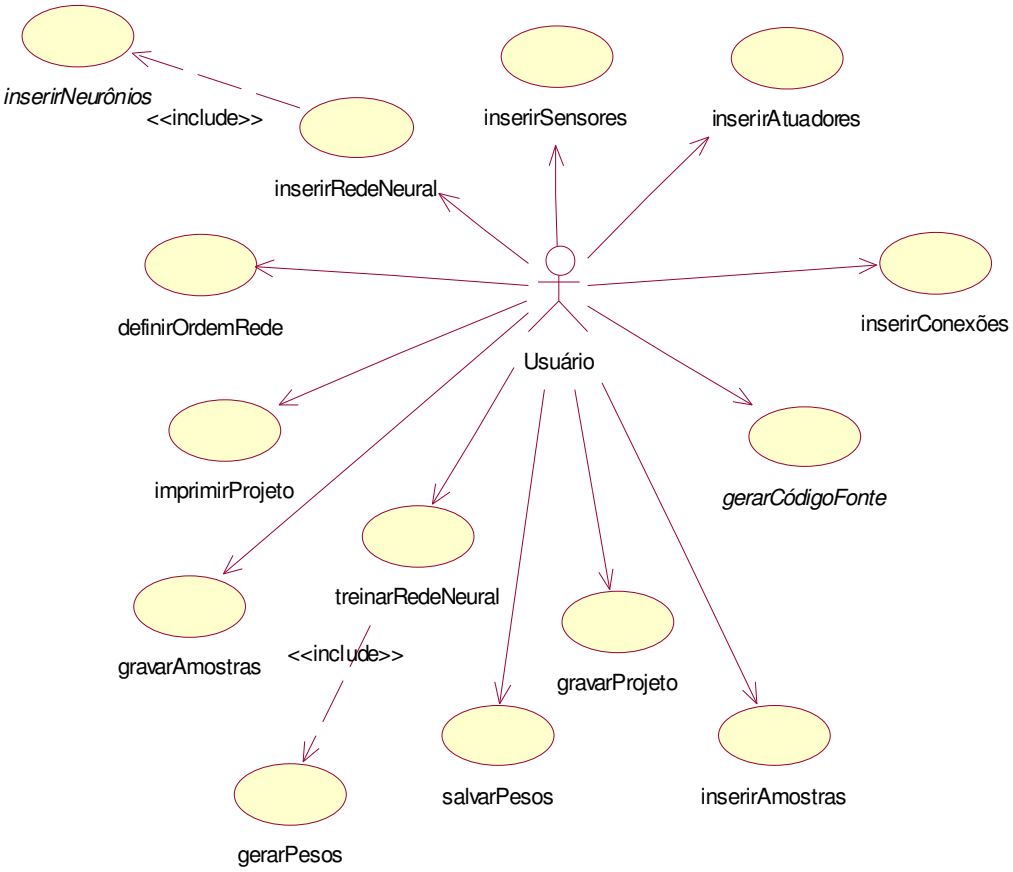

Figura 6 – Montando o Projeto - Diagrama de casos de uso da Ferramenta PiramidNet.

O objetivo da figura acima, é mostrar os recursos que a ferramenta dispõe para construir um projeto que envolva RNAs. O usuário (boneco) está representando a interação humana que irá disparar os objetos (balões). O diagrama acima possibilita ao usuário modelar o comportamento do sistema.

4.4.2 Diagrama de Classes

Boggs (2002) define diagrama de classes como: "*A class is something that encapsulates information and behavior*". Uma classe em um diagrama pode ser diretamente implementada quando exista uma linguagem de programação orientada a objetos que tenha suporte direto para construção de classes.

Para criar este tipo de diagrama, as classes devem estar identificadas, descritas e relacionadas entre si. As classes podem se relacionar com outras se valendo de algumas maneiras: associação (conectadas entre si), dependência (uma classe depende ou usa outra classe), especialização (uma classe é uma especialização de outra classe) ou em pacotes (classes agrupadas por características similares). Todos estes relacionamentos podem ser mostrados em um diagrama de classes juntamente com as suas estruturas internas - que são os atributos e as operações.

A técnica de modelagem chamada Diagrama de Classes tem seu objetivo focado na representação dos atributos e métodos de cada classe. Ilustra as especificações para as classes de programas e de interfaces de uma aplicação. Este diagrama é considerado como estático, uma vez que a estrutura descrita é sempre válida em qualquer ponto do ciclo de vida do sistema. Vale ressaltar sua importância, pois a maioria dos outros diagramas usará as classes definidas neste diagrama.

O diagrama de classes do projeto detalha as informações que são suficientes para gerar o código que será utilizado na camada de objetos do domínio. Larman (2000) cita: "A implementação em uma linguagem de programação orientada a objetos requer que escreva o código fonte para definições de classes e de métodos".

A seguir, Figura 7, apresenta-se um exemplo de classe abstrata, a Classe Rede Neural Artificial – CRNA**.** O diagrama de classes descreve o conjunto classes do sistema. Nesta descrição, constam: a estrutura, suas relações e os comportamentos. Uma classe comporta várias linhas de código. Através da identidade, atributos e métodos, molda-se uma classe.

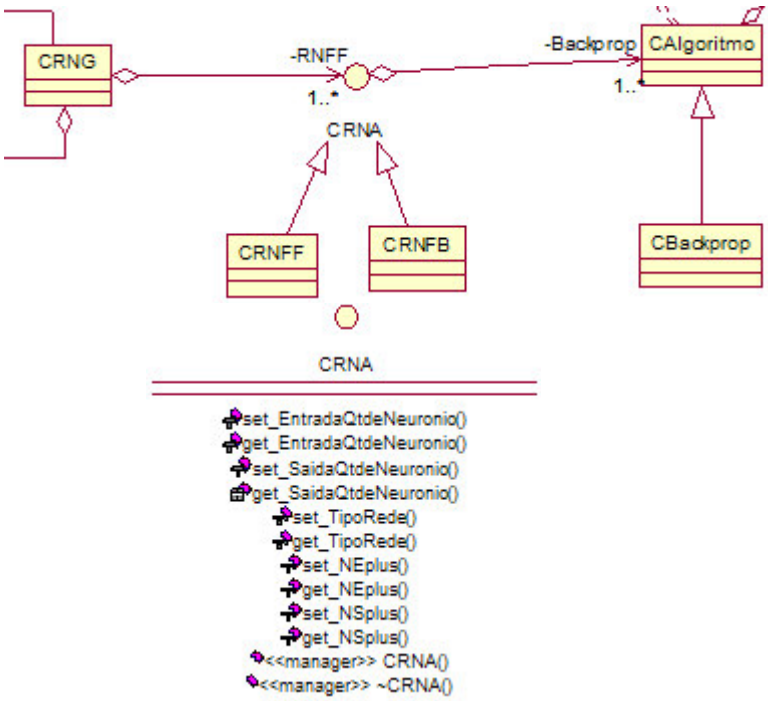

Figura 7 - CRNA – Classe abstrata do framework com seus relacionamentos.

A classe abstrata CRNA possui os seguintes métodos:

a) Método set_EntradaQtdeNeuronio: responsável por receber do usuário um valor inteiro. Este valor representa a quantidade de neurônios da camada de entrada da rede;

b) Método get_ EntradaQtdeNeuronio: responsável por retornar o valor inteiro que representa a quantidade de neurônios da camada de entrada da rede;

c) Método set_SaídaQtdeNeuronio: responsável por receber do usuário um valor inteiro. Este valor representa a quantidade de neurônios da camada de saída da rede;

d) Método get_SaídaQtdeNeuronio: responsável por retornar o valor inteiro que representa a quantidade de neurônios da camada de saída da rede;

e) Método set_TipoRede: Cada rede, no sistema, tem um Tipo definido. O Tipo de cada rede é representado no sistema por um valor inteiro (identificador). Quando o usuário seleciona a rede que deseja trabalhar, automaticamente é passado para este método o tipo daquela rede que foi escolhida. Este tipo será usado em rotinas que envolvam a distinção de propriedades específicas a cada RNA;

f) Método get_TipoRede: responsável por retornar o valor inteiro (tipo identificador) daquela RNA que foi escolhida pelo usuário;

g) Método set_NEplus: responsável por acrescentar ou não neurônios na camada de entrada. Este método guarda um valor booleano que representa o acréscimo ou decréscimo de neurônios na camada de entrada ou saída da rede;

h) Método get_NEplus: retorna o valor booleano;

i) Método set_NSplus: responsável por acrescentar ou não neurônios na camada de saída. Este método guarda um valor booleano que representa o acréscimo ou decréscimo de neurônios na camada de entrada ou saída da rede;

j) Método get_NSplus: retorna o valor booleano;

k) Relacionamento com a classe CAlgoritmo (agrega): neste relacionamento utilizou-se o padrão de projeto Factory Method. A intenção é definir uma interface para criar um objeto;

l) Relacionamento com a classe CRNG (é parte): CRNG agrega CRNA, pois CRNA é a parte conceitual das redes neurais artificiais e CRNG corresponde a uma generalizada construção gráfica do modelo destas redes. Este canal é o canal de comunicação entre o modelo conceitual e o modelo gráfico.

A seguir, descreve-se cada classe pertencente ao projeto do Framework, Figura 08:

Classes da Linguagem:

Classe typeinfo: o arquivo typeinfo.h armazena as declarações e protótipos para informações de classes em tempo de execução, seus dados membros e suas funções membro;

Classe list: é um tipo de classe de seqüência que suporta iteradores bidirecionais. Uma list<T,Allocator> permite constantemente inserir e apagar elementos dentro da seqüência;

Classe algorithm: nesta classe encontram-se vários algoritmos essenciais para executar diferentes operações em *containers* e *sequences*;

Classe iostream: suporta a transferência de dados entre um programa de dados e os periféricos;

Classe vcl: utilizada pelas classes de objetos gráficos. Nesta classe, encontram-se as rotinas gráficas responsáveis pelo desenho, cor etc;

Classe math: onde se encontram as rotinas de equações matemáticas;

Classe stdlib: biblioteca de comandos padrões da linguagem.

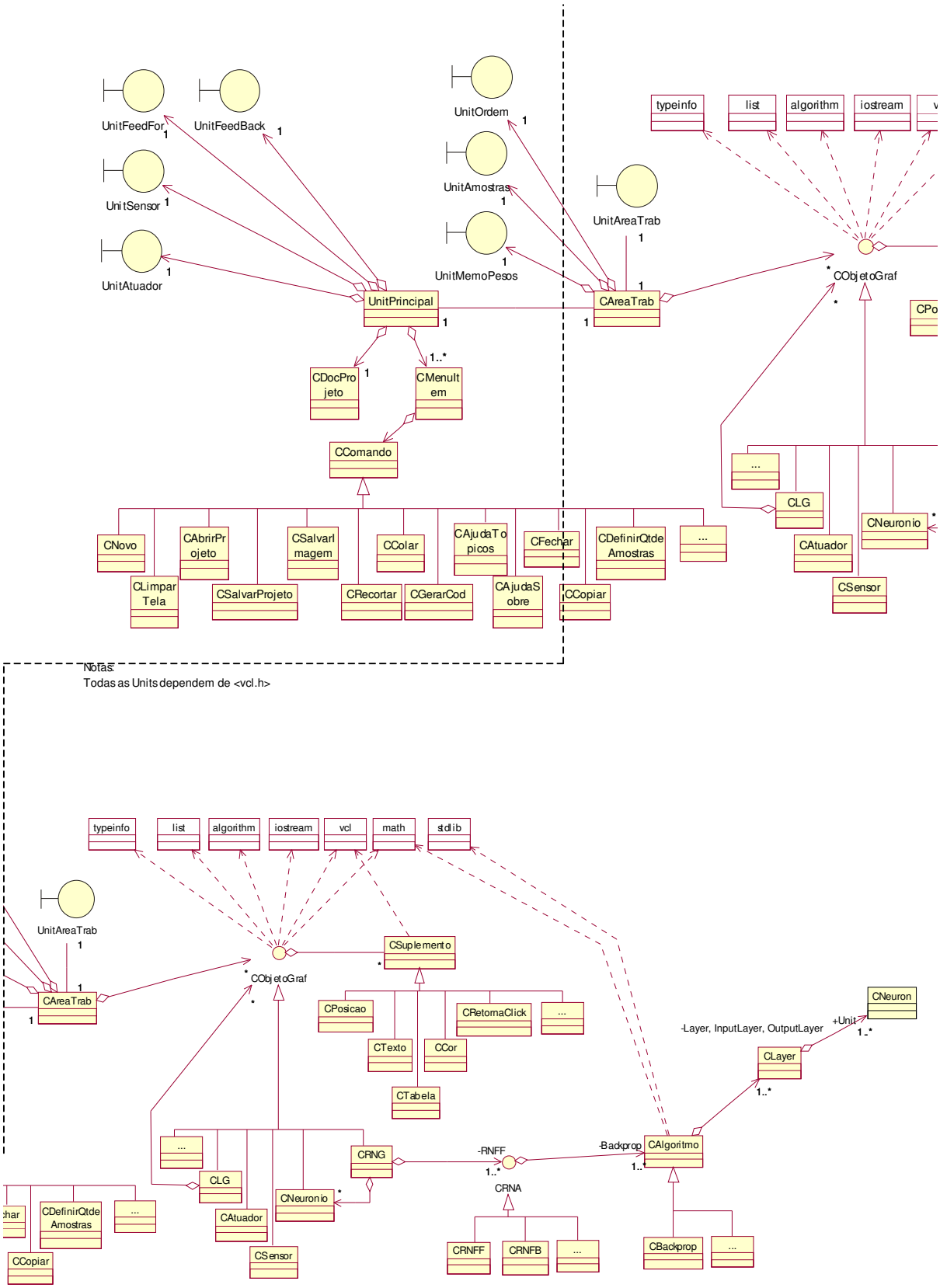

Figura 8 - Diagrama de classes do Framework.

Classes do Sistema: RNA – conceitual:

Classe CRNA: já explicada, esta classe suporta *hot spot*;

Classe CRNFF: contém as características pertinentes a uma RNA Direta;

Classe CRNFB: contém as características pertinentes a uma RNA Recorrente;

Classe CNeuron: contém o neurônio e suas propriedades;

Classe CLayer: suporta uma estrutura em forma de camadas que agregam neurônios;

Classe CAlgoritmo: contém uma matriz de pesos e o vetor de resultados. Esta classe suporta *hot spot*;

Classe CBackprop: contém o algoritmo backpropagation;

Classes do Sistema:

Classe CObjetoGraf: classe abstrata que suporta métodos comuns para outras classes. Esta classe suporta hot spot;

Classe CLG: responsável pela conexão entre os seguintes objetos: sensores, atuadores e neurônios de uma rede neural gráfica qualquer;

Classe CAtuador: responsável pelas características e definições gráficas primárias que um atuador deve possuir: suas coordenadas cartesianas, situação de ocupado quando conectado, mudanças de cores, autodesenho;

Classe CSensor : responsável pelas características e definições gráficas primárias que um sensor deve possuir: suas coordenadas cartesianas, situação de ocupado quando conectado, mudanças de cores e autodesenho;

Classe CNeuronio: responsável pelas características e definições gráficas primárias que um neurônio deve possuir: suas coordenadas cartesianas, situação de ocupado quando conectado, mudanças de cores, possibilidade de desenhar e até verificar se foi clicado;

Classe CRNG: Este modelo de classe é responsável por montar o quebra-cabeça da construção de uma rede neural artificial gráfica. Usando as Classes CNeuronio e CRNA, uma rede neural artificial gráfica pode ser desenhada. O objetivo desta classe é trabalhar com todas as características do desenho de uma rede na Área de Trabalho (área destinada aos objetos gráficos). Através de métodos como autodesenho, inserção dos neurônios nas camadas, captura de valores da CRNA, seleção da rede, seleção do neurônio, mudança de cor e outros;

Classe CSuplemento: utilizando-se do padrão Strategy, esta classe define uma família de algoritmos. Este padrão encapsula e torna intercambiável cada algoritmo. Esta classe suporta *hot spot*;

Classe CPosicao: recebe as posições de coluna e linha do mouse. Estas posições são passadas quando é executada a ação de clique sobre a Área de Trabalho;

Classe CTexto: responsável pelo texto, tamanho do texto e espaço vertical da linha;

Classe CCor: armazena a cor para colorir os objetos gráficos que serão desenhados na Área de Trabalho;

Classe CTabela: classe utilizada para montar, em memória, uma tabela que receberá valores (amostras) para treinamento da rede neural artificial;

Classe CRetornaClick: caso satisfeita a opção de seleção, tem como encargo retornar: características de um ponto de ligação, sensor, atuador, neurônio ou rede neural;

Classe UnitFeedFor: esta unidade fornece uma janela de configuração da RNA Direta. Através desta tela, entra-se com valores para que se efetuem os devidos ajustes às características desta rede;

Classe UnitFeedBack: esta unidade fornece uma janela de configuração da RNA Recorrente. Através desta tela, entra-se com valores para que se efetuem os devidos ajustes às características desta rede;

Classe UnitSensor: esta unidade fornece uma janela de configuração para o Sensor gráfico. Através desta tela, entra-se com valores para que se efetuem os devidos ajustes às características deste objeto gráfico;

Classe UnitAtuador: esta unidade fornece uma janela de configuração para o Atuador gráfico. Através desta tela, é possível entrar com valores para que se efetuem os devidos ajustes às características deste objeto gráfico;

Classe UnitOrdem: esta unidade fornece uma janela de configuração para ordenar as RNAs gráficas dispostas na Área de Trabalho. Esta ordem não se refere a disposição na tela, mas a ordem de treinamento de todo o conjunto de redes que existem no projeto. Nesta janela, o usuário entra com valores positivos de ordem crescente para cadastrar a ordem de treinamento das redes;

Classe UnitAmostras: esta unidade fornece uma janela de configuração para cadastrar valores de treinamento (exemplos ou amostras) e resultados desejáveis;

Classe UnitMemoPesos: esta unidade fornece uma janela de demonstração de valores (pesos sinápticos). Estes valores são formatados e mostrados em uma coluna. Caso a condição de aceitação dos pesos seja validada, estes pesos serão armazenados geração do código ANSI C;

Classe UnitAreaTrab: esta unidade fornece uma a janela principal do sistema. É a responsável por receber objetos gráficos e gerenciar outras janelas;

Classe CAreaTrab: utilizando-se do padrão Mediator, define um objeto que encapsula o comportamento coletivo em CAreaTrab. O uso deste padrão torna-se válido quando é necessário para controlar e coordenar as interações de um grupo de objetos. Esta unidade faz seu diferencial por receber/enviar dados para serem processados e validados por outras classes;

Classe UnitPrincipal: é a tela de abertura do sistema. Nesta unidade encontra-se a janela de entrada de objetos da Ferramenta PiramidNet, constituída de menus suspensos, botões e a Área de Trabalho;

Classe CMenuItem: item de menu que chama a função execute() pelo seu objeto Comando da classe CComando. Este item de menu dispara uma ordem pertinente ou uma seqüência de ordens;

Classe CComando: usando-se do padrão Command, esta classe abstrata encapsula uma solicitação como um objeto, desta forma permite a parametrização de clientes com diferentes solicitações, podendo enfileirar ou registrar solicitações e suportar operações que possam ser desfeitas. Esta classe também suporta *hot spot*;

Classe CNovo: encarregada de iniciar uma nova Área de Trabalho;

Classe CLimparTela: encarregada de apagar os objetos gráficos alocados na Área de Trabalho;

Classe CAbrirProjeto: encarregada de ler projetos que anteriormente foram armazenados;

Classe CSalvarProjeto: encarregada de armazenar em disco os projetos desenvolvidos pelo usuário;

Classe CSalvarImagem: encarregada de armazenar em disco o projeto como formato de imagem;

Classe CRecortar: encarregada de remover o objeto gráfico da tela e disponibilizá-lo em memória;

Classe CColar: encarregada de copiar o objeto gráfico contido em memória e alocá-lo na Área de Trabalho utilizando as coordenadas retornadas pelo mouse;

Classe CGerarCod: encarregada de disparar a execução para gerar o código ANSI C;

Classe CAjudaTopicos: encarregada de mostrar os tópicos responsáveis por ajudar o usuário;

Classe CAjudaSobre: encarregada de mostrar informações pertinentes ao desenvolvimento do projeto;

Classe CFechar: encarregada de fechar a janela ativa;

Classe CCopiar: encarregada de fazer uma duplicata do objeto selecionado, colocando-o abaixo e a direita do objeto original;

Classe CDefinirQtdeAmostras: encarregada de receber o valor inteiro positivo que faz referência a quantidade de amostras.

4.4.3 Diagramas de Interação - Seqüência

A função de um diagrama de interação, de acordo com Craig Larman (2000), é ilustrar as interações com mensagens entre as instâncias no modelo de classes. José Davi Furlan (1998) também define um diagrama de interação como sendo um termo genérico que se aplica a vários tipos de diagramas. Fower (2000) ressalta que a utilização do diagrama de interação parte da necessidade de visualizar o comportamento de vários objetos dentro de um único caso de uso, a partir das mensagens que são passadas entre eles.

Logo, os diagramas de interação são apresentados sob duas formas na UML: diagrama de colaboração e diagrama de seqüência. O diagrama de colaboração ilustra as interações, colaborações dinâmicas, que diversos objetos trocam entre si na forma de grafo ou rede. O diagrama de seqüência também ilustra as interações, mas sua forma gráfica muda para um formato parecido com cercas (LARMAN, 2000). As setas de mensagens são nomeadas e existem para demonstrar a direção do fluxo de mensagens. O diagrama de seqüências pode conter objetos ativos que executam paralelamente com outros.

A Figura 09 é um exemplo de diagrama de seqüência, retratando o treinamento de uma RNA.

Este diagrama defini os principais itens para treinar RNAs. Estes passos são importantes, sem o qual não haveria possibilidade de gerar o programa-fonte para o robô. Este programa carrega as informações pertinentes a modelagem e valores finais do treinamento das redes. O treinamento de uma rede neural existe para definir os pesos (valores) entre as conexões sinápticas e sem estes pesos não há aprendizado. O robô utilizará o programa-fonte para demonstrar o comportamento adquirido.

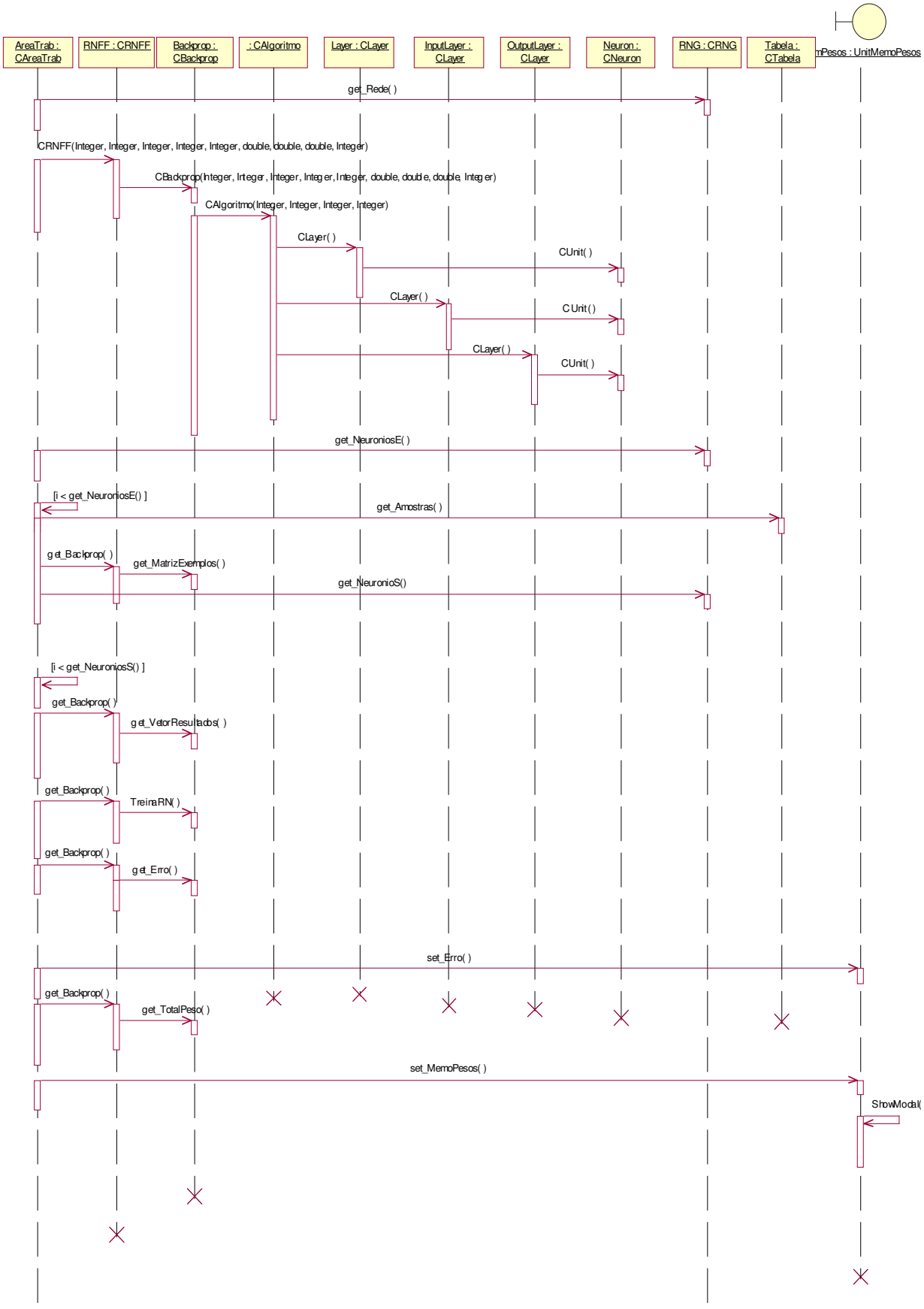

Figura 9 –Treinamento de RNAs - Exemplo de diagrama de seqüência da Ferramenta.

4.4.4 Diagramas de Comportamento - Atividade

O diagrama de atividades mostra o fluxo seqüencial das atividades em um sistema. Este fluxo é normalmente utilizado para demonstrar as atividades executadas por uma operação específica do sistema. Sendo o diagrama de atividades a variação do diagrama de estados, é uma maneira alternativa de mostrar as interações.

Este tipo de diagrama consiste em estados de ação, os que contêm a especificação de uma atividade a ser desempenhada por uma operação do sistema. Decisões e condições como execução paralela, também podem ser mostradas na diagramação. Esta representação gráfica também pode conter especificações de mensagens enviadas e recebidas como partes de ações executadas.

Optou-se pelo Diagrama de Atividades para evidenciar, através da Figura 10, as interações e a execução das ações.

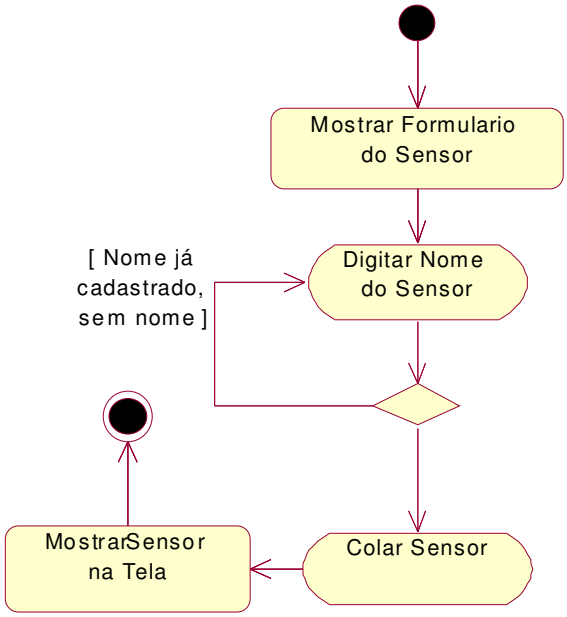

Figura 10 – Inserir Sensor – Exemplo de diagrama de atividades da Ferramenta.

É demonstrada como atividade do usuário no sistema, a inserção de um objeto gráfico que recebe o nome Sensor. Este objeto gráfico representa intuitivamente o elemento responsável por colher as informações oriundas do ambiente. Estas informações são necessárias para preencher a tabela de valores que será usada para treinar as RNAs e como conseqüência disso, formar o aprendizado do robô. Existe também a possibilidade do objeto

Sensor fazer conexão direta com o objeto Atuador, só que desta forma não haverá aprendizado.

4.4.5 Diagramas de Implementação - Componentes

O diagrama de componentes apresenta uma visão física de seu modelo, ou seja, os componentes implementados no seu sistema e o relacionamento entre eles. Larman (2000) define este diagrama da seguinte maneira: “Os diagramas de componentes mostram as dependências de compilação e tempo de execução entre componentes do sistema, tais como arquivos-fonte e dlls”. Analisando o diagrama de componentes e o de execução, percebe-se que são dois diagramas que mostram o sistema pelo lado funcional, expondo as relações entre seus componentes e a organização de seus módulos durante sua execução. Nesta dissertação optou-se pelo diagrama de componentes, visto que descreve bem os componentes do sistema e as dependências entre si, representando a estrutura do código gerado. Os componentes são considerados como peças que fazem parte da implementação na arquitetura física dos conceitos e da funcionalidade definidos na arquitetura lógica (classes, objetos e seus relacionamentos). A dependência entre componentes é mostrada como uma linha tracejada com uma seta, simbolizando que um tipo de componente precisa do outro, para possuir uma definição completa. Com o diagrama de componentes, demonstrado na Figura 11, facilmente se detecta quais são os arquivos necessários para a aplicação.

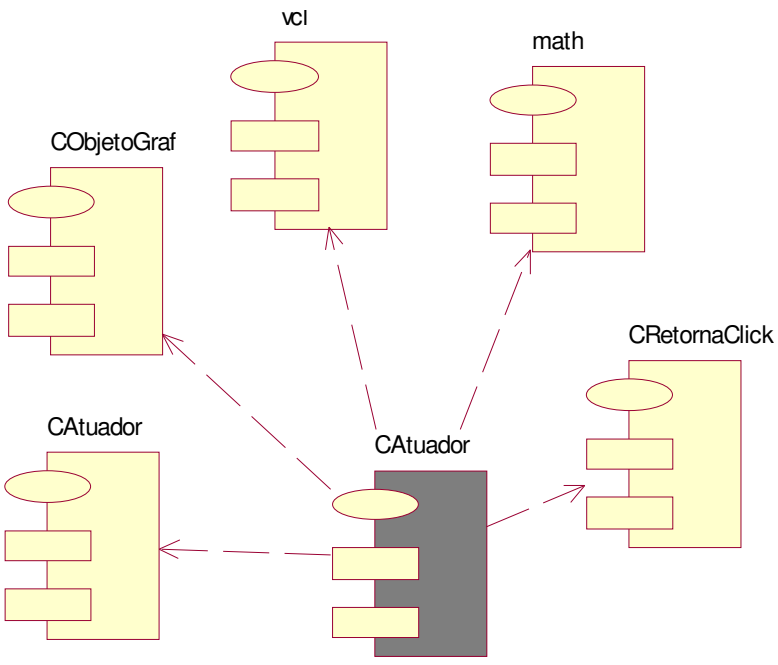

Figura 11 – Classe CAtuador - Exemplo de diagrama de componentes da Ferramenta.

No exemplo da Figura 11, a classe CAtuador precisa das seguintes classes porque:

a) Classe CObjetoGraf: é a classe responsável pela definição de métodos primários do objeto;
b) Classe CRetornaClick: é a classe responsável pelo retorno da localização de objetos, neste caso o objeto Atuador, como também outras informações pertinentes ao objeto;
c) Classe vcl: já definida, esta classe destina-se às rotinas gráficas para que se possa desenhar o objeto na tela;
d) Classe math: já definida, esta classe destina-se às rotinas matemáticas responsáveis pelo dimensionamento e cálculo de área o objeto que foi retornado pela classe CRetornaClick.

4.5 Interface Gráfica

Há muito que as telas com fundo preto e letras verdes deixaram de habitar o cotidiano dos nossos monitores. Os sistemas baseados em caracteres, que exigiam entre outras coisas memorização de comandos, não podem ser comparados pelo menos para os usuários comuns, aos aplicativos que se comunicam com interfaces gráficas intuitivas e atraentes.

Sendo possível fazer a comunicação entre o usuário e o computador por meio de uma interface, Paul Heckel (1991) cita que se deve pensar mais em comunicação do que em computação, afinal “[...] atingir as expectativas e atender questões subjetivas do público é tarefa para um bom comunicador”. Uma boa interface deve ser transparente para aquele que a utiliza, capacitando e auxiliando o usuário no acesso fácil as especificidades dos elementos e funcionalidades do programa.

Quando se fala em usabilidade, faz-se presente questões relacionadas a quão bem os usuários podem utilizar as funcionalidades da aplicação. Debora Hix (1993) resume muito bem esta questão quando afirma: "Uma boa interface é como o telefone ou como a luz elétrica; quando funciona ninguém a percebe". Nielsen (1993) associa o conceito de usabilidade a cinco atributos: facilidade de aprendizado; eficiência de uso; facilidade de memorização; baixa taxa de erros e a satisfação subjetiva.

A interface gráfica que compõe a Ferramenta PiramidNet foi modelada de acordo com as necessidades e especificidades dos pesquisadores que trabalham com modelos de RNAs do

grupo de pesquisa do Projeto PiramidNet e do Laboratório de Conexionismo e Ciências Cognitivas L3C. Usando os requisitos para construção de janelas (*Individual-Window Design*) estipulada por Shneiderman (1998), a interface gráfica é constituída de menus de comando e *popup*, caixas de diálogo, botões, várias sub-telas e uma área de modelagem gráfica que suporta RNAs. Este Framework contém uma tela central, interface de apresentação, responsável por gerir o ambiente; área para diagramação e sub-telas responsáveis por diversos tipos de elementos que compõe o ambiente da ferramenta.

A interface de apresentação da Ferramenta, demonstrada na Figura 12, suporta: uma barra de menus suspensos, uma barra de botões de elementos, uma tabela de ordem das RNAs e um retângulo central de fundo branco. Funcionalmente, estes componentes suprem as necessidades do usuário. Os botões e menus *popup* estão estrategicamente distribuídos para auxiliar o usuário a transcorrer a tarefa, estendendo-se dentro do programa para gerar uma ação futura. O retângulo branco posicionado no centro e nomeado "Área de Trabalho", recebe objetos como: sensores, modelos de redes neurais artificiais, atuadores e as conexões entre eles.

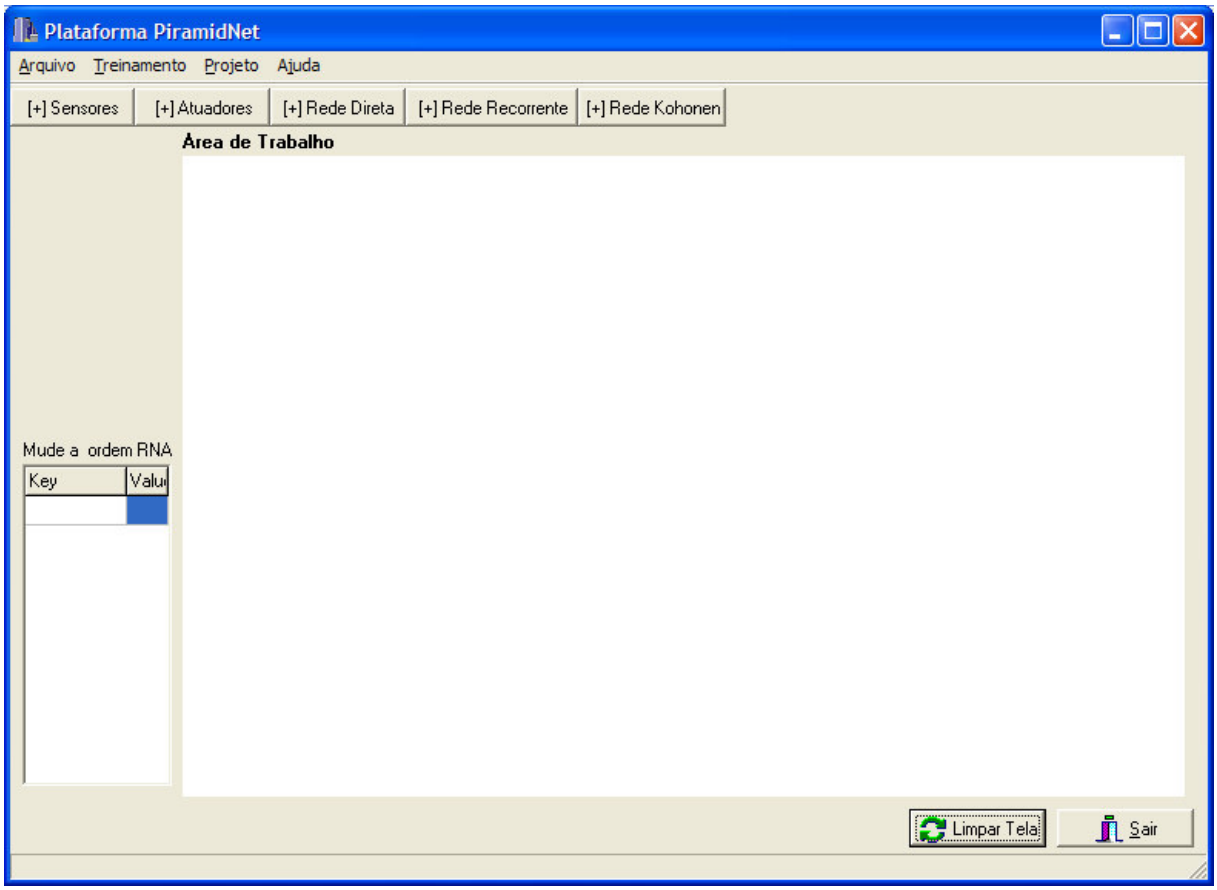

Figura 12 - Tela principal da Ferramenta PiramidNet.

4.6 Utilizando a Ferramenta

Referenciando a tela inicial, Figura 12, o usuário aproveita a localização dos botões disponíveis na barra de botões, Figura 13, para inserir seus objetos.

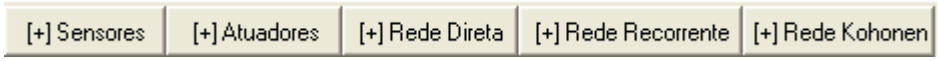

Figura 13 - Barra de Botões da Ferramenta PiramidNet.

No início da modelagem, escolhem-se os objetos a serem inseridos na Área de Trabalho. Esta escolha pode ser feita de forma aleatória. Para esta documentação, iniciou-se com o botão "[+] Sensores". Este botão tem a característica de apresentar um formulário, Figura 14, onde o usuário irá digitar o nome do sensor para depois colocá-lo na Área de Trabalho.

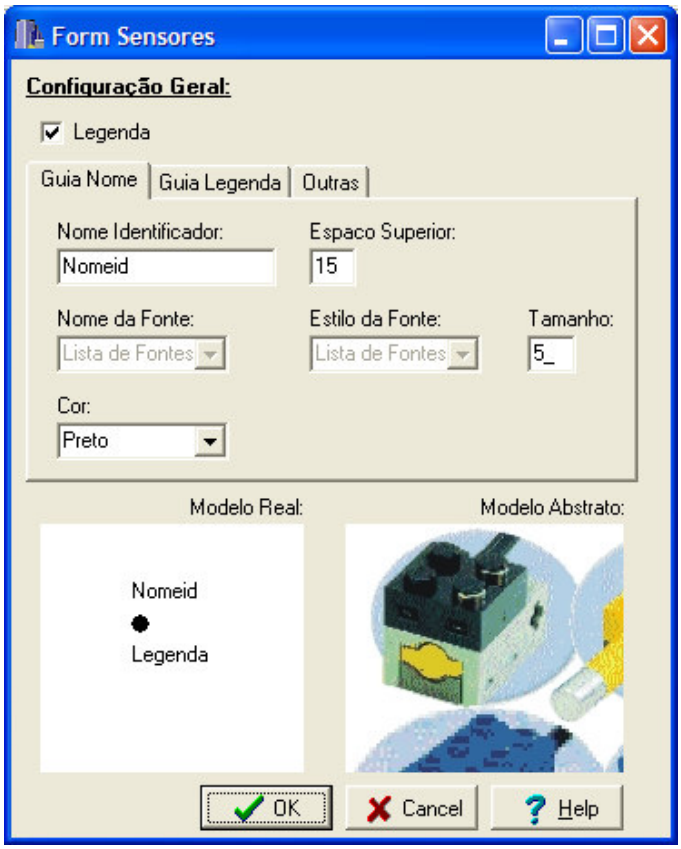

Figura 14 - Formulário Sensor da Ferramenta PiramidNet

Segue abaixo, a explicação dos campos que compõe o Form Sensores e Form Atuadores, Figura 14 e Figura 15, respectivamente:

- Legenda: quando selecionada, habilita a Guia Legenda que é a responsável por acrescentar uma legenda explicativa ao ícone do sensor;
- Guia Nome: comporta as opções de configuração do texto do nome do sensor,
    - Nome Identificador: usado para atribuir nome ao ícone sensor;
    - Espaço Superior: espaço usado para distanciar verticalmente o nome do sensor com o centro do ícone;
    - Tamanho: especifica o tamanho da fonte do texto;
    - Cor: especifica a cor da fonte do texto.
- Modelo Real: janela gráfica mostrando a real imagem do sensor que aparecerá na Área de Trabalho;
- Modelo Abstrato: mostra a foto de um sensor físico.

O botão "[+] Atuadores", Figura 15, mostra ao usuário um formulário parecido com o formulário Sensores. O item Atuador fará parte no processo de modelagem gráfica do projeto.

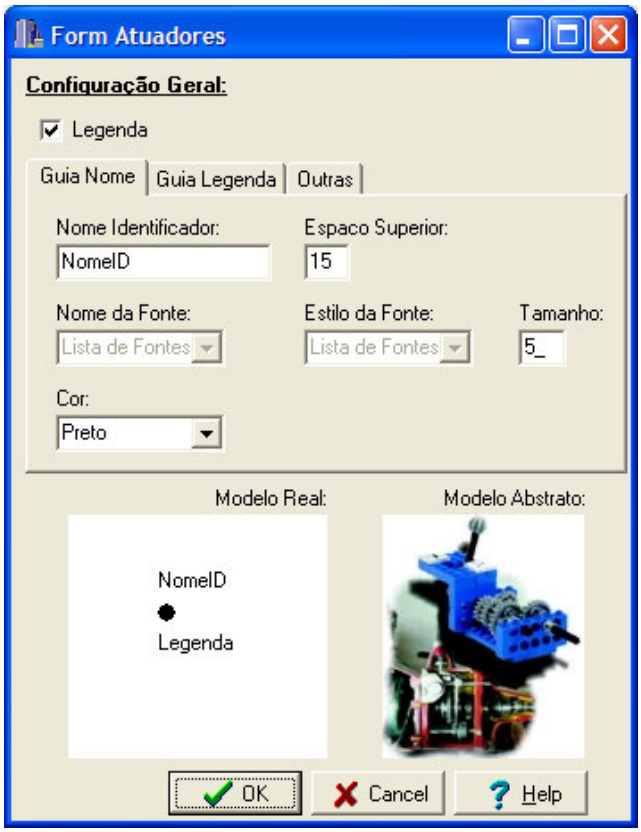

Figura 15 - Formulário Atuador da Ferramenta PiramidNet

Na seqüência, escolhe-se a rede que fará parte do projeto. Para esta modelagem inicial, o botão "[+] Direta" chama o formulário, Figura 16, que invoca os parâmetros necessários para modelar uma Rede Direta.

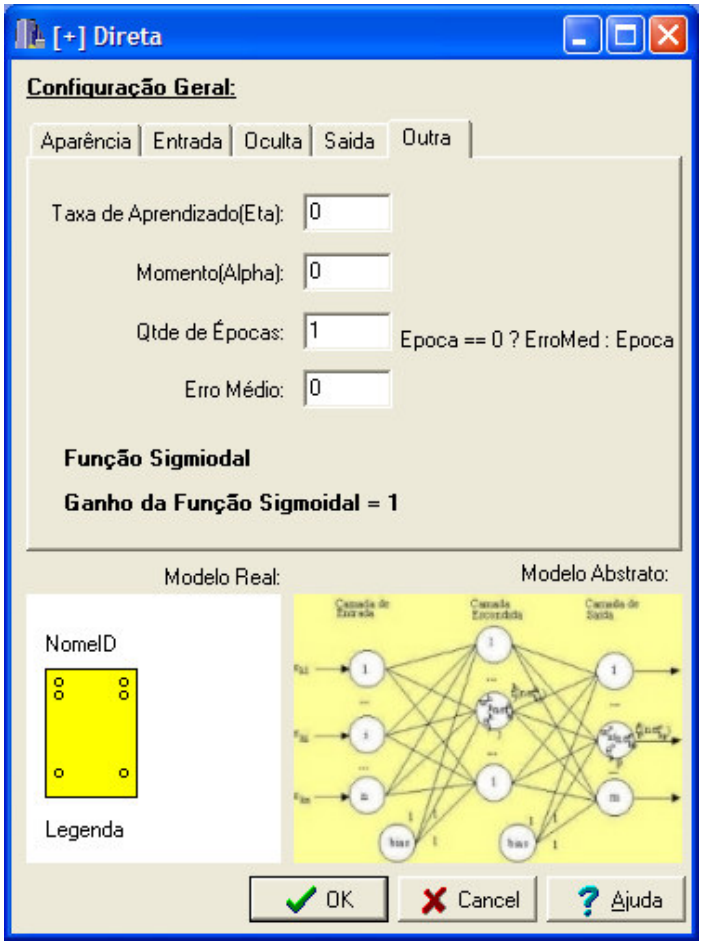

Figura 16 - Formulário [+] Direta da Ferramenta PiramidNet.

Documenta-se abaixo os campos que compõe o Formulário [+] Direta:

- Taxa de Aprendizado: é o valor de andamento que o vetor gradiente irá dar na superfície de erro. Valores elevados podem fazer o vetor gradiente ficar alternando de um lado para o outro na superfície de erros. Geralmente, usam-se valores entre 0.1 a 0.3;
- Momento: valores elevados – entre 0,8 e 0,9 – são necessários para que o vetor gradiente não vá à direção dos mínimos locais e mantenha uma direção adequada a ser seguida;
- Quantidades de Épocas: o treinamento se encerra quando a quantidade de iterações for atingida. É importante ressaltar que o exagero no treinamento pode levar a rede a se tornar especialista;

- Erro Médio: o treinamento se encerra quando o erro médio quadrático aceitável for alcançado. É importante ressaltar que o exagero no treinamento pode levar a rede a piorar sua capacidade de generalização.

Depois que cada objeto gráfico foi distribuído na Área de Trabalho, o usuário tem a flexibilidade de mudar as características. Este recurso está disponível em diferentes menus *popup* e pode ser requisitado através do clique do botão direito do mouse. Cada objeto gráfico agrega um tipo de menu. A Figura 17 apresenta o menu de uma rede escolhida. Desmontando-o em duas partes, a parte superior relaciona-se com a RNA e a parte inferior relaciona-se com as conexões e os neurônios. A Figura 18 apresenta o menu de um Sensor e a Figura 19 apresenta o menu de um Atuador.

Figura 17 - Menu popup Rede/Link da Ferramenta PiramidNet

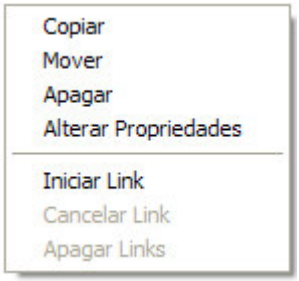

Figura 18 - Menu popup Sensor da Ferramenta PiramidNet

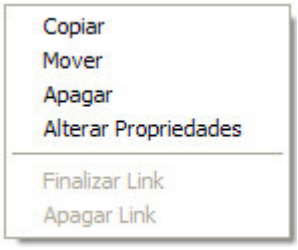

Figura 19 - Menu popup Atuador da Ferramenta PiramidNet

Utilizando os recursos que a ferramenta dispõe - modelos de RNAs - é possível chegar a esta tela de experimento, Figura 20.

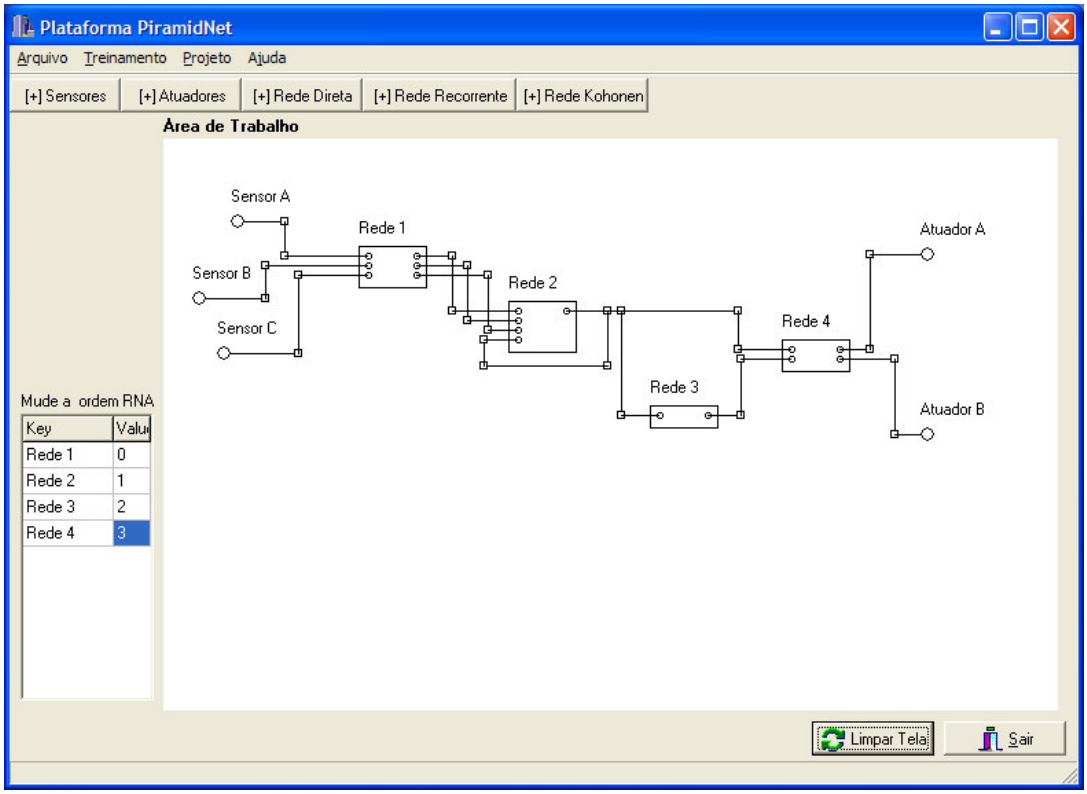

Figura 20 - Tela de experimento da Ferramenta PiramidNet.

4.7 Considerações Finais

Observou-se que, o Framework PiramidNet tem a potencialidade e a robustez para gerar ferramentas que apóiem os desenvolvedores de RNAs. O diagrama de classes (estrutura estática) apresentou a estrutura e definição do Framework PiramidNet e os diagramas de comportamentos apresentaram os principais recursos da Ferramenta PiramidNet. No próximo capítulo, apresenta-se um roteiro para trabalhar com a Ferramenta.

5 POTENCIALIDADE DA FERRAMENTA PIRAMIDNET

5.1 Considerações Iniciais

Este capítulo comenta as potencialidades alcançadas da Ferramenta PiramidNet em um projeto específico de RNAs. Usando-se do projeto manual de RNAs hierárquicas e heterogêneas de Silva (2001), é possível construir um projeto gráfico automatizado de redes neurais artificiais para simular os mesmos comportamentos inteligentes, em um artefato robótico, só que agora mais rápido e organizado.

5.2 Projeto Manual de RNAs: Seguir Paredes

O texto foi extraído da dissertação de Silva (2001):

> Imaginemos a seguinte situação, o robô está andando pelo mundo e detecta um foco luminoso e deverá ir a sua direção até estar muito próximo. No caso do ambiente simulado o foco luminoso também é considerado como obstáculo, já que se trata de um 'abajur'. No caso de encontrar um obstáculo, quando estiver indo para o foco luminoso, que impeça a sua aproximação em relação ao foco, ele deverá desviar primeiro para depois ir de encontro ao foco. Então, seu comportamento deverá ser não se chocar com este obstáculo que impede a aproximação, e sim tentar dar a volta e chegar ao objetivo. Conjeturou-se nesse trabalho que, encontrar um foco luminoso é um comportamento que deve emergir logo que um foco for detectado, no entanto para isto acontecer o robô terá que desviar dos obstáculos. Os obstáculos formando paredes ou somente 'tijolos' espalhados pelo mundo são translúcidos, possibilitando a detecção do foco de luz.
>
> Para a implementação dos comportamentos no robô simulado, fez-se necessário a identificação dos valores dos seus sensores, pois através deles determinamos o seu 'comportamento'. O posicionamento do robô no 'mundo', se ele está perto ou distante de um obstáculo ou de um foco luminoso, é dado pelos sensores. A etapa seguinte foi a implementação de uma hierarquia de rede neural capaz de selecionar e coordenar o comportamento do robô fazendo-o 'desviar de obstáculos' e 'ir de encontro a um foco luminoso'. Podemos compará-lo ao comportamento de uma criança aprendendo a andar, uma vez que, terá que se manter em pé e, ao mesmo tempo, desviar de obstáculos para não se machucar. Após a verificação de que o robô conseguiu desviar de obstáculos e também foi ao encontro de um foco luminoso, assim que o detectou, modificamos a hierarquia criando mais comportamentos reflexivos, reativos e um controle de energia. O intuito foi conseguir fazê-lo seguir paredes enquanto sua energia estivesse normal. E fazê-lo 'perceber' quando sua energia chegou a um nível baixo a ponto dele ter que percorrer o mundo em busca de um ponto de energia para reabastecer suas 'baterias'.

Deste projeto, construiu-se uma estrutura nervosa com modelos de RNAs hierárquicas e heterogêneas. Para chegar no resultado, utilizaram-se comportamentos reflexivos, comportamentos reativos e um controle de energia. Estes comportamentos são caracterizados por possuírem fases distintas de funcionamento. Por exemplo, um robô que está em um

ambiente dinâmico, deve usar um comportamento básico como evitar colisões com obstáculos. Também deve ser capaz de seguir paredes, percebendo que exista algo ao seu redor. Enquanto a energia do robô estiver normal, o processo de seguir paredes é contínuo, mas quando estiver com seu nível de energia muito baixo, o robô deve interromper o comportamento anterior, ou seja, parar de seguir paredes e passar a vagar pelo ambiente a procura de uma fonte luminosa para ir de encontro a uma fonte de energia para recarga.

Para visualizar graficamente os comportamentos anteriormente citados, utilizaram-se Autômatos de Estados Finitos (AEFs). Menezes (1998) cita que "[...] é um modelo matemático de sistema com entradas e saídas [...]" e complementa: "Cada estado resume somente as informações do passado necessárias para determinar as ações para a próxima entrada". O uso de AEFs sintetiza o fato de poderem ser associados a diversos tipos de sistemas naturais e artificiais.

Demonstra-se abaixo, A explosão de comportamentos do estado 0 (Rede de Controle Seguir Parede – RCSP) e estado 1 (Rede de Controle de Energia – RCE) do Controle de Energia, Figura 21.

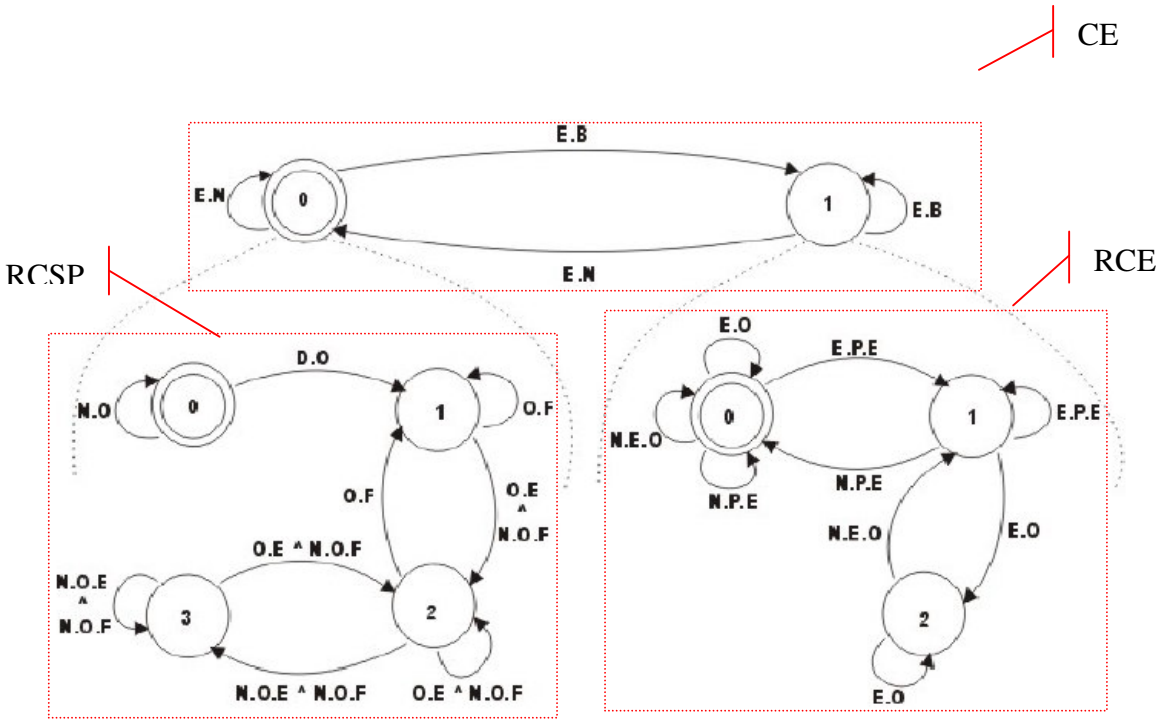

Figura 21 – Hierarquia dos AEFs, extraído de Silva (2001).

A tabela abaixo, Tabela 2, contém as siglas que foram utilizadas na Figura 21.

Tabela 2 – Descrição dos estados da hierarquia dos AEFs.

Estado	Descrição
DO	Detectou Obstáculo
EB	Energia Baixa
EM	Energia Normal
EO	Encontrou Obstáculo
EPE	Encontrou Ponto de Energia
NEO	Não Encontrou Obstáculo
NO	Nenhum Obstáculo
NOE	Nenhum Obstáculo Esquerdo
NOF	Nenhum Obstáculo Frontal
NPE	Não Encontrou Ponto de Energia
OE	Obstáculo Esquerdo
OF	Obstáculo Frontal

Para cada fase do processo, existe um comportamento que responde diferentemente aos estímulos que recebe. As ações de controle previstas em uma fase não são necessariamente válidas em outras fases. A navegação autônoma depende do reconhecimento do ambiente e tratamentos específicos para cada situação encontrada. O sistema de controle deve ser capaz de identificar e se adaptar a cada uma das fases do processo.

O projeto é composto pelos seguintes elementos:

a) 8 sensores e 2 atuadores (robô Khepera);
b) modelo de rede direta para alcançar comportamentos básicos reflexivo: evitar colisões e ir de encontro a um foco luminoso;
c) modelo de rede recorrente para alcançar comportamentos complexos: coordenar a seleção dos comportamentos reflexivos.

A seguir, o diagrama construído manualmente, Figura 22, representando o projeto de RNAs hierárquicas e heterogêneas, Figura 21.

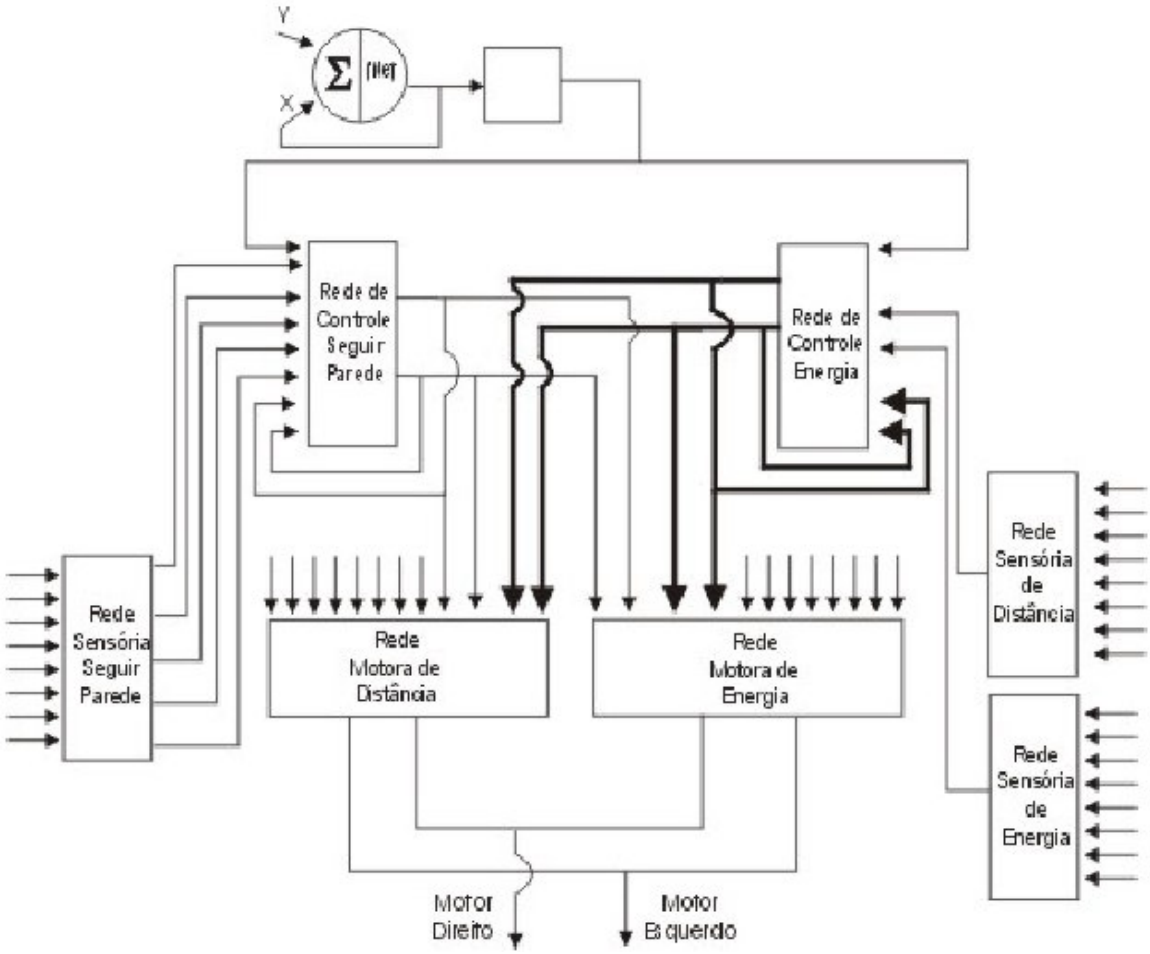

Figura 22 - Diagrama feito manualmente, Projeto de RNAs, extraído de Silva (2001).

Apresenta-se abaixo, a descrição da quantidade de neurônios na camada de entrada, armazenamento e saída de cada rede neural, Figura 22:

a) Rede Sensória de Distância (RSD): 8 neurônios de entrada, 10 neurônios ocultos e 1 neurônio de saída;
b) Rede Sensória de Energia (SER): 8 neurônios de entrada, 10 neurônios ocultos e 1 neurônio de saída;
c) Rede Sensória Seguir Parede (RSSP): 8 neurônios de entrada, 10 neurônios ocultos e 5 neurônios de saída;
d) Rede de Controle Seguir Parede (RCSP): 8 neurônios de entrada, 10 neurônios ocultos e 2 neurônios de saída;
e) Rede de Controle de Energia (RCE): 8 neurônios de entrada, 10 neurônios ocultos e 2 neurônios de saída;
f) Rede Motora de Distância (RMD): 12 neurônios de entrada, 14 neurônios ocultos e 2 neurônios de saída;
g) Rede Motora de Energia (RME): 12 neurônios de entrada, 14 neurônios ocultos e 2 neurônios de saída.

5.3 Projeto Automatizado de RNAs: Seguir Paredes

O uso Ferramenta envolve a manipulação de objetos gráficos como sensores, atuadores, RNAs e conexões. Apresenta-se abaixo, Figura 23, o Projeto de RNA Seguir Paredes, construído usando a Ferramenta PiramidNet. Os comportamentos que o robô gerou no projeto de Silva, também foram conseguidos no projeto automatizado.

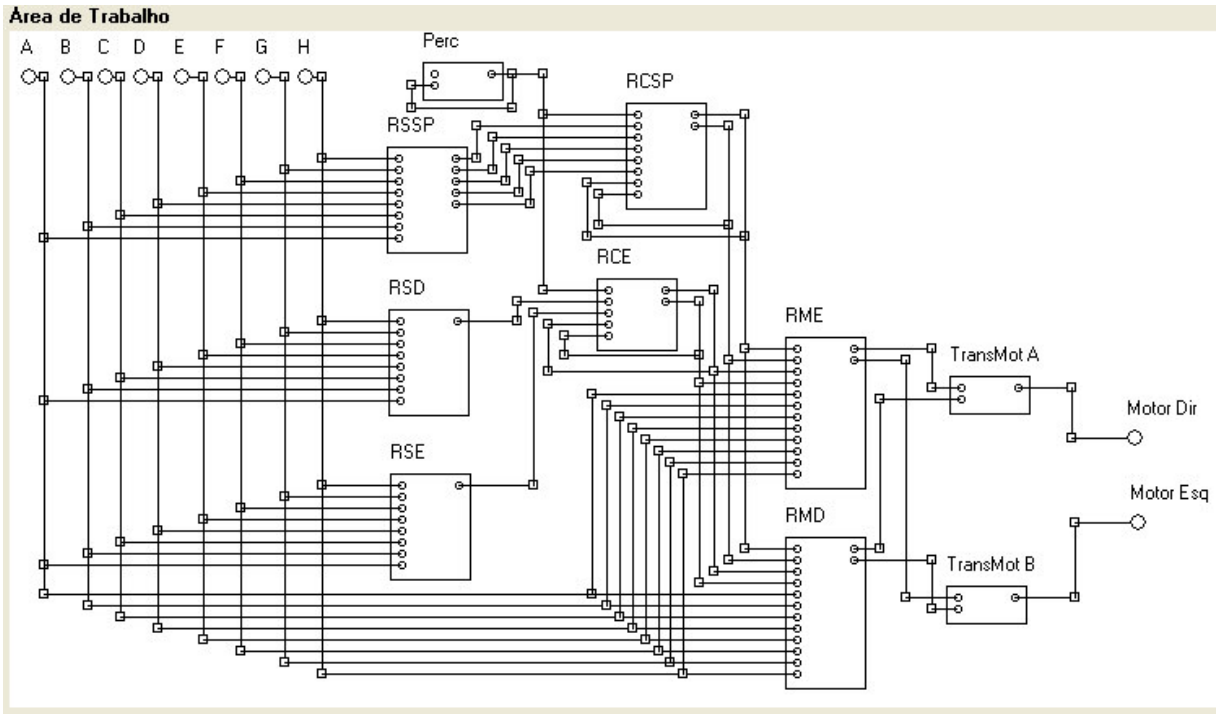

Figura 23 - Diagrama proposto pela Ferramenta PiramidNet, Projeto de RNAs.

5.3.1 Gerando o Código ANSI C

A Ferramenta oferece o recurso de gerar código ANSI C, a partir do diagrama construído. Depois de criado o código, é possível integrá-lo a um código já existente ou hospedá-lo diretamente no robô. Além disso, todas as linhas de código consideradas importantes, foram comentadas.

5.4 O Potencial da Ferramenta PiramidNet

A existência de características que permitem interconectar RNAs e gerar código-fonte de forma clara, faz o diferencial da Ferramenta PiramidNet com outras ferramentas. Estas características permitem traduzir diagramas de RNAs feitos a mão, com uma distribuição não uniforme das redes e sua construção, para um diagrama formal e ordenado.

Para inserir objetos gráficos na Área de Trabalho, a Ferramenta oferece botões que estão dispostos para o usuário de forma intuitiva. Cada objeto faz relação com um menu suspenso. O menu suspenso é acionado pelo mouse. Estes menus são responsáveis pelas características plásticas dos objetos. É possível deslizar o objeto sobre a área gráfica.

Cada projeto usa um ou vários modelos de RNAs. A escolha do modelo de RNA depende do domínio do problema (contexto) que o usuário está trabalhando. A Ferramenta suporta o recurso de expansão de modelos de RNAs, inserindo novas classes, caso não exista um modelo específico. A potencialidade da performance não é alterada pela quantidade e a disposição das redes. Cada modelo de rede tem seu formulário com suas respectivas características.

A Ferramenta gera, a partir de um diagrama modelado, código-fonte em ANSI C. Este código já está pronto para que seja hospedado no robô Lego. O código-fonte primaria ser o mais enxuto possível, sem perder a qualidade e modularidade da engenharia de software. É um código portável para qualquer plataforma. Esta é uma grande potencialidade. Agora, o usuário não precisa saber ou entender ‘a fundo’ uma linguagem de programação para construir classes de RNAs.

As conexões entre objetos gráficos são construídas mantendo a forma de linhas e ângulos de 90 graus (ortogonal). Isto facilita a leitura do diagrama para o usuário. Caso uma conexão esteja sobre outra, é possível movê-la ou apagá-la.

A Área de Trabalho da Ferramenta, disponibiliza um espaço suficiente para suportar vários sensores/atuadores, várias RNAs com vários neurônios em cada entrada ou saída. Caso o usuário necessite de uma área maior, é possível redimensioná-la. A Área de Trabalho se atualiza a cada evento do objeto que está sendo trabalhado.

5.5 Considerações Finais

A Ferramenta foi capaz de automatizar o diagrama construído manualmente. Isto facilita e muito a vida do usuário. Quando enfocamos a falta de tempo e a busca por resultados imediatos, esta é a Ferramenta para se trabalhar. A construção desse tipo de ferramenta vem de encontro a necessidades que a muito tempo se tinha, ou seja, construir projetos de RNAs para artefatos robóticos sem precisar digitar uma linha de código!

6 CONCLUSÕES

Esta dissertação foi desenvolvida com o propósito de implementar comportamentos biologicamente inspirados em artefatos robóticos. Não existindo ferramentas com suporte a criação de conexões entre modelos de RNAs heterogêneas e que gere código-fonte a partir do projeto gráfico modelado, inferiu-se a construção de uma Ferramenta que oferecesse estas facilidades. Porém, desejava-se ter uma estrutura orientada a objetos que fosse capaz de dar suporte a outras ferramentas usando de sua estrutura básica (extensão de estrutura). Logo, a construção de um framework orientado a objetos que use RNAs é a melhor decisão. Isto definido, estudou-se o comportamento dos seres vivos, as redes neurais artificiais, o Projeto PiramidNet - buscando identificar os modelos de redes neurais que melhor se adequavam aos comportamentos – e frameworks.

Para projetar frameworks, é necessário estudar profundamente UML, Padrões de Projeto e Tipos de Frameworks. Abrangendo todo este conhecimento, dá-se início a construção do Framework PiramidNet.

A estrutura do Framework PiramidNet começa a ser desenvolvida a partir da construção de classes genéricas somado ao uso de Padrões de Projeto, funcionalidades dos objetos, modelos de RNAs e suas características. A construção de frameworks passa por várias iterações para se aproximar de uma estrutura mais apurada. Terminada a etapa de construção do Framework, o próximo passo é a construção de uma ferramenta. Usando as estruturas pré-estabelecidas do Framework PiramidNet, pôde-se criar a Ferramenta PiramidNet. A Ferramenta habilita o usuário – sem conhecimento de RNAs ou linguagem de programação – a definir um projeto de RNAs heterogêneas. Contendo os modelos de redes necessários para gerar comportamentos no robô, o usuário é capaz de convertê-los em código-fonte para ser implementado em um robô móvel.

Para provar a superioridade de resultados na experiência, escolheu-se um projeto de RNAs compatível com a Ferramenta gerada. O projeto de RNAs hierárquicas de Silva (2001) tinha como objetivo fazer um robô seguir paredes. Depois de construído o projeto de Silva na Ferramenta, percebeu-se a equivalência na obtenção dos comportamentos do robô.

As melhorias que a Ferramenta oferece em relação ao trabalho de Silva (2001) são:

a) Disponibilidade de recursos em um ambiente visual;
b) Otimização de tempo para desenvolver o projeto, conseqüentemente baixo custo;
c) Enfoca todas as fases de projeto de RNAs: criação, treinamento e codificação;
d) Adaptação na configuração da RNA escolhida - característica visual e específica (modelo);
e) Visualização prévia de um modelo de RNA graficamente especificado (diagrama);
f) Facilidade nas conexões entre objetos gráficos;
g) Escolha da ordem de treinamento das RNAs;
h) Visualização da execução do treinamento das redes;
i) Desprendimento da programação;
j) Importação e exportação de modelos de RNAs;
k) Facilidade para gerar o código-fonte para o robô;
l) Criação de código-fonte final otimizado e com comentários;
m) Dentre outras já comentadas.

Os usuários de RNAs do laboratório L3C-UFSC já estão usando a Ferramenta, para desenvolvimento de comportamentos complexos em ambientes dinâmicos com artefatos robóticos.

Por fim, esta dissertação vem facilitar e acelerar o desenvolvimento de projetos, programas (código-fonte) e ferramentas que trabalham com RNAs. Antigamente, para se construir um programa com duas RNAs interconectadas que fosse impelementada em um robô móvel, necessitava de um conhecimento profundo da linguagem e ambiente de programação, conceitos e técnicas para se trabalhar com RNAs, conhecimento do artefato robótico, sua linguagem de programação interna e muito, muito tempo. Com o Framework PiramidNet, é possível que se construa ferramentas em pouquíssimo tempo e com um alto grau de desempenho. Outro importante ganho está no desenvolvimento de programas sem a necessidade de entender a fundo qualquer linguagem ou ambiente de programação. A Engenharia de Software é de fundamental importância para construção sistemas baseados em comportamentos. Ela não só proporciona um adequando embasamento teórico necessário para um correto e amplo entendimento da modelagem de sistemas, como também introduz os conceitos fundamentais para um correto desenvolvimento. O objetivo não é somente capacitar

o usuário a desenvolver projetos de RNAs, mas também informalmente construir um raciocínio para combinar estes comportamentos biologicamente inspirados, seguindo algumas regras simples e aplicáveis que podem chegar a gerar um sistema nervoso para o robô móvel.

6.1 Limitações

As limitações do Framework são:

a) Construção de uma rede neural a partir de neurônios independentes;
b) Ferramenta multiplataforma.
c) Exportação direta para o artefato robótico.

6.2 Sugestões para Trabalhos Futuros

Como sugestões para trabalhos futuros, pode-se citar:

a) Framework,
    a. uma expansão do modelo estrutural do Framework, envolvendo a inserção de novas classes;
b) Ferramenta,
    a. inserir de novas redes neurais artificiais e seus respectivos algoritmos de treinamento;
    b. implementar a transferência do código para o artefato robótico a partir do sistema;
    c. construir um ambiente gráfico para modelar autômatos;
    d. desenvolver novas ferramentas utilizando a estrutura do framework.

REFERÊNCIAS BIBLIOGRÁFICAS

ALVES, J. B. M. Ficção, realidade e expectativa de robôs inteligentes baseados em comportamento. In: I SIMPÓSIO BRASILEIRO DE AUTOMAÇÃO INTELIGENTE, 1993, Rio Claro, SP. **Anais do I Simpósio Brasileiro de Automação Inteligente**, São Paulo: [s.n.], 1993.

ANDERSON, T. L.; DONATH, M. Animal behavior as a paradigm for developing robot autonomy. In: MAES, P. (ed). **Designing autonomous agents: theory and practice from biology to engineering and back**. Massachusetts: MIT, 1990. p. 145-168. ISBN: 0-262-63135-0.

ARKIN, R. C. **Behavior-based robotics**. 2. ed. Massachusetts: MIT, 1999. 491 p. Inclui índice. ISBN 0-262-01165-4.

BARRETO, J. M. **Inteligência artificial no limiar do século XXI**. 3 ed. rev e aum. Florianópolis: PPP, 2001. 392 p. 26 cm. Inclui índice. ISBN: 85-900382-5-4.

BARTO, A. G. Connectionist learning for control. In: MILLER III, W. T.; SUTTON, R. S.; WERBOS, P. J. (ed). **Neural networks for control.** Cambridge: MIT, 1990. p. 5-58.

BEER, R. D.; CHIEL, H. J.; STERLING, L. S. A biological perspective on autonomous agent design. In: MAES, P. (ed). **Designing autonomous agents:** theory and practice from biology to engineering and back. Massachusetts: MIT, 1990. p. 169-186. ISBN: 0-262-63135-0.

BITTENCOURT, G. **Inteligência artificial**: ferramentas e teorias. Florianópolis: UFSC, 1998. 362 p. ISBN: 85-328-0138-2.

BOERS, E. J. W.; KUIPER, H. **Biological metaphors and the design of modular artificial neural networks**. 1992. Dissertação (Mestrado em Ciências da Computação) - Departamento da Ciência da Computação e Psicologia Experimental e Teórica. Leiden University, 1992.

BOGGS, W.; BOGGS, M. **Mastering UML with rational rose 2002**. London: SYBEX, 2002. ISBN: 0-7821-4017-3.

BOURGINE, P.; VARELA, F. J. Introduction: towards a practice of autonomous systems. In: FIRST EUROPEAN CONFERENCE ON ARTIFICIAL LIFE, 1991, Paris. **Proceedings of the First European Conference on Artificial Life.** Paris, MIT, 1991. p. 11-17.

BLUM, A. **Neural networks in C++:** an object-oriented framework for building connectionist system. [S.l.]: Wiley, 1992.

BROOKS, R. A. A robust layered control system for a mobile robot. **Artificial Intelligence Memo 864.** Massachusetts Institute of Tecnology, Artificial Intelligence Laboratory. 1985.

BROOKS, R. A. A robot that walks: emergent. **Artificial Intelligence Memo 1091.** Massachusetts Institute of Tecnology, Artificial Intelligence Laboratory, 1989.

BROOKS, R. A. Elephants don´t play chess. In: MAES, P (ed). **Designing autonomous agents:** theory and practice from biology to engineering and back. Massachusetts: MIT, 1990. p. 3-15. ISBN: 0-262-63135-0.

BROOKS, R. A. **Artificial life and real robots**. Paris: MIT, 1991. p. 3-10.

CARBONEL, J. G.; MICHALSKI, R. S.; MITCHELL, T. An overview of machine learning. In: **Machine learning**: an artificial intelligence approach. [S.l.]: Springer-Verlag, 1984, p. 3-23.

CARDOSO, S. H. Comportamento instintivo: padrão fixo de ação (pfa). Brain & Mind: eletronic magazine on neuroscience – **Revista da Informática Biomédica da UNICAMP**. Campinas, 2002. Seção Comportamento Instintivo. Disponível em: <http://www.epub.org.br/cm/n09/fastfacts/comportold.htm>. Acesso em: 20 mai. 2002. ISSN: 1414-4018.

DEITEL, H. M.; DEITEL, P.J. **C++ como programar**. 3 ed. rev e aum. Porto Alegre: Bookman, 2001. 1098 p. ISBN: 85-7307-740-9.

DIAS, A. S. **Desenvolvimento em borland C++ builder 5.0**. Rio de Janeiro: Ciência Moderna, 2000. 193 p. ISBN: 85-7393-095-0.

EILB-EIBESFELDT, J. **Etologia**: introducción al estudio comparado del comportamiento. Barcelona: Omega, 1979. 643 p.

FOWLER, M.; SCOTT, K. **UML essencial**: um breve guia para a linguagem padrão de modelagem de objetos. Porto Alegre: Bookman, 2000. 169 p.

FRONTEIRA, N.(ed). **Novo dicionário Aurélio - século xxi**. Rio de Janeiro: Nova Fronteira, nov. 1999. 1 CD-ROM. Produzido por MGB Informática Ltda.

FAYAD, M. E., SCHMIDT, Douglas C., **Object-oriented application frameworks**, Communications of the ACM, vol. 40 n. 10, pp. 32-38, 1997.

FURLAN, J. D. **Modelagem de objetos através da UML**. São Paulo: Makron Books, 1998.

GAMMA, E.; *et al*., **Padrões de projeto: soluções reutilizáveis de software orientado a objetos**. Porto Alegre: Bookman, 2000. 364p. ISBN 85-7307-610-0.

GARRAS.JPG. Altura: 100 pixels. Largura: 85 pixels. 75 dpi. 32 BIT CMYK. 162 KB. Formato JPG. Compactado. Disponível em: <http://www.epub.org.br/cm/n09/fastfacts/garras.jpg>. 1999. Acesso em: 20 mai. 2002.

HAFNER, V. V. Learning places in newly explored environments. In: SAB2000, 2000, Honolulu. **Proceedings Supplement Book**. Honolulu: International Society for Adaptive Behavior, 2000.

HEBB, D. O. **The organization of behavior**. New York: Wiley, 1949.

HECKEL, P. **Software amigável**: técnicas de projeto para uma melhor interface com o usuário. Rio de Janeiro: Campus, 1991.

HEYLIGHEN, F. **(Meta)systems as constraints on variation**. [S.l.]:World Futures, 1995. p. 59-85.

HIX, D.; HARTSON, H. **Developing user interfaces**: ensuring usability through product & process. [S.l.]: John Wiley & Sons, 1993.

HOGG, D. W.; Martin, F.; RESNIK, M. **Braitenberg creatures**. [S.l.: s.n.], Jun. 1991.

JACOBSON, I. *et al*. **Object-oriented software engineering**: a use case driven approach. MA: Addison-Wesley, 1992.

JOHNSON, R. E. **Frameworks = components + patterns.** New York: Communications of the ACM, 1997. v. 40. n. 10. p. 39-42.

JOHNSON, R. E. **How to design framework**s. [S.l.: s.n.], 1993. Disponível em <ftp://st.cs.uiuc.edu/pub/framework>. Acesso em fev. 2002.

LANGTON, C. *et al*. **Artificial life II**. Massachusetts: Addison-Wesley, 1991.

LAPRADE, A.; LAMBERT-TORRES, G. Controle de um veículo utilizando a teoria dos conjuntos nebulosos. In: I SIMPÓSIO BRASILEIRO DE AUTOMAÇÃO INTELIGENTE, 1993, São Paulo. **Anais do I Simpósio Brasileiro de Automação Inteligente**. São Paulo, 1993. p. 166-174.

LARMAN, C. **Utilizando UML e padrões**. Porto Alegre: Bookman, 2000. 492 p. ISBN:85-7307-651-8.

LEE, R. C. **UML and C++:** a pratical guide to object-oriented development. New Jersey: Prentice-Hall, 1997. 446p. ISBN:0-13-619719-1.

LEITE, A.; ALVES, J. B. M.; RESENDE, M. F. Sistema para depuração de um robô móvel autônomo inteligente. In: I SIMPÓSIO BRASILEIRO DE AUTOMAÇÃO INTELIGENTE, 1993, São Paulo. **Anais do I Simpósio Brasileiro de Automação Inteligente**. São Paulo, 1993. p. 175-184.

MATEUS, C. A. **C++ builder 5**: guia prático. São Paulo: Érica, 2000. 286 p. ISBN: 85-7194-747-3.

MCFARLAND, D.; BÖSSER, T. **Intelligent behavior in animals and robots**. Massachusetts: MIT, 1993. 305 p. Inclui índice. ISBN:0-262-13293-1.

MENEZES, P. F. B. "**Linguagens formais e autômatos**". 2. ed. Porto Alegre: Instituto de Informática da UFRGS: Sagra Luzzato, 1998. 165 p. ISBN: 85-241-0554-2.

MEYER, B. **Object-oriented software construction**. Englewood Cliffs: Prentice Hall, 1988.

MÖLLER, R.; MARIS, M.; LAMBRINOS, D. **A neural model of landmark navigation in insects**. [S.l]: Elsevier project, 1998.

NIELSEN, J. **Usability engineering**. [S.l.]: AP Professional, 1993.

OWEN, C.; NEHMZOW, U. Route learning in mobile robots through self-organizing. In: **EUROMICRO.** [S.l], 1996. Workshop on advanced mobile robots. [S.l]: IEEE, 1996.

OLIVEIRA, L. O. L. **Mapas auto-organizáveis de Kohonen aplicados ao mapeamento de ambientes de robótica móvel**. 2001. 69 f. Dissertação (Mestrado em Ciências da Computação) - Curso de Pós-Graduação em Ciência da Computação, Universidade Federal de Santa Catarina, Florianópolis, 2001.

PALLAZO, L. A. M.; CASTILHO, J. M. V. **Sistemas de informação inteligentes**: uma perspectiva cibernética. UCPel - Universidade Católica de Pelotas. Pelotas, 2002. Seção Biblioteca Virtual. Disponível em: <http://esin.ucpel.tche.br/bbvirt/art/Art-educ.htm>. Acesso em: 20 mai. 2002.

PARBERRY, I. **Circuit complexity and neural networks**. Massachussetts: MIT Cambridge, 1994.

PATO.JPG. Altura: 100 pixels. Largura: 93 pixels. 75 dpi. 32 BIT CMYK. 162 KB. Formato JPG. Compactado. Disponível em: <http://www.epub.org.br/cm/n09/fastfacts/pato.jpg>. 1999. Acesso em: 20 mai. 2002.

PREE, W. **Design patterns for object oriented software development**. [S.l.]: Addison-Wesley, 1994.

PRESSMAN, R. S. **Engenharia de software**. São Paulo: Makron Books, 1995.

RASHEVSKY, N. The principle of adequate design. In: ROSEN, R. (ed). **Foundations of mathematical biology:** supercellular systems**.** New York and London: Academic, 1973. v. 1. p. 143-175.

REINALDO, F. A. F.; ROISENBERG, M. Sistemas baseados em comportamentos robóticos. In: IV ENCONTRO DE ATIVIDADES CIENTÍFICAS - UNOPAR, 2001, Arapongas. **Anais do IV Encontro de Atividades Científicas da UNOPAR**. Londrina: UNOPAR, 2001. v. 1. 1CD. ISBN: 85-87686-0.

RIBEIRO, J. A. **UML - Linguagem de modelagem unificada**: em português. Rio de Janeiro, 2000. Seção UML: tutorial. Disponível em <http://www.eng.uerj.br/~araujo/disciplinas/uml>. Acesso em 13 jul 2002.

ROISENBERG, M.; BARRETO, J. M.; AZEVEDO, F. M. Biological inspirations in neural network implementations of autonomous agents. In: 13th BRAZILIAN SYMPOSIUM ON ARTIFICIAL INTELLIGENCE, 1159, oct. 1996. **Advances in artificial intelligence:** Lecture notes in computer science; Lecture notes in artificial intelligence. Berlin: Springer-Verlag, 1996. p. 211-220.

ROISENBERG, M.; BARRETO, J. M.; AZEVEDO, F. M. Modeling behaviors with artificial neural networks. In: WRI'97 INTELLIGENT ROBOTICS WORKSHOP, 1997, Brasília, DF. **Anais do WRI'97 Intelligent Robotics Workshop**. Brasília:[s.n], 1997. p. 34-45.

ROISENBERG, M.; BARRETO, J. M.; AZEVEDO, F. M. On a formal concept of autonomous agents. In: AI'98 IASTED INTERNATIONAL CONFERENCE ON APPLIED INFORMATICS, 1998. **Anais do AI'98 IASTED International Conference on Applied Informatics**. Germany: Garmisch-Paterkirchen, 1998a.

ROISENBERG, M. **Emergência da inteligência em agentes autônomos através de modelos inspirados na natureza**. 1998, 206 f. Tese (Doutorado em Engenharia Elétrica). Grupo de Pesquisas em Engenharia Biomédica, Universidade Federal de Santa Catarina, Florianópolis, 1998b.

ROISENBERG, M.; BARRETO, J. M.; AZEVEDO, F. M. Um ambiente evolucionário para geração de redes neurais em agentes autônomos. In: IV CONGRESSO BRASILEIRO DE REDES NEURAIS, July 20-22, 1999, ITA, São José dos Campos, SP. **Proceedings of IV Brazilian Conference on Neural Networks.** São Paulo:[s.n.], 1999. p. 888-999.

ROISENBERG, M. **PiramidNet - redes neurais modulares e hierárquicas:** fundamentos e aplicações em robótica. 2000. 13 f. Projeto de Pesquisa. Departamento de Informática e de Estatística, Universidade Federal de Santa Catarina, Florianópolis, 2000.

ROSEN, R. **Is there unified mathematical biology?** In: ROSEN, R. (ed). **Foundations of mathematical biology:** supercellular systems**.** New York and London: Academic, 1973. v. 1. p. 361-393.

RUMBAUGH, J.; BLAHA, M.; PREMERLANI, W.; EDDY, F.; LORENSEN, W. **Object-oriented modeling and design.** Englewood Cliffs, New Jersey: Prentice-Hall, 1991.

SAUVÉ, J. P. **Definições de frameworks.** Disponível por www em <http://www.dsc.ufpb.br/~jacques/cursos/1999.2/map/material/frame/deffw.htm>. Acesso em fev. 2002.

SCHILDT, H.; GUNTLE, G. **Borland C++ builder**: a referência completa. Rio de Janeiro: Campus, 2001. 780 p. ISBN: 85-352-0871-2.

SCHMID, H. A. **Systematic framework design.** New York: Communications of the ACM, 1997. v. 40. n. 10. p. 48-51.

SHELDRAKE, R. **Renascimento da Natureza**. São Paulo: Cultrix, 1991.

SHNEIDERMAN, B. **Design the user interface:** strategies for effective human-computer interaction. 3 ed. California: Addison-Wesley, 1998. 639p. ISBN: 0-201-69497-2.

SILVA, R. P. **Suporte ao desenvolvimento e uso de frameworks e componentes**. Porto Alegre: PPGC da UFRGS, 2000.

SILVA, F. A. **Redes neurais hierárquicas para implementação de comportamentos em agentes autônomos**. 2001. 132 f. Dissertação (Mestrado em Ciências da Computação) - Curso de Pós-Graduação em Ciência da Computação, Universidade Federal de Santa Catarina, Florianópolis, 2001.

TALIGENT(ed.). **Building object-oriented framework**s. [S.l.]: Taligent, 1994. White paper.

TALIGENT(ed.). **Leveraging object-oriented framework**s. [S.l.]: Taligent, 1995. White paper.

TURCHIN V. **The phenomenon of science:** a cybernetic approach to human evolution. New York: Columbia University, 1977.

WIRFS-BROCK, R.; JOHNSON, R. E. **Surveying current research in object-oriented design**. New York: Communications of the ACM, set. 1990, v.33, n. 9.

WIRFS-BROCK, A. *et al*. **Designing reusable designs:** experiences designing object-oriented frameworks. In: OBJECT-ORIENTED PROGRAMMING SYSTEMS, LANGUAGES AND APPLICATIONS CONFERENCE; EUROPEAN CONFERENCE ON OBJECT-ORIENTED PROGRAMMING, 1991, Ottawa. Addendum to the proceedings. Ottawa: [s.n.], 1991.

WOOLDRIGE, M.; JENNINGS, N. **Intelligent agents:** theory and practice. [S.l.]: Knowledge Engineering Review, 1994.

YOURDON, E. **Analise baseada em objetos**. 2. ed. rev. Rio de Janeiro: Campus, 1991.