UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA ELÉTRICA E COMPUTAÇÃO
DEPARTAMENTO DE COMPUTAÇÃO E AUTOMAÇÃO

Dissertação submetida à Faculdade de Engenharia Elétrica e Computação da Universidade Estadual de Campinas, para preenchimento dos pré-requisitos parciais para obtenção do Título de Mestre em Engenharia Elétrica

Kards: Ambiente para desenvolvimento de aplicações inteligentes para o gerenciamento das atividades de um ambiente especializado

Autor : Marcos Paulo Ferreira Rebello
Orientador: Prof. Dr. Armando Freitas da Rocha

Julho de 1997

UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA ELÉTRICA E COMPUTAÇÃO
DEPARTAMENTO DE COMPUTAÇÃO E AUTOMAÇÃO

Tese de Mestrado

Título da tese: **Kards: Ambiente para desenvolvimento de aplicações inteligentes para o gerenciamento das atividades de um ambiente especializado**

Autor: **Marcos Paulo Ferreira Rebello**

Orientador: **Prof. Dr. Armando Freitas da Rocha**

Banca Examinadora:

Prof. Dr. Armando Freitas da Rocha (Presidente)

Prof. Dr. Márcio Luiz de Andrade Netto

Prof. Dr. Ricardo José Machado

"Se o cérebro humano fosse tão simples que pudéssemos entendê-lo, seríamos tão simples que não o entenderíamos."

Lyall Watson

Agradecimentos

Ao Prof. Armando Freitas da Rocha, com o qual diversas pessoas já o confundiram como sendo meu pai pela forma com que me trata.

Novamente ao Prof. Armando Freitas da Rocha, pela orientação.

Ao apoio dos meus pais, Antonio e Maria Tereza que contribuíram para a continuação de meus estudos.

Aos colegas Ivan Rizzo Guilherme e Adriane Beatriz de Souza Serapião pelas contribuições no desenvolvimento deste trabalho.

Ao colega e amigo Antonio Rodrigues Patricio por todo seu apoio e colaboração.

Aos colegas Edson Bolonhini, Benno, Eugênio e Pedro pelo apoio principalmente nos testes em campo.

Aos colegas da Petrobrás que através do uso do sistema desenvolvido neste trabalho forneceram uma valiosa contribuição para a sua realização.

Ao pessoal do DEP/FEM pela gentileza com que sempre me atenderam.

À minha irmã Ana Laura pela ajuda na redação desta tese.

E por fim, à FAPESP, por financiar parte desta pesquisa.

Resumo

O Kards tem como objetivo ser um ambiente para programação inteligente de atividades em um ambiente especializado onde se utilize um conhecimento específico. Desenvolvido como um sistema para processamento de Linguagens Formais Nebulosas suportado pelo conceito de Sistema Distribuído, onde agentes primitivos se encarregam da implementação das propriedades básicas do sistema e agentes complexos são criados como equipes de agentes primitivos para processar uma Linguagem Formal Nebulosa. Esta Linguagem Formal Nebulosa é aquela requerida para implementar uma aplicação específica, isto é, aquela utilizada para processar o conhecimento especializado que define uma aplicação. As sentenças desta Linguagem Formal Nebulosa são armazenadas na base de dados sob uma sintaxe de redes. O sistema Kards possui ferramentas para: a) implementação e manuseio de bases de dados; b) representação do conhecimento especializado através de redes de processamento e redes de raciocínio; c) simulação do raciocínio especializado através da navegação das redes de raciocínio; d) aquisição sensorial de dados através de sistemas de conversão analógico-digital, *scanner* e mesa digitalizadora, e) reconhecimento sensorial de padrões através de paradigmas de aprendizado e reconhecimento sintático de padrões; f) aprendizado e processamento do jargão utilizado para comunicação em ambientes especializados; g) atuação inteligente no ambiente externo dentro dos paradigmas de controle nebuloso através do uso de conversão digital-analógico. O Kards é um sistema auto-referenciado, pois utiliza a própria sintaxe do sistema para descrever suas estruturas básicas. É apresentado um exemplo de aplicação na área de engenharia de petróleo.

Palavras-Chaves: Linguagem Nebulosa, Sistemas Especialistas, Sistemas Distribuídos, Inteligência Artificial

Abstract

KARDS provides a computational environment for intelligent programming of activities supported by expert knowledge. It is a tool for Fuzzy Formal Language processing using the paradigm of Distributed Processing Systems, where primitive agents are in charge of handling the basic system's basic rules and complex agents are created as teams of these primitive agents in charge of processing a given Fuzzy Formal Language. This language is the one required to implement the desired application, that is the one supporting the expert knowledge used in the application. The sentences of this Fuzzy Formal Language compose a Data Base and a network syntax is used for storing purpose. The systems provides tools for a) Data Base handling and implementation; b) knowledge representation by means of processing and reasoning networks; c) expert knowledge simulation by means of network navigation; d) sensory data acquisition from A/D devices, scanners and tablets; e) sensory pattern recognition by means of syntactical and network paradigms; f) learning and processing the restricted language (jargon) used in the expert environment, and g) intelligent action over the external environment using the paradigm of fuzzy control. KARDS is a self-referred system, because it takes profit of its own syntax to represent its basic structures. An example of application in the oil industry is presented.

Keywords: Fuzzy Language, Expert Systems, Distributed Systems, Artificial Intelligence

Índice

Lista de Figuras

Lista de Tabelas

1. Introdução

1.1. Sistemas Especialistas

Nossa cultura tem privilegiado a lógica como a ferramenta mais importante de raciocínio por quase três mil anos. Consequentemente, é perfeitamente natural a predominância do logicismo na Inteligência Artificial (IA) e a proposição de uma clara separação entre conhecimento e inferência, que suporte a crença na existência de uma linguagem geral (lógica, é claro) para ser usada na implementação de uma máquina de raciocínio universal. De acordo com este ponto de vista, a solução para um problema requer somente uma completa e consistente descrição do domínio do problema por meio do conhecimento declarativo composto por um conjunto de sentenças do tipo

se é verdadeira a composição (conjunção/disjunção) das informações fornecidas por distintas fontes de informação então a decisão é D;

porque a inferência lógica é só o que é necessário para raciocinar usando esta base de conhecimento. A solução do problema é alcançada manuseando binariamente uma dimensão de incerteza (a verdade é a incerteza de comparação da informação atual com uma informação esperada) através de operadores booleanos de sentença. A base de conhecimento sobre o domínio do problema tem que ser adquirida ad hoc, porque completude e consistência não suportam qualquer aprendizado real, subentendido aqui, como a descoberta de novas peças de informação sobre o domínio do problema ainda não incorporados no conjunto inicial de axiomas (Base de Conhecimento).

Por um lado, completude e consistência implicam fechar o mundo observável, isto é, requerem um estudo de um universo estático, uma noção que a ciência moderna não mais aceita desde que o mundo newtoniano foi substituído pela teoria da relatividade, e Darwin salientou as características evolucionárias da vida. Qualquer teoria científica deve ser falseável de acordo com Popper, de modo a ser substituída por uma nova hipótese sempre que sua quantidade de falhas em explicar o mundo observável tornar-se grande [KoHe81]. Por outro lado, pesquisas sobre aquisição de conhecimento têm revelado algumas propriedades interessantes do raciocínio especialista que contradizem algumas das proposições clássicas do logicismo [Roc92]. A tomada inteligente de decisão envolve um processamento em um espaço de incerteza multi-dimensional. Relevância é uma medida da incerteza estatística em um mundo

observável parcialmente aberto e utilidade é uma medida de custo/benefício associada ao processamento da informação. Em geral, os especialistas usam conhecimento do tipo [Kac86]:

*se a **maioria** das fontes **relevantes** fornecerem informação*
*com **confiança** c então a **confiança** na decisão D é c'*

Além disso, o raciocínio pode ser aproximado porque a soma da confiança na proposição *p* e na sua complementar *não p* pode assumir valor menor que um, ou envolver conflito se esta soma for maior que um. O universo real torna-se mais complexo que o mundo dos pingüins. Novos tipos de lógica foram propostas como tentativas para o tratamento do problema de aumento da complexidade.

1.2. Redes Neurais

A descoberta do cérebro como a máquina biológica que suporta o raciocínio humano é um dos mais excitantes empreendimentos da ciência moderna. A pesquisa sobre os mecanismos envolvidos no processamento neural e aprendizado tem sido fonte de inspiração para outro importante paradigma da IA, chamado conexionismo. Este paradigma foi o mais importante no início da IA, e sua popularidade na comunidade aumentou novamente nos anos oitenta. De acordo com este ponto de vista, conhecimento é um conjunto de relações estabelecidas entre dispositivos (neurônios) com uma capacidade de processamento limitada e semelhante. Conhecimento pode ser aprendido pela modificação de conexões entre neurônios guiada por regras de punição/recompensa. Se **m** fontes $\mathbf{s_i}$ fornecem a informação de entrada $\mathbf{v_i}$ para um neurônio $\mathbf{n_j}$ com relevância $\mathbf{w_i}$ cada, então sua saída $\mathbf{s_j}$ é obtida como (e.g., Hinton, 1989):

(1) $$\mathbf{a = \Sigma\, v_i * w_i}$$

(2) $$\mathbf{s_j = \mathit{f}\,(a)}$$

isto é, $\mathbf{s_j}$ é uma função f da média ponderada **a** das entradas $\mathbf{s_i}$. f é em geral uma função booleana do tipo

(3) $$\mathbf{s_j} = \begin{cases} \mathbf{1 \text{ se } a > \alpha} \\ \mathbf{0 \text{ caso contrário}} \end{cases}$$

mas ela pode ser definida como qualquer outra função de filtragem. A relevância $\mathbf{w_i}$ é o peso da conexão (sinapse) entre o i-ésimo neurônio pré-sináptico (fonte de informação) e o neurônio pós-sináptico $\mathbf{n_j}$ (dispositivo processador). Procedimentos de aprendizado (*backpropagation* foi o mais famoso deles) modificam pesos sinápticos de acordo com o sucesso (recompensa) ou falha (punição) de $\mathbf{n_j}$ em fornecer a solução para o problema presente

estudado [Hin89] [Roc92]. Nesta condição $\mathbf{w_i}$ é uma medida da incerteza estatística da contribuição de $\mathbf{s_i}$ para a solução do problema. Neurônios são usualmente organizados em camadas para formar redes especializadas em resolver problemas particularmente complexos. Se neurônios da camada de entrada (sensorial) forem programados para comparar valores atuais **x** de variáveis específicas com o conhecimento prototipado **x'** sobre a conduta dessas mesmas variáveis, então $\mathbf{s_i}$ fornece uma medida de incerteza da comparação $\mathbf{x \cong x'}$. Se $\mathbf{s_i}$, $\mathbf{s_i'}$ fornecem as confianças em $\mathbf{x \cong x'}$ e $\mathbf{x \cong}$ **não** $\mathbf{x'}$, respectivamente, então um conflito é estabelecido se $\mathbf{s_i + s_i' > 1}$ e a ignorância é definida se $\mathbf{s_i + s_i' < 1}$. As propriedades de filtragem de f podem ser usadas para codificar a noção de maioria de, e média pode ser aceita como um cálculo de consenso para resolver conflitos. Embora redes neurais desfrutem dessas propriedades interessantes, elas são criticadas por terem baixo poder de expressão, porque não existe uma maneira fácil de descrever claramente o conhecimento codificado na conectividade da rede usada para resolver o problema estudado.

1.3. Sistemas Especialistas x Redes Neurais

Logicismo e coneccionismo têm sido usados pela comunidade de IA como métodos antagônicos para explicar a inteligência [AnRo89] [CGA88] [Gal88] [Hin89] [MiPa69] [Ros58]. Esta competição tem resultado em uma alternância da popularidade de uma dessas teorias como ferramenta para modelar a mente humana [Roc92]. Nesta competição, conexionismo é acusado de não ser capaz de manusear o raciocínio simbólico adequadamente, ao passo que lógica é acusada de confiar na heurística porque ela não pode manipular aprendizado e incerteza adequadamente [Bir91] [Fox81] [Kir91] [Nil91] [Roc92].

De acordo com [Kir91], as seguintes suposições básicas têm se tornado foco de debate e servem como linha divisória da competição entre essas duas abordagens da IA:

a) preeminência de conhecimento e conceitualização: inteligência que transcende o nível de inteligência dos insetos requer conhecimento declarativo e alguma forma de raciocínio - como computação;

b) desincorporação: pressupõe que cognição e o conhecimento podem ser estudados com abstração de detalhes da percepção e controle motor;

c) a dinâmica da cognição tem propriedades semelhantes a da linguagem: é possível descrever o estado de conhecimento ou estado de informação criados durante a cognição usando um vocabulário muito mais parecido com a linguagem natural ou alguma versão lógico-matemática arregimentada desta linguagem;

d) aprendizado pode ser incluído posteriormente: a dinâmica e o domínio do conhecimento necessário para a cognição podem ser estudados separadamente do estudo do aprendizado e desenvolvimento psicológico, e

e) arquitetura uniforme: existe uma arquitetura única que suporta toda cognição.

O logicismo aceita a visão sedutora de que cognição é inferência e que a inteligência depende apenas de uma base de conhecimento declarativo e um mecanismo de inferência [Kir91]. O mecanismo de inferência é relativamente simples e formalizado dentro de uma lógica definida, sendo considerado um problema independente do domínio do problema. A base de conhecimento é específica do domínio e tão complexa quanto a habilidade cognitiva requeira. Embora seja sabido que nem toda habilidade que requeira um controle inteligente exija uma base de conhecimento declarativa [Kir91], o logicismo muitas vezes nega a utilidade de qualquer conhecimento procedural (ou compilado) suportando cognição, apesar do fato de que especialistas tiram grande proveito do raciocínio procedural [Roc92].

O conexionismo, em contraste, assume que o conhecimento seja codificado nas relações compartilhadas por simples dispositivos de processamento e que o aprendizado é capaz de modificar a relevância dessas conexões, de modo que unidades internas que não são partes da entrada e da saída, venham a representar características importantes no domínio da tarefa [Hin89]. Nesta condição, a topologia do mecanismo de processamento é contexto-dependente ou domínio-específica. Por causa disto, o aprendizado pode ser usado para compilar o conhecimento procedural envolvido na resolução de problemas inteligentes [Roc92].

1.4. Sistemas Distribuídos

A falha tanto do logicismo quanto do conexionismo proporcionarem uma explicação decisiva ou um paradigma universal de inteligência e aprendizado tem estimulado a IA a buscar por outras formas de criar sistemas de inteligência artificial. Dentre essas novas proposições pode-se comentar que Modelos Evolucionários e Distribuídos tem ganho popularidade ultimamente.

Sistemas de Resolução Distribuída de Problemas Inteligentes, aqui chamado SRDPI, são definidos como uma coleção de agentes (inteligentes ou não) que interagem para resolver um problema em um domínio de conhecimento definido [Cha81] [CKLM92] [DaSm83] [DuMo91] [Fox81] [Hew77] [HeIn81] [KoHe81] [LeCo81] [Les91]. De acordo com esta abordagem, a solução de problemas pode ser decomposta em um conjunto de soluções de sub-tarefas (problemas), e agentes SRDPI são designados para resolverem essas sub-tarefas. As sub-tarefas podem ser de qualquer ordem, e.g., atividades de banco de dados, dedução de microteorias, controle de dispositivos, aquisição de dados, etc. Como sub-tarefas supostamente não necessariamente requerem solução inteligente, esta propriedade depende principalmente na maneira pela qual os agentes interagem para resolver o problema em questão. Basicamente, agentes perguntam e fornecem informações de e para outros agentes. O fluxo de informação pode ser especificado por meio de sistemas de correio, quando pelo menos um agente conhece precisamente o endereço do outro, ou por meio de difusão de mensagem, quando agentes selecionam mensagens que lhe interessam baseados em seu contexto e essas mensagens são distribuídas no sistema por meio de sistemas não especificados como quadro de avisos, canais paralelos de informação, etc. Implícita em ambas as estratégias é que comunicação é suportada por um conjunto de relações específicas ou genéricas. Em

ambos os casos, algumas noções de conexionismo são assumidas. Uma maneira de organizar fluxo de informações SRDPI é redes de contrato. Contudo, agentes não são proibidos de usar lógica para cuidar de obrigações relativas ao SRDPI. Por isso, os paradigmas distribuídos parecem tentar tirar proveito das melhores qualidades das teorias logicistas e conexionistas.

Centrais na noção de usar um conjunto de agentes para obter uma solução inteligente para um dado problema, são os conceitos de conflito, consenso e negociação. Conflito pode ser gerado, pois cada agente pode agir de maneira independente na busca da solução das tarefas para os quais são especializados. Consenso pode ser usado como uma forma de resolver conflito se informações opostas exibirem diferentes graus de confiança e/ou relevância que permitam uma avaliação adequada do problema. A negociação pode ser usada para modificar a confiança e/ou relevância de agentes. Hierarquia é outro processo a ser usado na solução de conflito e negociação. Contudo a hierarquia deve ficar dentro dos limites, caso contrário a inteligência do sistema será reduzida àquela (se existente) do agente mais alto na hierarquia do poder. A melhor solução é alguma ordem de democracia hierarquizada, onde cada agente goza de um certo grau de liberdade enquanto usar esta liberdade em benefício do sistema como um todo. A inteligência do SRDPI deriva, portanto, da combinação adequada de fatores distintos tais como especialização de agentes, comunicação, conflito, negociação, hierarquia, autonomia, etc.

A complexidade em desenhar um SRDPI força o desenvolvimento de Sistemas Evolucionários (SE). A idéia central aqui, é que tanto a especialização de agentes quanto as interações requeridas para resolver um problema são obtidas através de um processo interativo guiado por regras de punição/recompensa, cujos propósitos são selecionar os agentes mais adequados e os processos de comunicação mais eficientes para resolver o problema em consideração. Algoritmo Genético (AG) foi a primeira ferramenta evolucionária a obter sucesso entre os cientistas de IA. As características chaves do AG são: 1) codificar a solução do problema em cadeias; 2) atribuir um mérito a cada cadeia de acordo com sua contribuição para a solução do problema; 3) gerar variabilidade de cadeias através de mutação e recombinação gênica das cadeias de maior sucesso; e finalmente, 4) selecionar as cadeias de maior sucesso em cada geração para compor um sistema inteligente final. AG foca a atenção na geração de agentes especializados através de um processo combinatorial de gênese de cadeias e não dá a atenção devida aos problemas de comunicação entre esses agentes. Um passo nessa direção é a proposição do tipo feita por Booker [BGH89] para implementar o processo de raciocínio em sistemas de classificação. Nesta abordagem, regras são codificadas por cadeias complexas que tem sua porção esquerda (sub-cadeia de entrada) descrevendo os antecedentes da regra e sua porção direita (sub-cadeia de saída) codificando o conseqüente. Além disso, uma interface codifica a informação externa em cadeias compatíveis, gerando mensagens em um quadro de aviso. Cada nova mensagem é comparada com as sub-cadeias de entrada de todas as regras na base de conhecimento, e aquelas ativadas pela similaridade de cadeias escrevem mensagens (sub-cadeia de saída) no quadro de aviso. Créditos são atribuídos à essas regras em função de sua contribuição para o sucesso da solução dos problemas. AG é usado para gerar novas regras sempre que necessário. Darwinismo Neural [Ede87] é outro sistema SE, que assume a existência de uma população muito ampla de conjuntos de neurônios compartilhando uma variabilidade de conexões muito grande. Seleção Natural é o processo de aprendizado responsável pela seleção desses conjuntos de neurônios com maior sucesso na solução de

problemas definidos. Outra idéia popular na teoria do SE, é a de distribuir marcas entre agentes como parte do processo de recompensa, marcas essas que podem ser usadas como moeda para negociações entre agentes. Agentes ricos podem adquirir privilégios de expansão, mas devem ser proibidos de explorar outros agentes. A ecologia é algumas vezes usada como uma metáfora rica em SE, desde que predadores e presa possam coexistir em um ambiente adaptado.

1.5. Sistemas Neurais Distribuídos

O passo atual de aquisição de conhecimento em Neurociência é tremendo e demanda o desenvolvimento de ferramentas matemáticas adequadas à serem usadas tanto na análise de dados experimentais coletados, quanto na construção de poderosos modelos teóricos do cérebro. Rocha [RFB80] propôs o uso da teoria de Linguagens Formais Nebulosas (LFN) para modelar o comportamento de Sistemas Neurais (cérebro) de acordo com o conhecimento fornecido pela biologia. Quase duas décadas de pesquisas nesta linha resultaram na proposição de considerar em Sistema Neural (**SN**) como um Sistema de Resolução Distribuída de Problemas Inteligentes compostos por agentes (neurônios) que trocam mensagens especializadas, suportado por uma **LFN** para resolver tarefas específicas (inteligentes ou não) necessárias para garantir a sobrevivência do sistema como um todo em um ambiente variável ou universo **U**. Agentes organizam a si mesmos em times especializados de acordo com seu sucesso em fornecer soluções para problemas que podem ser decompostos em sub-problemas para as quais esses agentes são especializados [PeRo93] [PLR95] [Roc92].

O cérebro é considerado, portanto, como um SN formado por uma coleção de neurônios e células gliais que trocam mensagens químicas e elétricas na tentativa de resolver problemas impostos por mudanças no ambiente. Essas mensagens são ou cadeias químicas ou palavras de um código elétrico suportadas por uma gramática nebulosa [Roc92] [SRRP96]. Em resumo, transmissores e neuropeptídeos soltos pela célula pré-sináptica ligam-se ao receptor pós-sináptico (a) para ativar vias enzimáticas que controlam ou a leitura do DNA ou processos metabólicos, ou (b) para controlar canais iônicos para modular correntes iônicas e a atividade elétrica da membrana pós-sináptica. Desta forma, as transações na sinapse são assumidas como sendo descritas por um conjunto de processamentos suportados por uma dada **LFN**. Esta linguagem nebulosa sináptica (**LNS**) é definida como

$$\mathbf{LNS} = \{ S_o, S_r, S_i, S_t, G \} \tag{4}$$

onde: S_o é o conjunto das mensagens (hormônios, neuropeptídeos ou transmissores) soltos pelo agente pré-sináptico (neurônios ou células gliais) e reconhecidos pelo conjunto de receptores pós-sinápticos S_r , e S_i e S_t são, respectivamente, os conjuntos de mensagens intermediárias e terminais produzidas pela seqüência de derivação ativada por S_o de acordo com as regras especificadas por **G**. Neste caso, a possibilidade de que a mensagem $s_o^i \in S_o$ solta pelo agente pré-sináptico n_i ative o neurônio pós-sináptico n_j para produzir uma mensagem $s_i^j \in S_t$ é dada por

(5) $$\rho(d(s_o^i, s_i^j)) = \Phi (\Gamma (A(s_o^i), A(s_r^j), \mu(s_o,s_r)))$$

$$\Gamma : A(s_o^i) \times A(s_r^j) \times \mu(s_o , s_r) \rightarrow [0;1]$$

onde $A(s_o^i)$, $A(s_r^j)$ são, respectivamente, a quantidade total de cadeias s_o^i disponíveis em n_i e s_r^j disponíveis em n_j. O número $A(s_i^j)$ de cópias da cadeia s_i^j ativada por n_j é calculada como

(6) $$A(s_i^j) = \partial (A(s_o^i), \rho(d(s_o^i, s_r^j)))$$

onde $A(s_o^i)$ é a quantidade real de cadeias s_o^i soltas por n_j. Agora, o conjunto real S_t^j de mensagens terminais produzidas por n_j, é obtido do conjunto S_i^j de cadeias intermediárias ativadas por todos **m** de seus agentes pré-sinápticos:

(7) $$S_t^j = \{ s_i^j \rightarrow s_t^j \text{ se } \rho(d(s_i^j, s_t^j)) > 0 \}_{i=1 \text{ até } m}$$

$$A(s_t^j) = \partial (A(s_i^j), \rho(d(s_i^j, s_t^j)))$$

No caso em que todos os neurônios pré-sinápticos utilizam somente um tipo de mensagem a ser reconhecido por apenas um tipo de receptor pós-sináptico, a equação (5) torna-se uma medida do peso sináptico definido por McCulloch e Pitts [McPi49] e as equações (6) e (7) podem ser combinadas para calcular a atividade axônica a_j em n_j como

(8) $$a_j = f (\sum_{i=1}^{m} a_i * w_i)$$

onde f é a função de filtro de McCulloch e Pitts. Conseqüentemente, o neurônio de McCulloch e Pitts é um caso especial do neurônio formal nebuloso definido acima. Nesta linha de raciocínio, a Teoria de **Redes Neurais** clássica vem a ser um tipo especial de **Sistema Neural**.

1.6. O Sistema Kards

Junto com essas novas propostas (LFN, Sistemas Neurais, etc.) de paradigmas para implementação de sistemas inteligentes desenvolvidas pelo nosso grupo de pesquisa, surgiu a necessidade de testar essas novas idéias. Por isso, a necessidade do desenvolvimento do ambiente computacional adequado. O propósito da presente tese é, portanto, a criação do sistema Kards, construído como sendo um manipulador de LFN e onde aplicações inteligentes utilizando essas novas propostas podem ser implementadas a partir da definição da LFN que as

suportam. Com a criação dessas aplicações pode-se demonstrar tanto a veracidade dessas novas propostas quanto a viabilidade computacional destas.

A presente tese está assim organizada: no capítulo 2 o objetivo do presente trabalho e a estrutura geral são apresentados; no capítulo 3 é descrita a noção de Linguagens Formais Nebulosas utilizada na implementação do Kards; o capítulo 4 descreve o sistema Kards funcionalmente, enquanto que o capítulo 5 possui uma descrição básica de sua implementação; no capítulo 6 foi descrita um aplicação desenvolvida no sistema Kards, onde pode ser observada a utilização de diversos conceitos de forma prática; e o capítulo 7 apresenta as conclusões do presente trabalho.

2. Objetivo

2.1. O Sistema

O Kards tem como objetivo ser um ambiente para programação inteligente de atividades em um ambiente especializado onde se utilize um conhecimento específico (e.g. diversas especialidades médicas, diferentes áreas de engenharia, etc.). Para tanto, foi desenvolvido como sendo um sistema para processamento de LFNs suportado pelo conceito de Sistema Distribuído, onde agentes primitivos se encarregam da implementação das propriedades básicas do sistema e agentes complexos são criados como equipes de agentes primitivos para processar uma LFN definida. Esta LFN é aquela requerida para implementar uma aplicação específica, isto é, aquela utilizada para processar o conhecimento especializado.

2.2. Estrutura Geral

Desenvolvido como uma Linguagem Formal Nebulosa, o sistema Kards possui ferramentas para:

a) implementação e manuseio de bases de dados;

b) representação do conhecimento especializado (redes de raciocínio, funções nebulosas, etc.);

c) simulação do raciocínio especializado através da navegação das redes de raciocínio;

d) processamento sensorial de dados adquiridos do ambiente externo via sistemas A/D, *scanner* e mesa digitalizadora, através de paradigmas de aprendizado e reconhecimento sintático de padrões;

e) aprendizado e processamento do jargão utilizado para comunicação em ambientes especializados;

f) atuação inteligente dentro dos paradigmas de controle nebuloso.

O sistema Kards compõe-se de um manipulador de Linguagem Formal Nebulosa (LFN) que utiliza a noção de Redes Nebulosas para implementar o processamento de uma LFN, que especifica o raciocínio a ser implementado no sistema utilizando as sentenças desta LFN que descrevem o conhecimento especializado armazenado na base de dados sob uma sintaxe de redes. Essas sentenças descrevem as atividades do ambiente especializado a ser implementado no Kards - chamadas *aplicação*.

O Kards é um sistema auto-referenciado, pois utiliza a própria sintaxe (LFN) do sistema para descrever suas estruturas básicas (a LFN do sistema em si). Isto é, as sentenças da LFN do Kards que definem a sintaxe da LFN do Kards são armazenadas em sua base de dados, que é também descrita por estas sentenças.

3. Linguagens Formais Nebulosas

3.1. Definição

A teoria das linguagens formais foi introduzida como um formalismo para suportar a análise da linguagem humana bem como da linguagem artificial usada em computadores. De acordo com esta teoria, uma gramática **G** [Cho65] [Miz73] [NeRa75] [RFB80] [Roc92] [Sea93] é uma estrutura definida como

(9) $$\mathbf{G = \{ V_s, V_n, V_t, P, \eta)}$$

onde:

a) $\mathbf{V_s}$: é um conjunto de símbolos iniciais;
b) $\mathbf{V_n}$: é um conjunto de símbolos não-terminais
c) $\mathbf{V_t}$: é um conjunto de símbolos terminais;
d) η : é o elemento vazio, e
e) **P**: é o conjunto de regras **p** de reescrita definidas como

(10) $$\mathbf{p : \alpha\, s_i\, \beta \rightarrow \alpha\, s_j\, \beta}$$
$$\mathbf{\alpha , \beta , s_j \in V_s \cup V_n \cup V_t \cup \eta \text{ e } s_i \in V_s \cup V_n}$$

Em outras palavras, $\mathbf{p \in P}$ reescreve a cadeia $\mathbf{s_i}$ como a cadeia $\mathbf{s_j}$. $\mathbf{s_i}$ é definida como uma cadeia de símbolos de $\mathbf{V_s \cup V_n}$, que é a união do conjunto de símbolos iniciais e símbolos não-terminais. $\mathbf{s_j}$ é definida como uma cadeia de símbolos da união de todos os conjuntos de símbolos, que é $\mathbf{V_s \cup V_n \cup V_t \cup \eta}$. α e β denotam cadeias contextuais, isto é $\mathbf{s_i}$ é reescrito em $\mathbf{s_j}$ no contexto definido por α e β. Para simplificar, atribui-se

(11) $$\mathbf{V^+ = V_S \cup V_n \text{ e}}$$
$$\mathbf{V^* = V_s \cup V_n \cup V_t}$$

A seqüência de derivação $\mathbf{d(s_o,s_m)}$ da cadeia $s_m \in V^*$ de **G** é o conjunto ordenado de reescrições requeridas para transformar o símbolo inicial $s_o \in V_s$ em s_m. Em outras palavras,

(12) $$\mathbf{d(s_o,s_m)} = \alpha\ s_o\ \beta \rightarrow \alpha\ s_1\ \beta \rightarrow \ldots\ldots \alpha\ s_k\ \beta \rightarrow \alpha\ s_m\ \beta$$

Uma linguagem formal **L** é definida como um subconjunto de cadeias geradas por uma gramática **G.** As cadeias geradas por **G** e aceitas como pertencentes à **L** são chamadas **fórmulas bem formadas (fbf)** de **L** de acordo com **G**. A cadeia s_j produzida por **G** é uma **fbf** de **L** se ela pertencer a V_t. Em outras palavras, a cadeia s_m aceita pela linguagem **L(G)** suportada por **G** são aquelas $\mathbf{fbf(s_o,s_m)}$ obtidas em

(13) $$\mathbf{fbf(s_o,s_m)} = \mathbf{d(s_o,s_m)} = s_o \rightarrow \alpha\ s_1\ \beta \rightarrow \ldots\ \alpha\ s_k\ \beta \rightarrow s_m\ ,\ s_m \in V_t$$

Assim, a sentença $\alpha\ s_o\ \beta$ é aceita como uma sentença de **L** somente se existir pelo menos uma $\mathbf{fbf}(\alpha\ s_o\ \beta,\ \alpha\ s_m\ \beta)$ para reescrever ela em $\alpha\ s_m\ \beta$ pertencente à V_t.

O processamento de qualquer sentença da seqüência de derivação $\mathbf{d(s_o,s_m)}$ é um conjunto **seqüencialmente ordenado** de operações de reescrita, cada uma envolvendo os seguintes passos:

1) **Comparação**: o lado esquerdo da provável regra de reescrita é comparado com os símbolos da cadeia s_i a ser processada. Se esta comparação for bem sucedida, então

2) **Reescrita:** a subcadeia selecionada em s_i é substituída pelo lado direito da regra de reescrita aceita. Finalmente;

3) **Aceitação:** a pertinência da cadeia derivada s_j a V_t é calculada como uma medida definida no intervalo fechado **[0,1].**

Agora, assumindo as seguintes definições [SRRP96]:

a) O **grau de similaridade** (comparação) $\mu(\ s_i\ ,\ s_j\ )$ de duas cadeias s_i , s_j é uma medida do espaço cartesiano $(V^*)^l$ para o intervalo fechado **[0,1]**, onde **l** é o comprimento da maior dessas cadeias. Em outras palavras,

(14) $$\mu : (V^*)^l \rightarrow [\ 0,\ 1\ ]$$

$\mu(s_i,s_j) \rightarrow 1$ se s_i tende a ser igual a s_j; $\mu(s_i,s_j) \rightarrow 0$ caso contrário

b) O **grau de aceitação** $\mu(s_j,V_t)$ de s_j como uma **fbf** de **L** é calculado como o máximo grau de similaridade $\mu(s_j,s_t)$ de s_j para a cadeia $s_t \in V_t$. Em outras palavras,

(15) $$\mu(s_j,V_t) = \vee_{V_t} \mu(s_j,s_t)$$

As restrições em (a) e (b) definem **L** como uma linguagem nebulosa [Miz73] [NeRa75] [SRRP96] [RFB80].

Agora, se

a) $\mu(s_i,s_j)$ em (14) é assumido como sendo uma função binária assumindo um valor sendo 1 ou 0, isto é

(16) $$\mu: (V^*)^l \rightarrow (0,1)$$

$\mu(s_i,s_j) = 1$ se s_i igual a s_j ; $\mu(s_i,s_j) = 0$ caso contrário

e

b) $\mu(s_i,V_t)$ em (15) é assumido como sendo similar a uma função identidade, isto é

(17) $\mu(s_j,V_t) = 1$ se $s_j \in V_t$, caso contrário $\mu(s_j,V_t) = 0$

então, **L** torna-se uma linguagem clássica do tipo discutido por Chomsky [Cho65], e outros [Sea93], e aqui será chamada de LFN booleana.

3.2. Ambigüidade

Linguagens nebulosas exibem propriedades distintas relativas às linguagens clássicas porque, para qualquer $s_i \in V^*$, podem existir muitos $s_k \in V^*$ tal que $\mu(s_i,s_k) > 0$. Isto porque muitas regras de derivação $\alpha\ s_k\ \beta \rightarrow \alpha\ s_j\ \beta$ podem permitir que se reescreva s_i em diferentes $s_j \in V^*$. Como a ambigüidade da sentença s_o de **L** depende de quantas cadeias de derivação $d(s_o,s_t)$ existam fornecendo $s_t \in V_t$, então as linguagens nebulosas são naturalmente ambíguas. O processamento das Linguagens Nebulosas irá requerer **operações concorrentes** em um **processo espacialmente distribuído**, pois muitas cadeias de derivação irão ser concomitantemente ativadas pelo mesmo conjunto de símbolos iniciais.

A ambigüidade das Linguagens Nebulosas pode ser mantida dentro de faixas aceitáveis por restrições impostas sobre:

1) **tamanho da seqüência de derivação $l(s_o,s_t)$:** o poder combinatorial das linguagens nebulosas pode ser controlado pela redução do número de passos de derivação permitido de qualquer **fbf** de **L**. Em outras palavras,

(18) $$l(s_o,s_t) < \tau$$

o valor efetivo de τ será um dos parâmetros determinando o grau **A(L)** de ambigüidade de **L.** Isto implica em restringir o tamanho das possíveis **fbf**s.

2) **cardinalidade de V_t:** outro importante parâmetro determinando o valor real de **A(L)** é o número vigente de cadeias em V_t, porque a possibilidade de qualquer cadeia s_i de ser uma **fbf** de **L** depende da similaridade $\mu(s_i,s_t)$ de s_i para qualquer $s_t \in V_t$. Deste modo, o número **A(L)** de possíveis **fbf**s de **L** é uma função da cardinalidade $C(V_t)$ de V_t:

(19) $$A(L) = f\ C(V_t)$$

Isto mantém o número de **fbf**s finito e o mais baixo possível.

3) **número total $A(s_k)$ de cópias disponíveis de $s_k \in V^+$:** pelo menos uma cópia da cadeia $\alpha\ s_k\ \beta$ deve estar disponível para que a regra $\alpha\ s_k\ \beta \rightarrow \alpha\ s_j\ \beta$ possa ser usada dado que $\mu(s_i,s_k) > 0$;

4) **número total $A(s_j)$ de cópias disponíveis de $s_j \in V^*$**: pelo menos uma cópia da cadeia $s_j \in V^*$ deve estar disponível para que $\alpha\ s_i\ \beta$ seja reescrito em $\alpha\ s_j\ \beta$ dado que $\mu(s_i,s_k) > 0$;

5) **número atual $A(\alpha\ s_i\ \beta)$ de cópias ativas da sentença $\alpha\ s_i\ \beta$ a ser processada:** pelo menos uma cópia da cadeia $\alpha\ s_i\ \beta$ deve ser selecionada (ativada) para ser reescrita.

Estas restrições implicam que tanto o tipo de símbolo quanto sua quantidade têm papel importante na determinação da reescrita de sentenças nebulosas. Esta é uma grande diferença entre as linguagens nebulosas e clássicas, que não impõem nenhuma restrição à disponibilidade de símbolos, assumindo suas quantidades como infinitas.

6) **prioridade de comparação**: sempre que não houver cópias de $A(\alpha\ s_i\ \beta)$ suficientes para disparar todas as possíveis regras de derivação $\alpha\ s_k\ \beta \rightarrow \alpha\ s_j\ \beta$, $\mu(s_i,s_k) > 0$, então deve ser dada prioridade para aquelas regras $\alpha\ s_k\ \beta \rightarrow \alpha\ s_j\ \beta$ que possuírem o maior valor de $\mu(s_i,s_k)$. Do mesmo modo, prioridade deve ser dada para aquelas regras suportadas por $s_k \in V^*$ tendo o maior número $A(s_k)$ de cópias disponíveis.

Do exposto acima, pode-se assumir que a **possibilidade** $\rho(d(s_i,s_j))$ de uso de uma seqüência de derivação $d(s_i,s_j)$, suportada por $\alpha\ s_k\ \beta \rightarrow \alpha\ s_j\ \beta$, $\mu(s_i,s_k) > 0$, $k=1,...,n$, para reescrever s_i, deve ser uma função (Φ) de $A(s_j)$; $A(s_k)$ e $\mu(s_i,s_k)$, isto é

$$(20) \qquad \rho(d(s_i,s_j)) = \Phi \left(\prod_{k=1}^{n} \Gamma \left(A(s_j), A(s_k), \mu(s_i,s_k) \right) \right)$$

$$\Gamma : A(s_j) * A(s_k) * \mu(s_i,s_k) \rightarrow [0;1]$$

O número $A(\alpha\ s_j\ \beta)$ de cópias da cadeia $\alpha\ s_j\ \beta$ produzida (ativada) por $\alpha\ s_i\ \beta$ deve ser assumido como sendo dependente de $A(\alpha\ s_i\ \beta)$ e $\rho(d(s_i,s_j))$:

$$(21) \qquad A(\alpha\ s_j\ \beta) = \partial \left(A(\alpha\ s_i\ \beta), \rho(d(s_i,s_j)) \right)$$

ou

$$A(\alpha\ s_j\ \beta) = \Phi \left(\prod_{k=1}^{n} \partial \left(A(\alpha\ s_i\ \beta), \rho(d(s_k,s_j)) \right) \right)$$

onde Φ, Γ e ∂ são ***t*** ou ***s*-norms** (Rocha,1992, Serapião et al., 1996). Uma ***t*-norm** é uma operação matemática satisfazendo os seguintes requisitos:

a) condição de limite: $t(a,1) = a$ e $t(0,0) = 0$

b) monotonicidade: $t(a,b) \leq t(c,d)$ se $a \leq c$; $b \leq d$

c) simetria: $t(a,b) = t(b,a)$

d) associatividade: $t(a, t(b,c)) = t(t(a, b),c)$

Uma ***s*-norm** é definida se

a) condição de limite: $s(a,0) = a$ e $s(1,a) = 1$

e as condições acima de monotonicidade, simetria e associatividade mantidas.

Neurônios podem, agora, ser considerados como agentes que trocam mensagens [Roc92]. Então uma classe de **SN**s pode ser definida suportada pelas **LFN**s. Neste caso, transmissores

liberados pelo neurônio pré-sináptico podem ser considerados como sendo símbolos iniciais s_i da seqüência de derivação a ser processada pela célula pós-sináptica. Desta forma, a célula pós-sináptica é ativada pelo neurônio pré-sináptico se houver uma regra pós-sináptica $\alpha\ s_k\ \beta \rightarrow \alpha\ s_j\ \beta$ para reescrever o símbolo pré-sináptico s_i na cadeia pós-sináptica s_j. A possibilidade de enviar informação de um neurônio para outro é, portanto, dependente não somente da quantidade disponível de s_j ,s_k, mas também de $\mu(s_i,s_k)$. Mostra-se que estes neurônios nebulosos possuem propriedades interessantes [PeRo93] [PLR95] [Roc92]. Agora se $\mu(s_i,s_k)$ é assumida sendo sempre um, porque a qualidade das mensagens trocadas não é considerada importante, então (20) torna-se dependente somente da quantidade numérica como o proposto pela teoria das Redes Neurais Clássica. Nesta linha de raciocínio, o neurônio definido em (21) pode ser reduzido àquele definido por McCulloch-Pits. Com isso, é possível considerar as Redes Neurais Clássicas como sendo uma família especial dos Sistemas Neurais Nebulosos.

3.3. Capacidade Lógica

Assumindo serem dadas as seguintes condições

(22)
$$\alpha\ s_i\ \beta \rightarrow \alpha\ s_j\ \beta \textbf{ dado } \alpha\ s_k\ \beta \rightarrow \alpha\ s_j\ \beta\ ,\ \mu(s_i,s_k) > 0$$
$$\alpha\ s_i\ \beta \rightarrow \alpha\ s_h\ \beta \textbf{ dado } \alpha\ s_m\ \beta \rightarrow \alpha\ s_h\ \beta\ ,\ \mu(s_i,s_m) > 0$$

pode-se considerar que as gramáticas nebulosas suportam diferentes classes de lógicas não-monotônicas, tais como:

a) **Lógica por Default**: porque a prioridade de comparação suporta seqüências de derivação do tipo

(23)
$$\mu(s_i,s_m) \rightarrow ,5 \Rightarrow \alpha\ s_i\ \beta \rightarrow \alpha\ s_h\ \beta$$
$$\text{a menos que } \mu(s_i,s_k) \rightarrow 1 \Rightarrow \alpha\ s_i\ \beta \rightarrow \alpha\ s_j\ \beta$$

isto é, dado $\mu(s_i,s_m) < \mu(s_i,s_k)$, s_i reescreve s_m a menos que s_k esteja disponível para reescrever s_i. Aqui, o símbolo $\Rightarrow$ indica *suportar*.

b) **Lógica de Recursos Finitos**: porque pode ser assumido que as cópias disponíveis $\mathbf{A}(s_k)$ de $s_k \in \mathbf{V}^*$ são decrementadas por uma quantia proporcional à quantidade $\mathbf{A}(\alpha\ s_j\ \beta)$ de sentenças copiadas $\alpha\ s_j\ \beta$ sempre que s_k reescreve s_i em (22) então

(24)
$$\alpha\ s_i\ \beta \rightarrow \alpha\ s_j\ \beta \text{ enquanto } \mathbf{A}(s_k) > 0$$

Isto implica que uma dada seqüência de derivação pode ser ativada enquanto recursos estiverem disponíveis.

c) **Lógica de Filtros**: porque pode-se assumir que a sentença $\alpha\ s_j\ \beta$ ou $\alpha\ s_h\ \beta$ pode ser produzida somente se o número de cópias disponíveis $\mathbf{A}(\alpha\ s_i\ \beta)$ de $\alpha\ s_i\ \beta$ em (22) está contido em um intervalo da função, isto é

(25) $$\theta < \mathbf{A}(\alpha\ s_i\ \beta) < \upsilon$$

d) **Lógica Temporal**: reações temporais podem ser suportadas por linguagens nebulosas, pois pode ser assumido que restrições de tempo podem ser impostas sobre a disponibilidade de cópias $\mathbf{A}(\alpha\ s_i\ \beta)$ da cadeia $\alpha\ s_i\ \beta$:

(26) $$\mathbf{A}(\alpha\ s_i\ \beta) = f(\ t\ )$$

De fato, as lógicas acima descritas podem ser consideradas como casos especiais de uma família geral de lógicas, chamada

e) **Lógica Nebulosa:** porque pode-se assumir que a sentença $\alpha\ s_i\ \beta$ é reescrita como

(27) $$\mathbf{Q}\ (\ \rho(\mathbf{d}(s_i,s_j)) \Rightarrow \alpha\ s_i\ \beta \rightarrow \alpha\ s_j\ \beta \vee \ldots \wedge \rho(\mathbf{d}(s_i,s_k)) \Rightarrow \alpha\ s_i\ \beta \rightarrow \alpha\ s_k\ \beta\ )$$
$$\mathbf{A}(\alpha\ s_j\ \beta) = f(\ \rho(\mathbf{d}(s_i,s_j)) \ldots\ldots \rho(\mathbf{d}(s_i,s_k))\ )$$

onde **Q** é um quantificador lógico (e.g., *a maioria da regras, **x** de **n** regras etc.)*; $\vee$ e $\wedge$ denotam **t** e/ou **s-normas**. $\rho(\mathbf{d}(s_i,s_j))$ é uma função de $\mathbf{A}(s_i)$, $\mathbf{A}(s_k)$, $\mu(s_i,s_k)$ e f pode ser definido como uma função de filtragem. A interpretação de (27) é

(28) *A cadeia* $\alpha\ s_i\ \beta$ *é reescrita por* ***Q*** *(regras relevantes* $\alpha\ s_k\ \beta \rightarrow \alpha\ s_j\ \beta \vee \ldots \wedge \ldots$ *) nas cadeias* $\alpha\ s_j\ \beta$

onde $\rho(\mathbf{d}(s_i,s_k))$ mede a relevância (ou suporte) da regras $\alpha\ s_k\ \beta \rightarrow \alpha\ s_j\ \beta$ para reescrever $\alpha\ s_i\ \beta$.

3.4. Aprendizado

As **LFN** têm capacidade de aprendizado, porque a quantidade disponível $\mathbf{A}(s_j)$, $\mathbf{A}(s_k)$ de $s_j,s_k \in \mathbf{V}^*$ em $\alpha\ s_k\ \beta \rightarrow \alpha\ s_j\ \beta$ para reescrever s_i em s_j em $\mathbf{d}(s_o,\ldots s_i,s_k,\ldots,s_t)$, bem como a estrutura s_i,s_k, pode ser considerada como sendo dependente do sucesso da regra de reescrita na contribuição para o processamento das **fbf**s de **L**, isto é, dependente do valor vigente de $\mu(s_t,V_t)$. Assim, os seguintes paradigmas gerais de aprendizado podem ser definidos [SRRP96]:

a) **recompensa**: incrementa $\mathbf{A(s_j)}$, $\mathbf{A(s_k)}$ sempre que $\mathbf{d(s_o,..s_i,s_k,...,s_t)}$ produzir (ativar) s_t para o qual $\mu(s_t,V_t)\rightarrow 1$. Este paradigma pode ser implementado por regras de cópia de cadeias do tipo:

(29) $$\rho\ s_j \rightarrow s_j\ ,\ \rho\ s_k \rightarrow s_k$$

no contexto de recompensa ρ.

b) **punição**: decrementa $\mathbf{A(s_j)}$, $\mathbf{A(s_k)}$ sempre que $\mathbf{d(s_o,..s_i,s_k,...,s_t)}$ ativar s_t para qual $\mu(s_t,V_t)\rightarrow,5$, $\mu(s_t,V_t)$ <,5. Este paradigma pode ser implementado por regras de cópia de cadeias do tipo:

(30) $$\pi\ s_j \rightarrow \eta\ ,\ \pi\ s_k \rightarrow \eta$$

no contexto de punição π. Aqui, η é o símbolo vazio, e (30) mostra a destruição (ou desativação) de s_j, s_k.

c) **reconsideração**: decrementa $\mu(s_i,s_k)$ sempre que $\mathbf{d(s_o,..s_i,s_k,...,s_t)}$ ativar s_t para qual $\mu(s_t,V_t)\rightarrow 0$. Este paradigma pode ser implementado por regras de cópia de cadeias do tipo:

(31) $$\lambda\ s_i \rightarrow s_i'\ ,\ \lambda\ s_k \rightarrow s_k'\ ,\ \mu(s_i,s_k) > \mu(s_i',s_k')$$

no contexto λ, que troca a estrutura interna das cadeias s_i, s_k para torná-las outras cadeias menos similares.

d) **reforço**: incrementa $\mu(s_i,s_k)$ sempre que $\mathbf{d(s_o,..s_i,s_k,...,s_t)}$ ativar s_t para qual $\mu(s_t,V_t)\rightarrow,5$, $\mu(s_t,V_t)$ >,5. Este paradigma pode ser implementado por regras de cópia de cadeias do tipo:

(32) $$\nu\ s_i \rightarrow s_i'\ ,\ \nu\ s_k \rightarrow s_k'\ ,\ \mu(s_i,s_k) < \mu(s_i',s_k')$$

no contexto ν, que troca a estrutura interna das cadeias s_i, s_k para torná-las outras cadeias mais similares.

Desta forma, tanto aprendizados quantitativo (a e b acima) quanto qualitativo (c e d acima) podem ser usados para reduzir a ambigüidade das linguagens nebulosas, porque **reconsiderar** e **reforçar** implicam modificação na estrutura do símbolos, enquanto **recompensa** e **punição** estão associadas à modificação das quantidades de símbolos. Todos os processos acima resultam em modificação da possibilidade $\rho(\mathbf{d(s_o,..s_i,s_k,...,s_t)})$ de disparar a seqüência de derivação $\mathbf{d(s_o,..s_i,s_k,...,s_t)}$, e podem ser usados para controlar a ambigüidade da linguagem. Aprendizado pode ser uma ferramenta poderosa no controle da ambigüidade da **LFN**, e esta propriedade da **LFN** é um componente chave para qualquer sistema inteligente, se inteligência

é assumida como sendo a capacidade de sistemas não-determinísticos de resolver novos problemas ou recusar propostas por uma contínua variação ambiental.

3.5. Cadeias Nebulosas Não Lineares

Outra característica importante da **LNF** é suportada pelas suas propriedades de comparação de cadeias, porque pode-se considerar que a comparação é uma operação a ser realizada em um espaço n-dimensional e linearidade não é um requisito necessário no processo de comparar símbolos de cadeias diferentes.

Consideremos o caso onde s_i, $s_j \in \mathbf{V}^*$ são cadeias do tipo

(33) $$s_i = \alpha\ s_p\ \{..\gamma..\}_k\ s_q\ \beta\ ,\ s_j = \alpha\ s_r\ \{..\lambda..\}_l\ s_s\ \beta$$

onde $\{...\}_n$ forma qualquer cadeia de tamanho **n**. Uma comparação não-linear entre $\mu(s_i,s_j)$ e s_i, $s_j \in \mathbf{V}^*$ é definida [SRRP96] no caso

(34) $$\mu(s_i,s_j) = f'(\ \mu(s_p,s_r),\ \mu(s_q,s_s),\ \mu(\ k, l\ ),\ \mu(\ \gamma,\ \lambda)\ )$$

isto é, a comparação entre s_i,s_j é uma função de comparações individuais $\mu(s_p,s_r)$, $\mu(s_q,s_s)$, a similaridade do comprimento de $\{..\gamma..\}_k$ e $\{..\lambda..\}_l$, e a similaridade estrutural $\mu(\ \gamma,\ \lambda)$ de γ e λ.

Neste contexto as cadeias

(35) $$\{..\gamma..\}_k \ \ \mathbf{e}\ \ \{..\lambda..\}_l$$

são chamadas de **dobras** $\phi\ (s_i)$ e $\phi\ (s_j)$ de s_i e s_j, respectivamente, porque as cadeias $\alpha\ s_i\ \beta$, $\alpha\ s_j\ \beta$ podem ser dobradas em sub-cadeias $\{..\gamma..\}_k$, $\{..\lambda..\}_l$, para tornar suas funcionalidades similares às das cadeias lineares $\alpha\ s_p s_q\ \beta$ e $\alpha\ s_r s_s\ \beta$. A possibilidade de dobra $\rho(\ \Gamma(s_i)\ ,\ \Gamma(s_j)\ )$ é definida como uma função de ambos $\mu(\ k, l\ )$ e $\mu(\ \gamma,\ \lambda)$, isto é

(36) $$\rho(\ \phi\ (s_i),\ \phi\ (s_i)\ ) = f''\ (\ \mu(\ k\ ,\ l\ ),\ \mu(\ \gamma,\ \lambda)\ )$$

Desta forma,

(37) $$\mu(s_i,s_j) = f(\ \mu(s_p,s_r),\ \mu(s_q,s_s),\ \rho(\ \phi\ (s_i),\ \phi\ (s_j)\ )\ )$$

isto é, a comparação entre s_i, s_j torna-se dependente das similaridades $\mu(s_p,s_r)$, $\mu(s_q,s_s)$ entre s_p, s_q e s_r, s_s e a possibilidade de dobra $\rho(\ \phi\ (s_i),\ \phi\ (s_j)\ )$.

Nesta condição:

a) $\mu(s_p,s_r)$, $\mu(s_q,s_s)$ é assumido para suportar a informação **codificada primariamente** em s_i,s_j,

b) $\mu(k, l)$ é assumido para suportar a informação **codificada secundariamente** em s_i, s_j, e

c) $\mu(\gamma,\lambda)$ é assumido para suportar a informação **codificada terciariamente** em s_i, s_j.

A linguagem suportada pela comparação não-linear definida em (33) a (37) é chamada de **Linguagem Nebulosa Não-Linear Ln** [SRRP96]. A gramática não-linear **Gn** suportando qualquer **Ln(Gn)** tem uma complexidade maior que a gramática linear **G** suportando uma linguagem linear **L**, porque ela inclui um conjunto específico **F** de regras associadas ao processamento das dobras das cadeias de **Ln**, isto é,

(38) $$\mathbf{Gn = \{ V_s, V_n, V_t, P, F, \eta \}}$$

onde $\mathbf{V_s}$, $\mathbf{V_n}$, $\mathbf{V_t}$, **P**, η são definidos como em (9) e (10), e **F** é definido como o conjunto de regras de dobras entre as quais:

a) **dobra:**

(39) $$\alpha\ s_i\ ..\gamma..\ s_j\ \beta \rightarrow \alpha\ s_i\ \{..\gamma..\}_s\ s_j\ \beta$$

define $\{..\gamma..\}_s$ como a dobra de $\alpha\ s_i\ ..\gamma..\ s_j\ \beta$, isto é, $\Gamma(\alpha\ s_i\ ..\gamma..\ s_j\ \beta) = \{..\gamma..\}_s$;

b) **corte:**

(40) $$\alpha\ s_i\ \{..\gamma..\}_s\ s_j\ \beta \rightarrow \alpha\ s_is_j\ \beta,\ ..\gamma..$$

desconcatena $\alpha\ s_is_j\ \beta$ e $\{..\gamma..\}_s$;

c) **inserção:**

(41) $$\alpha\ s_is_j\ \beta,\ ..\gamma..\ \rightarrow \alpha\ s_i\ ..\gamma..\ s_j\ \beta$$

insere a cadeia $..\gamma..$ na cadeia $\alpha\ s_is_j\ \beta$;

d) **reordenação:**

(42) $\alpha\ s_i\ ..\gamma..\ s_j\ s_k\ \beta \rightarrow \alpha\ s_i s_j..\gamma..\ s_k\ \beta$

dobra, corte e inserção podem ser usados para reordenar os símbolos da cadeia $\alpha\ s_i..\gamma..\ s_j\ s_k\ \beta$; etc.

3.6. Redes Nebulosas

A manutenção da ambigüidade natural da **LFN** dentro de limites para torná-la usável para qualquer tipo de computação requer a imposição de restrições sobre o espaço computacional suportado por um sistema distribuído nebuloso.

Dada uma gramática **G**

$$\mathbf{G = \{ V_s, V_n, V_t, P, \eta \}}$$

definida como em (9), e uma seqüência de derivação $\mathbf{d(s_o,s_m)}$ de **G**

$$\mathbf{d(s_o,s_m)} = \alpha\ s_o\ \beta \rightarrow \alpha\ s_1\ \beta \rightarrow \ldots\ldots\ \alpha\ s_k\ \beta \rightarrow \alpha\ s_m\ \beta$$

definida como em (12), um espaço computacional pode ser definido a partir de um conjunto de símbolos **M** cercando componentes de $\mathbf{V^*}$ em um espaço interno definido. Em outras palavras, **M** é o conjunto de fronteiras onde cada $\mathbf{M_m}$ particiona $\mathbf{V^*}$ em:

a) $\mathbf{V^o}$: o conjunto de símbolos localizados fora do cerco fornecido por $\mathbf{M_m}$;

b) $\mathbf{V^i}$: o conjunto de símbolos localizados dentro do cerco fornecido por $\mathbf{M_m}$, e

c) $\mathbf{V^m}$: o conjunto de símbolos localizados na própria fronteira $\mathbf{M_m}$.

Se aí existir um $\mathbf{M_e}$ chamado fronteira externa, definindo um espaço interno contendo todas as outras fronteiras, então **M** particiona **G** em um família de espaços de processamento parcialmente ordenados.

Três subconjuntos especiais de elementos de $\mathbf{V^m}$ são:

a) o conjunto $\mathbf{R}^m$ de fronteiras receptoras: compostas por aqueles elementos s_1 em $\mathbf{d}(s_0, s_1, ..., s_j)$ ativados por $s_0 \in V_s^o$, isto quer dizer ativados pelo símbolo inicial disponível fora do cerco de $\mathbf{M}_m$;

b) o conjunto $\mathbf{T}^m$ de fronteiras transportadoras: composta por aqueles elementos s_1 em $\mathbf{d}(s_q, s_1, s_q)$ movendo $s_q \in V_s^o$ de fora para o interior do cerco de $\mathbf{M}_m$, ou por elementos s_{t+1} em $\mathbf{d}(s_t, s_{t+1}, s_t)$ movendo $s_t \in V_s^t$ do interior para fora do cerco de $\mathbf{M}_m$, e

c) o conjunto $\mathbf{A}^m$ de fronteiras âncoras: composto por aqueles elementos s_k em $\mathbf{d}(s_0,..., s_i, s_k, s_j,..., s_q)$ localizados nos lugares específicos **k** de $\mathbf{M}_m$, determinando que a regras de reescrita $s_i ... \rightarrow s_j$ são para ocorrer na vizinhança do local **k.**

Deste modo, **M** organiza **G** em um conjunto de espaços de processamento que suportam seqüências de derivações do tipo:

(43) $$\mathbf{d}(s_0, ..., s_t) = \mathbf{d}(s_0, s_1 ,..., s_i, s_k, s_j ,..., s_{t+1}, s_t)$$

onde $s_0, s_t \in V^o$, $s_1 \in \mathbf{T}^m$ ou $\mathbf{R}^m$, $s_k \in \mathbf{A}^m$ e $s_{t+1} \in \mathbf{T}^m$ definidos por cada $\mathbf{M}_m \in \mathbf{M}$. Aqui, $\mathbf{d}(s_0, ..., s_t)$ é definido como a seqüência derivando s_0 em s_t.

Duas diferentes $\mathbf{G}_i$, $\mathbf{G}_j$ são ditas funcionalmente associadas se elas compartilham o subconjunto de símbolos $\mathbf{V}_c$ que são símbolos terminais de $\mathbf{G}_i$ e símbolos iniciais de $\mathbf{G}_j$ (ou vice-versa), isto é

a) $\mathbf{V}_i \cap \mathbf{V}_c \neq \varnothing$ em $\mathbf{G}_i$, e

b) $\mathbf{V}_t \cap \mathbf{V}_c \neq \varnothing$ em $\mathbf{G}_j$,

Uma vez que elementos de $\mathbf{V}_c$ podem ser usados para especificar um subconjunto de **G** em $\mathbf{G}_j$ como o conjunto ativo de regras de reescrita, a interação funcional entre dois $\mathbf{G}_i$, $\mathbf{G}_j$ diferentes pode resultar na especialização de $\mathbf{G}_j$ em produzir um subconjunto definido $\mathbf{V}_s$ dos produtos finais $\mathbf{V}_t$. Desta maneira, um sistema distribuído pode ser considerado como uma família de agentes definidos por **M** e compartilhando a mesma gramática **G**. Do ponto de vista formal, uma família de agentes usando uma gramática nebulosa **G** pode ser especializada em resolver sub-tarefas definidas por empregar subconjuntos restritos de regras (chamado aqui de regras legíveis desta LFN). Desta forma, cada agente pode tornar-se especializado em processar uma ou mais seqüências de derivação e eles podem interagir inteligentemente para resolver um problema complexo, se a cadeia terminal produzida por algum deles puder ser aceita como cadeia inicial por um outro agente qualquer.

Aqui, uma rede de agentes $\mathbf{R_k}$ suportada por uma gramática **G** é formalmente definida como:

(44) $$\mathbf{R_k = \{ V_i , V_n , V_t , P, N \}}$$

onde **N** é definida por

a) um conjunto **A** de agentes $\mathbf{a_k}$, cada qual definido por um conjunto $\mathbf{D_k}$ de endereços espaciais ou lógicos, e especializado em processar um conjunto $\mathbf{B^k}$ de uma ou mais cadeias de derivação $\mathbf{b_j \in P}$. Assim:

(45) $$\mathbf{a_k = \{ V_i^k \subset V_i , V_n^k \subset V_n , V_t^k \subset V_t , P^k \subset P, I_k \}}$$

b) um conjunto **I** de canais de informação (correio ou quadro de avisos) suportando a troca de símbolos entre os elementos de **A**.

Dado $\mathbf{S_i}$ como o conjunto de símbolos produzidos pelo conjunto de agentes $\mathbf{A_i}$, o canal de informação $\mathbf{I_{i,k} \subset I}$ suportando a comunicação entre um subconjunto de agentes (emissores) $\mathbf{A_i}$ e outro conjunto de agentes (receptores) $\mathbf{A_k}$ é especificado por:

c) um conjunto $\mathbf{D_{i,k}}$ de endereços físicos ou lógicos $\mathbf{d_{i,k}}$ referindo-se aos agentes $\mathbf{a_i \in A_i}$ e/ou $\mathbf{a_k \in A_k}$;

d) uma família de funções de custo $\mathbf{\Delta_{i,k}}$ que descrevem as restrições de tempo e espaço na veiculação de $\mathbf{S_i}$ em $\mathbf{I_{i,k}}$.

Desta maneira

e) um sistema de correio para distribuição de $\mathbf{S_i}$ é definido quando o emissor e/ou receptor conhece perfeitamente a direção física do receptor e/ou emissor.

f) Um sistema de quadro de avisos é definido sempre que o receptor seleciona o conjunto $\mathbf{S_k}$ de símbolos ou mensagens cujo conteúdo ele está interessado, de um conjunto $\mathbf{S_i}$ produzido por $\mathbf{A_i}$, e veiculado por um canal $\mathbf{I_k}$ menos seletivo.

Assim :

(46) $$\mathbf{I_j = \{D_j, S_j , \Delta_j \}}$$

Dois conjuntos de canais de informação são associados à $\mathbf{a_j \in A_j}$: o conjunto $\mathbf{I_{i,j}}$ de canais de entrada tendo $\mathbf{a_j}$ como o receptor e o conjunto $\mathbf{I_{j,k}}$ de canais de saída tendo $\mathbf{a_j}$ como o emissor.

O conjunto **A** de agentes de um sistema distribuído é, portanto, particionado nos seguintes subconjuntos de acordo com seus símbolos de entrada:

1) $\mathbf{A_i}$: o conjunto de agentes de entrada usando símbolos iniciais para produzir símbolos não-terminais. No caso destes agentes $\mathbf{V_t}^k=\varnothing$;

2) $\mathbf{A_n}$: o conjunto de agentes intermediários usando símbolos não-terminais para produzir símbolos não-terminais. No caso destes agentes $\mathbf{V_i}^k=\varnothing$ e $\mathbf{V_t}^k=\varnothing$;

3) $\mathbf{A_t}$: o conjunto de agentes de saída usando símbolos não-terminais para produzir símbolos terminais. No caso destes agentes $\mathbf{V_i}^k=\varnothing$.

Do ponto de vista formal, a rede de processamento **N** definida por uma dada família **I** de canais de comunicação é usada para organizar um processamento concorrente da **LFN** suportada por **G**, para manter sua ambigüidade entre limites. **N** pode ser tanto fisicamente quanto logicamente definida, dependendo da via de canais de informação operada, se por sistema de correio ou quadro de avisos. Neste caso, sincronização tem um papel importante na definição de **N**.

As Linguagens Formais clássicas se utilizam do conceito de grafos para descrever as transações simbólicas permitidas pela linguagem representadas por arcos entre seus símbolos que são representados por vértices. Aqui necessitamos de um conceito mais abrangente para implementar a noção de custo da transação, isto é, necessitamos do conceito de canais de informação. Isto é feito através do uso de redes, onde pode-se definir tanto o endereçamento necessário para realizar um processamento completo quanto as funções de custo para a ativação de um agente receptor.

Essa rede pode ser organizada em três camadas bem definidas, onde na primeira está $\mathbf{A_i}$, na segunda $\mathbf{A_n}$ e na terceira $\mathbf{A_t}$, sendo que a segunda camada pode ser subdividida em mais camadas, pois símbolos intermediários podem ativar regras que geram outros símbolos intermediários. Desta forma, uma **fbf** em $\mathbf{R_k}$ é descrita por qualquer caminho formado entre um agente da primeira camada até um agente da última camada. Essas redes serão aqui chamadas de **Redes Nebulosas**.

4. KARDS

O sistema Kards foi desenvolvido como um manipulador de **LFN** usando a noção de **Redes Nebulosas** para implementar o processamento desta LFN. O sistema foi organizado nos seguintes módulos operacionais: *interpretador* (calculadora), *base de dados*, *raciocínio especializado* (quest), *processamento de linguagem especializada* (jargão) e *aquisição de dados* (sensor). A especificação do Kards para manuseio de uma LFN é feita através da especificação de uma base de dados, um raciocínio e uma linguagem especializada quando esta for necessária. Esta especificação é aqui chamada de *Aplicação*.

4.1. A Linguagem Kards

Para o processamento das regras da **LFN**, o Kards possuiu um interpretador interno que manuseia uma LFN Booleana, pois as máquinas atuais nas quais o Kards pode ser executado não fornecem meios para tratamento de uma LFN. Com esta LFN booleana aceita pelo interpretador pode-se, entretanto, desenvolver aplicações suportadas por diversos tipos de LFNs.

O interpretador realiza duas funções básicas:

a) Executar funções primitivas que geram as cadeias ou sentenças terminais: a execução é feita através de agentes, cuja estrutura básica envolve uma interface para aquisição do símbolo ou sub-cadeia a processar, uma unidade de processamento encarregada da execução da regra e uma interface de distribuição da cadeia processada. Esses agentes utilizados para a execução das funções básicas do sistema são denominados aqui de agentes primitivos.

As funções primitivas manuseadas por esses agentes podem ser funções de cálculo numérico, manuseio de arquivos, interface com o usuário etc., pois os agentes de um sistema distribuído podem dedicar-se a tarefas outras que o processamento da LFN.

b) Executar *Redes de Processamento* que manuseiam o conjunto de sentenças iniciais e intermediárias: processamentos mais complexos são programados através da organização de um conjunto de agentes primitivos em uma rede de processamento. Essa rede de processamento pode ser vista como um novo agente do sistema, sendo que essa impõe uma hierarquia que tem por objetivo implementar uma estrutura de herança de propriedades a

serem partilhadas entre os agentes e organizar o fluxo de informação com o intuito de reduzir o custo de processamento na obtenção do símbolo terminal.

4.2. Base de Dados

Um módulo de base de dados foi implementada no sistema para que ele pudesse suportar o conceito de memória. Todo o conhecimento contido no sistema, inclusive sobre ele mesmo, é armazenado nessa base de dados: as redes de processamento, a própria definição das bases de dados a serem utilizadas, o conhecimento dos especialistas, etc.

A base de dados do Kards foi organizada baseada no conceito de armários, onde um armário contém gavetas, cada gaveta contém pastas e cada pasta contém fichas com os dados de cada sentença armazenada. Essa hierarquia é armazenada no sistema através de uma rede:

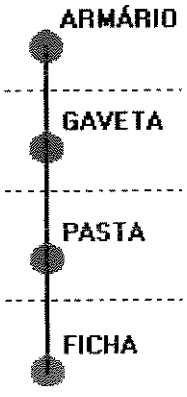


Figura 1 - Rede armário-gaveta-pasta-ficha

Cada agente dessa rede possui a definição dos atributos e métodos referentes a ele. Como a *gaveta* pertence ao *armário*, então é possível definir um método no *armário* que seja 'herdado' pela *gaveta*, e também é possível redefinir para a *gaveta* um método que já havia sido definido para o *armário*. Com isso é possível definir métodos que sejam genéricos para todo um conjunto de agentes que está abaixo de um determinado agente especifico na hierarquia e definir um método específico para apenas um desses agentes. Existe um conjunto de métodos 'padrões' que são herdados por todos os agentes, desde que esses métodos não sejam redefinidos. Este tipo de herança permite, por exemplo, que as características associadas às gavetas de um armário de uma aplicação sejam herdadas ou não por todas as pastas deste armário, como por exemplo a lista de fichas que compõe uma pasta pode ser a mesma ou diferente para gavetas distintas.

Em cada ficha da base de dados são armazenadas sentenças da LFN que suporta a aplicação. Essas sentenças, dependendo do tipo de ficha utilizado, são interpretadas por gramáticas diferentes, de forma que, com a mesma estrutura básica computacional, podem ser armazenados diversos tipos de fichas e diferentes tipos de ficha correspondem a diferentes tipos de formatação das sentenças de uma LFN. As definições das fichas utilizadas no sistema

(na aplicação) são armazenadas em redes na base de dados. Essas redes seguem uma estrutura básica dependendo do tipo de ficha a ser definido.

4.2.1.Tipos padrões de ficha do sistema

a) Normal: armazena campos ou sub-cadeias que compõem a sentença, sendo que esses campos são organizados em linhas e seções. A estrutura da ficha é armazenada em uma rede onde o nível superior possui o agente referente à ficha, o segundo nível possui as seções, o terceiro as linhas e o quarto os campos, como mostra a Figura 2. Este tipo de ficha deve ser usado para armazenar sentenças de estrutura complexa e bem definida quer do ponto de vista sintático, como por exemplo as sentenças de processamento definindo cálculo numérico, lingüístico ou lógico, quer do ponto de vista semântico como por exemplo as sentenças de uma base de dados, onde campos são associados pelos seus valores semânticos. A rede da Figura 2 representa a estrutura de ficha mostrada na Figura 3.

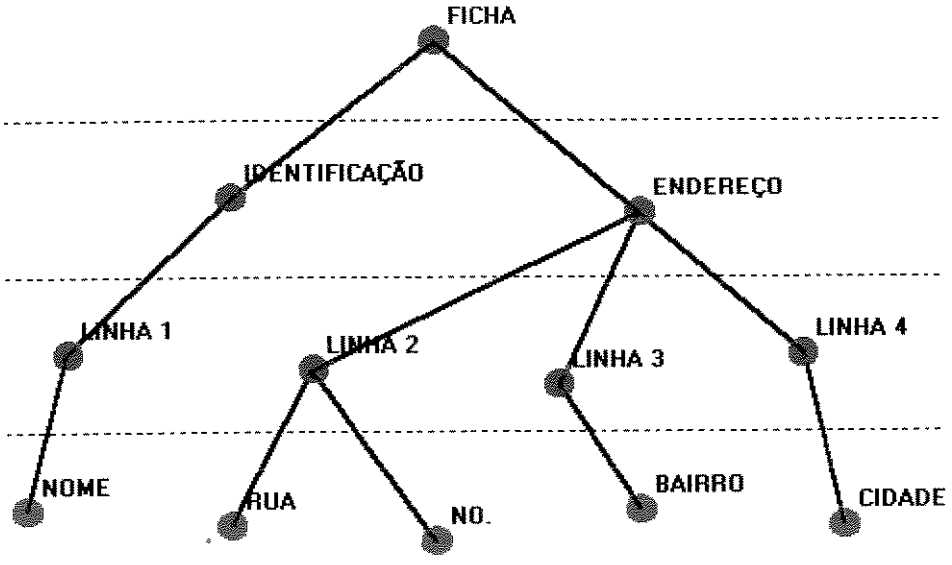


Figura 2 - Exemplo de rede de uma ficha tipo Normal

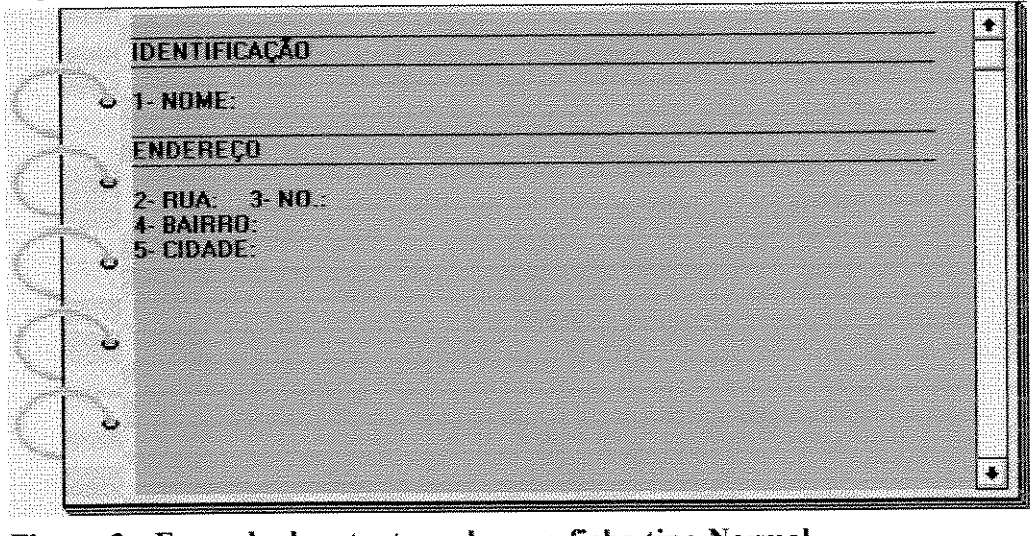


Figura 3 - Exemplo de estrutura de uma ficha tipo Normal

b) Texto: armazena uma sentença qualquer sem formatação conhecida *a priori*, ou cuja formatação esteja descrita pela sintaxe da LFN armazenada no armário Linguagem da aplicação. Neste caso, o Jargão pode ser usado para ajudar na aquisição da sintaxe da LFN utilizada em um ambiente especializado. A rede dessa ficha é composta apenas por dois agentes, como mostra a Figura 4.

Figura 4 - Exemplo de rede de uma ficha tipo Texto

O primeiro agente contém os métodos referentes à ficha e o segundo representa os dados da ficha, ou seja, a sentença propriamente dita.

c) Planilha: armazena os campos que compõem uma sentença sob o formato de grade (campos organizados em linhas e colunas). Essa formatação está baseada na de uma ficha de tipo normal, guardando em cada campo da ficha normal uma linha da grade e no nome da seção a definição das colunas que serão utilizadas, como mostra o exemplo da Figura 5. Em cada campo da ficha normal a cadeia é armazenada com um símbolo separador para as respectivas colunas. Este tipo de ficha também deve ser usado para armazenar sentenças de estrutura complexa e bem definida.

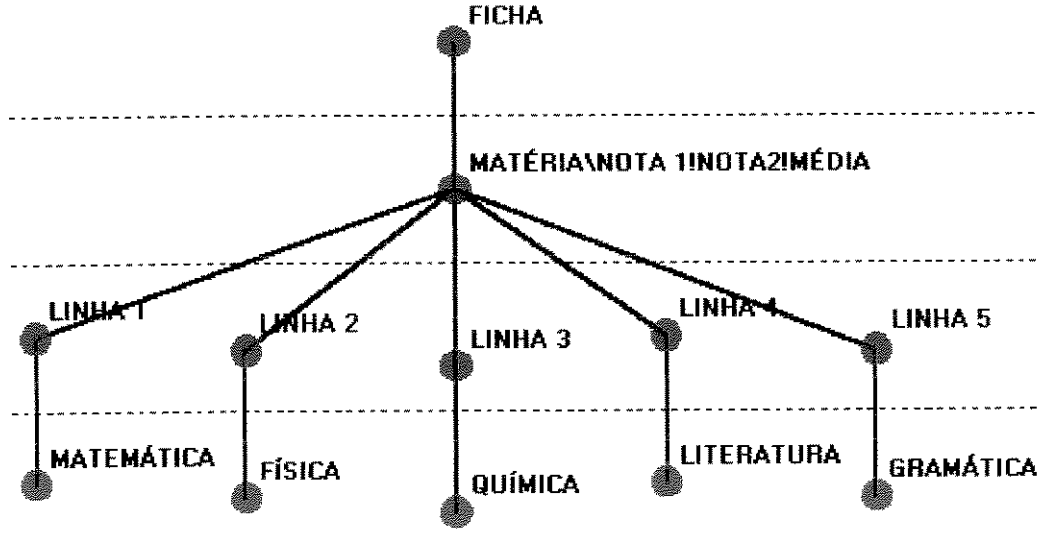


Figura 5 - Exemplo de rede de uma ficha tipo Planilha

A rede da Figura 5 representa a estrutura de ficha mostrada na Figura 6.

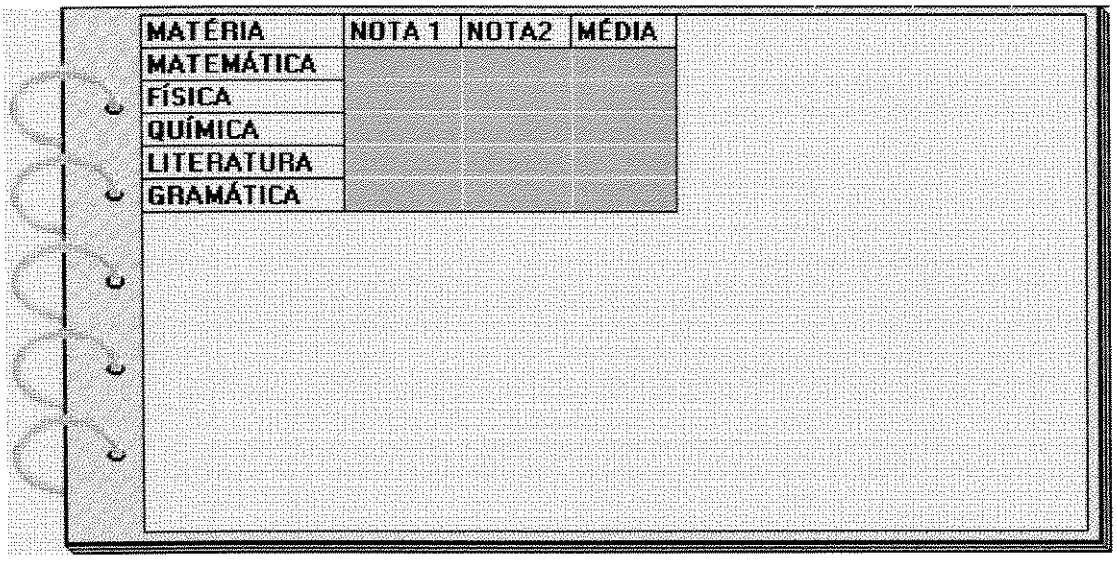


Figura 6 - Exemplo de estrutura de uma ficha tipo Planilha

d) Planilha Texto: baseada na junção de uma ficha tipo *planilha* e uma ficha tipo *texto*, este tipo de ficha armazena uma grade de tamanho indeterminado, do mesmo modo que na ficha texto não há um número fixo de linhas; porém com colunas definidas. Este tipo de ficha deve ser utilizado para armazenar sentenças de estrutura complexa e bem definida, porém de comprimento não fixo quanto ao número de sub-cadeias, de forma que todas as sub-cadeias são tratadas de acordo com a mesma sintaxe da LFN. Ex.:

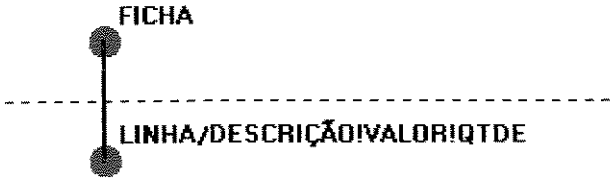


Figura 7 - Exemplo de rede de uma ficha tipo Planilha Texto

A rede da Figura 7 representa a estrutura de ficha mostrada na Figura 8.

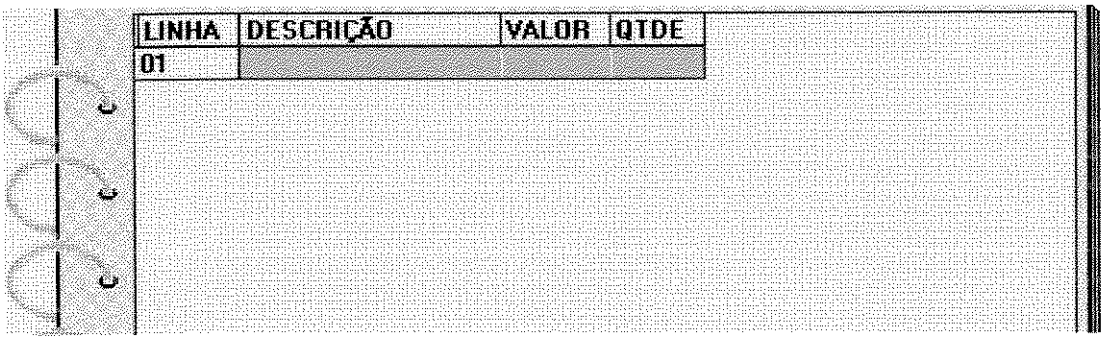


Figura 8 - Exemplo de estrutura de uma ficha tipo Planilha Texto

e) Rede: baseada na ficha *planilha texto*, essa ficha foi criada para armazenar redes. Ela é uma ficha tipo planilha texto com colunas predefinidas, onde são armazenadas as definições de cada agente e as ligações entre esses agentes. É um tipo muito específico de ficha, porém é essencial para o sistema, já que todo o sistema é baseado no uso de redes.

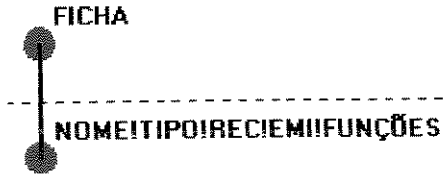


Figura 9 - Exemplo de rede de uma ficha tipo Rede

A rede da Figura 9 representa a estrutura de ficha mostrada na Figura 10.

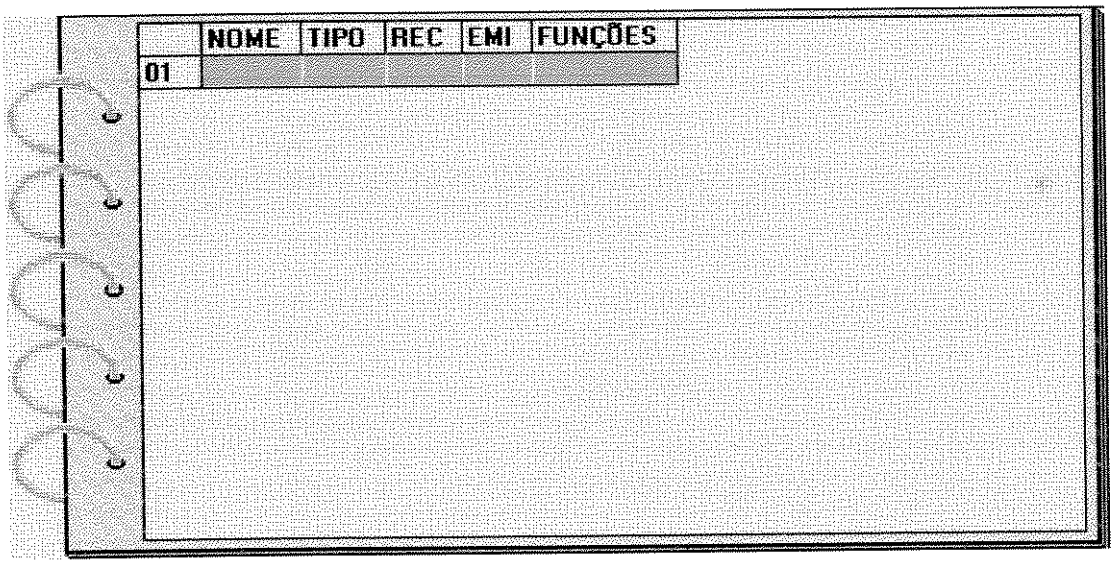


Figura 10 - Exemplo de estrutura de uma ficha tipo Rede

4.3. Raciocínio (QUEST)

O conhecimento utilizado para implementar um raciocínio suportado pela LFN é armazenado no sistema através de **Redes de Raciocínio**. Essas redes são armazenadas em fichas do tipo *Rede* (uma ficha compõe uma sentença), de forma que o conhecimento pode ser armazenado num conjunto de sentenças associadas às distintas fichas de uma pasta. Desta maneira as pastas de cada gaveta da base de conhecimento descrevem os diversos módulos do raciocínio para suportar a aplicação desejada.

Cada rede possui quatro níveis, como mostra a Figura 11.

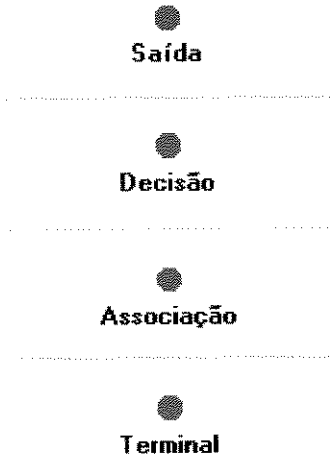


Figura 11 - Níveis de uma Rede de Raciocínio

O nível mais baixo (*Terminal*) é o responsável pela aquisição dos dados a serem processados pela rede (símbolo inicial). No nível *Associação* se dá a manipulação dos dados adquiridos no nível *Terminal*. O nível *Decisão* é o responsável pela distribuição da sentença processada e é portanto responsável pela ativação da *Saída* a ser acionada de acordo com os dados obtidos no nível *Terminal* e trabalhados no nível *Associação*. O nível *Saída* é o responsável pela finalização da rede e sua possível conexão com as demais redes que compõem o raciocínio.

Cada agente de uma rede possui métodos, sendo que no módulo de raciocínio são utilizados três métodos para cada agente:

a) método ***inicial***: responsável pela aquisição dos dados que o agente processará,

b) método ***intermediário***: responsável pelo processamento da sentença,

c) método ***final***: este método é o responsável pela distribuição da sentença processada entre os diversos agentes do sistema, sendo que o fluxo da rede é definido através das ligações entre os agentes do mesmo modo que é feito nas Redes de Processamento.

Para a construção de cada um desses métodos, podem ser utilizados quaisquer agentes primitivos, tanto quanto quaisquer redes de processamento. O sistema Kards possui agentes que permitem a implementação dos conceitos de pertinência, variáveis lingüísticas, etc. que são utilizados para implementações que envolvam raciocínio aproximado, além dos agentes de cálculo numérico e de lógica simples, dentre outros.

A rede mostrada como exemplo na Figura 12 checa que tipo de problema existe com um separador de uma planta processadora primária de petróleo (aonde o óleo é separado do gás). O nível terminal (mais abaixo) adquire do ambiente as variáveis utilizadas no raciocínio.

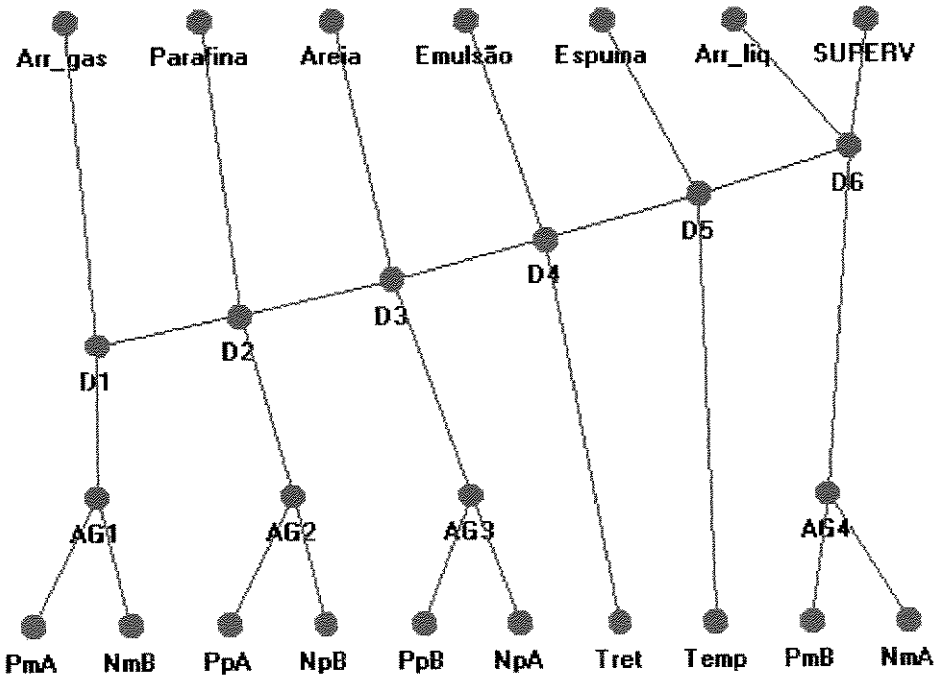


Figura 12 - Exemplo de Rede de Raciocínio

O armário de estrutura da aplicação (descrito em 4.6) possui uma gaveta chamada *Funções Nebulosas*, onde podem ser definidas todas as funções necessárias para a implementação de conceitos de variáveis lingüísticas e funções de pertinência para avaliações nebulosas. Desta maneira o Kards pode ser utilizado para implementações inteligentes suportadas por raciocínio aproximado e parcial. A Figura 13 mostra um exemplo de uma tela para a definição de funções nebulosas, onde a variável lingüística *pressão* pode ser observada e modificada visualmente.

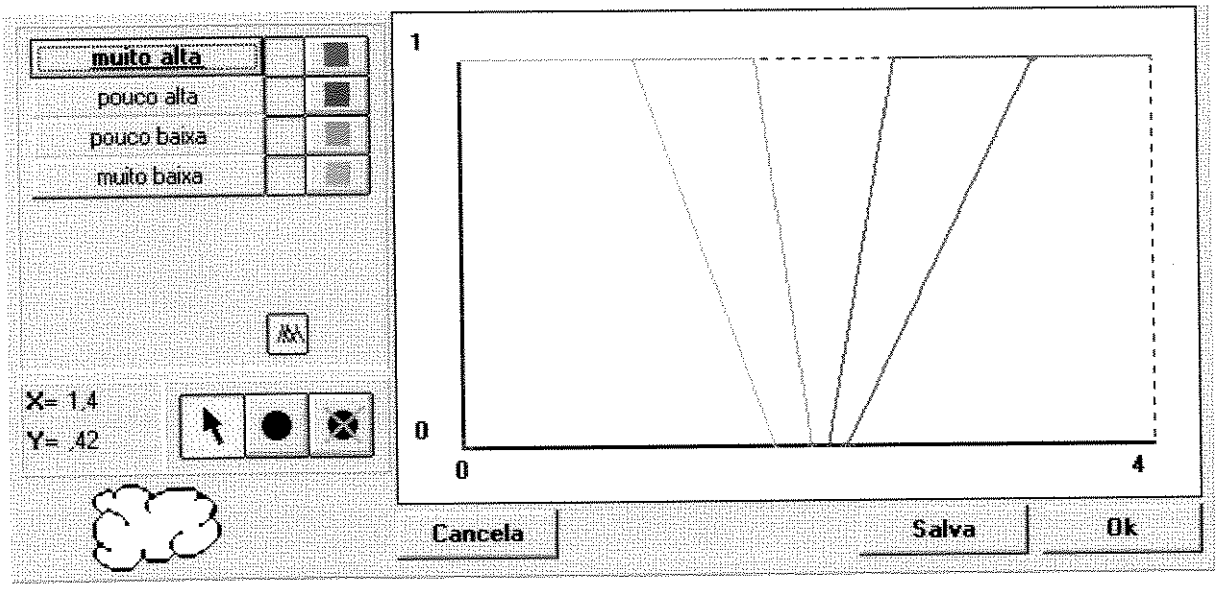


Figura 13 - Exemplo de definição de variável lingüística

Essas funções nebulosas são utilizadas pelos agentes do nível de associação e decisão, sendo que neste a pertinência obtida através dessas funções é utilizada para definir qual agente de saída será ativado. Os agentes do nível de saída definem então, qual o problema encontrado no separador analisado.

4.4. Linguagem (JARGÃO)

O jargão é o módulo destinado a ajudar na análise do conteúdo de sentenças descritas em linguagem natural. O propósito desta análise é de extrair de um conjunto de sentenças as palavras significativas associadas a um conjunto de conceitos primitivos, a serem usados para identificar conceitos complexos ou teorias, descritos por frases ou conjuntos de frases. Essas frases são **fbf** para uma dada sintaxe (gramática), descrevendo as possíveis relações entre os conceitos. Desta forma, o usuário pode codificar suas consultas em um conjunto de possíveis relações descritas por fbfs, que serão usadas pelo Jargão para identificar quais frases ou conjuntos de frases obedecem às fbfs. As fbfs suportadas pelas sentenças dadas são, então, assumidas como sendo as respostas às consultas propostas. A sintaxe pode ser uma sintaxe ordinária de uma dada linguagem natural, ou pode ser qualquer outra linguagem formal usada para codificar o conhecimento específico de qualquer área do conhecimento humano, isto é, para codificar o jargão usado nessa área de conhecimento.

Uma **fbf** é descrita por uma rede com quatro níveis, onde no nível mais baixo estão os conceitos primitivos onde agentes são ativados pela similaridade com as palavras significativas associadas à esse agente (o conjunto dessas palavras compõem um dos métodos desse agente), e o nível mais alto corresponde à teoria formada pelas frases que são compostas pelos conceitos primitivos. Os métodos associados à cada agente dos níveis intermediários descrevem o tipo de associação. As sentenças que contém as redes que descrevem essas fbf

são armazenadas no armário Linguagem da aplicação (descrito em 4.6), em uma ficha tipo *rede*. A Figura 14 mostra um exemplo de uma rede.

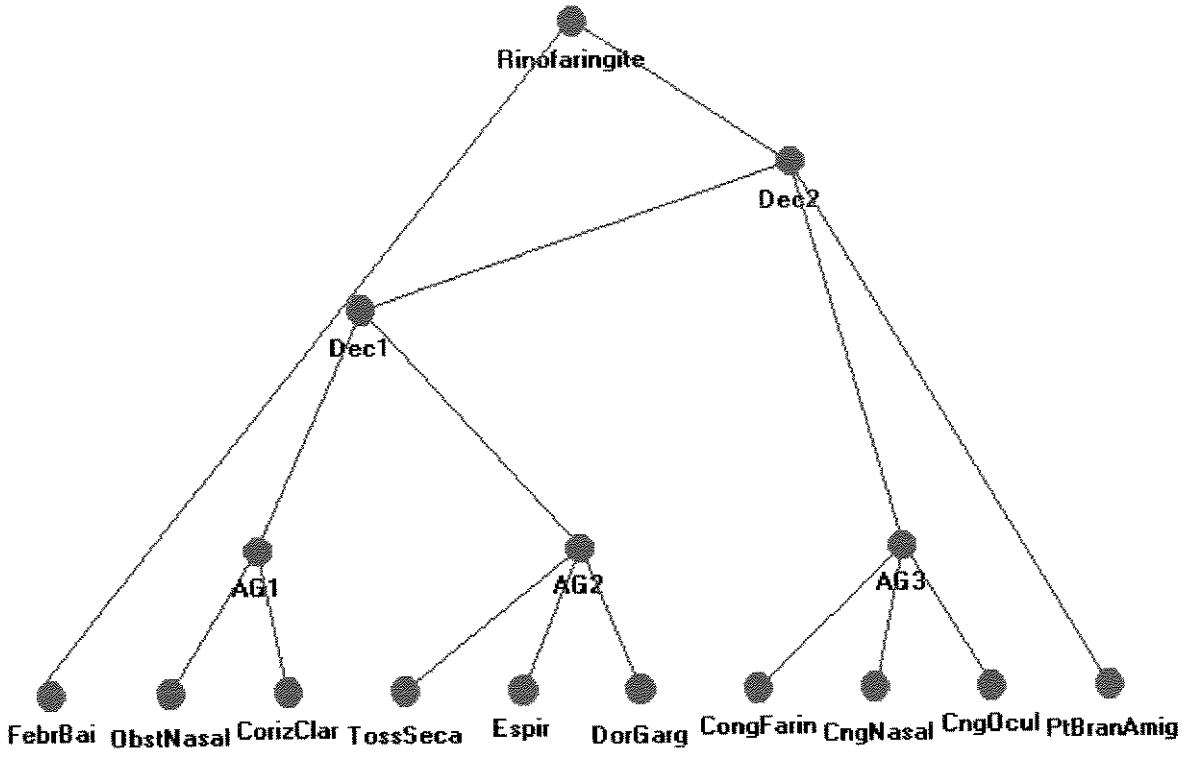


Figura 14 - Exemplo de rede do Jargão

Os agentes do nível mais baixo são ativados de acordo com a existência das palavras do texto associadas à cada conceito. Esses conceitos são agrupados de modo que na rede da Figura 14, o conceito complexo *Rinofaringite* depende da existência de febre baixa (FebrBai), obstrução nasal (ObstNasal), coriza clara (CorizClar), tosse seca (TossSeca), etc.

O primeiro agente do nível mais baixo da rede é utilizado como *gatilho* para a rede, ou seja, se for encontrada a palavra chave associada à ele ('Febre') a rede é analisada, senão o sistema passa a analisar as outras redes. Quando a palavra chave é encontrada, as outras palavras são procuradas em uma vizinhança desta. Os limites para essa vizinhança são definidos nos métodos de cada agente. Além da vizinhança, é associado uma certeza ao conceito, que vai variando de acordo com as palavras encontradas e com a vizinhança aonde foi encontrada, de forma que ao final da análise da rede o sistema fornece uma certeza sobre o conceito analisado.

A gaveta com o conjunto de pastas que contêm essas redes é armazenada no armário de *Linguagem* e este conjunto forma a LFN da aplicação. Fichas texto são utilizadas no armário da aplicação para armazenar as frases que serão analisadas, pois as fichas texto não possuem formatação, ou seja, sua sintaxe não é definida na ficha, e sim nos agentes que utilizarem as sentenças contidas nessas fichas.

A rede mostrada como exemplo na Figura 14 é ativada, por exemplo, através da análise da sentença mostra na Figura 15.

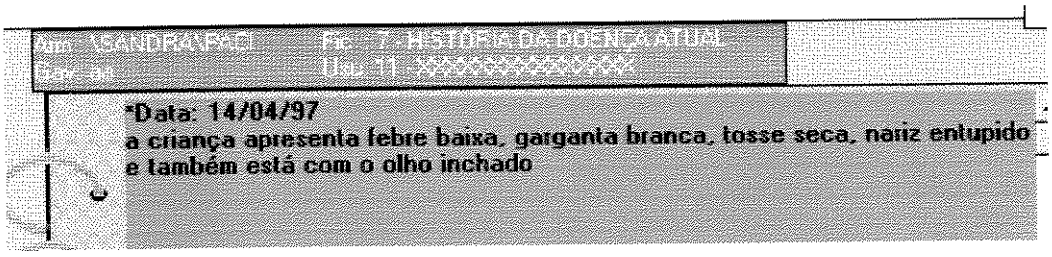


Figura 15 - Exemplo de ficha texto a ser analisada no Jargão

Após a análise desta sentença, no sistema de onde foi retirado o exemplo, é ativada uma rede de raciocínio onde cada um dos conceitos detectados pelo Jargão é analisado levando-se em consideração outros dados sobre o paciente retirados da base de dados, bem como alguns fatores retirados do próprio texto, como se a febre está *alta* ou *baixa* e uma conduta pode ser definida de acordo com esse conceito, como mostra a Figura 16.

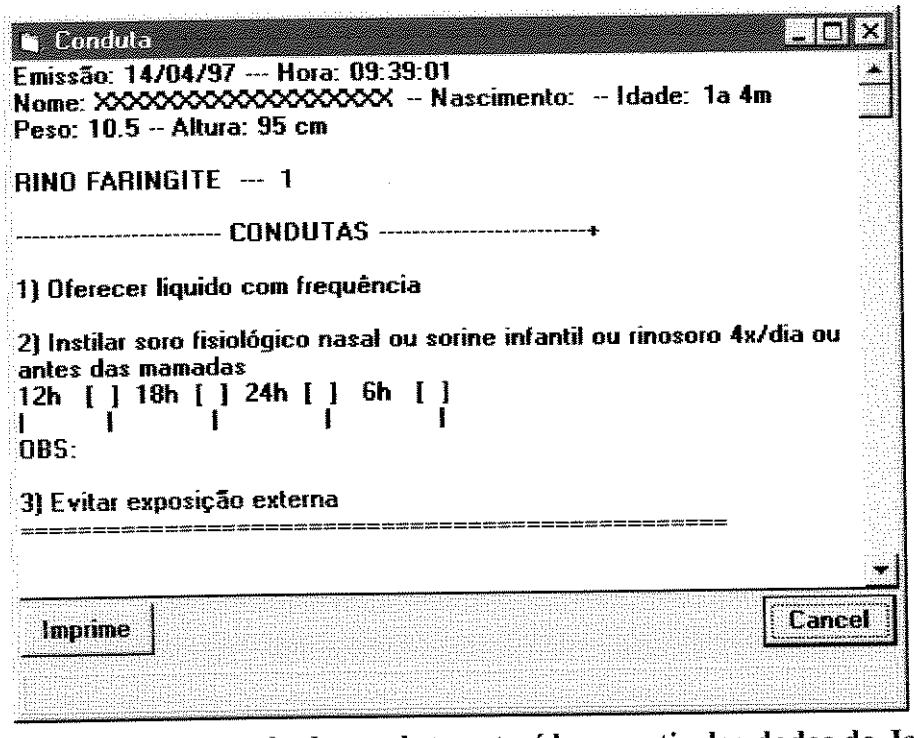


Figura 16 - Exemplo de conduta extraída a partir dos dados do Jargão

4.5. Sensor

O módulo *Sensor* foi implementado visando a aquisição do símbolo inicial a ser processado diretamente do ambiente externo, e também agentes primitivos que permitam atuar diretamente no ambiente externo.

4.5.1.Aquisição do símbolo inicial

A aquisição dos dados propriamente dita. Para fazer essa aquisição, o Kards possui ferramentas para aquisição através de:

- Scanners:

A ferramenta utilizada neste caso é a importação de imagens, que podem ser provenientes de scanners (que normalmente necessitam da intervenção de um usuário) ou através da integração com um sistema de digitalização de vídeo. A imagem é transformada em um símbolo que representa um mapa das cores de cada ponto da imagem importada.

- Mesa Digitalizadora:

O Kards possui um formulário especialmente desenhado para a aquisição de dados através de mesa digitalizadora, onde podem ser adquiridos dados de curvas e pontos isolados. Como saída é fornecido um símbolo descrevendo o conjunto dos pontos isolados ou que descrevem a curva.

- Conversão Analógica-Digital:

Foram incorporados ao Kards agentes que acessam diretamente placas conversoras analógico/digital, de forma que o Kards pode ser utilizado para gerenciar as atividades de qualquer ambiente que possua sensores elétricos, estando limitado somente às restrições de velocidade e memória do sistema.

A aquisição do símbolo sensorial inicial envolve não só a digitalização de uma variável do ambiente externo, como também o tratamento desse dado digitalizado. Por exemplo, quando uma imagem é adquirida, é necessário retirar dela a informação a ser processada pelo sistema. Foram implementados agentes que imitam a visão humana para extrair informações de imagens, bem como podem ser criadas *Redes de Processamento* para realizar essa tarefa, além de agentes para o reconhecimento de padrões.

Como exemplo, a Figura 17 mostra uma carta dinamométrica de uma unidade de bombeio de petróleo. Essa carta foi adquirida através de uma mesa digitalizadora, onde foram introduzidos os pontos que unidos formam a curva contínua da carta. Ao lado da carta principal são mostradas outras três cartas anteriores para que o usuário possa compará-las visualmente. A carta é armazenada no sistema através de uma sentença que descreve seus pontos.

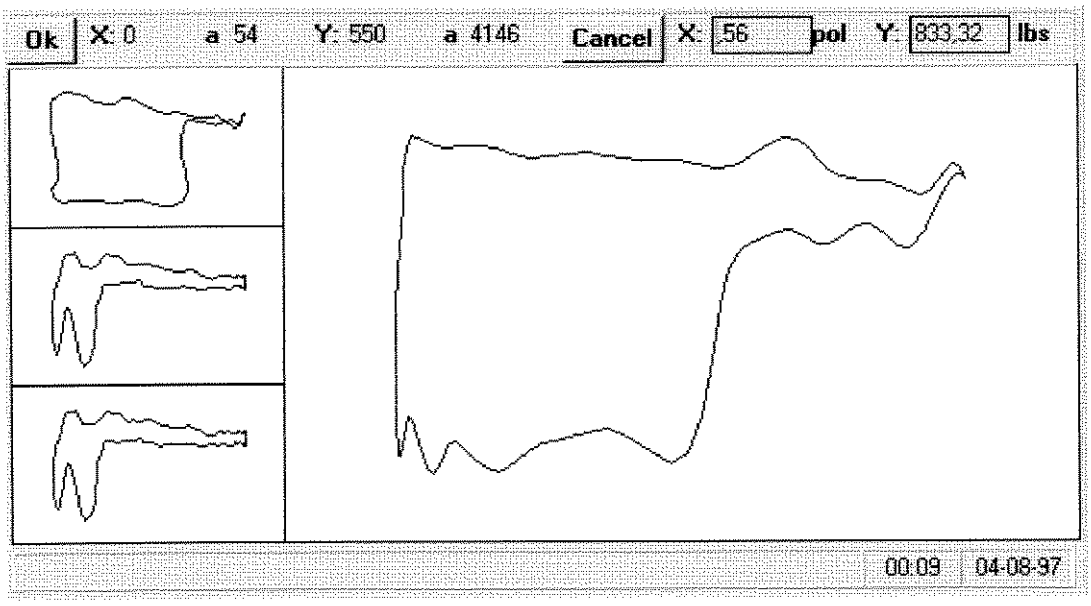


Figura 17 - Exemplo de curva adquirida através de mesa digitalizadora

Após a digitalização, essa carta (sentença) é submetida a um programa externo ao Kards através do agente primitivo *EXEC* que executa um programa qualquer no Sistema Operacional. Nesse programa essa carta medida na superfície do poço de petróleo é transformada por um algoritmo para uma carta que equivale a que seria medida no fundo do poço [Bar96], como mostra a Figura 18.

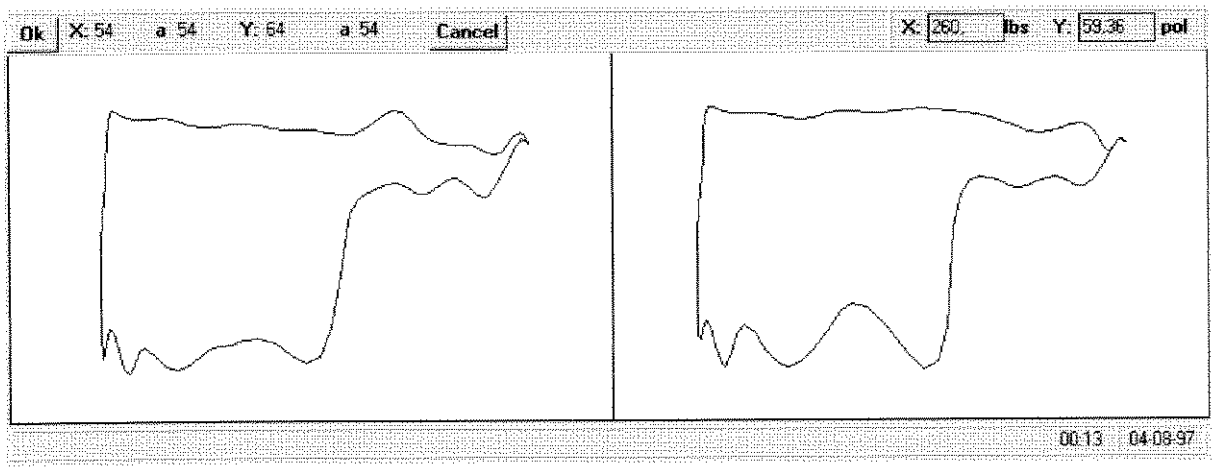


Figura 18 - Exemplo de carta de superfície e sua respectiva carta de fundo

Após esse tratamento inicial a sentença descrevendo a carta é então submetida ao reconhecimento de padrões, onde os pontos da carta são comparados aos pontos de modelos (padrões) predefinidos no sistema. O conjunto desses modelos compõem parte da sintaxe da LFN da aplicação, e são armazenados como redes [Ale90] na base de dados. No exemplo utilizado, um limiar é utilizado para definir a aceitação do padrão.

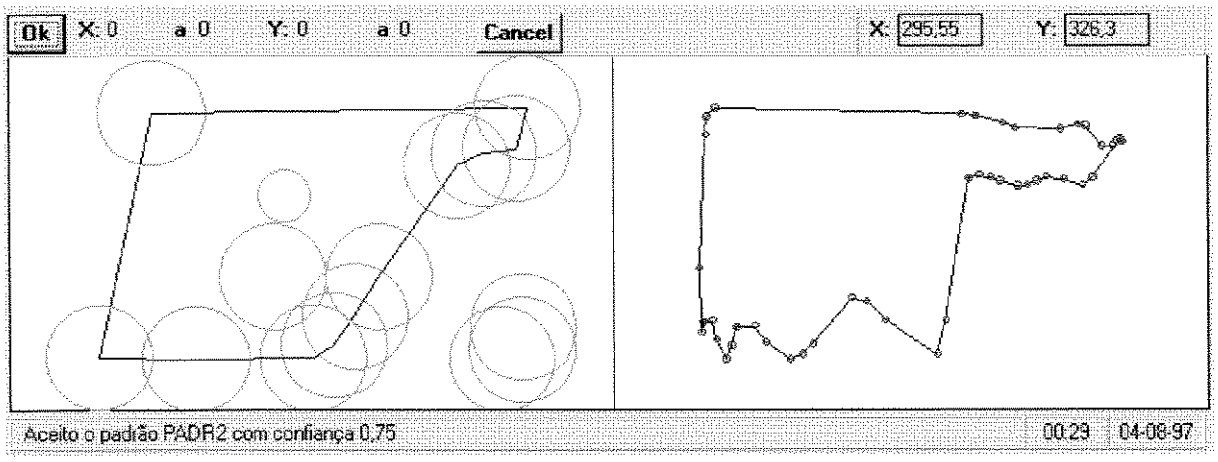


Figura 19 - Exemplo de reconhecimento de padrão

Após o reconhecimento do padrão o módulo de raciocínio é acionado para tratar o símbolo inicial gerado pelo sensor e tomar as decisões (chegar ao símbolo terminal) navegando outro conjunto de redes de raciocínio [Ale90].

4.5.2.Atuação no ambiente externo

A atuação no meio externo é feita através agentes primitivos que realizam uma **Conversão Digital-Analógica**. Esses agentes acessam diretamente placas conversoras digital/analógicas, gerando como símbolo terminal a atuação no meio externo.

A utilização de agentes primitivos de conversão analógica-digital para obtenção dos símbolos iniciais, em conjunto com agentes primitivos de conversão digital-analógica para gerar os símbolos terminais possibilitam o uso do Kards como um sistema de automação inteligente para ambientes especializados.

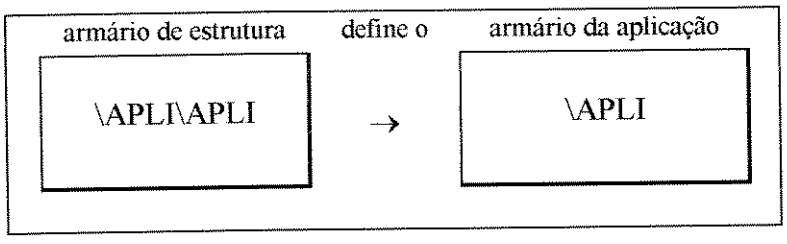
4.6. Aplicação

Uma ***aplicação*** é a especificação do *Kards* para o manuseio de uma LFN, e é feita através da especificação de um armário (base de dados), um raciocínio, uma linguagem especializada quando esta for necessária, redes de processamento, etc. A especificação de um armário (base de dados) é feita através da definição de suas gavetas e fichas, e dos métodos para cada agente da rede *armário-gaveta-pasta-ficha*. A especificação de um raciocínio é feita através da definição das redes de raciocínio. Essas especificações são compostas por redes que são armazenadas na base de dados do Kards. Portanto, para a criação de uma *Aplicação*, é necessário a criação de um armário aonde serão armazenadas essas informações (as sentenças que definem essas redes). Esse armário é chamado de *armário de estrutura da aplicação*, ou simplesmente, *armário de estrutura*.

Deste modo, existe uma gaveta no *armário de estrutura* - gaveta de *fichas* - na qual são armazenadas as sentenças que definem as redes das fichas da base de dados (armário) da aplicação; uma outra gaveta contém a sentença que define a rede *armário-gaveta-pasta-ficha* e seus métodos e em outra gaveta estão as sentenças das *redes de raciocínio* (gaveta *quest*). As gavetas e seus conteúdos existentes nesse armário são:

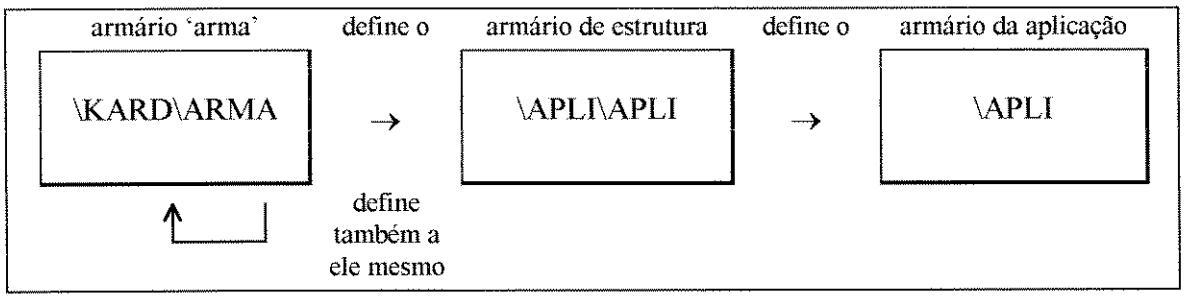
- **Fichas**: sentenças que definem as redes de cada ficha utilizada no armário
- **Quest**: sentenças que definem as redes de raciocínio da aplicação
- **Funções Nebulosas**: sentenças que definem as variáveis lingüísticas para a aplicação
- **Gavetas**: sentenças que definem as gavetas que serão utilizadas no armário da aplicação
- **Armário**: sentença que define a rede *armário-gaveta-pasta-ficha* do armário da aplicação
- **Buscas**: sentenças que definem buscas (consultas) a serem feitas no armário da aplicação
- **Formulários**: sentenças que definem formulários para impressão
- **Simulador**: sentenças que definem telas com agentes para a criação de simulações
- **Servidores**: sentenças que definem servidores para o processamento paralelo

Com isso, a especificação da estrutura de um armário é feita através das sentenças armazenadas em seu armário de estrutura:



Como padrão, o nome do armário da aplicação é um nome de diretório (diretório do sistema operacional aonde ficará a aplicação) seguido do nome do armário (com até quatro letras - vide 5.5.2), e o nome do armário de estrutura é o nome do diretório mais o nome do armário da aplicação mais o nome do armário da aplicação novamente (é criado um sub-diretório no diretório do armário da aplicação com o mesmo nome do *armário da aplicação*, e nesse sub-diretório é colocado o armário de estrutura com o mesmo nome do armário da aplicação).

Como a definição de um armário é feita através das sentenças em seu armário de estrutura, é necessário que haja um armário com as sentenças que definem a estrutura do *armário de estrutura*. Esse é um armário muito especial no sistema, chamado de *'arma'*, e ele contém sentenças que definem a estrutura do armário que contém sentenças que definem um outro armário. Com isso, as sentenças armazenadas nesse armário *'arma'* podem ser utilizadas para definir ele mesmo, ou seja, nesse armário estão armazenadas as sentenças que definem sua própria estrutura.



Além do armário de estrutura, outros três armários são utilizados para definir uma aplicação. São eles:

- armário Padrão ('**PADR**'): este armário é utilizado para armazenar sentenças com as redes de processamento. Essas sentenças, apesar de representarem redes, são armazenadas em fichas tipo texto, onde seguem a sintaxe da linguagem Kards (descrita em 5.1). Também é reservada uma ficha tipo texto neste armário onde as redes podem ser documentadas (sua função, sintaxe, etc.)

- armário Ajuda ('**HELP**'): contém gavetas e fichas que compõem o *help* da aplicação (fichas texto com explicações, descrições, etc.). Este armário pode ser consultado diretamente da tela principal do Kards a qualquer hora pelo usuário

- armário Linguagem ('**LING**'): contém as sentenças que definem a sintaxe da LFN da aplicação

5. Estrutura Computacional

O sistema Kards foi desenvolvido em Visual Basic 3.0 sob o Windows 3.1. Sendo o Kards um manipulador de LFN, é necessário haver uma representação computacional para os símbolos a serem processados, e para tal foi utilizado o tipo *cadeia de caracteres* (*string do Visual Basic*). Isso impôs uma restrição ao Kards, pois na versão utilizada, o *Visual Basic* possui uma limitação para o tamanho deste tipo de dado que é de aproximadamente 32000 caracteres.

5.1. Sintaxe do Interpretador

O interpretador do Kards manuseia uma LFN Booleana na qual são descritas redes de processamento. Uma rede de processamento do Kards é organizada em 7 camadas, sendo estas:

- terminal
- seqüenciamento
- execução condicional / loop
- comparação
- agentes de interfaceamento kards/máquina/usuário , base de dados / atribuição e redes de processamento
- cálculo simbólico/numérico
- símbolos iniciais

Como a saída de um agente de uma camada intermediária não é obrigatoriamente conectada a um agente de um nível superior, a hierarquia da rede é descrita da esquerda (nível mais alto) para a direita (nível mais baixo), de modo que uma interface de entrada recebe sentenças de agentes à sua direita (em vez receber de baixo, como de costume) e sua interface de saída é à esquerda (e não acima). A rede é executada pelo interpretador da esquerda para a direita de modo a minimizar o custo de processamento para a obtenção do símbolo terminal. Os agentes condicionais são responsáveis pela minimização do custo através da ativação apenas das subredes que lhe fornecerão o símbolo de entrada necessários para a obtenção de sua saída.

Além da minimização de custo, a organização da rede também impõe uma hierarquia, de forma que mensagens colocadas em um quadro de avisos por um agente são visíveis apenas por eles

mesmos e pelos agentes que compõem suas subredes (que equivale então ao uso de variáveis locais em uma linguagem procedural), e as mensagens criadas por um agente são apagadas do quadro de aviso assim que este fornecer seu símbolo terminal ao próximo agente acima dele na hierarquia (à esquerda na rede). Há também um segundo quadro de avisos que não está sujeito a essa hierarquia, permitindo que todos os agentes possam se comunicar (equivalente ao uso de variáveis globais em uma linguagem procedural).

Essa minimização de custo permite que um mapeamento de um procedimento procedural seja feito para este tipo de processamento de rede e vice-versa (exemplos mais abaixo).

Todo o processamento do Kards se dá em torno desse interpretador, que foi denominado de "*calculadora*". Essa calculadora recebe um símbolo inicial com a descrição da rede a ser efetuada e devolve o símbolo resultante do processamento dessa rede (símbolo inicial). O símbolo inicial segue uma sintaxe definida internamente no sistema, que para facilitar o usuário na criação das redes de processamento, foi criada de modo que ficasse muito semelhante à sintaxe de uma linguagem procedural baseada em funções, como por exemplo a linguagem "C".

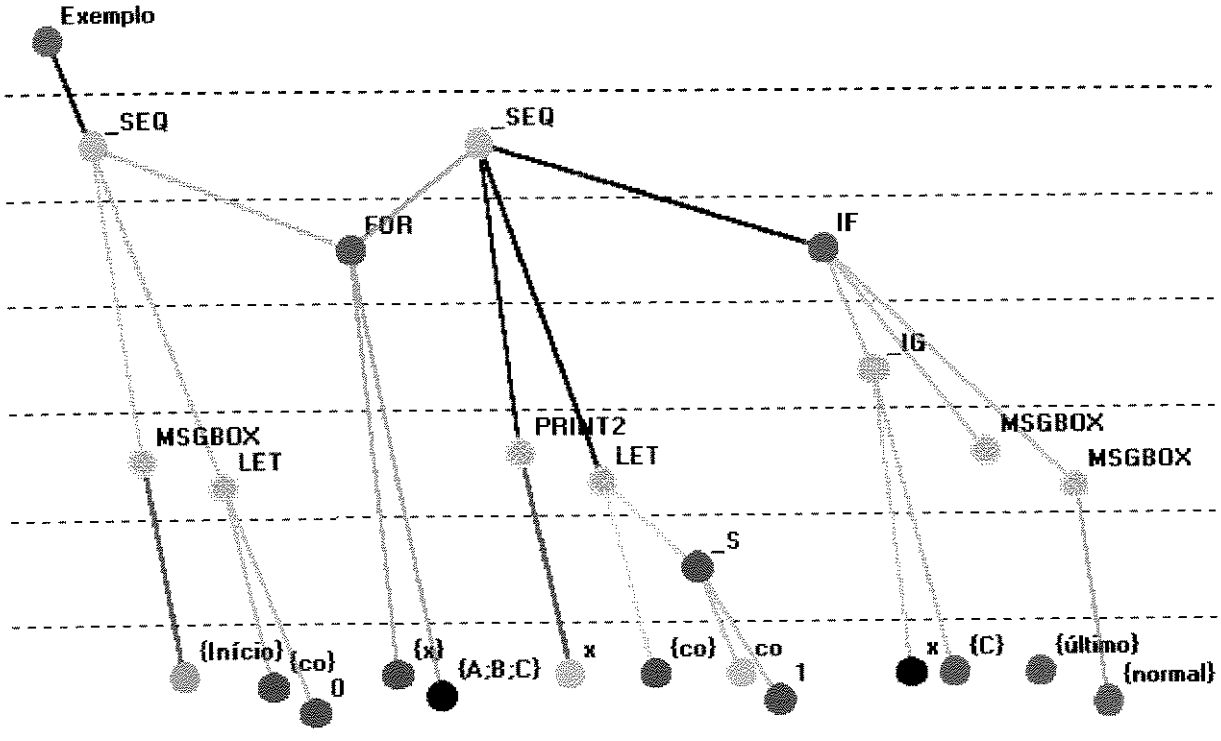


Figura 20 - Exemplo de rede de processamento

A rede mostrada na Figura 20 é representada pela sentença (explicada detalhadamente a seguir)

```
msgbox({Início});let({co};0);for({x};{A;B;C};(print2(x);let({co};co+1);if(x
={C};msgbox({último});msgbox({normal})))
```

Como o interpretador ignora espaços extras, tabulações e mudanças de linha, então pode-se descrever o símbolo em formato de programação estruturada:

```
msgbox( {Início} );
let( {co}; 0 );
for( {x}; {A;B;C};
 ( print2(x);
   let( {co}; co+1 );
   if( x={C};
     msgbox({último});
     msgbox({normal})
   )
 )
```

Como pode-se observar na Figura 20, um agente especial foi introduzido para realizar a serialização de procedimentos encontrada em linguagens procedurais. Esse agente é o agente primitivo *_SEQ*. Sua sintaxe é:

```
_SEQ( a; b; c; d  ... )
```

Esse agente pode possuir várias entradas, sendo que ele ativa uma entrada por vez, da primeira a última. O símbolo de saída do agente *_SEQ* é o símbolo recebido de sua última entrada, ou seja, será o símbolo de saída do último agente ligado às suas entradas.

A inclusão desse agente é realizada pelo interpretador, pois a sintaxe permite o uso de algumas otimizações de forma a facilitar sua utilização. Essas otimizações envolve a inclusão automática do agente *_SEQ*, e a substituição de operadores por funções, como no caso da soma observada na rede, onde *"co+1"* é substituído por *"_S(co;1)"*. Os agentes de comparação também podem ser descritos na sintaxe da linguagem como operadores, como no caso de *"x={c}"*, que é substituído por *"_IG(x;{c})"*.

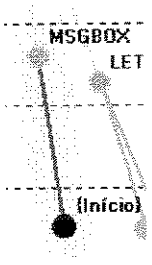


Figura 21 - Subrede 1

A execução da primeira subrede (Figura 21) do agente _SEQ inicial possui o agente *MSGBOX*. Esse agente recebe um símbolo "{Início}", onde as chaves servem para indicar que o símbolo 'Início' deve ser passado ao agente *'MSGBOX'* diretamente, de forma a não causar

ambigüidade quanto à possibilidade de 'Início' ser um símbolo qualquer ou a identificação de um agente. Esse agente coloca uma janela na tela com o símbolo que recebe, de modo que é apresentada uma janela com a mensagem 'Início' e então esse agente retorna como símbolo terminal "-1" para indicar que a mensagem foi mostrada.

Após a execução dessa primeira subrede, a subrede referente à segunda entrada do agente *_SEQ* principal é ativada (Figura 22).

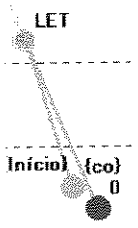


Figura 22 - Subrede 2

Essa subrede é composta pelo agente *LET* que recebe o símbolo "{co}" e o símbolo "0". O símbolo "0" é convertido para o valor numérico 0 (zero), pois nomes de agentes não podem iniciar com um caracter numérico, evitando com isso esse tipo de ambigüidade, de forma que símbolos a serem tratados como números são convertidos **pelo agente** que os trata para seus respectivos valores numéricos.

O agente *LET* é o agente responsável por colocar sentenças no quadro de avisos. Ele coloca a sentença contida em sua segunda entrada no quadro de avisos, junto com sua identificação que corresponde à primeira entrada do agente. De modo que no quadro de avisos é colocada uma sentença do tipo:

co $\Rightarrow$ 0

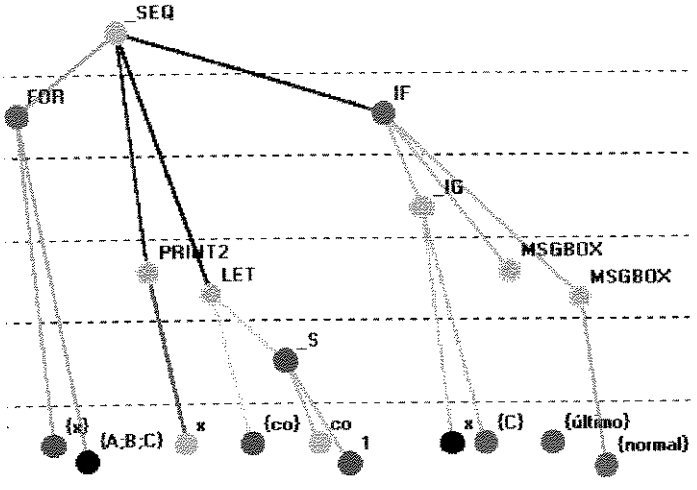


Figura 23 - Subrede 3

A terceira subrede (Figura 23) é formada pelo agente *FOR*. Esse agente é um agente de *Loop*. Ele é um agente primitivo complexo, que realiza o seguinte processamento: o símbolo de sua primeira entrada representa um identificador a ser usado para colocar mensagens no quadro de avisos; a segunda entrada representa uma lista (uma sentença que descreve um vetor - vide 5.2.1), onde para cada subsentença (cada elemento do vetor) é colocada uma mensagem no quadro de aviso com o identificador obtido na primeira entrada e com o conteúdo sendo essa subsentença. Esse *FOR* difere do comando *for* utilizado na maioria das linguagens procedurais, pois ele não é numérico, e sim simbólico, pois percorre um símbolo tipo vetor. O Kards possui também um *FOR* do tipo numérico. Após colocar a mensagem no quadro de avisos a terceira entrada é ativada. Como resultado, o agente *FOR* irá ativar a subrede de sua terceira entrada tantas vezes quanto forem o número de itens da sentença de sua segunda entrada.

A subrede ativada pelo agente *FOR* é composta pelo agente *_SEQ*, que como já visto acima, irá ativar cada uma de suas subredes de entrada. Então a sua primeira subrede (Figura 24) é ativada.

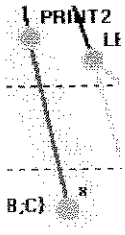


Figura 24 - Subrede 3.1

Essa subrede é composta pelo agente *PRINT2*, que coloca uma mensagem na tela e retorna com o símbolo "-1" indicando que a mensagem foi mostrada. A mensagem a ser mostrada é recebida em sua entrada, porém este agente não recebe diretamente um símbolo. Como mostrado na rede, ele recebe o resultado de um agente da camada de símbolos iniciais que é identificado como sendo "x". Esse agente não representa o agente "x", mas sim, um agente interno do sistema que realiza uma leitura do quadro de avisos, e retorna com a sentença do quadro de avisos que corresponder à sentença com identificador "x". Como o agente *FOR*, como visto acima, colocou uma mensagem com a identificação "x" no quadro de avisos antes que esta subrede fosse ativada, o agente *PRINT2* colocará a mensagem do quadro de avisos na tela, ou seja, da primeira vez que esta subrede é ativada, onde o agente *FOR* colocou o símbolo "A" no quadro de avisos, a mensagem mostrada é "A"; da segunda vez, "B", e assim por diante.

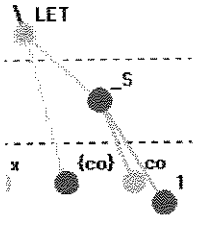


Figura 25 - Subrede 3.2

A próxima subrede ativada (Figura 25) possui novamente o agente *LET*. Porém, desta vez ele coloca uma mensagem no quadro de avisos que provém do agente *_S* (que soma dois valores). Esse agente recebe um dos valores a serem somados do próprio quadro de avisos. Como o agente *LET* ativa as duas entradas antes de realizar seu processamento, o valor proveniente da soma é primeiramente passado ao agente *LET*, para então este colocar a nova sentença no quadro de avisos. Quando o agente *LET* encontra uma sentença no quadro de avisos com o mesmo identificador da sentença que ele deve colocar, a sentença antiga é retirada do quadro e substituída pela nova. Isso equivale, em uma linguagem procedural como "C", à "co = co + 1", ou em pascal "co := co + 1".

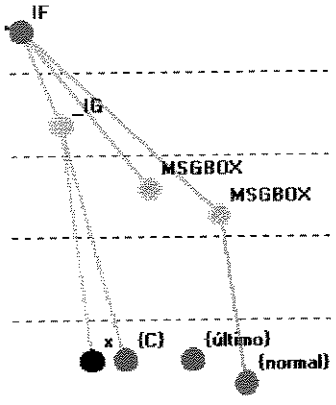


Figura 26 - Subrede 3.3

A subrede seguinte (Figura 26) possui o agente *IF*. Esse é um agente de execução condicional, onde sua sintaxe é:

IF(condição; expressão1; expressão2 *)*

Sua primeira entrada representa um valor numérico que, quando diferente de zero, causa a ativação de sua segunda entrada e quando igual a zero causa a ativação da terceira entrada. Sua primeira entrada é composta pelo agente *_IG*, que realiza uma comparação de símbolos e retorna uma sentença representando o valor "-1" se suas entradas forem iguais ou "0" caso sejam diferentes. O resultado final desta subrede é que a mensagem "último" é mostrada caso o quadro de avisos possua a sentença "{C}" com identificador "x" e a mensagem "normal" caso possua outra mensagem com esse identificador.

Devido à minimalização de custo na execução da rede, apenas uma mensagem é mostrada quando da ativação dessa subrede. Caso não houvesse essa minimalização, as duas mensagens seriam mostradas para se obter o símbolo final desta subrede.

Os agentes *_AND* e *_OU*, que realizam as operações lógicas "E" e "OU", são, no Kards, agentes condicionais, pois o agente "E" somente ativará sua segunda entrada caso a primeira entrada seja diferente de "0", e o agente "OU" só ativará sua segunda entrada caso a primeira seja igual a "0".

5.1.1.Descrição da Sintaxe

A sintaxe básica da sentença a ser processado pela calculadora foi baseada na sintaxe da linguagem C, onde há somente funções e não procedimentos, de modo que cada agente é descrito como uma função, onde os parâmetros da função referem-se à interface de aquisição de dados de cada agente.

Além dos agentes, a calculadora do Kards é dotada também de duas pilhas de sentenças, uma para o quadro de avisos local (o quadro de avisos que está sujeito à hierarquia da rede) e uma para o quadro de avisos global (o quadro de avisos que não está sujeito à hierarquia).

A sintaxe do Kards permite uma ambigüidade no sentido de que quando está sendo executada a rede, o nome do identificador retirado da sentença inicial pode referenciar um agente primitivo, uma rede de processamento, ou uma mensagem do quadro de avisos local ou global. Exemplo:

"MOSTRA({1})"

O identificador "MOSTRA" pode ser tanto um agente primitivo quanto uma rede de processamento. Este identificador não poderia ser uma mensagem do quadro de avisos porque foi definida uma entrada para esse agente, e agentes que representam mensagens do quadro de aviso não possuem entradas. Esta é uma maneira de reduzir a ambigüidade da sintaxe, porém, nem todos os agentes possuem entradas, de modo que a sentença

"MOSTRA"

poderia representar qualquer um dos casos citados acima.

Dessa forma, para 'executar' um agente, a calculadora do Kards procura identificar o *identificador* através da seguinte ordem:

1- mensagem do quadro de avisos local (caso não seja passado nenhuma entrada ao agente)
2- mensagem do quadro de avisos global (caso não seja passado nenhuma entrada ao agente)
3- agente primitivo
4- rede de processamento

De modo que se houver uma ambigüidade - ou seja, um agente primitivo com o mesmo nome de uma rede de processamento, por exemplo - o interpretador executará apenas o que for identificado primeiro - no caso, o agente primitivo.

Cada agente é responsável pela ativação de suas entradas. O processamento normal de um agente é ativar suas entradas ordenadamente, porém, alguns agentes primitivos como os

agentes condicionais, não ativam todas as suas entradas (como o agente '*IF*'), ou ativam a mesma entrada várias vezes (como o agente '*WHILE*' e o agente '*FOR*'), e não necessariamente na ordem em que foram definidas. Também existem agentes com número de entradas variáveis, onde seu processamento depende da quantidade de entradas (ou para realizar processamentos diferentes de acordo com o número de entradas ou assumindo um valor padrão quando a respectiva entrada não é definida), e também agentes onde o processamento é diferenciado caso a sentença de entrada represente um valor numérico ou uma sentença, como é o caso do agente de soma '*_S*'.

Para facilitar ao usuário, foram implementados operadores que facilitam a construção de expressões matemáticas no sistema. Esses operadores são:

-'+' : adição - realiza adição numérica caso ambos os operadores sejam numéricos ou a concatenação de símbolos caso contrário

-'-' : subtração numérica

-'*' : multiplicação numérica

-'/' : divisão numérica

-'^' : potenciação

-'&' : 'E' lógico

-'|' : 'OU' lógico

-'=', '>', '<', '>=', '<=', '<>' : comparação

Esses operadores representam os agentes '*_S*', '*_U*', '*_M*', '*_D*', '*_P*', '*_A*', '*_O*', '*_IG*', '*_MA*', '*_ME*', '*_AI*', '*_EI*' e '*_MM*' respectivamente, e são substituídos levando-se em consideração a precedência matemática do operador (na ordem potenciação, multiplicação e divisão, soma e subtração, 'e' e 'ou' lógicos e comparações).

5.1.1.1.Sintaxe BNF

A sintaxe da linguagem do Kards descrita em formato BNF seria:

```
Expression =
      Expression '+' Expression
|     Expression '-' Expression
|     Expression '*' Expression
|     Expression '/' Expression
```

```
|     Expression '^' Expression
|     Expression '&' Expression
|     Expression '|' Expression
|     Expression '=' Expression
|     Expression '<' Expression
|     Expression '>' Expression
|     Expression '<>' Expression
|     Expression '=>' Expression
|     Expression '<=' Expression
|     Expression '?' Expression ':' Expression
|     '(' Expression ')'
|     Expression ';' Expression
|     FloatLiteral
|     String
|     Identifier
|     Identifier '(' ArgList* ')'
;

ArgList =
      Expression (';' Expression)*
;

FloatLiteral =
      DecimalDigits '.' DecimalDigits? ExponentPart?
|     '.' DecimalDigits ExponentPart?
|     DecimalDigits ExponentPart?
;

DecimalDigits =
      ('0'|..|'9')+
;

ExponentPart =
      ( 'E' | 'e' ) ( '+' | '-' )? DecimalDigits
;

String =
      '{' Character* '}'
|     '"' Character* '"'
;
```

5.2. Tipos de Dados

A *calculadora* do Kards não trabalha com diversos tipos de dados como uma linguagem computacional normal; mas somente com símbolos. Alguns agentes do Kards, principalmente os agentes numéricos, tratam esses símbolos de forma especial, convertendo o conteúdo do símbolo no seu respectivo valor numérico, realizando o cálculo e convertendo o valor resultante novamente em um símbolo.

Para a utilização de tipos de dados mais complexos no Kards é necessário fazer um mapeamento da estrutura ou tipo de dado a ser utilizada para um símbolo. A seguir alguns exemplos desses mapeamentos utilizados pelo Kards:

5.2.1.vetor ↔ símbolo:

{1;2;3;4}

É utilizado um símbolo separador entre os elementos do vetor. O Kards possui diversos agentes internos especializados no tratamento desse tipo de dado, como por exemplo os agentes:

- *ITEM*: tem como saída um determinado item do símbolo de entrada. Uma de suas entradas é o número do item a ser retirado

- *ALTERA_COL*: tem como saída um símbolo onde determinado item é substituído por um novo símbolo

- *FIND_LIST*: usado para encontrar um item do vetor que contenha um determinado sub-símbolo

- *OCORRENCIAS*: conta o número de ocorrências de um símbolo dentro de outro. Serve para contar o número de itens quando o símbolo contado é o separador

Este tipo de mapeamento possui como restrição que o símbolo usado como separador não pode estar contido em nenhum dos elementos do vetor.

5.2.2.matriz ↔ símbolo:

{11;12;13;14 / 21;22;23;24 / 31;32;33;34}

Seguindo o mesmo esquema do mapeamento do vetor, é utilizado um símbolo entre cada vetor da matriz, assim como um outro símbolo entre os elementos de cada vetor da matriz.

Os agentes utilizados neste tipo de dado são os mesmos utilizados para os vetores, porém de forma encadeada. Alguns outros agentes primitivos tratam diretamente este tipo de mapeamento, como por exemplo:

- *TRANSPOSTA*: tem como símbolo de saída a sentença que eqüivale à matriz transposta da matriz mapeada no símbolo de entrada.

- *SOMATORIA*: tem como saída a soma de determinados elementos da matriz delimitados por uma coluna e linha inicial e uma coluna e linha final.

- *EXTRAI_COLUNAS*: tem como símbolo de saída a sentença que mapeia a matriz composta por determinadas colunas da matriz mapeada pelo símbolo de entrada.

Este tipo de mapeamento possui como restrição que os símbolos usado como separadores das colunas e das linhas não podem estar contidos em nenhum dos elementos do vetor.

5.2.3.lista identificada ↔ símbolo:

{A=v1;B=v2;C=v3}

Também pode ser tratado como uma combinação dos agentes utilizados com os vetores, mas o Kards também possui agentes especializados para este tipo de mapeamento, como por exemplo:

- *CONF_LIS*: tem como saída o símbolo associado à um identificador da lista, como por exemplo, se for dado como entrada para o *conf_lis* a lista acima e o símbolo "B", terá como símbolo de saída "v2".

- *SETA_CONF_LIS*: tem como saída a sentença de entrada com o subsímbolo referente ao símbolo identificador passado à este agente alterado para um novo símbolo também passado como entrada.

Este tipo de mapeamento é utilizado, por exemplo, para descrever os métodos de um agente de uma sentença de rede, onde é também utilizado o mapeamento do tipo matriz para descrever a rede inteira.

5.2.4.rede ↔ símbolo:

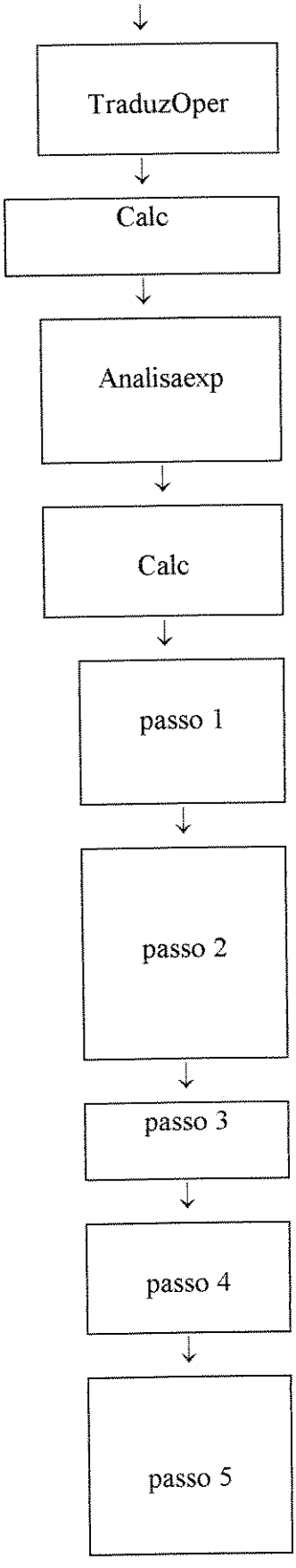
Para mapear as redes utilizadas no sistema em sentenças que podem ser armazenados na base de dados, foi utilizado um mapeamento equivalente ao da matriz, onde em cada linha é armazenado um agente da rede, e para cada agente são armazenadas informações sobre o nome do agente, o tipo de agente, suas ligações com outros agentes e os métodos associados à esse agente em cada uma das colunas. É utilizado um número fixo de colunas (os métodos estão em apenas uma coluna, sendo que esta coluna em si é uma *lista identificada*, e o mesmo se aplica à coluna das ligações de entrada e saída), e o número de linhas (nós) é indefinido, ficando limitado apenas ao tamanho máximo de um símbolo do sistema (32k)

Com a utilização de símbolos como separadores sucessivamente é possível mapear estruturas complexas em apenas uma sentença, como no caso das redes. Os símbolos utilizados como separadores não necessitam ser de apenas um caracter, podendo, com isso, criar uma infinidade de símbolos separadores.

5.3. Interpretador

O módulo responsável pelo interpretador do Kards é o *CALCMACR*, e para o processamento da sentença (símbolo), são seguidos os seguintes passos, na ordem em que estão abaixo:

função	descrição
Calcula	é a primeira a ser chamada, tendo como argumentos a sentença a ser executada, uma variável para retornar a descrição do erro, se este ocorrer e uma estrutura que representa o armário em que será executada a sentença (necessário para as funções de manipulação de armário - vide 5.5.1.)
↓	
ConverteExp	O primeiro passo aqui é reescrever o símbolo inicial em um outro símbolo contendo somente caracteres referentes aos agentes que serão interpretados, excluindo desse os símbolos definidos como parâmetros de outros agentes (Ex.: A sentença "print({f(a)})" é convertida para "printf({....})"). A sentença original e a nova passam a ser tratadas pelas funções posteriores. Essa conversão também faz a mesma coisa com os números (pois existem formatos de números que podem confundir o interpretador posteriormente, como "1E-2")
↓	



Realiza a substituição dos operadores matemáticos e lógicos pelas suas funções equivalentes, levando em consideração a precedência destes de acordo com a ordem das operações.

é quem realmente realiza a análise do símbolo a ser executado. Também é subdividida:

faz a análise do símbolo dividindo-o em *identificador* e uma lista de *parâmetros*. Essa lista é colocada em uma estrutura em que são guardados tanto o símbolo de controle quanto o símbolo original de cada um dos parâmetros passados à função.

Após obter o identificador e a lista de parâmetros, a função *Calc* passa então a tentar identificar esse *identificador*, através dos seguintes passos:

No caso da lista de parâmetros estar vazia, o interpretador verifica se o identificador está descrevendo diretamente um símbolo (numérico ou não). Caso esteja, retorna com o próprio símbolo, caso contrário, continua com o passo 2

Também no caso da lista de parâmetros estar vazia, verifica se o identificador está no quadro de avisos. Caso afirmativo, retorna a mensagem do quadro de avisos associada à ele. Primeiro é verificado o quadro de avisos local, e em seguida o quadro de avisos global. Caso ainda não tenha obtido sucesso, segue com o passo 3

Checa se o identificador é um agente primitivo. Caso afirmativo ativa o agente, caso contrário, vai para o passo 4

Checa se o identificador se refere à uma rede de processamento, que estão armazenadas no armário de estrutura da aplicação. Caso não obtenha sucesso, passa ao passo 5

Verifica se existe um arquivo '*Identificador.MAC*' no diretório da aplicação. Caso exista, esse arquivo é lido e assumido como sendo um arquivo que contêm uma sentença que descreve a rede de processamento a ser efetuada. Caso não obtenha sucesso, gera um código de erro.

Na execução de cada agente primitivo, as subredes de entrada passam a estar descritas na estrutura usada para armazenar os parâmetros da função, de modo que para a execução (ativação) destas, a função *Calc* é chamada recursivamente com os símbolos do parâmetro a ser ativado. As redes de processamento podem receber parâmetros através do quadro de avisos. Para tal, a primeira linha da sentença da rede de processamento contém a definição dos identificadores a serem colocados no quadro de avisos e opcionalmente um valor padrão para o caso do parâmetro não ter sido definido quando da chamada para essa rede.

5.4. Redes

O Kards possui um módulo destinado ao tratamento de redes. Este módulo possui além da definição de uma estrutura interna de rede, funções para o tratamento dessa estrutura.

A estrutura de *Rede* do Kards possui as seguintes variáveis:

- **NumNós**: variável numérica com o número de agentes da rede
- **Nó(..)**: vetor do tipo estrutura *Nó* com as seguintes variáveis:
    - **Nome**: string com o nome do agente
    - **Tipo**: string com o tipo de agente
    - **Rec**: string com a lista identificada dos agentes dos quais este agente recebe símbolos
    - **Emi**: string com a lista identificada para os quais este agente manda símbolos
    - **Fun**: string com a lista identificada de métodos associados ao agente
    - **Valor**: string com o valor atual do agente
    - **Descr**: string com uma descrição do agente (utilizado para mostrar as redes na tela)
    - **Cor**: cor do agente (utilizado para mostrar as redes na tela)
    - **Pos**: posição do agente (utilizado para mostrar as redes na tela)

O nível de cada agente é definido pela estrutura de ligação dos agentes e pelo seu tipo, sendo que o primeiro agente é sempre o do nível superior, e a partir dele se chega aos outros agentes e níveis. É utilizada uma lista duplamente ligada para facilitar a navegação da rede.

Para melhorar o desempenho, uma função de 'compilação' da rede percorre toda a estrutura colocando nas listas **Rec** e **Emi,** para cada agente destas, o respectivo número no vetor **Nó** do agente ao qual se referem, de forma a otimizar a procura do agente.

Várias funções foram criadas para a utilização dessa estrutura, dentre as principais:

- Num_Nó: devolve o número de um agente no vetor (recebendo o nome deste)

- Compila_Rede: otimiza a devolução do número do agente

- Ajusta_Rede: ajusta a posição e cor dos agentes para que sejam mostrados na tela

- Mostra_Rede: mostra uma rede na tela

- Seta_Valor: ajusta o valor atual de um determinado agente

- Valor: devolve o valor atual de um determinado agente

- Seta_Meto_Nó: ajusta um determinado método de um agente

- Meto_Nó: devolve o método especificado de um agente. Esta função é a responsável pela característica de *herança* da rede, pois é ela quem percorre a rede nos níveis acima quando não é encontrado o método desejado no agente especificado. A lista de nós **EMI** é percorrida sucessivamente (nível após nível) até que se encontre uma definição de método herdável ou se chegue ao topo da rede. Neste caso, é verificado se existe o método especificado em uma lista de métodos padrões.

- Calc_Meto: calcula o método especificado de um nó (utilizando a função *Meto_Nó* para obter o método a ser calculado)

- Expr_Rede: converte um símbolo da sintaxe da linguagem do Kards para a estrutura de rede

- Rede_Expr: converte uma rede para a sintaxe da linguagem do Kards

- Str_Rede: converte um símbolo com a definição de uma rede para a estrutura interna de rede

- Rede_Str: converte da estrutura interna de rede para um símbolo com a definição da rede

- Exec_Filhos: executa determinada expressão para todos os nós da lista **Rec** do nó especificado

5.5. Base de Dados

5.5.1.Estrutura

Para a manipulação dos agentes da base de dados, o Kards utiliza a estrutura interna de *Redes*. A estrutura interna para o tratamento de um armário compreende uma rede onde é armazenada a estrutura básica do armário (*armário-gaveta-pasta-ficha*) e os valores referentes ao armário, gaveta, pasta e ficha atualmente selecionados, uma outra rede onde é armazenada tanto a estrutura da ficha atual quanto os valores dos campos da ficha selecionada, e além dessas duas redes são utilizadas algumas outras variáveis internas para otimização.

Como praticamente todos os agentes de base de dados trabalham em cima do armário atual, foi criada uma forma de se poder trabalhar com mais de um armário ao mesmo tempo: o sistema trabalha com um vetor de até cinco armários, e um agente especial (*push_arm*) faz com que o armário atual seja copiado para o próximo armário disponível, permitindo com que uma sentença seja executada em cima dessa nova cópia do armário e após a execução dessa sentença o novo armário seja liberado e o armário atual volte a ser o original, sem que este tenha sofrido qualquer alteração com relação aos valores das redes que estão em memória. Com isso é possível que se tenha várias janelas diferentes do sistema atuando em armários diferentes ao mesmo tempo (por exemplo uma janela acessando o armário com o *Help* da aplicação enquanto se utiliza o armário da aplicação), pois alguns agentes especiais criam novas janelas que executam redes de processamento enquanto o processamento da janela principal é feito concorrentemente pelo sistema operacional.

Devido ao fato do armário 'arma' se autodefinir, o sistema Kards possui um conhecimento predefinido que define a sintaxe de uma sentença tipo rede armazenada na base de dados, onde a partir da obtenção e interpretação dessa sentença ele passa a poder definir toda a leitura e entendimento da sentenças que definem a aplicação e o próprio armário 'arma'.

5.5.2.Arquivos

A organização dos arquivos da base de dados foi feita utilizando-se vários arquivos para cada armário, ao invés de um único arquivo com todos os dados. Isso simplificou as rotinas a serem implementadas sem comprometer o sistema, pois o sistema operacional possui diversas otimizações para tratar arquivos e diretórios.

Os nomes dos arquivos são definidos nos métodos da rede com a estrutura básica do armário *armário-gaveta-pasta-ficha*, e, como padrão, o sistema utiliza a seguinte convenção:

ARMA*fi*GA.DOC

onde:
ARMA é o nome do armário
fi é o número da ficha
GA é a sigla da gaveta

de forma que em um arquivo '.DOC' são armazenadas todas as sentenças (todos os registros) referentes à ficha '*fi*' da gaveta 'GA' do armário 'ARMA'. Com isso o número de arquivos criados para armazenar o armário é igual ao número de fichas vezes o número de gavetas, sendo que somente são criados os arquivos que conterão sentenças, isto é, se a ficha 15 for utilizada somente em uma gaveta, será criado apenas o arquivo 'arma15ga.DOC' relativo a essa gaveta. Cada arquivo '.DOC' é um arquivo aleatório, com registros de tamanho 'tam_fic' (tamanho da ficha, definido como método da ficha na criação de cada estrutura de ficha). Em cada um desses registro é armazenado uma sentença (símbolo). Devido à uma característica do Sistema Operacional, quando se grava, por exemplo, o décimo registro de um arquivo aleatório sem que existam os registros 8 e 9, os registros 8 e 9 são preenchidos com um 'lixo' indefinido. Para evitar que o Kards leia esses registros preenchidos com 'lixo', os registros do Kards são gravados com uma assinatura (um '#*') no final de cada sentença, de forma que se não houver essa assinatura, o registro lido é assumido como sendo um registro vazio.

Como para cada tipo de ficha essa sentença é interpretada de um jeito, não existe nenhuma outra restrição sobre os símbolos armazenados na base de dados (as restrições são de que o registro não contenha a assinatura dentro deles - a sentença não pode conter um '#*' - e o tamanho do símbolo não pode exceder 'tam_fic'-2).

Além desses arquivos '.DOC', o sistema utiliza arquivos '.ORD' para indexar os arquivos '.DOC'. O nome dos arquivos '.ORD' são definidos no padrão como sendo:

ARMA01**GA**.ORD

onde:
ARMA é o nome do armário
GA é a sigla da gaveta

Em cada '.ORD' é armazenada uma sentença (na forma de lista - um arquivo texto cujo símbolo separador é o 'New-Line') sendo que o conteúdo de cada linha é referente à cada pasta da gaveta e está no formato

nn,GA / nome_índice [; comentários]

onde
nn é o número do registro da pasta nos arquivos '.DOC'
GA é a sigla da gaveta, utilizada em índices secundários (descritos mais abaixo)
nome_índice é o símbolo que descreve a pasta ('nome' do registro / chave de procura)
; comentários são dados extras que podem ser colocados no arquivo '.ORD' e que não são exibidos na hora de selecionar uma pasta na tela

Um outro arquivo é utilizado opcionalmente para cada armário. Esse arquivo é

ARMA.DEF

onde
ARMA é o nome do armário

Esse arquivo contém as seguintes informações:

IMPORT=dados_de_importação
ESTRUTURA=armário_de_estrutura

onde
dados_de_importação pode estar em um de dois formatos diferentes: ou descreve um método que realiza a importação rápida de registros ou contém uma lista com o tamanho das fichas do armário. Esse dado é utilizado para otimizar a leitura rápida de dados da base de dados.
armário_de_estrutura é o nome do armário de estrutura do armário. Como padrão, é utilizado o nome do armário duplicado. Exemplo: o padrão para o armário \CONTABIL\CONT é \CONTABIL\CONT\CONT

Devido as convenções utilizadas na formação dos nomes de arquivos, e ao Kards ter sido feito em cima do Sistema Operacional Windows 3.1, onde os nomes de arquivos são no formato 8.3, têm-se mais três restrições à base de dados:

- O nome do armário não pode exceder 4 caracteres

- Podem ser utilizadas no máximo 99 fichas (a numeração começa em 01)

- As siglas de gavetas devem ser de dois caracteres

5.5.3.Criação de Armários

Um sub-módulo do módulo de banco de dados do Kards é o módulo *GERAR*. Nesse módulo estão as funções responsáveis pela criação de novos armários. Essa criação é feita criando-se o diretório para o novo armário, o diretório para o armário de estrutura e copiando-se a gaveta 'AR' (que contém a definição da rede do armário padrão: *armário-gaveta-pasta-ficha*). Depois desta etapa, será necessário editar o armário de estrutura para criar as gavetas (editando a gaveta *gavetas*), criar as fichas (editando a gaveta *fichas*) e as outras gavetas que forem necessárias (*funções nebulosas*, etc). Há também a possibilidade de *especializar* o novo armário, quando então são copiadas as estruturas dos outros armários auxiliares (de macros, dicionário e *help*).

Foram criadas funções para auxiliar a criação das fichas de redes de estrutura das novas fichas, visto que a edição da ficha tipo rede que descreve a ficha não é prática. Para tal, primeiro foi criado um formulário onde são informados o tipo de ficha e tamanho do registro da nova ficha, como mostra a Figura 27. De posse do tipo de ficha, o sistema faz então a edição da rede de acordo com o tipo de ficha, sendo que a rede de uma ficha do tipo *rede* e a de uma ficha do tipo *texto* são montadas automaticamente. Para os outros tipos de ficha, é feita uma conversão da sentença que descreve a rede para um formato texto, onde o usuário edita essa definição em uma janela tipo editor de textos simples (Figura 28).

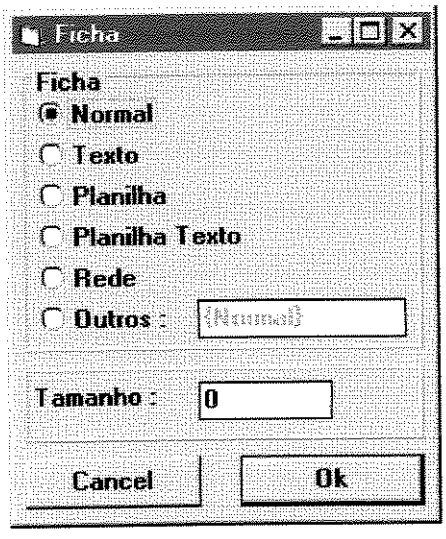


Figura 27 - Formulário de seleção do tipo de ficha

Após a edição da estrutura da ficha é possível definir os métodos para cada agente da rede através de um formulário especial, acessado diretamente a partir da edição da ficha tipo rede.

É importante salientar que na conversão para o formato texto não são incluídos os métodos associados à cada agente, sendo que toda vez que for alterada a estrutura de uma ficha deve-se checar os métodos, que são armazenados antes da edição e recolocados na rede após esta na mesma ordem apenas para o caso de se desejar alterar apenas o nome de um agente.

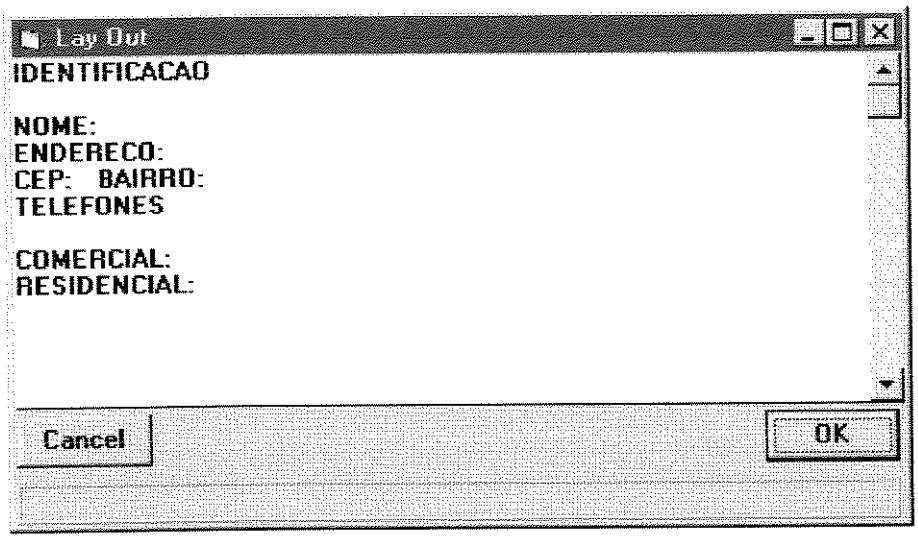


Figura 28 - Exemplo de definição de uma ficha no formato texto

Este exemplo gera a rede da Figura 29, que descreve o formato da Figura 30.

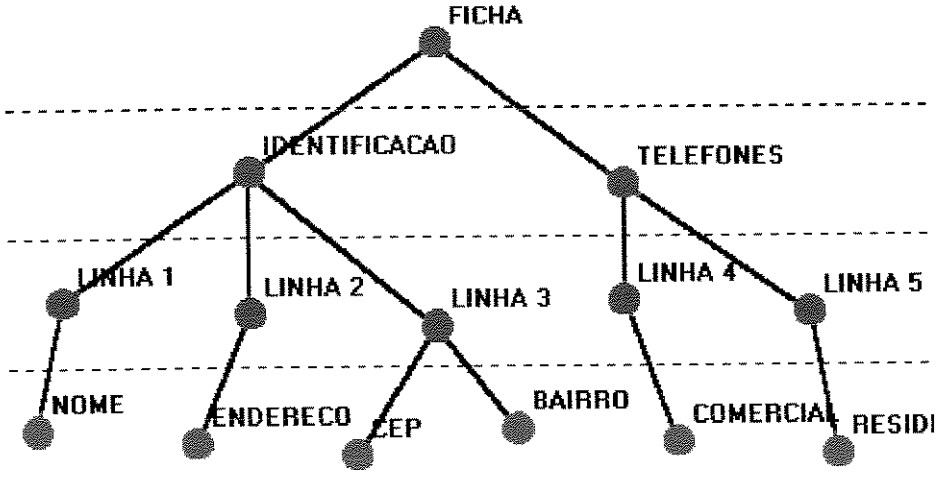


Figura 29 - Rede que foi descrita no formato texto

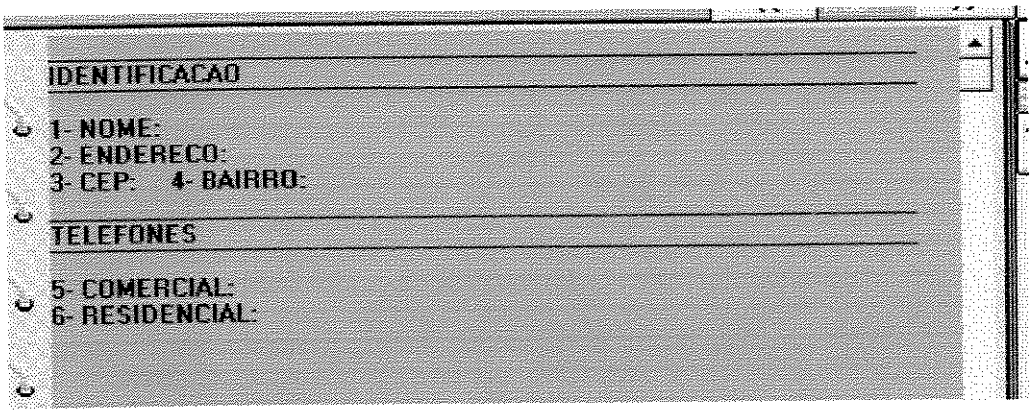


Figura 30 - Estrutura da ficha descrita no editor em modo texto

5.6. Quest

As redes de raciocínio descritas em (4.3) são implementadas através do processamento direto das sentenças que definem essas redes. Essas sentenças são armazenadas no sistema em uma ficha do tipo rede e a edição dessas redes é feita de forma visual através de um editor de redes, mostradas na Figura 31, onde os métodos de cada agente podem ser definidos através de uma janela.

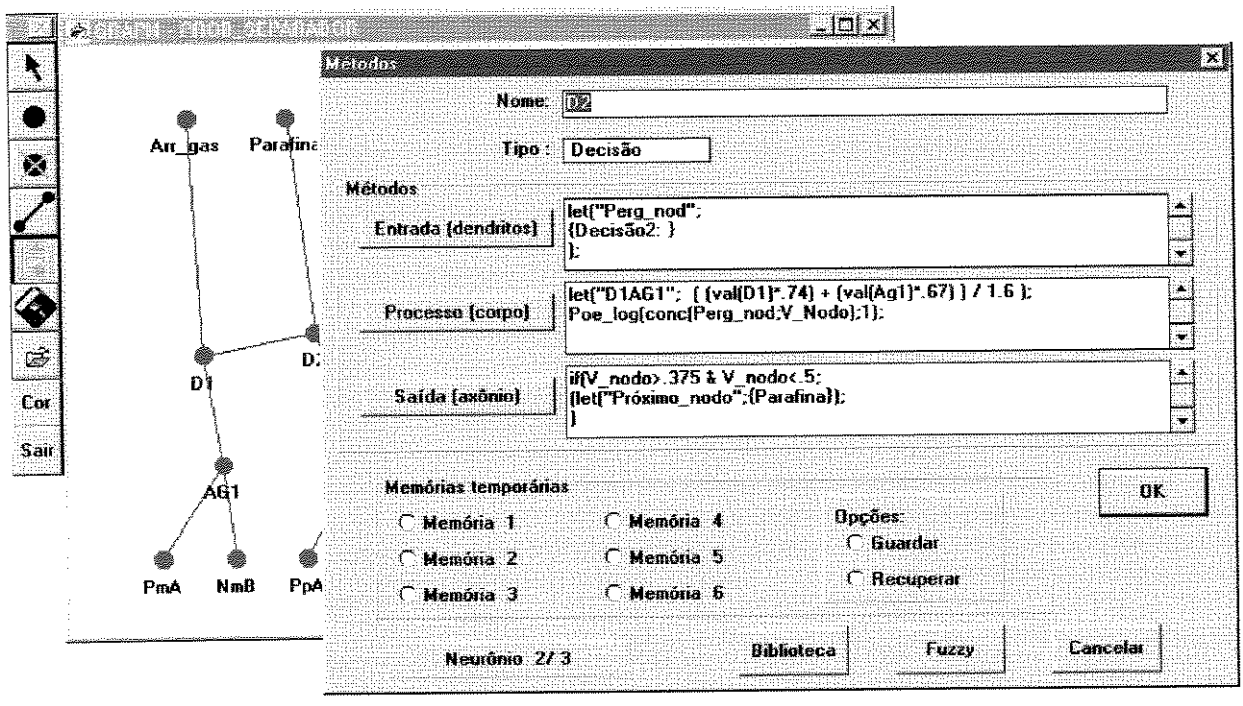


Figura 31 - Telas do editor de redes de raciocínio

A execução da rede é feita carregando-se a sentença que a descreve para a memória em um vetor de strings do Visual Basic onde em cada variável é armazenada uma célula da planilha texto que a sentença representa. O editor de redes grava a rede de forma que a primeira linha da sentença seja a linha referente ao primeiro agente que deve ser executado (ativado). Antes da execução dos métodos de um agente, são colocados no quadro de avisos global o nome do agente, o tipo de agente ('T' - Terminal, 'A' - Associação, 'D' - Decisão ou 'S' - Saída) e os métodos que serão executados. Esses métodos formam uma sentença do Kards descrevendo uma rede de processamento.

Em seguida, o Kards executa as redes de processamento referentes ao método inicial, método intermediário e método final do agente. O símbolo de saída do método final é colocado no quadro de avisos com um identificador igual ao nome do agente, de forma que qualquer agente executado (ativado) após ele poderá ter acesso ao seu símbolo de saída através do quadro de avisos.

No caso dos agentes de *Decisão* e dos agentes de *saída*, o quadro de avisos pode ser utilizado para definir qual será o próximo agente ou rede a ser ativado. Para o caso de agentes de

Decisão, isto está claro nos casos em que um agente pode causar a ativação de um ou de outro agente da rede, de forma que a definição de qual agente será ativado é feita através da colocação do nome do próximo agente a ser ativado no quadro de avisos com um identificador interno ('Próximo_Nodo'). Para o caso dos agentes de saída, o quadro de avisos pode ser usado para informar ao sistema que uma outra rede de raciocínio deve ser acionada a partir deste agente de saída. Isto é feito colocando-se no quadro de avisos, com o mesmo identificador interno citado acima, uma mensagem definindo que a rede de raciocínio *x* deve ser ativada (a mensagem colocada é composta pelo símbolo '#' seguido do número *x* da rede).

O acesso à base de dados e aos outros recursos do sistema é feito através da definição da rede de processamento utilizada em cada agente, pois essas redes podem envolver qualquer agente do sistema, estando aí incluídos os agentes de acesso à base de dados, às funções de pertinência, às funções de cálculo numérico, etc. A ativação de determinada rede de raciocínio também se dá através de um agente primitivo.

5.7. Jargão

O agente primitivo responsável pelo processamento de um dicionário contido no armário de linguagem é o agente '*Frase*', que recebe a sentença com o dicionário que é lida do armário de linguagem, e uma sentença **A** que será analisada com o uso desse dicionário (a sentença que é lida da ficha texto a ser analisada).

A análise é feita a partir do dicionário composto de uma rede que define a sintaxe para determindado conceito. A rede é analisada de cima para baixo, a fim de reduzir o custo de processamento. O primeiro agente conectado ao agente de saída (conceito) é o agente que define o gatilho para aquele conceito, ou seja, o resultado da consulta (análise da rede) depende da ativação de seu gatilho. Após o gatilho, o próximo agente é analisado. Os agentes do nível de decisão (logo abaixo do conceito) são agentes que exigem que todas as suas entradas sejam ativadas, ou seja, agem como agentes com uma operação lógica do tipo 'E'. Os agentes do nível de associação (logo acima dos terminais) possuem o tipo de operação definido em um de seus métodos, de forma que podem agir como 'E' (necessitando da ativação de todas a entradas) ou como 'OU' (necessitando da ativação de pelo menos uma entrada).

Para a análise de um agente da rede de dicionário, o Kards realiza uma procura na sentença **A** pelas palavras que estão associadas ao agente, sendo que esta pesquisa é feita levando-se em consideração a distância máxima em que essas palavras podem estar em relação à última palavra encontrada (começando pelo gatilho). No final, uma certeza é atribuída ao conceito de acordo com o número de agentes ativados e a distância em que as palavras associadas à esses agentes foram encontradas.

Como saída, o agente '*Frase*' retorna uma sentença com a lista de todos os agentes ativados, as palavras que ativaram esses agentes, e o valor de certeza do conceito em relação à sentença

analisada. Uma rede de processamento é responsável por realizar a pesquisa em todas as redes definidas no armário de linguagem, e, se necessário, ativar a rede de raciocínio correspondente.

A Figura 32 mostra o exemplo da sentença retornada pelo agente '*Frase*' para a análise realizada no exemplo mostrado com a Figura 14, Figura 15 e Figura 16.

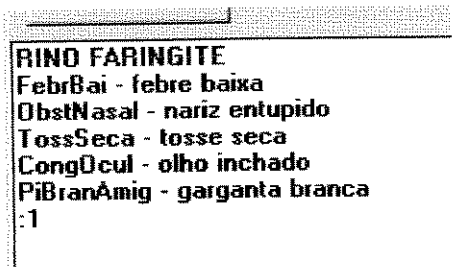


Figura 32 - Exemplo do resultado do agente '*Frase*' do Jargão

5.8. Sensor

A aquisição de dados através de scanners foi implementada através da importação de arquivos de imagens nos formatos ".PCX" e ".BMP" suportados pelo Visual Basic. O tratamento desses arquivos importados possui a restrição de que apenas são tratados os dados visíveis em tela, de modo que o tamanho da figura e o número de cores tratado dependem do adaptador de vídeo e da configuração utilizados pelo Windows. O agente responsável pela importação da imagem retorna uma sentença com a codificação de cada ponto da imagem.

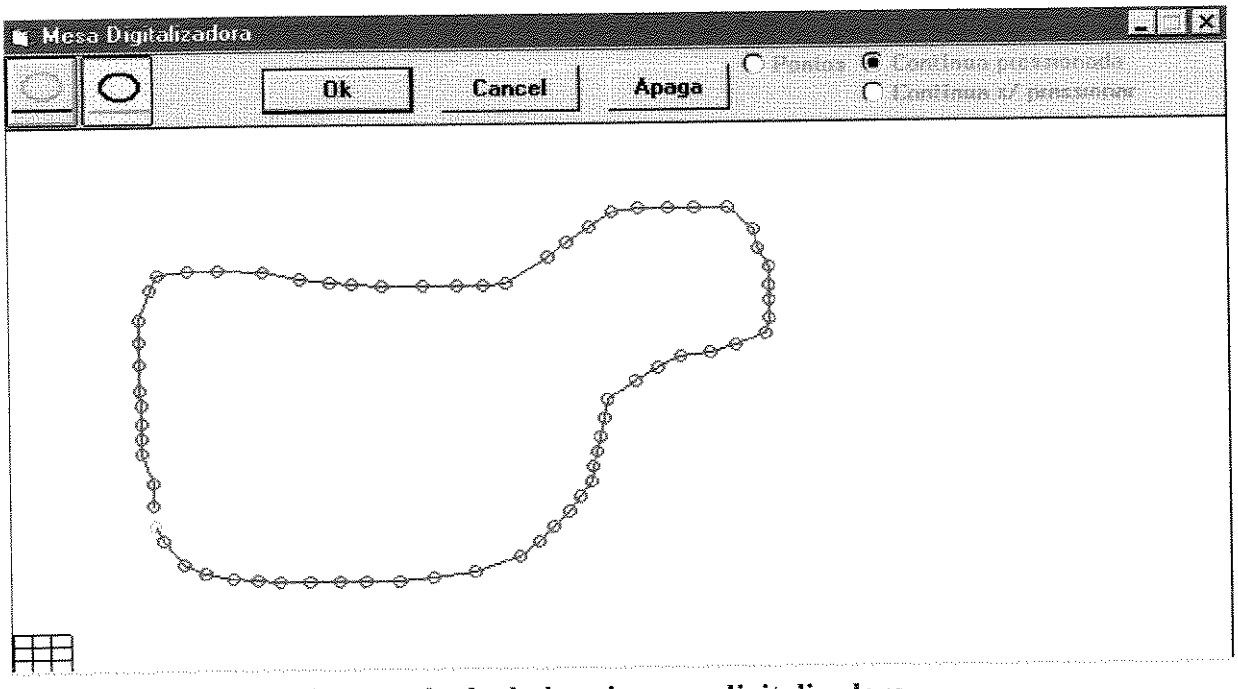


Figura 33 - Exemplo de entrada de dados via mesa digitalizadora

A aquisição de dados através de Mesa Digitalizadora é feita em um formulário especial para a entrada de curvas. A comunicação com a mesa digitalizadora é feita através do *driver* desta para o Windows, de modo que a mesa digitalizadora funciona como se fosse um *mouse*, porém com coordenadas absolutas, o que permite uma digitalização adequada. Assim como na aquisição de imagens, também há uma restrição quanto a resolução utilizada, pois como os dados são adquiridos através de uma janela, há um limite de tamanho para essa janela, que depende do adaptador de vídeo e da configuração utilizados. A Figura 33 mostra um exemplo de entrada de dados pela mesa digitalizadora.

A entrada de dados via placa de conversão analógica-digital e a atuação via placa digital-analógica podem ser feitas através de três placas diferentes:

- CAD 12/32: Placa conversora analógico-digital com 32 canais de entrada de 12 bits, e 16 entradas e 16 saídas digitais nível TTL
- CDA 12/08: Placa conversora digital-analógico com 8 canais de saída de 8 bits;
- CAD 12/36: Placa conversora analógico-digital e digital-analógico com 32 canais de entrada e 2 ou 4 canais de saída, e 16 entradas e 16 saídas digitais nível TTL

A comunicação com as placas é feita através de chamadas a DLLs fornecidas pelo fabricante das placas. Podem ser instaladas mais de uma placa de cada modelo em um mesmo computador, bem como podem ser misturadas placas de modelos diferentes.

O sistema Kards possui redes de processamento pré-definidas que permitem um fácil acesso a essas placas, de modo que a operação delas a nível de construção do raciocínio é simplificada. Para isso, basta criar uma sentença com as devidas conversões de valores que devem ser realizadas para cada canal lógico a ser utilizado e também em que placa física e canal físico da placa os canais lógicos serão mapeados. Com isso, para a construção do raciocínio essa rede de processamento se encarrega de realizar a leitura nas diversas placas, fazer as conversões necessárias e fornecer como símbolo de saída dessa rede um vetor com os valores respectivos a cada canal da sentença de configuração; e para a atuação possui uma rede de processamento que recebe um vetor com os valores a serem ajustados nas placas realizando as devidas conversões definidas em uma sentença de configuração para a atuação.

Para a análise de imagens adquiridas através de scanners, o Kards possui agentes que realizam as seguintes tarefas (Figura 34):

- **luminância**: transforma uma imagem colorida em preto e branco
- **contraste**: detecta os contrastes de luminosidade na imagem
- **bordas**: extrai contornos da imagem
- **direções**: extrai direções de uma seqüência de pontos (borda) da imagem
- **pontos_sig**: extrai pontos significativos a partir das direções
- **classifica**: classifica pontos significativos de acordo com um padrão
- **perceptron**: classificação avançada de uma curva em relação a um padrão

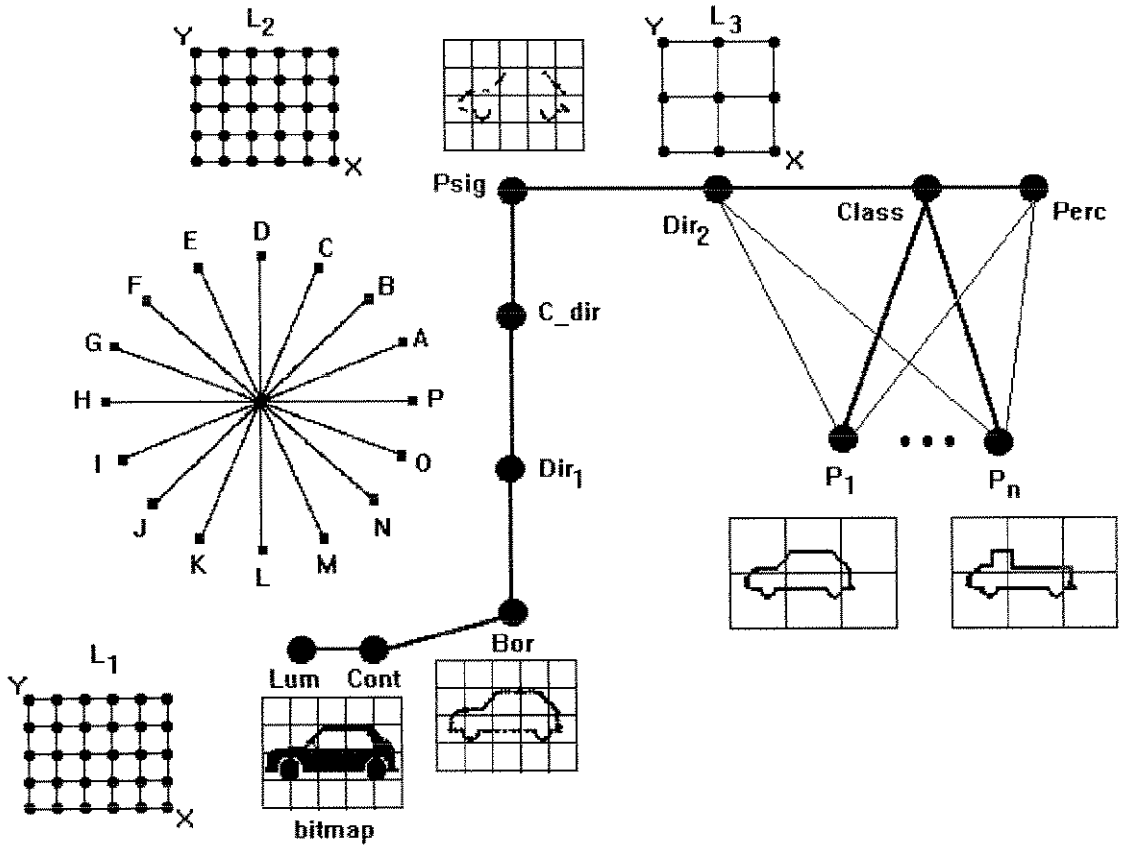


Figura 34 - Seqüência usada pelo sensor para análise de objetos

Esses agentes são os responsáveis pelo reconhecimento de padrões no Kards quando da aquisição de uma imagem ou uma curva através de mesa digitalizadora.

6. Sistema SIEP - Exemplo de Aplicação

6.1. O Sistema SIEP

O Sistema Inteligente para Elevação de Poços e Controle de Processos Petrolíferos (**SIEP**) consiste de um conjunto de aplicações Kards que integram o gerenciamento de métodos de elevação artificial de petróleo e os respectivos processos de tratamento do petróleo produzido. O **SIEP** foi implementado e testado na Planta da Estação de Compressão e Coleta de Livramento (PETROBRÁS - RN), e foi objeto de uma tese de doutorado [Pat96], que tratou da parte relativa ao conhecimento específico sobre o assunto e da presente tese que trata do desenvolvimento do ambiente (software e interface com placas para aquisição e atuação) do sistema Kards que permitiu a construção deste sistema.

Este sistema é composto de quatro aplicações Kards (Figura 35) que são:

- uma aplicação para controle automático de uma **planta de separação** óleo-gás;
- uma aplicação para controle automático das operações de um **poço de "gas lift"** contínuo;
- uma aplicação para o gerenciamento das operações de um poço de bombeio mecânico (**SICAD**), e
- uma aplicação para **gerenciar** estes módulos, fazendo com que eles possam interagir.

O SIEP apresenta várias inovações na atividade de automação do controle de poços e processos. Dentre elas destacam-se: a centralização de todas as operações dos vasos e dos equipamentos da planta na sala de controle e a interação do operador com o sistema.

Em sua grande maioria, os sistemas de controle manuais e automatizados de processos têm como filosofia de construção e operação o funcionamento local em cada vaso-equipamento, usando periféricos instalados juntos aos componentes da planta. No SIEP todo controle é centralizado em uma sala de operação e todas as redundâncias ditadas por medidas de segurança de operação estão projetadas com a mesma filosofia. Isto facilita, em muito, a operação e a manutenção dos instrumentos dos poços e da planta de processo.

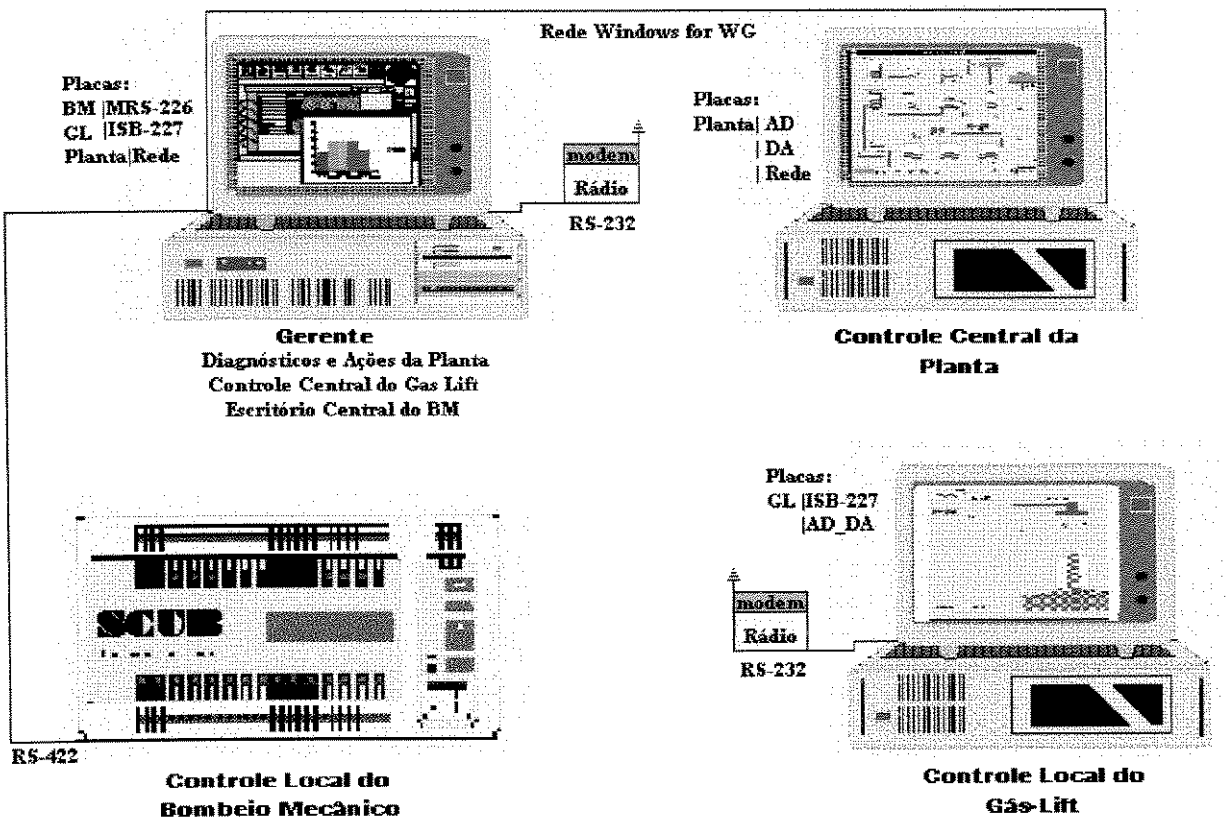


Figura 35 - O Sistema SIEP

6.2. Descrição Geral

A descrição do sistema no que se refere a conhecimento específico de petróleo é um resumo do que se encontra em [Pat96].

6.2.1.Planta de Separação

A **Planta de Separação**, opera com dois trens de separação óleo-gás, caracterizados pelo recebimento de fluidos provenientes de dois reservatórios adjacentes à estação de produção.

Os equipamentos que constituem a planta são os seguintes:

Trem I

"Manifold" de Bombeio Mecânico (Man BM): Equipamento constituído de tubos e válvulas que recebe e distribui os fluidos produzidos pelos poços de bombeio mecânico do campo produtor correspondente.

"Manifold" de "Gas Lift" (Man GL): Recebe e distribui os fluidos provenientes do poço de "gas lift".

Separador de Produção (Sep 1): Vaso de pressão que separa o óleo (com água) do gás, de todos os poços do campo, exceto o poço que é dirigido para o separador de teste.

Separador de Teste (Sep Test 1): Semelhante ao separador de produção, porém opera com um só poço.

Tratador de Óleo (Trat 1): Vaso também de pressão, que separa o óleo da água produzida pelos poços do trem.

Trem II

Este Trem é praticamente igual ao Trem I, diferenciando-se do primeiro pelo fato de que não possui o *"Manifold" de "Gas Lift"* e o número total de poços é menor.

Vasos e Equipamentos Comuns aos dois Trens

Depurador de gás (Dep): Vaso fundamentalmente de pressão, que elimina do gás as últimas gotículas de óleo proveniente dos separadores.

Compressor de Gás Natural (Compressor): Comprime o gás elevando sua pressão aos valores necessários para execução do "gas lift" e para envio à UPGN (Unidade de Processamento de Gás Natural).

Distribuidor de Gás: Equipamento semelhante ao "manifold", porém de alta pressão, que faz a distribuição do gás comprimido para os consumidores prioritários que são os poços de "gas lift", "plunger lift" e "pig lift", envia o volume contratado para a UPGN e permite a queima do excesso

Queimador de Gás: Considerado equipamento de segurança da planta, recebe todo o excesso de gás oriundo de situações naturais ou emergências. Tem como característica não poder operar apagado.

Tanques de Coleta e Armazenagem de Óleo para Bombeio (Tanque A B C): Vasos que operam à pressão atmosférica, responsáveis pela coleta e estabilização do óleo que será bombeado.

Bombas de Transferência (Bomba): Transfere o óleo à pressão necessária de envio aos terminais de carregamento.

Válvulas de Controle e Medidores: Instrumentos de aquisição de dados, supervisão, atuação e aferição do controle.

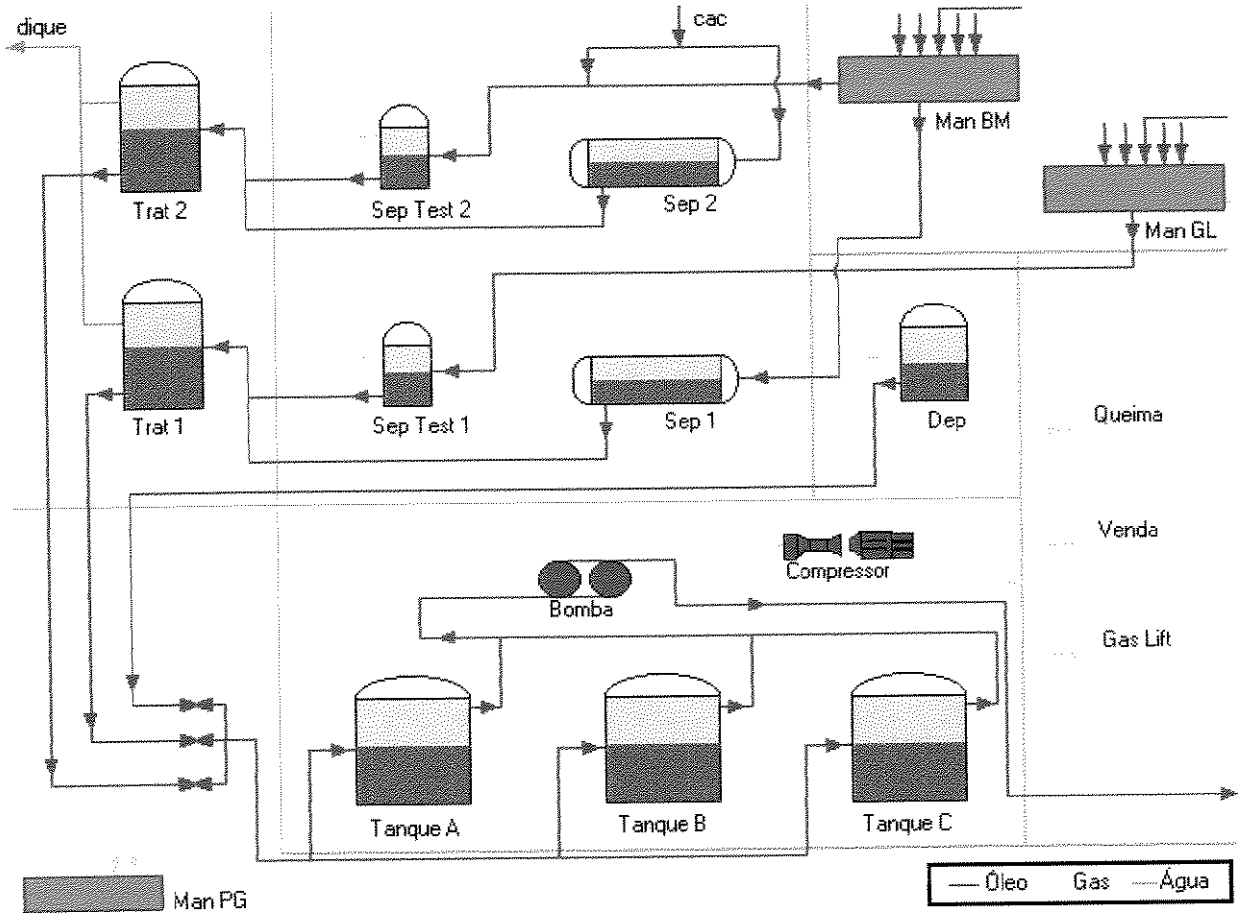


Figura 36 - Planta de Separação

6.2.1.1. Descrição do Processo de Separação

Na planta ocorrem os processamentos do **óleo cru**, da **água** e do **gás natural**. O petróleo dos poços produtores (mistura básica de óleo/gás/água) é transferido até os **"manifolds"** dos trens correspondentes, e de lá para os **separadores de teste** (apenas um poço por separador) e **de produção** (os demais poços).

Após a separação, o gás sai pelo topo dos **vasos de separação**, seguindo até o **depurador de gás**, onde são eliminadas as últimas gotículas remanescentes de óleo. O gás é então enviado para a **compressão** e em seguida, para o **processo de distribuição**.

O óleo cru e a água saem pela parte inferior dos separadores, seguindo até o respectivo **tratador**. O tratador a água e o óleo são separados. O óleo vai para os tanques, onde permanece até a estabilização; seguindo então para as **bombas,** via oleoduto. O pouco de gás que chega junto com o óleo nos tratadores é enviado controladamente para o **queimador**. A água é drenada manualmente para o **dique**.

Os integrantes dos trens de produção são:

Tanques: que tem como características de operação a ventilação do gás pelo topo e a decantação da água para posterior bombeio do óleo;

Vasos de Pressão: com uma entrada para os fluidos (internas às câmaras de separação) e saídas (linhas) para o líquido e para o gás natural. Nas linhas de saída estão instaladas as válvulas de controle e os respectivos medidores.

A operação normal dos vasos de pressão se dá pelo controle da pressão do gás e do nível de líquido, através dos controladores que atuam sobre as válvulas. Estas válvulas são de grande importância para a operação do processo como um todo, visto que em caso de falha do instrumento e/ou do operador, os problemas resultantes são de grandes proporções no sentido de perdas e/ou acidentes.

6.2.2.Poço de "GAS LIFT" Contínuo

6.2.2.1.Conceitos Básicos

O **"Gas Lift" Contínuo (GLC)** é a forma de elevação artificial que mais se aproxima do processo de surgência natural. Este método pode ser considerado até mesmo uma extensão do regime natural de fluxo.

O sistema de GLC é recomendado tanto para aumentar a produção de poços surgentes, como para promover a produção de poços naturalmente sem condições de surgência. O sistema é composto dos seguintes elementos:

- fonte de gás de alta pressão;
- "choke" de superfície;
- coluna de produção;
- válvula de "gas lift";
- mandris.

A **fonte de gás natural de alta pressão** é normalmente um compressor ou um poço de gás não associado (poço de gás livre). O tipo de compressor depende fundamentalmente do volume de gás e do espaço disponível, e o motor de acionamento do mesmo pode ser movido

a gás ou a energia elétrica. Quando existe poço de gás livre, ele só é usado temporariamente neste tipo de aplicação. É muito importante observar que esta fonte de gás natural de alta pressão é um componente considerado fundamental do sistema, visto que sem ele não há como se implementar o "gas lift" contínuo em um campo ou em um poço produtor.

O "choke" de superfície (ou "bean" de injeção de gás) consiste basicamente de uma válvula tipo agulha, que o controla o gás injetado no poço. Sua responsabilidade é manter o poço produzindo na vazão ótima de líquido.

Na **coluna de produção**, uma faixa de 200 a 20000 bbl/d de líquido pode ser produzido com tubulações de diâmetros internos de 2", 2 1/2", 3" e 3 1/2". Em geral o fluxo de produção pode ser classificado como via "tubing" ou via espaço anular do poço.

O coração de qualquer sistema de "gas lift" contínuo é a sua **válvula**. É ela que permite rigorosamente a passagem do gás do revestimento para a coluna de produção, e considerando as características técnicas de construção, elas podem ser de pressão, atuando basicamente como uma reguladora da pressão de fundo, ou de orifício, dependendo da função da mesma no fundo do poço. Por permitirem a comunicação entre o espaço anular e a coluna de produção do poço, estas válvulas são de extrema importância neste método de elevação tanto na fase de descarga, como na de produção do poço, sendo responsáveis portanto, por inúmeros problemas operacionais.

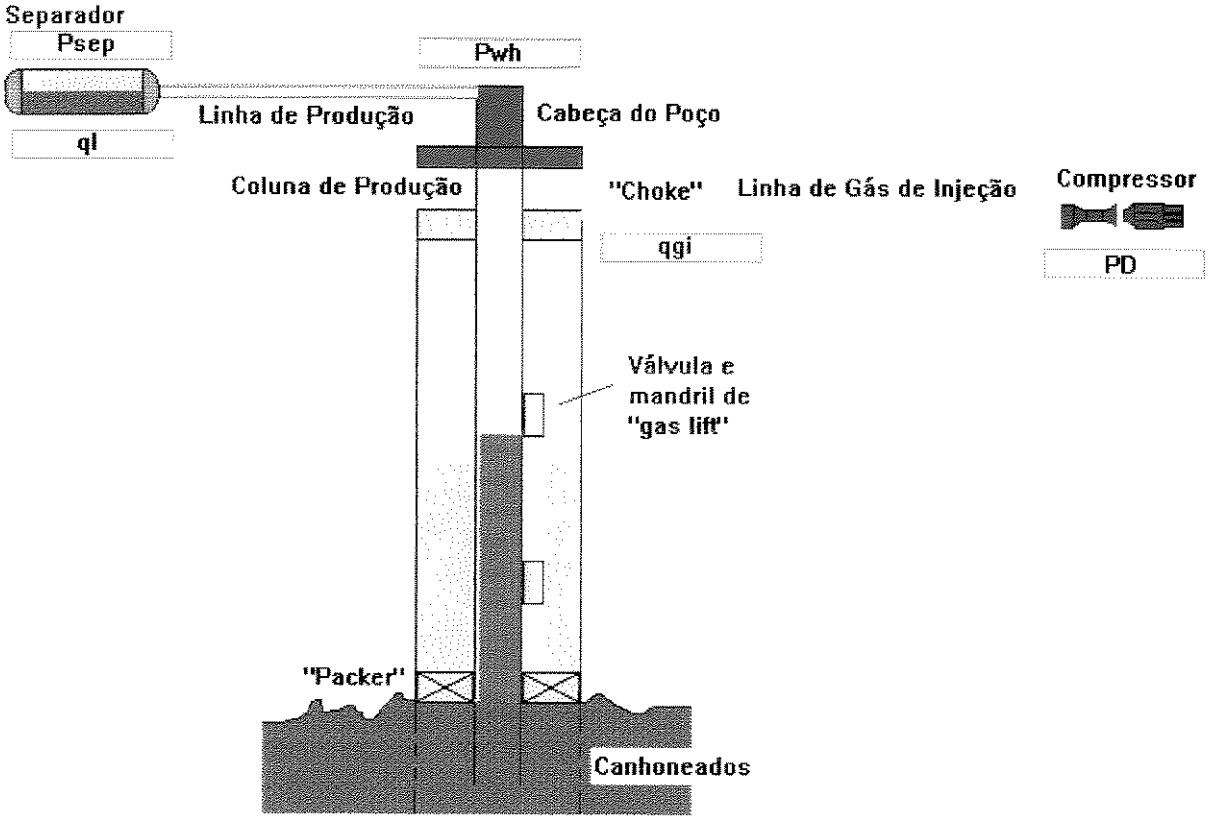


Figura 37 - O poço de "Gas Lift" Contínuo

As válvulas de pressão podem ser operadas pela pressão da coluna de produção ou pela pressão do espaço anular, sendo o último caso, o adotado nas operações dos poços das bacias nacionais. As válvulas de pressão normalmente usadas nas bacias locais são do tipo insertável, operada pela pressão do revestimento, o fole carregado com nitrogênio, sem mola e não balanceada. Seus componentes básicos são: o fole, a haste, a esfera, a sede e uma válvula anti-retorno.

Os **mandris** usados para alojar as válvulas de "gas lift" são de dois tipos: o convencional, que desce no poço fixo na coluna de produção com a válvula e o não convencional (tipo MM), que permite o assentamento da válvula através de operação com arame.

Nem sempre os mandris e as válvulas de descarga do GLC são os mesmos utilizados na operação. Algumas limitações inerentes a fonte de gás de alta pressão tais como: disposição física dos poços, disponibilidade de gás e pressão de injeção (principalmente no "gas lift" contínuo em campos marítimos), fazem com que os projetos de dimensionamento deste método considerem as duas fases (descarga e operação).

6.2.2.2. Descrição do Processo de Elevação por "GAS LIFT"

O método de elevação por "gas lift" contínuo consiste da injeção contínua de gás natural em um ponto da coluna de produção, a fim de que com o aumento da razão gás-líquido, se consiga uma redução no gradiente de pressão de fluxo promovendo uma maior produção de líquido do reservatório. Quanto mais profundo for injetado o gás, maior será a vazão de óleo produzida. Devido as limitações de disponibilidade da pressão do gás de injeção, são instaladas válvulas de "gas lift" na coluna de produção de forma a se conseguir a descarga do poço e permitir ao gás chegar ao ponto de injeção do projeto.

Na fase de colocação de um poço de GLC em produção, o gás é injetado a alta pressão na superfície de encontro ao fluido de completação (amortecimento) que se encontra no espaço anular e na coluna de produção do poço, com o objetivo de se reduzir o diferencial de pressão entre o reservatório e o poço. O mecanismo implica em que o gás empurre o fluido de amortecimento via os mandris de descarga cujas válvulas, abertas no início da operação, se fecham devido a redução da pressão do gás no espaço anular, ao tempo em que a próxima válvula situada mais abaixo é descoberta pela ação do gás.

Com a chegada do gás na válvula operadora (assentada no mandril mais profundo da coluna de produção), a injeção se dá estritamente neste ponto, pois as válvulas situadas acima da operadora deverão permanecer fechadas. As válvulas de "gas lift" têm proteção contra reversão de fluxo ("check valves") e são previamente calibradas com as devidas pressões de atuação.

6.2.3.Poço de Bombeio Mecânico - Sicad

6.2.3.1.Conceitos Básicos

O bombeio mecânico é o método de elevação artificial de petróleo que, no cenário mundial, tem o maior número de poços equipados em campos produtores. O sistema consiste dos seguintes componentes (Figura 38):

- unidade de bombeio
- motor
- redutor
- coluna de hastes
- coluna de produção
- coluna de fluido
- bomba de fundo

A unidade de bombeio é o componente que providencia potência para a coluna de hastes. Esta unidade converte o movimento de rotação do motor em movimento alternativo, e transmite esse movimento às hastes. Basicamente, é composta de uma base, de uma estrutura com uma viga oscilante e de um redutor. A viga é conectada à manivela por meio de dois braços e a unidade recebe contrapesos na viga ou na manivela para contrabalançar as cargas do poço.

Os motores utilizados para funcionamento da unidade de bombeio podem ser de acionamento elétrico ou de acionamento a explosão. Os motores a explosão são geralmente empregados em campos onde não existe energia elétrica.

O redutor se caracteriza por um conjunto de engrenagens que tem como função reduzir a velocidade de rotação do motor para um valor conveniente da ordem de 8 - 30 RPM.

A coluna de hastes é responsável pela transmissão da potência fornecida na superfície para a bomba de fundo. Essa potência é aplicada em uma extremidade da coluna de hastes e usada na outra, onde é fixado o pistão da bomba. A durabilidade da coluna de hastes é de suma importância para o sistema, sendo portanto seu dimensionamento um dos pontos críticos do projeto.

A tubulação de produção é o conjunto de tubos dentro do qual opera a coluna de hastes e fluem os fluidos desde a bomba até a superfície. As principais características de especificação desta coluna são: os diâmetros interno e externo, o peso e a quantidade total de tubos.

Os fluidos produzidos exercem um papel muito importante neste método de elevação, uma vez que os mesmos estão diretamente relacionados com a eficiência do bombeio. As propriedades

mais importantes dos fluidos produzidos são: viscosidade, temperatura, densidade, produção de água (BSW) e de materiais abrasivos e suas corrosividades.

A bomba de fundo usada no bombeio mecânico é do tipo alternativa de simples efeito, que bombeia em um único sentido do curso, e é constituída principalmente de: camisa, pistão e válvulas. A camisa é um tubo com revestimento interno de material endurecido, o pistão é do tipo inteiriço e as válvulas são a de pé, fixa no tubo de produção e a de passeio, fixa no pistão.

Estas bombas de subsuperfície podem ser ainda do tipo tubular ou insertável. As bombas tubulares são assentadas diretamente na coluna de tubos, e só podem ser sacadas dos poços com a retirada total das colunas de produção. Quanto as bombas insertáveis, são assentadas na coluna de hastes e podem ser sacadas dos poços apenas com manobras das colunas de hastes.

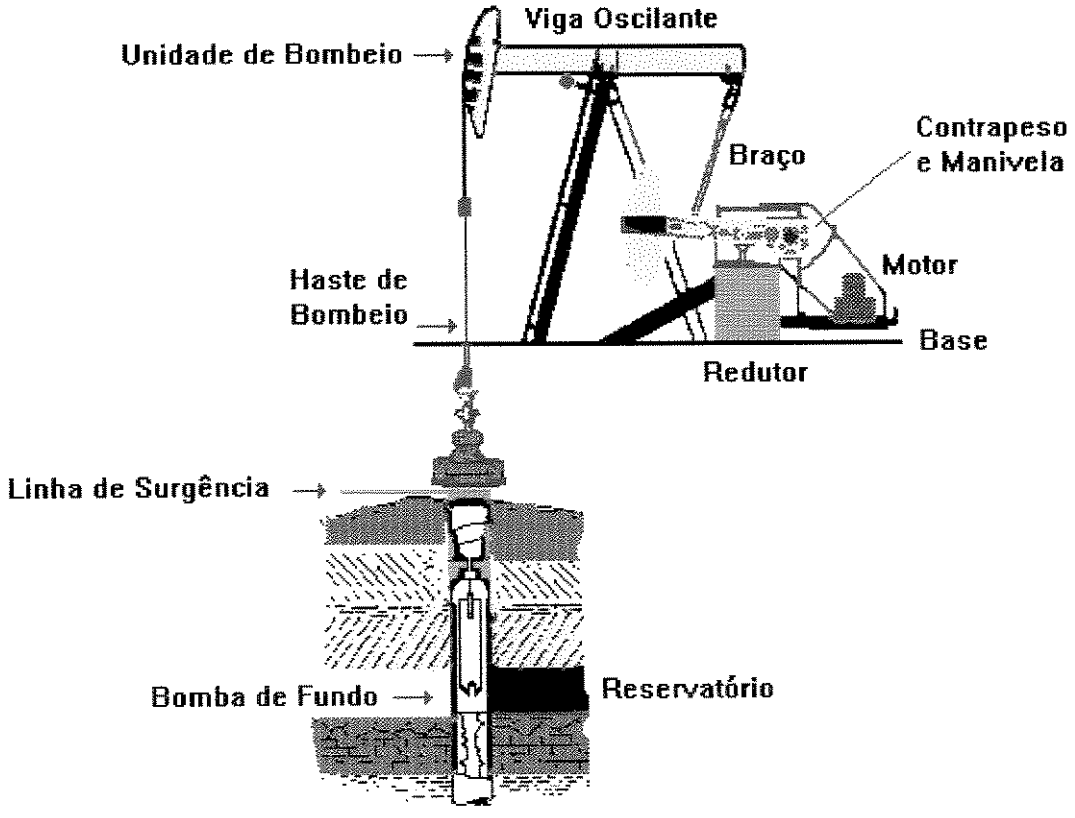


Figura 38 - O Poço de Bombeio Mecânico

6.3. Implementação

6.3.1.Planta de Separação

6.3.1.1.Variáveis do Processo de Separação

As condições operacionais dos vasos que compreendem a **Planta de Processo** são estabelecidas basicamente por:

- pressão de separação
- temperatura
- nível dos vasos

A **pressão de separação** é ajustada com base na pressão ótima de separação. Quando esta pressão sofre uma redução, o volume de gás liberado pelo óleo é acrescido. Portanto, quanto menor a pressão de separação, maior será o volume de gás ocupando o mesmo espaço nos vasos.

A **temperatura de separação** não é um parâmetro normalmente controlável nos vasos. Um aumento na temperatura tem o mesmo efeito da redução da pressão, isto é, aumenta o volume de gás.

Quanto ao **nível dos vasos**, é possível manter a velocidade dos fluídos, o nível do líquido e a circulação do gás estáveis, obtendo a máxima eficiência de operação. Para isto basta que os vasos recebam fluxos regulares. Um mesmo vaso pode ser ajustado para as condições diferentes de operação tais como máxima capacidade ao óleo com redução da capacidade ao gás (solução para campos com baixas razão gás/óleo), máxima capacidade ao óleo e ao gás (a operação se dá nas máximas condições permitidas pelo vaso) e máxima capacidade ao gás (solução para campos com altas razão gás/óleo).

6.3.1.2.Problemas Operacionais

Todos os vasos, equipamentos com mecanismos alternativos (por exemplo: bombas e compressores) e instrumentos que fazem parte deste processo, estão sujeitos a problemas operacionais devido ao mau funcionamento ou erros de operação. Alguns destes problemas foram modelados e fazem parte do controle do sistema.

A seguir, enumeram-se alguns dos problemas que ocorrem nos diferentes vasos separadores:

Separadores de Produção e Teste: arraste de gás, obstrução por parafina, produção de areia, formação de emulsão, formação de espuma e arraste de líquido.

Tratadores: arraste de óleo, vaporização e redução do tempo de retenção

Depurador: arraste de gás e arraste de líquido

Tanques: derramamento de líquido e sem transferir óleo

6.3.1.3. Procedimentos de Operação

Este tipo de processo funciona alternando duas modalidades operativas que são os estados de operação permanente e transiente. No estado permanente todos os vasos operam normalmente e nenhum dos problemas operacionais identificados anteriormente ocorre, caracterizando um estado de operação no equilíbrio. Neste estado os níveis e pressões dos vasos têm comportamento estáveis, segundo os valores ajustados nos "set points".

O que caracteriza o estado transiente de operação do processo são os tempos de abertura e de fechamento das válvulas de nível ou de pressão destes vasos. Pode-se observar que se o volume de óleo (ou de gás) na entrada da planta é pequeno (ou grande), as vazões de saída dos vasos podem ser reduzidas (ou aumentadas), a fim de que se consiga a estabilização dos níveis (ou das pressões). Isto protege os vasos de forma a não trabalharem vazios (ou com sobrecargas).

A observância dos tempos de abertura e fechamento das válvulas de controle da saída dos vasos, por razões de segurança, primeiro para as válvulas de pressão e segundo para as válvulas de nível, garante que as vazões de saída dos vasos não mudem bruscamente e que não haja modificação na abertura (ou fechamento) das válvulas enquanto operando no estado de transiência.

6.3.1.4. Fluxograma de Funcionamento do Processo de Separação

Uma visão global do funcionamento da planta de processamento primário pode ser visto na Figura 39.

O funcionamento da planta, representada no fluxograma acima, inicia-se com a leitura dos dados, a definição da pressão e do nível de "set point", e a verificação das variáveis de produção dos poços (os três primeiros blocos no topo da Figura 39).

Como as variáveis pressão e nível de operação são as entradas do controle, a seguir o algoritmo analisa os sinais de cada equipamento que compõe a planta de processo. Portanto, os valores de pressão e de nível destes vasos são analisados pelo controle (bloco central da Figura 39), que considera basicamente duas situações:

- **"erro zero" e "erro diferente de zero"**, para os separadores, tratadores, depurador e tanques,

- **"operando" e "não operando"**, para as bombas de transferência, o compressor de gás natural, o oleoduto, o gasoduto e os poços que utilizam gás para elevação.

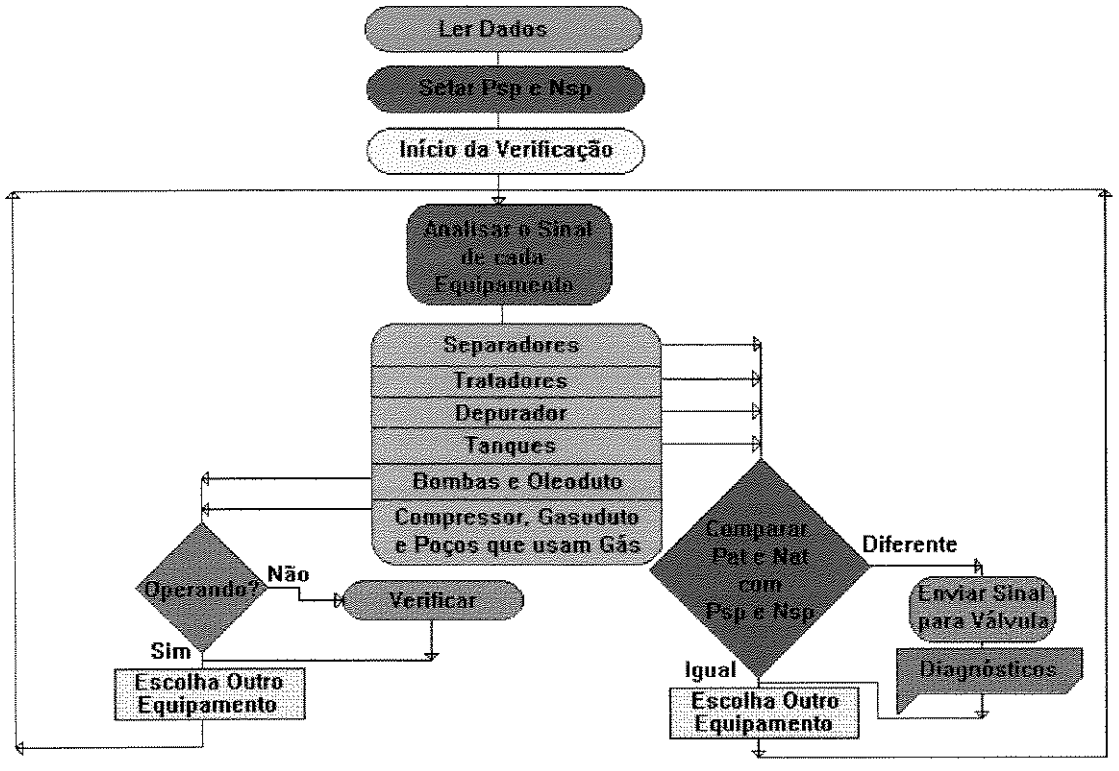


Figura 39 - Fluxograma de Funcionamento do Processo

A primeira situação está representada na parte lateral direita do controle pelas comparações das pressões atual (Pat) e de "set point" (Psp), e dos níveis atual (Nat) e de "set point" (Nsp). A expressão **Igual** permite a escolha de outro vaso para supervisão, enquanto que a expressão **Diferente** libera um sinal de abertura ou fechamento para a válvula de controle do vaso em verificação e a respectiva mensagem de diagnóstico.

A segunda situação está representada na parte lateral esquerda do controle pelas verificações se estão ou não operando. A expressão **Sim** permite ao sistema escolher outro equipamento para supervisionar, enquanto que a expressão **Não** libera um sinal de verificação para orientação do operador, uma vez que nesse ramo do diagrama não há atuação em válvulas.

As mensagens de diagnósticos que o controle emite para identificar problemas operacionais nos vasos são as seguintes:

Separadores: Arr_gás, Parafina, Areia, Emulsão, Espuma e Arr_óleo;

Tratadores: Arr_liq, Vaporização e Red_ret;

Depurador: Arr_gás e Arr_óleo;

Tanques: Nivelliq _muitoalto e Derr_líq.

6.3.1.5. Controle Inteligente

Os propósitos deste controle são escoar óleo e gás dos vasos mudando suavemente as aberturas das válvulas e manter estáveis o nível e a pressão de trabalho de cada vaso. O controle desenvolvido para a planta de processo apresenta a seguinte estrutura de coordenação, de acordo com o fluxograma da Figura 39:

1. o agente denominado "Inicializa" lê os dados de entrada e introduz estes valores no controle;

2. a rede de "controle" dos vasos e dos equipamentos, (que tem o nome do próprio elemento do processo), detecta um erro na pressão e/ou no nível no elemento da planta;

3. na rede anterior, a pressão atual de operação (Pat) é comparada com a pressão de ajuste (Psp) e o nível atual é comparado com o nível de ajuste (Nsp), sendo os valores de ajustes ("set point") armazenados no controle;

4. se for detectado erro, a própria rede de controle aciona a válvula controladora de pressão ou a de nível, para modificar a abertura das mesmas, no sentido de corrigir o problema;

5. quando a válvula controladora de saída do vaso é acionada, a rede de "Problemas" da planta, denominada "Prob_Vaso", gera um diagnóstico (se for o caso);

6. uma vez gerado o diagnóstico, uma das redes de "Ações" recomenda uma ação a ser tomada (se for o caso) em função do diagnóstico emitido.

Além do agente de inicialização de variáveis, o controle tem como base de atuação três tipos de estruturas de redes para a supervisão do processo:

- uma rede de controle de operação,
- uma rede de diagnósticos de problemas operacionais e
- uma rede de ações recomendadas em função dos problemas detectados.

Estas redes atuam nos vasos de separação e equipamentos de bombeio e compressão.

Ao ser ativado, o agente de inicialização de variáveis busca dados através de redes de processamento da aplicação que utilizam agentes do sensor para adquirir dados dos vasos e equipamentos.

Para compensar os volumes de óleo e de gás no tempo, na saída dos vasos, a rede de controle de operação dos vasos mostrada na Figura 40 (no caso os vasos separadores) compara pressão (Pat) e nível (Nat) atuais do vaso com os respectivos valores (Psp e Nsp) de ajustes ("set point") e, em função do valor do erro, executa um controle de abertura e/ou fechamento das válvulas de controle correspondentes, até retornar estes parâmetros às suas condições de equilíbrio.

Se os erros da pressão e do nível forem nulos a situação indica normalidade, então o controle não modifica a abertura das válvulas, sendo daí chamado o agente *supervisor* para que o próximo vaso seja analisado. Se um dos erros for diferente de zero, após a correção nas aberturas das válvulas, o agente acionado vai ser o de problemas (PROB), que ativa imediatamente a outra rede (Figura 41) para diagnosticar os problemas operacionais do vaso.

As variáveis de entrada do controle são: a pressão e o nível para os separadores, para os tratadores de óleo e para o depurador de gás, pressão e volume para o distribuidor de gás, bombas de transferência e compressor de gás natural e só nível para os tanques de armazenagem e transferência. As variáveis de saída são as compensações para as válvulas que controlam o nível e pressão dos respectivos vasos.

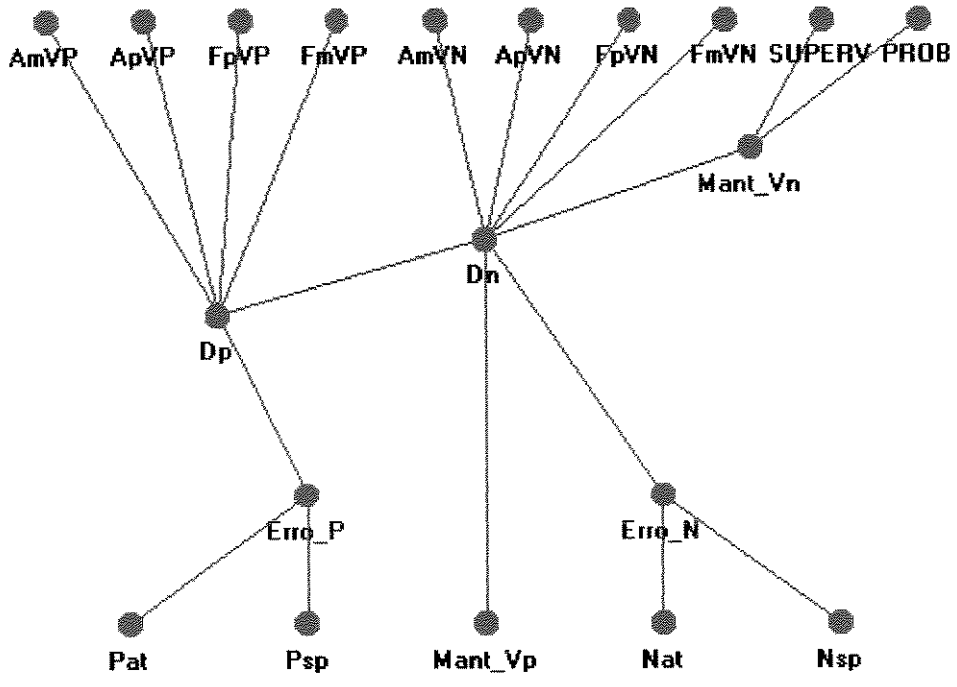


Figura 40 - Rede de Funcionamento do Separador

Todas as funções de pertinência são do tipo: Abre Muito, Abre Pouco, Fecha Muito e Fecha Pouco, para as válvulas de pressão e nível respectivamente. Estas funções de pertinência são simbolizadas da forma: AmVp, ApVp, FmVp e FpVp, para a válvula de controle da pressão e, AmVn, ApVn, FmVN e FpVn, para a válvula de controle do nível.

Com vistas a diagnosticar os problemas operacionais que ocorrem nos vasos de separação de óleo e gás e nos equipamentos de bombeio e compressão da planta de processo, é utilizada a segunda estrutura de rede construída para o controle (Figura 41).

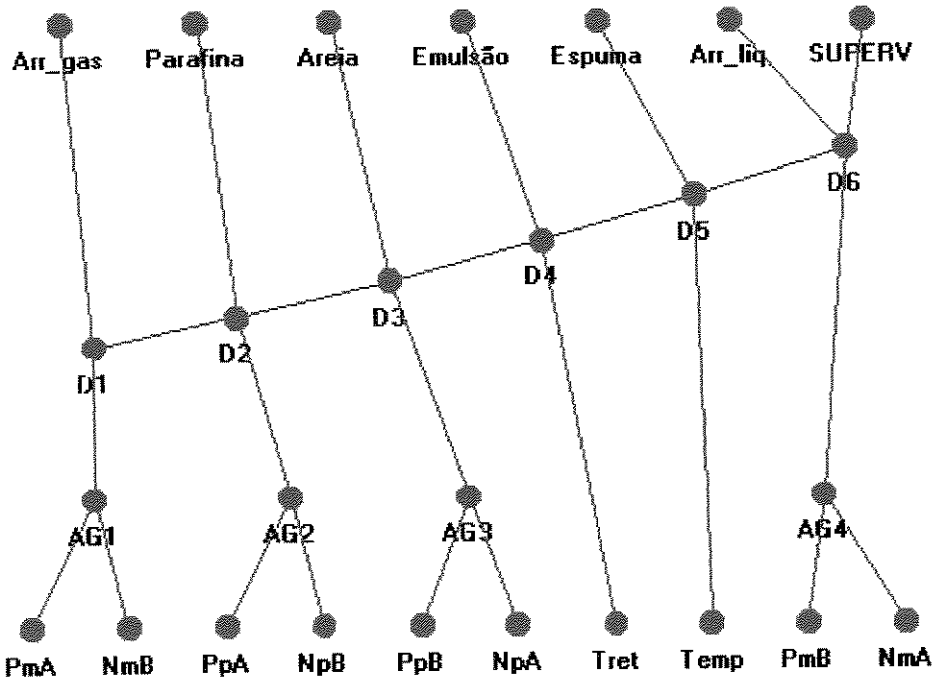


Figura 41 - Rede de Problemas dos Separadores

Na supervisão, quando há atuação da rede de controle, a rede de problemas é acionada pelo agente "PROB" da rede de controle, indicando alteração na abertura da válvula de controle da pressão e/ou de nível, e investiga se houve alguma anormalidade que possa levar a um diagnóstico. Como já descrito anteriormente, se os valores das variáveis de pressão e de nível investigadas não indicarem nenhuma anomalia mais grave, o último agente das redes de diagnósticos devolve os parâmetros normais ao agente intitulado de "supervisor".

Um dos pontos de suma importância deste controle é a confirmação dos diagnósticos, levantados no campo por especialistas, e armazenados nas redes de detecção de problemas operacionais.

Em caso de ocorrência de dois ou mais diagnósticos simultaneamente, o controle detectará, mostrará e recomendará as ações sobre os mesmos seqüencialmente, só liberando o retorno do sistema à operação após as resoluções dos problemas assinalados. Faz-se importante ressaltar que para algumas das ações recomendadas pela rede, mais de um diagnóstico, em caso de existência, são resolvidos com a adoção de apenas uma ação.

Vale ainda acrescentar que estes problemas causam mau funcionamento nos vasos e nos equipamentos, provocando em conseqüência quase sempre parada da planta de processo.

Observe-se também que se nenhum dos diagnósticos for confirmado, a rede ativa o agente *supervisor* que encerra a navegação da mesma, continuando o sistema a efetuar sua supervisão programada.

O terceiro tipo de estrutura de rede do controle é o que recomenda as ações para o operador, em função dos diagnósticos de problemas detectados (Figura 42).

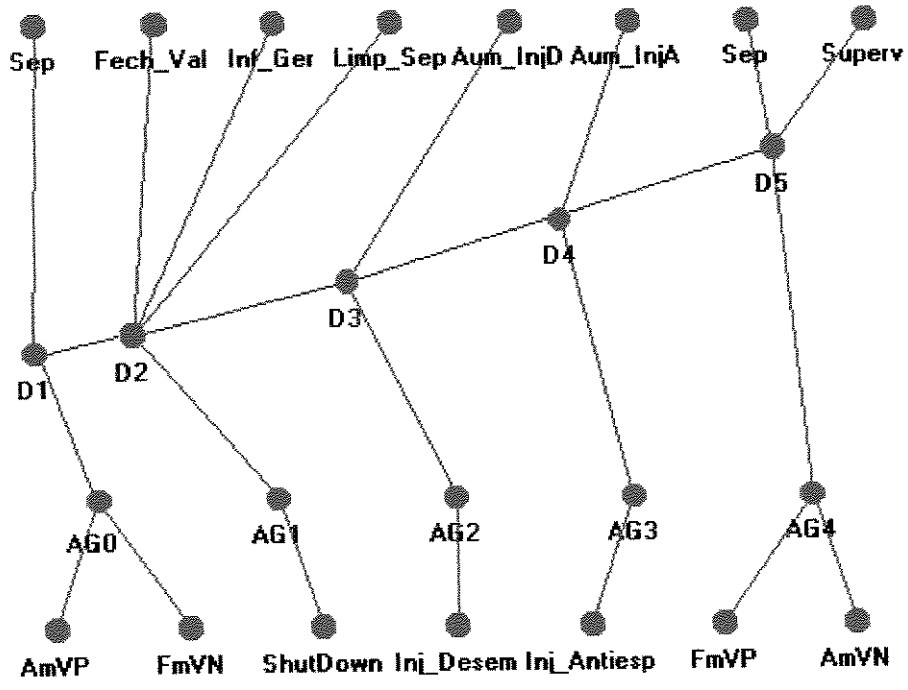


Figura 42 - Rede de Ações do Controle

Esta rede é acionada após um problema ser detectado pela rede de diagnósticos. O objetivo da mesma é o de somente auxiliar o operador nesta tarefa, considerando que tanto o controle de funcionamento como o diagnóstico de problemas operacionais são feitos pelas outras redes. A forma de navegação desta rede é a mesma mostrada para as redes anteriormente descritas, porém o que se destaca na mesma é a introdução de atuações sobre problemas, até então só tratados manualmente pelo operador.

6.3.2.Poço de "GAS LIFT"Contínuo

6.3.2.1.Operação de um Poço de "Gas Lift" Contínuo

Para se avaliar corretamente a eficiência operacional de um poço que produz por "gas lift" contínuo, é fundamental que se analise a instalação como um todo.

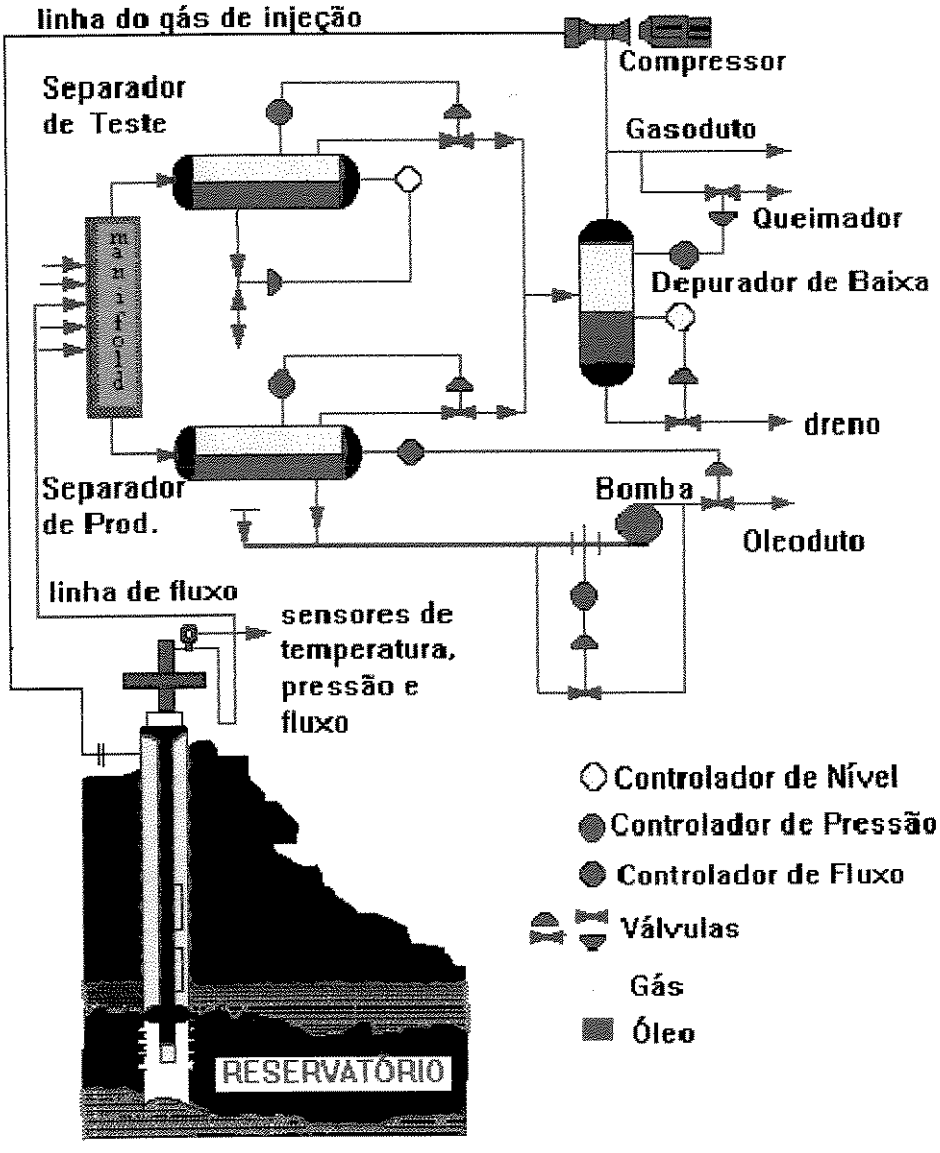


Figura 43 - Operação do "Gas Lift" Contínuo e seu Processo

Muitas vezes, se o sistema é propriamente conhecido, se consegue ganhos de produção só com o manuseio da razão gás/líquido; porém, a injeção excessiva de gás pode aumentar o gradiente de pressão de fluxo provocando uma redução na produção.

Existem várias formas de se analisar corretamente a instalação de um poço de "gas lift" contínuo:

- *acompanhamento de superfície*:
  registro da pressão do tubo e da pressão do revestimento;
  medição do gás injetado;
  medição da temperatura;
  testes de produção;
  automação inteligente da instalação.

- *acompanhamento de subsuperfície*:
  registro das pressões de fundo;
  registro da temperatura de fundo;
  determinação do nível de fluidos;
  programa computacional de cálculos.

A operação eficiente de um poço de "gas lift" contínuo requer que, na fase de projeto, um correto dimensionamento da coluna de produção, dos mandris e válvulas de "gas lift", do volume e da pressão do gás na descarga do compressor, e das linhas de injeção de gás e surgência do poço. Na fase de operação, é preciso dispor da medição dos volumes de fluidos no tempo, da razão gás-líquido de injeção e de produção, da produção de água (BSW ou FW), e das pressões no espaço anular e na cabeça do poço.

Para otimizar a vazão de injeção de gás de um poço, é fundamental o conhecimento da curva de produção versus vazão de gás injetado. Assumir um valor constante para a pressão na cabeça do poço e aumentar a vazão de injeção de gás para elevar a produção pode gerar graves distorções nos resultados de operação. O uso da pressão de injeção do gás para determinar o assentamento da válvula operadora na maior profundidade possível reduz a vazão de injeção de gás e a potência de compressão requerida.

A produção de um poço equipado com "gas lift" contínuo pode ser acrescida de forma significativa analisando-se as condições de operação e executando-se as correções necessárias. A análise operacional deve realizar o acompanhamento das variações da pressão do revestimento em função de qualquer alteração na pressão da cabeça do poço.

Variações na pressão da cabeça do poço ocasionam alterações também na vazão de produção requerendo um aumento ou redução no volume de gás injetado, fato que sugere por onde o poço pode ser controlado.

6.3.2.2. O Programa OGLC

A metodologia de otimização do poço de "gas lift" contínuo compreende um programa de cálculo de variáveis do poço, e um controle inteligente. O programa de cálculo foi

desenvolvido fora do sistema Kards, e é acionado por agentes do sistema de controle implementado no Kards que permitem a execução de programas externos.

O desenvolvimento do programa de cálculo exigiu a escolha das correlações para as propriedades de fluidos, determinação dos gradientes de pressão na linha de produção e na tubulação de produção.

A correlação de Beggs-Brill foi adotada para o trecho horizontal, e as correlações de Hagedorn-Brown e Duns-Ros, para o trecho vertical. Para as propriedades de fluidos, as correlações adotadas foram:

- *Bo* e *Rs*: Correlação de Vasquez;
- *z*: Método de Dranchuck e Abou-Kassem;
- *Viscosidade do Gás*: Correlação de Lee et al.;
- *Viscosidade do Óleo Morto*: Correlação de Beggs-Robinson;
- *Correção para o Gás em Solução*: Beggs-Robinson;
- *Viscosidade da Água*: Correlação de Van Wingen.

Além das correlações e métodos mencionados acima, para o cálculo da vazão de equilíbrio foi utilizada a equação de IPR de Vogel.

Assim sendo, o programa **OGLC** (Otimizador de "Gas Lift" Contínuo) utilizou as correlações acima mencionadas. Caso a utilização do programa em outros poços, com estas correlações, não indique bons resultados, o próprio **OGLC** pode ser usado para a escolha de outras correlações mais adequadas aos novos casos.

O programa **OGLC** foi desenvolvido e implementado com base no seguinte algoritmo de cálculo, que junto com o controle mantém o poço produzindo sempre com a vazão ótima de líquido (ql ótima):

1. quando há variação nas condições de reservatório, de escoamento ou de separação, varia também a pressão na cabeça do poço (Pwh);

2. esta nova Pwh implica na determinação de uma nova ql de equilíbrio;

3. esta ql de equilíbrio permite o cálculo de uma nova vazão de injeção de gás (qgi). Uma vez consideradas adequadas ql e qgi, ajusta-se a abertura da válvula de controle de injeção de gás e continua a operação do poço;

4. se as novas ql e qgi forem consideradas inadequadas, calcula-se uma nova qgi ótima, determina-se a nova ql de equilíbrio (ql ótima) e ajusta-se a nova abertura da válvula de controle do gás de injeção;

5. estes novos valores das variáveis devem permitir um novo ajuste do valor de Pwh e a continuidade operacional do poço.

6.3.2.3. O Controle Inteligente

O controle inteligente tem como objetivo manter o poço nas condições ótimas de projeto (em termos de vazão de gás injetado e vazão de líquido produzido), observando-se as evidências, os diagnósticos e as ações para os possíveis problemas; e os transientes envolvidos com a operação do poço.

A descrição do procedimento para os passos do controle desenvolvido para o poço é a seguinte:

1. a rede Cont_Pwh detecta um erro na pressão da cabeça do poço Pwh (Figura 44);

2. o programa OGLC é acionado, através do agente Calc_Poço, e fornece uma vazão de líquido ql (Figura 45);

3. a vazão ql é comparada com a ql de teste;

4. se houver erro, a rede Calc_Poço aciona a válvula de controle do gás injetado, para modificação da abertura da mesma;

5. quando a válvula controladora de injeção de gás é acionada, uma rede de "Problemas" do poço gera um diagnóstico, se for o caso (Figura 46e Figura 47);

6. uma vez gerado o diagnóstico, uma das redes de "Ações" recomenda uma ação a ser tomada, se for o caso, em função do diagnóstico emitido.

A primeira rede do controle Cont_Pwh (controle da pressão da cabeça do poço), que introduz a base de conhecimento deste subsistema, é a responsável pela supervisão do mesmo (Figura 44).

Ao ser ativada, através do agente *le_valores*, a pressão lida na cabeça do poço (Pwh) é comparada com a pressão de "set point" pré-definida para o mesmo. Se o erro for nulo, a decisão a ser tomada pelo sistema é a de encerramento da supervisão, via agente termina. Se o erro for diferente de zero, a navegação da rede se dá pelo agente *OGLC*, que acionará a segunda rede de supervisão do controle, que é a rede Calc_poco (Figura 45).

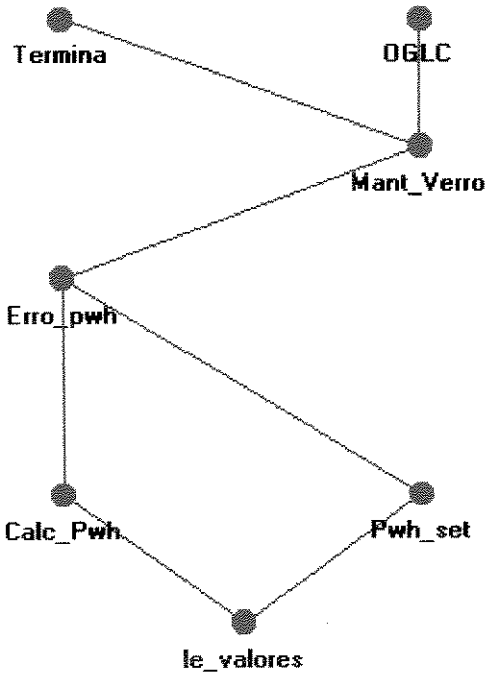


Figura 44 - Rede de Supervisão da Pressão da Cabeça do Poço (Cont_Pwh)

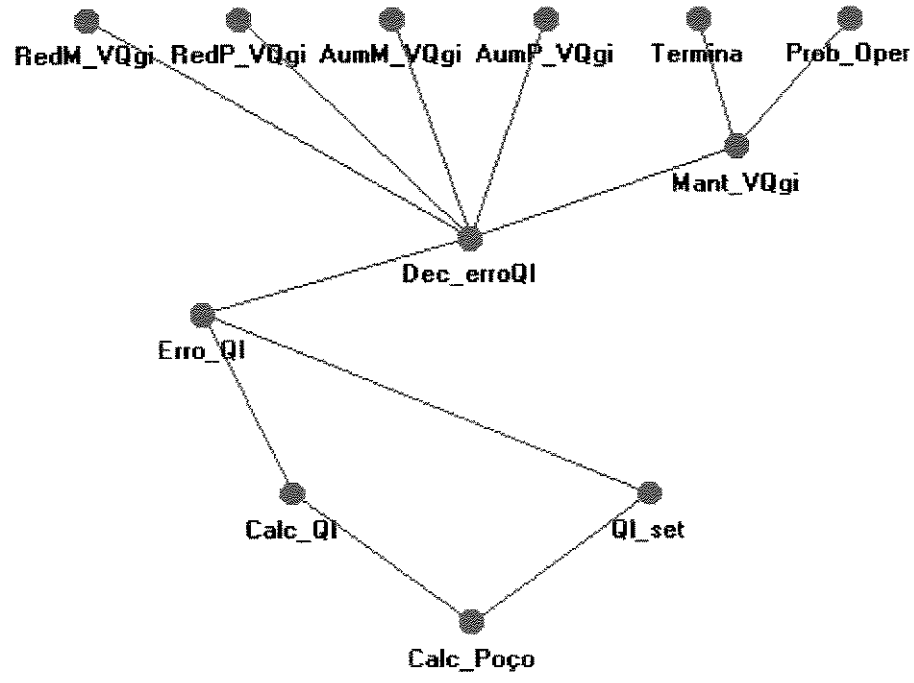


Figura 45 - Rede de Supervisão da Vazão de Líquido do Poço (Calc_Poço)

O programa **OGLC** é então acionado pelo agente *Calc_Poço*, que acessa as bases de dados do Kards e armazena os dados a serem transferidos para o **OGLC** em um arquivo tipo texto, executado o programa **OGLC**, e lê de um arquivo tipo texto o resultado do **OGLC**, que devolve uma vazão de líquido (ql). Esta vazão também é comparada com uma vazão de "set point", resultante do teste de produção. Novamente se o erro for nulo, o valor da abertura da

válvula de controle de injeção de gás é mantido e termina a rodada de supervisão. Se o erro for diferente de zero e dependendo do valor deste erro, a válvula de controle de injeção de gás terá sua abertura modificada segundo funções de pertinência do tipo: Reduz muito, Reduz pouco, Aumenta pouco e Aumenta muito o volume de gás injetado.

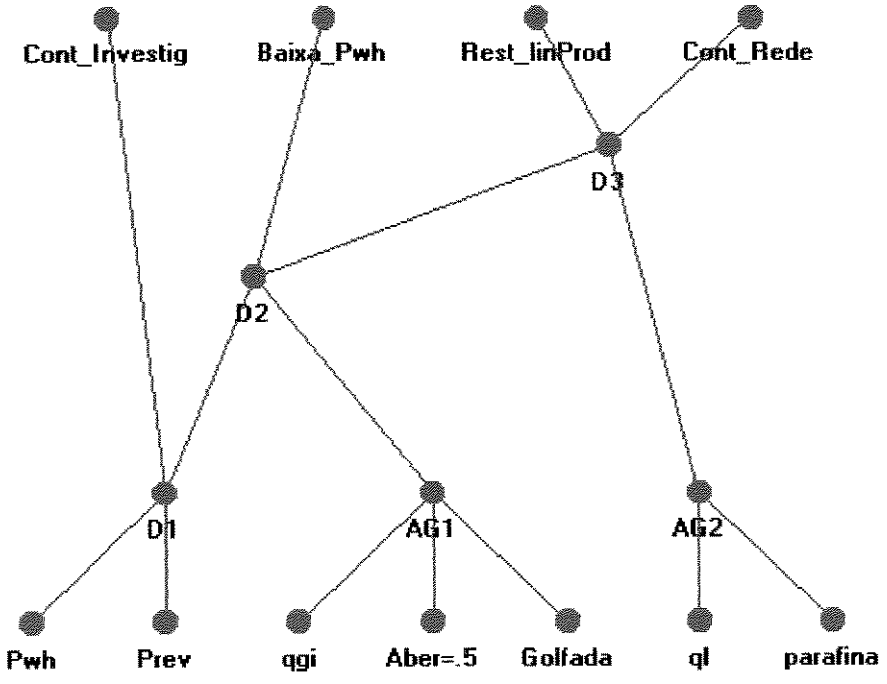


Figura 46 - Rede de Diagnósticos do Poço (Problemas)

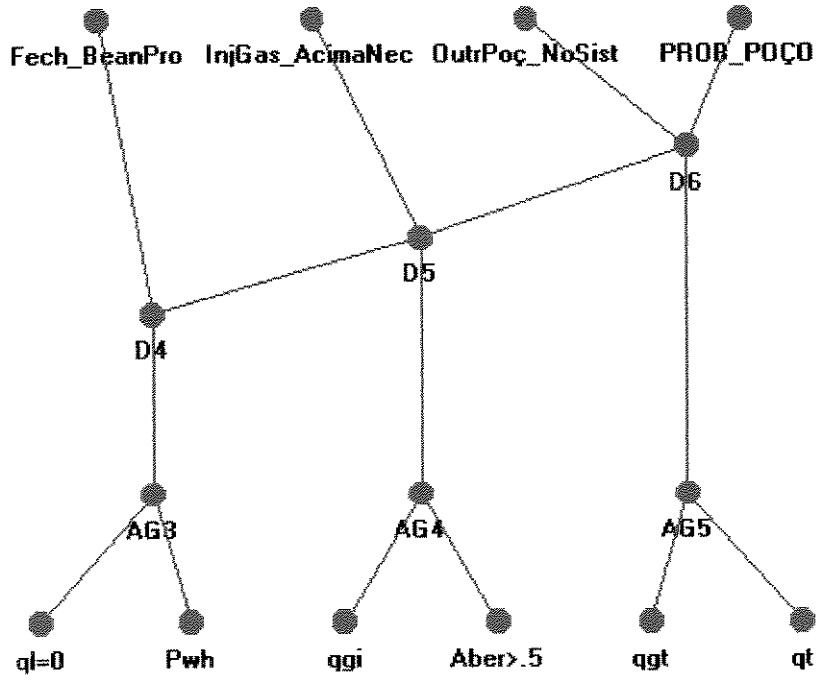


Figura 47 - Continuação da Rede de Diagnósticos do Poço

Ao ser denotada uma modificação na abertura desta válvula controladora, o controle navega seqüencialmente as redes de diagnósticos que são acionadas basicamente em função do comportamento das pressões do tubo e do revestimento, conforme mostrado na Figura 46 e na Figura 47.

Seguindo a mesma linha de navegação, ao ser identificado um diagnóstico em uma das redes, em função da falha detectada, o controle ativa uma das redes de ações (Figura 48), para correção do diagnóstico apontado na fase anterior.

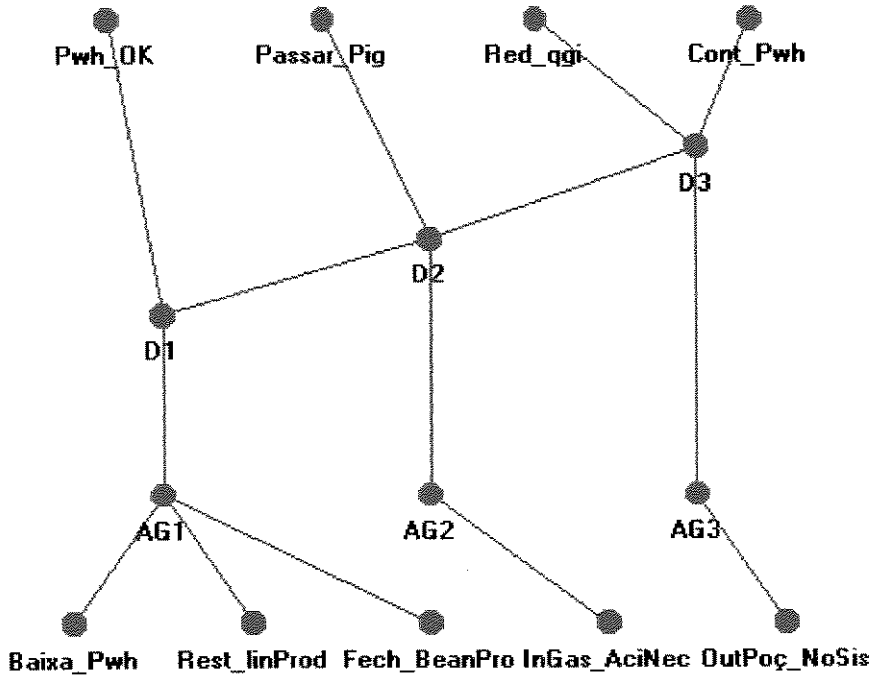


Figura 48 - Rede de Ações do Poço

6.3.3.SICAD

O SICAD é um sistema inteligente para análise de cartas dinamométricas, que tem como principal objetivo a automação e o controle de poços de petróleo que produzem por bombeio mecânico. Foi desenvolvido para a PETROBRAS através de convênios específicos e utiliza o Kards como o ambiente inteligente de supervisão e gerenciamento.

Em termos conceituais este sistema admite três possíveis arquiteturas básicas de operação do bombeio mecânico que são: o controle local, o escritório central e a unidade portátil tipo microcomputador (Figura 49).

Para execução das tarefas de reconhecimento de padrões e emprego do raciocínio especializado, o SICAD usa duas famílias de redes neurais distintas. Após a aquisição da carta dinamométrica de superfície e o cálculo da respectiva carta dinamométrica de fundo, as redes

de reconhecimento de padrões reconhecem o formato da carta dinamométrica de fundo e ativam, se necessário, as redes do raciocínio especializado para finalizar a análise.

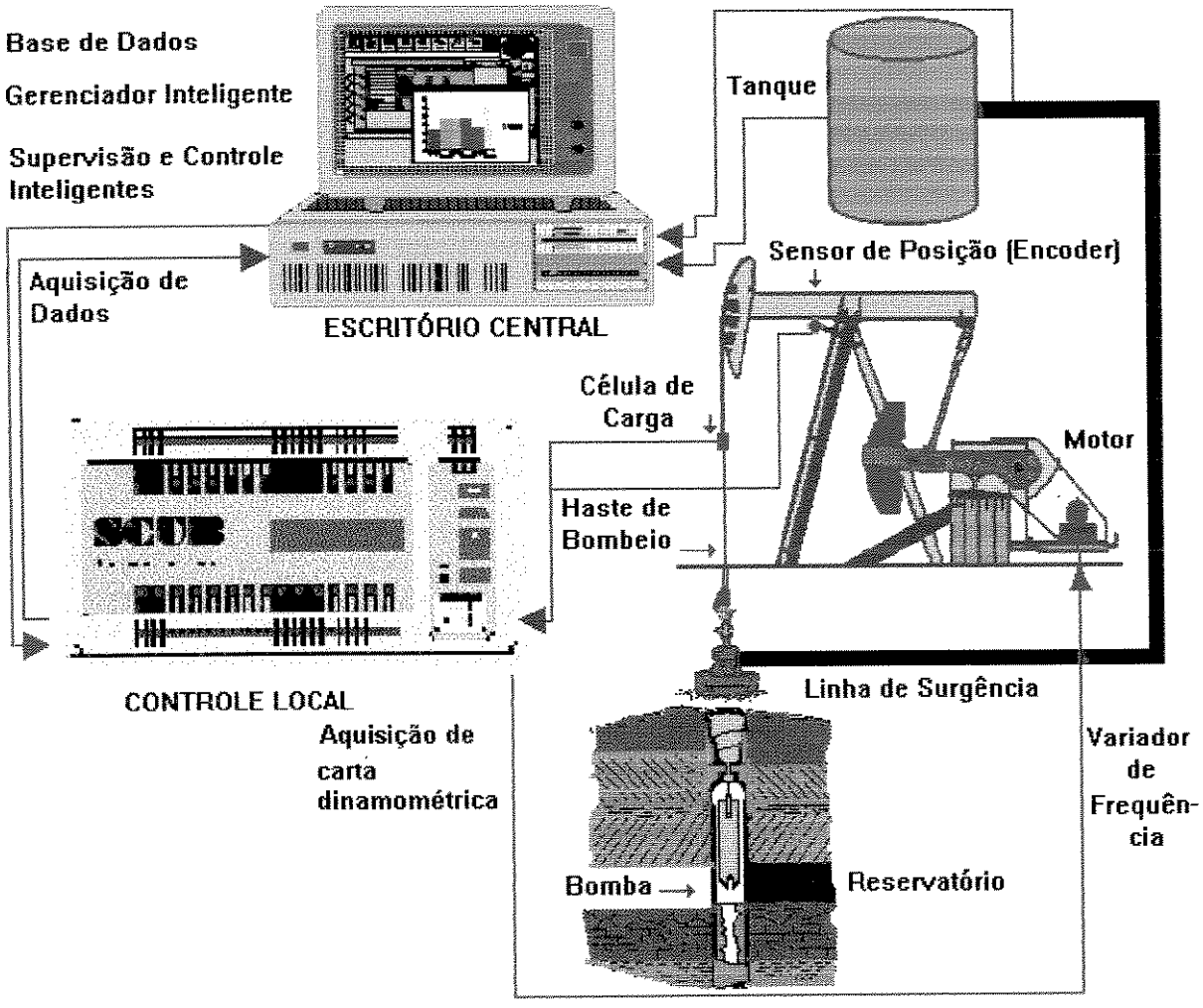


Figura 49 - Esquema do Sistema SICAD

As redes de reconhecimento de padrões são construídas pelo especialista de acordo com seu conhecimento sobre os diversos tipos de cartas de interesse. As redes de raciocínio seguem o raciocínio do especialista em bombeio mecânico, usando a classificação da carta dinamométrica provida pelo reconhecimento de padrões e os parâmetros de operação dos poços providos pela base de dados, para diagnosticar e atuar nas condições de bombeio. Identificado o diagnóstico, o sistema pode agir através do controle local, ligando e desligando o motor da unidade de bombeio, ou corrigir localmente alguns problemas usando um algoritmo que atua num equipamento variador de frequência para modificar as condições de bombeio, aumentando ou reduzindo a velocidade de bombeamento (CPM).

6.3.3.1. O Controle Local

O controle local do sistema consiste de um controlador inteligente da unidade de bombeio denominado **SCUB** (Sistema de Controle de Unidade de Bombeio) que uma vez ligado a uma

célula de carga e a um sensor de posição ("encoder"), executa a aquisição contínua da carta dinamométrica de superfície, através de medições das cargas na coluna de hastes na respectiva posição do ciclo de bombeio (Figura 49). Neste equipamento estão instalados o algoritmo de cálculo da carta de fundo, a base de dados contendo os dados do sistema de bombeio mecânico instalado e um grupo de padrões de cartas de fundo.

As redes instaladas reconhecem o padrão da carta dinamométrica de fundo, checam possíveis violações nos equipamentos de superfície e subsuperfície do poço e avaliam as condições do bombeio. Através do quadro de comando da própria unidade de bombeio ou através do variador de frequência é possível parar o poço, em caso de violação de cargas; aumentar a frequência de bombeio, em caso de carta que indique que a produção pode ser acrescida (formato cheio); ou reduzir a frequência de bombeio, em caso de carta que indique que a produção deve ser decrescida, acusando formatos de "pump-off" ou "pump-down".

6.3.3.2. O Escritório Central

O escritório central é uma aplicação Kards instalada em um micro na sala de operação para executar as tarefas executadas rotineiramente pelos técnicos responsáveis pela análise do bombeio mecânico (Figura 49). Este modo de operação analisa o sistema de bombeio mecânico de forma bem mais completa que o controle local. A aquisição da carta "on-line" com o funcionamento do poço, as bases de dados dos poços sempre atualizadas, e a disponibilidade de um maior número de padrões de cartas permitem um melhor manuseio dos dados no escritório central, implicando em diagnósticos bem mais amplos [Cor95].

O enlace via rádio e/ou cabo entre o controle local e o escritório central, possibilita o envio de novos dados a partir do escritório central. Porém o sistema também pode ser utilizado sem o controle local, pois a aplicação do escritório central possui também os passos realizados pelo controle local, com exceção da aquisição de cartas "on-line", que é substituída pela aquisição de uma carta dinamométrica de superfície através de "scanner" ou mesa digitalizadora (utilizando agentes do sensor). Isto permite que o sistema SICAD também seja utilizado em poços onde não se justifica a instalação do controlador local.

O escritório central, quando utilizado junto com o controle local, é utilizado também para transferir para o controle local as redes com os padrões a serem verificados bem como os programas utilizados para o cálculo e análise da carta de fundo, de forma que a aplicação do Kards (escritório central) possui total controle sobre o controle local.

No escritório central estão armazenados todos os padrões de cartas utilizados (Figura 51). Para o reconhecimento do padrão, a carta de fundo (Figura 50) adquirida diretamente do controle local ou calculada pelo escritório central a partir de uma carta de superfície digitalizada, é comparada com cada um dos padrões e um valor de aceitação no intervalo [0,1] é atribuído (Tabela 1). Com base nesse valor, os padrões reconhecidos (com aceitação acima de um limiar) são então analisados através da ativação de uma rede de raciocínio associada à

cada padrão (Figura 52), onde para a análise do padrão podem ser utilizados todos os dados armazenados na base de dados da aplicação [Cor95].

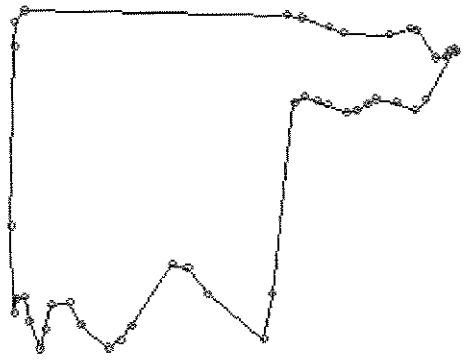


Figura 50 - Exemplo de carta de fundo

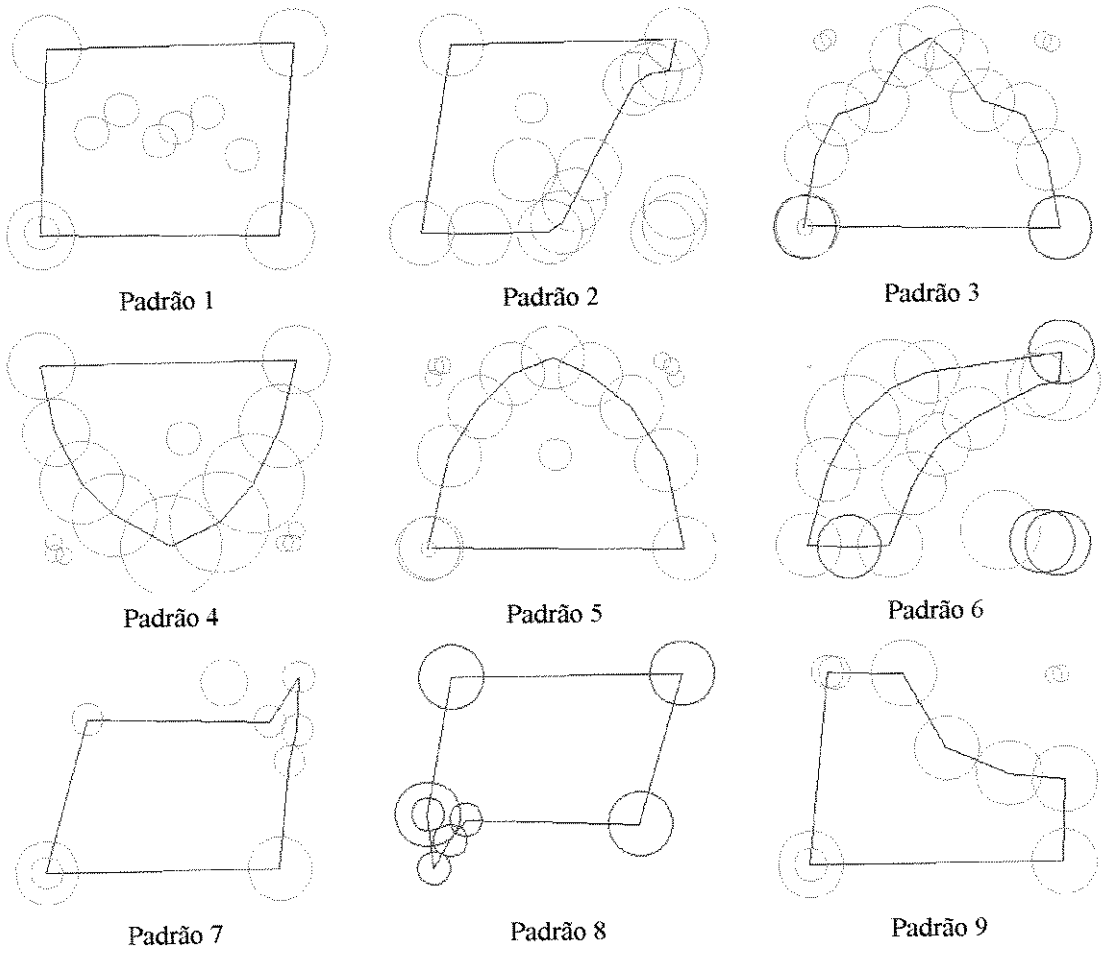


Figura 51 - Padrões armazenados no SICAD

Padrão	Aceitação
1	0,62
2	0,75
3	0,00
4	0,81
5	0,00
6	0,46
7	0,24
8	0,00
9	0,00

Tabela 1 - Exemplo dos valores de aceitação de cada padrão quanto à carta de fundo

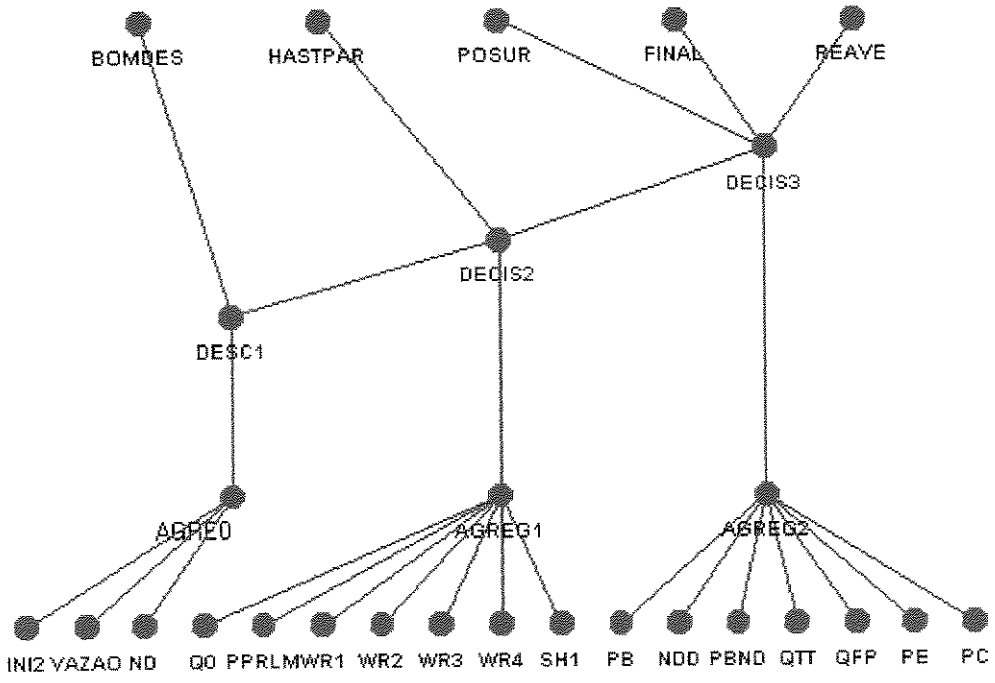


Figura 52 - Rede para análise de um padrão de carta de fundo

6.3.3.3. Unidades Portátil Tipo Microcomputador

Este modo de operação é utilizado quando da chamada "atuação isolada" do sistema SICAD. Esta atuação se dá quando o poço opera só com o controlador local e os problemas são detectados quando da ida ao poço. Neste modo de operação a carta dinamométrica de superfície é o dado disparador para análise.

6.3.4.O Módulo Gerenciador

O subsistema de gerenciamento do SIEP permite a atuação do gerente dentro do sistema como um todo. A arquitetura da Figura 35 foi adotada para o gerenciamento dos subsistemas do poço de bombeio mecânico (SICAD), do poço de "gas lift" contínuo e da planta de processamento dos fluidos produzidos pelos poços.

Como pode ser visto na Figura 35, há um microcomputador tipo industrial para controlar toda a operação da planta de processo; um outro microcomputador tipo industrial para controle local do poço de "gas lift" contínuo, e mais um microcomputador, de escritório, para gerenciar a operação do escritório central do SICAD, os diagnósticos de problemas e as ações correspondentes da planta de processo e do poço de "gas lift" contínuo.

6.3.4.1.Comunicação entre o Controle de Bombeio Mecânico e o Gerenciamento

Para realizar a comunicação entre o sistema de Gerenciamento e o Escritório Central do SICAD, ambos aplicações Kards, foram utilizadas as funções de paralelismo do Kards. O sistema de paralelismo do Kards funciona da seguinte maneira:

a) é utilizado um arquivo de controle de tarefas que contém um identificador da tarefa, um status, o nome da tarefa e uma sentença com informações adicionais;

b) o servidor checa esse arquivo com uma freqüência configurável a procura de uma tarefa com status 0. Esse status indica que um cliente enviou essa tarefa para o servidor executar;

c) quando o servidor encontra uma tarefa com status 0, ele lê a rede de processamento da sentença que deve ser executada, e então muda o status dessa tarefa para 1. Quando o servidor termina de executar essa rede, ele grava em um arquivo de resposta o símbolo terminal dessa rede e altera o status para 2. Em caso de erro na execução da rede, o erro é gravado no arquivo de resposta e o status é setado para 3.

O procedimento do cliente para pedir que o servidor execute uma rede é gravar no arquivo de controle a sentença que deve ser executada com o identificador que o servidor reconhece e ficar checando o arquivo de controle até que o status seja setado para 2 ou 3, então o cliente lê o símbolo terminal do arquivo de resposta e retira a tarefa do arquivo de controle. A freqüência de checagem do status e o tempo máximo de espera por uma resposta são determinados no arquivo de configuração do sistema - KARDIC.DEF, e os nomes de arquivos e identificações dos servidores são armazenados na gaveta *Servidores* do armário de estrutura da aplicação no Kards.

6.3.4.2.Comunicação entre a Planta de Processo e o Gerenciamento

A comunicação entre o micro da Planta de Processo e o micro do Gerente, foi feita utilizando uma rede local tipo ethernet com cabo coaxial interligando apenas os dois micros, que estavam na mesma sala, separados apenas por uma pequena distância. O software de rede utilizado foi o Windows for Workgroups, já que o Kards roda sob a plataforma Windows.

Para a comunicação entre os dois sistemas Kards (Plata de Processo e Gerenciamento), foi utilizado o mesmo esquema da comunicação entre o Escritório Central do SICAD e o Gerente, visto que toda a comunicação é baseada em arquivos. Para tal, bastou configurar o nome dos arquivos para que fizessem referência ao arquivo no computador da rede que fosse o servidor. Nessa configuração foi levada em questão a frequência de uso da comunicação, pois como o pooling no arquivo de controle é necessário tanto no servidor quanto no cliente (para aguardar a resposta), foi feita a configuração que fizesse menor uso da rede para o pooling, ou seja, os arquivos de controle ficaram nos micros do servidor. Deste modo, a rede apenas é utilizada quando um cliente pede uma informação ao servidor, então o cliente faz o pooling no arquivo de controle pela rede para aguardar o resultado, e lê pela rede o arquivo de resposta. O tempo do pooling utilizando a rede foi ajustado para 500 milisegundos.

6.3.4.3. Comunicação entre o Controle de Bombeio Gas-lift e o Gerenciamento

O poço de gas-lift usado no teste estava a uma distância de aproximadamente 2km da sala com o computador do Gerente, de modo que para o desenvolvimento do sistema **SIEP** foi implementado no sistema Kards a possibilidade de comunicação entre aplicações utilizando comunicação via rádio. Para tal, foram utilizadas placas de comunicação especialmente desenvolvidas para essa finalidade pela empresa *HI Tecnologia Indústria e Comércio Ltda* - a mesma que desenvolveu o hardware do controle local do SICAD.

A comunicação pode ser feita através de dois tipos de módulos (placas para o PC):

- Módulo ISB-227:

Deve ser utilizado para a comunicação via modem, no caso utilizando o canal de comunicação configurado como RS-232, com controle dos sinais de modem. As velocidades aceitas são 1.200, 2.400, 4.800 e 9.600 bps.

- Módulo MSR-226 R2:

Deve ser utilizado para a comunicação serial RS-422 - via cabo.

O protocolo implementado corresponde ao protocolo proprietário da HI-Tecnologia denominado HI-SCP. No caso trata-se de um protocolo tipo mestre/escravo half-duplex. Assim em um microcomputador do tipo IBM/PC cada módulo de comunicação ISB-227 e/ou MRS-226 pode ser configurado em modo mestre ou escravo. Em um microcomputador do tipo IBM/PC pode-se colocar até 8 módulos de comunicação (ISB-227 e/ou MRS-226), sendo cada um configurado em modo mestre ou escravo independentemente e cada mestre pode gerenciar até 250 escravos. Assim, por exemplo, em um único mircocomputador é possível configurar um módulo de comunicação em modo mestre e outro em modo escravo, ou ambos em modo mestre, etc. Os módulos ISB-227 e MSR-226 são módulos 'inteligentes' que não dependem da CPU para fazer a comunicação. Neste protocolo, somente o *mestre* pode pedir um dado ao *escravo*, não sendo possível o *escravo* enviar um sinal ao *mestre* sem que o *mestre*

peça o dado. Desta forma, é necessário que o gerente faça um *pooling* no *escravo* a fim de saber se o escravo necessita se comunicar com o gerente. Esse *pooling* foi implementado internamente no sistema Kards, via software.

No Kards foram implementados agentes primitivos especiais para fazer a comunicação com estas placas, de forma a permitir a construção de aplicações que utilizem este recurso.

O controle local do SICAD utiliza esses mesmos módulos de comunicação, de forma que as rotinas implementadas para a comunicação via rádio entre microcomputadores podem ser utilizadas também para a comunicação com o SCUB.

A ligação física do micro do Gerente com o micro do Poço de Gás-Lift ou com o controle local do SICAD é feita como mostrado na Figura 53.

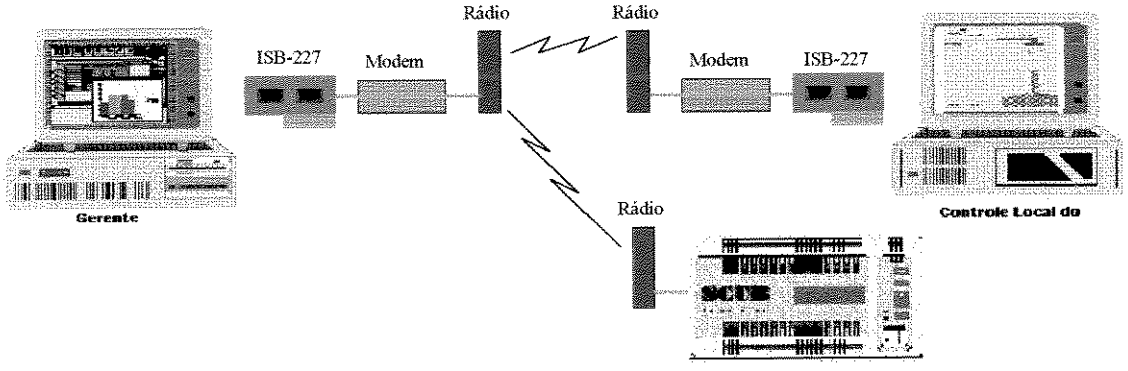


Figura 53 - Ligação física entre Gerente de Poço de "Gas Lift" e/ou SCUB

Centralizando as operações, o *gerente* é o responsável por checar se os controles do bombeio mecânico, da planta e do poço de "gas lift" estão funcionando corretamente. Isto é feito através de redes de raciocínio como a mostrada na Figura 54. Através do gerente também é possível acionar a maioria das funções de cada um de seus subsistemas.

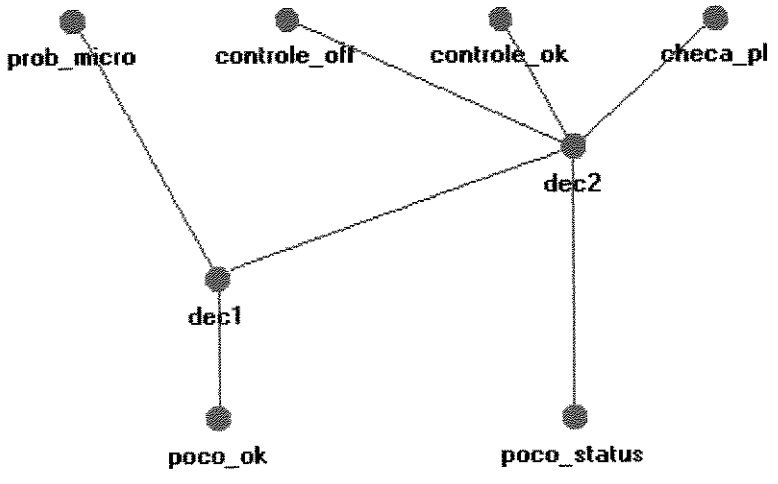


Figura 54 - Rede de problemas com o controle do poço de "gas lift"

6.4. Experimento de Campo

O sistema SIEP foi testado na Estação de Compressão e Coleta de Livramento, onde foram utilizados para o teste um poço de bombeio mecânico, um poço de "gas lift" contínuo e a planta de processamento dos fluidos produzidos.

As fotos mostradas em seguida, ilustram alguns aspectos do local onde se realizaram os testes, assim como permitem visualizar o ambiente de operação do sistema proposto neste estudo.

6.4.1.Automação do Poço de Bombeio Mecânico

As adaptações feitas para o experimento no poço em produção através de uma unidade de bombeio mecânico (UB) foram: a instalação de um sensor de posição tipo "encoder" na viga oscilante, de uma célula de carga na mesa de suporte do dinamômetro, e de um controlador local inteligente (SCUB), em um quadro de comando, instalado ao lado da UB.

O poço de bombeio mecânico utilizado no teste estava a aproximadamente 500 metros da sala da estação aonde estava o microcomputador do *gerente*, e a transmissão de dados foi feita através de cabo telefônico (RS-422).

No experimento de campo (vide Figura 55), o *gerente* acessou o sistema SICAD via o escritório central do mesmo, e as cartas dinamométricas de superfície e de fundo foram adquiridas e calculadas sincronamente pelo equipamento de controle local inteligente (SCUB).

Tendo em vista a distância do poço à sala de controle, localizada na estação de coleta e compressão, os dados para o escritório central, neste caso do SICAD, são enviados via cabo de comunicação. É importante ressaltar que o sistema SICAD foi instalado no microcomputador convencional (de escritório) operando em conjunto com o módulo gerenciador do sistema SIEP, conforme pode ser visto na Figura 35.

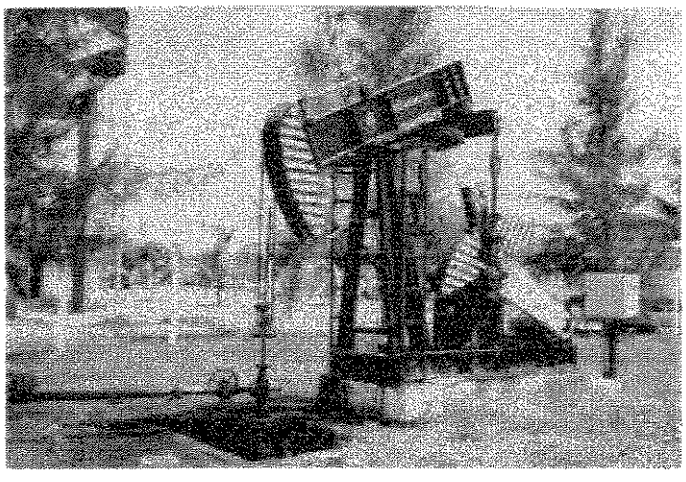


Figura 55 - Automação do Poço de Bombeio Mecânico

6.4.2.Automação do Poço de "Gas Lift" Contínuo

Para a automação do poço de "gas lift" contínuo foi necessária a instalação dos seguintes componentes: uma válvula de controle para substituir o "choke" de injeção de gás (vide Figura 56), e sensores com os respectivo transmissores para medição da pressão na cabeça do poço, do fluxo de gás injetado, da temperatura do revestimento e da pressão do revestimento (indicados na Figura 57), além da construção de uma casa de controle onde foi instalado o microcomputador industrial do controle local (Figura 58).

A monitoração das variáveis chegam através de cabo de instrumentação ao microcomputador do controlador local desse subsistema. Estes dados são então transmitidos via rádio/modem para o microcomputador onde foi instalado o subsistema de gerenciamento do sistema SIEP, na sala de controle da estação coletora. O módulo de gerenciamento do "gas lift" responde ao operador sempre que solicitado, sendo que a cada solicitação a informação é coletada dos sensores instalados no poço de "gas lift" e transmitida via rádio.

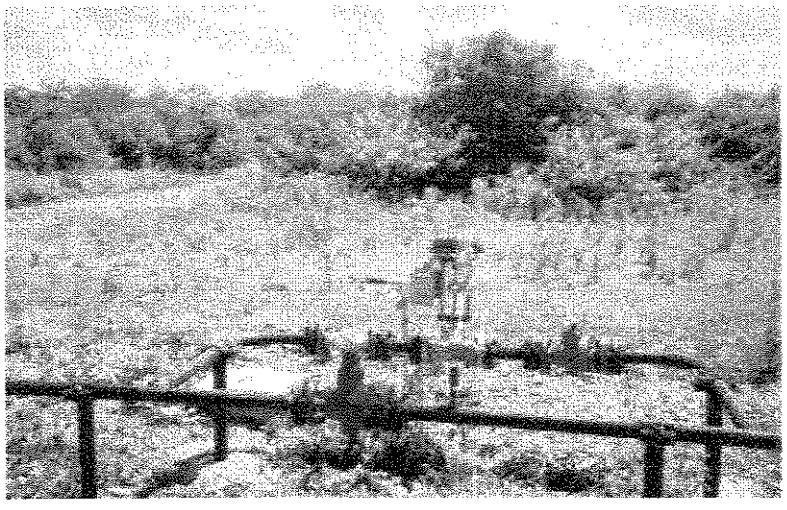


Figura 56 - Válvula Controladora de Injeção de Gás do Poço de "Gas Lift".

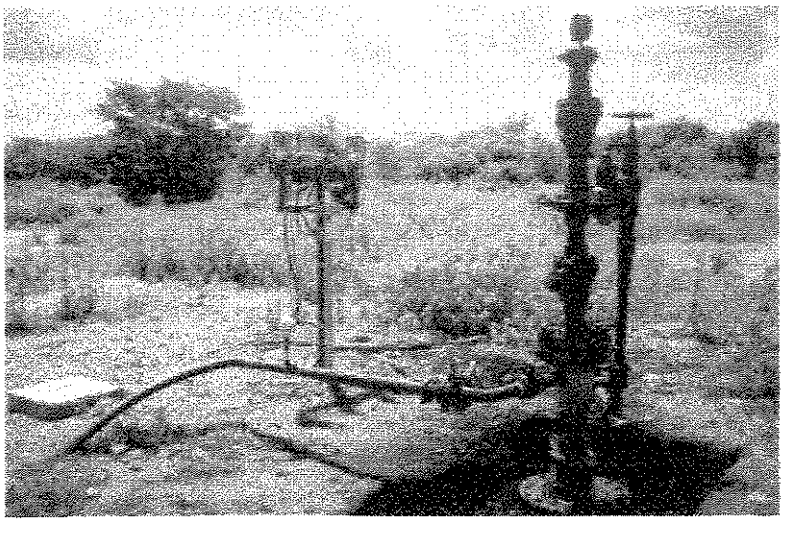


Figura 57 - Instrumentação para Aquisição das Variáveis do Poço de "Gas Lift".

O poço de "gas lift" utilizado no teste estava a aproximadamente 1800 metros da sala de controle e a transmissão de dados foi feita através de rádio/modem (RS-232).

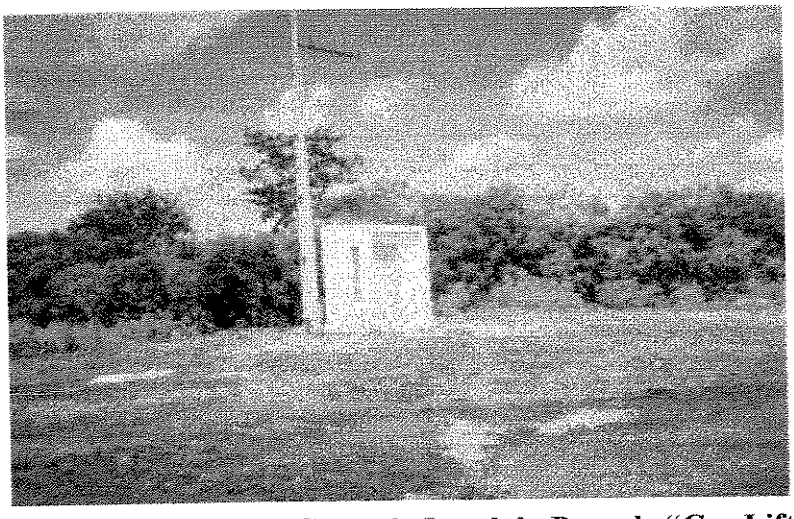


Figura 58 - Casa do Controle Local do Poço de "Gas Lift".

6.4.3.Automação da Planta de Processo

A automação dos vasos separadores e equipamentos de um processo requer a existência ou instalação de dois tipos de instrumentos: sensores (no caso de pressão e de nível) e atuadores.

Os vasos utilizados no teste estavam a uma distância que varia de 30 a 100 metros do microcomputador do *controle local da planta*, e a transmissão dos sinais foi feita via cabo de instrumentação.

Para automatizar a planta de processamento dos fluidos produzidos foi necessária a instalação de conversores de corrente para pressão para os atuadores e conversores de pressão para corrente para os sensores. Isto se deve ao fato de que todos os instrumentos disponíveis nesta planta para o experimento eram pneumáticos, utilizados nas válvulas acionadas até então por controladores do tipo PID (proporcional, integral e derivativo) instalados localmente nos vasos. Desta forma, todas as válvulas de controle, todos os sensores e todos os atuadores já existentes no processo foram reaproveitados neste experimento.

No teste de campo foi utilizada a atuação em apenas dois separadores de produção e dois separadores de teste, além dos sensores destes separadores e sensores de pressão de um tratador e do depurador bem como os sensores das bombas e compressor.

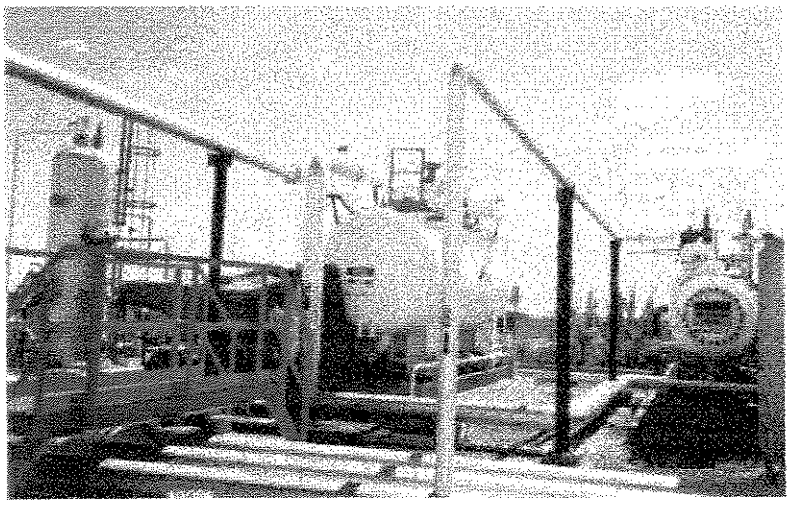


Figura 59 - Separadores de Teste e de produção.

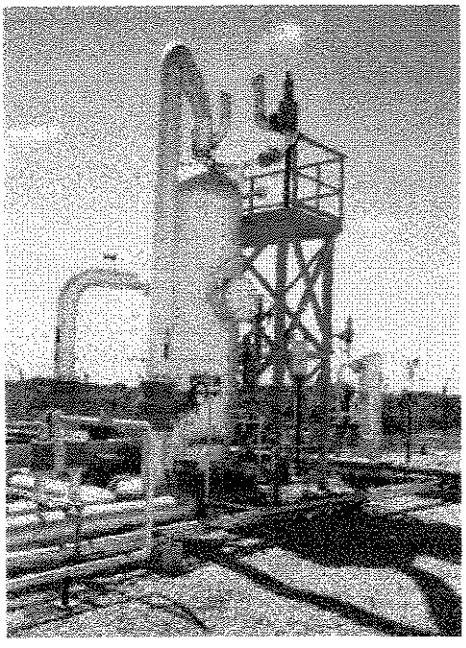


Figura 60 - Depurador de Baixa Pressão.

A ligação dos sensores e atuadores utilizados na planta com o microcomputador do controle foi feita através de um painel de controle onde foi também instalada uma fonte de alimentação de 24V dc, pois os conversores de pressão para corrente utilizados necessitavam deste tipo de alimentação. O esquema das ligações utilizadas pode ser observado na Figura 61.

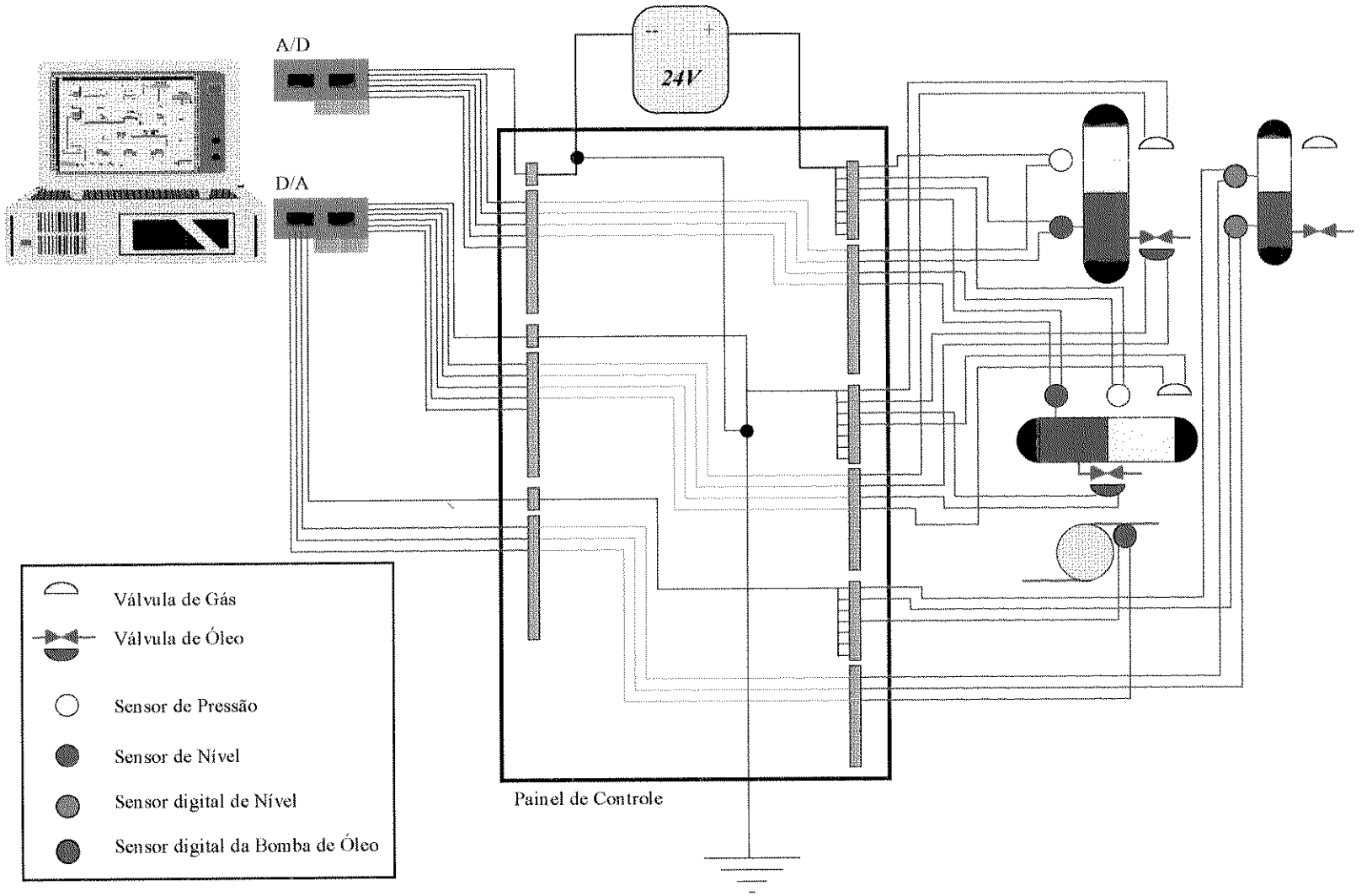


Figura 61 - Esquema da Ligação do Sistema A/D e D/A da Planta de Processamento

6.4.4. Resultados

O sistema SIEP está sendo utilizado a mais de seis meses na estação de compressão e coleta de livramento e tem obtidos resultados considerados bons pelo pessoal da PETROBRÁS responsável pelo desenvolvimento do sistema, bem como dos operadores que estão utilizando o sistema. Mais detalhes sobre os resultados obtidos podem ser vistos em Patrício [Pat96] e Correa [Cor95]. O SICAD, que foi desenvolvido à parte do SIEP, está em fase de implantação para seu uso em larga escala em duas regiões produtoras da PETROBRÁS, o que comprova a sua eficiência.

Estes resultados demonstram a capacidade do sistema Kards para suportar aplicações de controle inteligente em um ambiente especializado.

7. Conclusão

Do exposto anteriormente é lícito concluir que:

1) o Kards é uma plataforma computacional com a versatilidade requerida para a implementação de sistemas inteligentes utilizando tanto técnicas de processamento numérico como técnicas de processamento simbólico;

2) a teoria de Linguagens Formais Nebulosas que suporta o desenvolvimento do Kards mostrou-se adequada como ferramenta para a implementação de atividades inteligentes tão variadas quanto controle nebuloso a processamento de linguagem natural;

3) as implementações das aplicações desenvolvidas na área da indústria de petróleo obtiveram uma aprovação prática de sua eficiência;

4) a assimilação do manuseio do Kards por vários tipos diferentes de usuários mostra que se conseguiu desenvolver uma plataforma de trabalho eficiente e de baixa complexidade operacional.

A restrição mais importante a que o sistema está submetido está relacionada à escolha de sacrificar a velocidade pela versatilidade do processamento, principalmente levando-se em conta a escolha do Visual Basic como a linguagem básica para o desenvolvimento do Kards. Entretanto, este problema poderá ser minimizado em versões futuras que utilizem como linguagem básica uma linguagem tipo C++ que permite explorar melhor as capacidades da máquina.

Nomenclatura

Abreviações :

AG	Algoritmo Genético
fbf	fórmula bem formada
GLC	"Gas Lift" Contínuo
IA	Inteligência Artificial
LF	Linguagem Formal
LFN	Linguagem Formal Nebulosa
OGLC	Otimizador de "Gas Lift" Contínuo
SCUB	Sistema de Controle de Unidade de Bombeio
SE	Sistema Evolucionário
SIEP	Sistema Inteligente para Elevação de Poços
SN	Sistema Neural
SRDPI	Sistema de Resolução Distribuída de Problemas Inteligentes
UPGN	Unidade de Processamento de Gás Natural

Símbolos :

$d(\ ,\ )$	cadeia de derivação
$A(\ )$	número de cópias
$C(\ )$	cardinalidade
Δ	função de custo
$\phi(\ )$	dobra
$l(\ )$	tamanho
$\mu(\ )$	similaridade
$\rho(\ )$	possibilidade
$\Gamma, \partial, \Phi, \bullet$	t- ou s- norms
V	conjunto de símbolos
η	elemento vazio
P	conjunto de regras de reescrita
s	cadeia de símbolos
L	linguagem formal
G	gramática

Bibliografia

[Adl94] L. M. Adleman. "Molecular computation of solutions to combinatorial problems". *Science*, 266, pag. 1021-1024, 1994

[Ale90] Lideniro Alegre, 1990.

[AnRo89] J. A. Anderson e E. Rosenfeld. "Neurocomputing: foudations of research". MIT Press, Cambridge, 1989

[Bar96] M. A. Barreto Filho. "Generation of Bottomhole Dynamometric Chart for the Diagnostic of Sucker Rod Pumping in Oil Wells", *Tese de Mestrado*, Unicamp, dezembro, 1995.

[BGH89] L. B. Booker, D. E. Goldberg e J. H. Holland. "Classifier systems and genetic algorithm". *Artificial Intelligence*, vol 40, pag 235-282, 1989.

[Bir91] L. Birnbaum. "Rigor mortis: a response to Nilsson's 'Logic and Artificial Intelligence'". *Artificial Intelligence*, vol 47, pag 57-77, 1991.

[CGA88] B. Chandrasekaran, A. Goel e D. Allemang. "Connectionism and information". *AI Magazine*, pag 25-34, 1988.

[Cha81] B. Chandrasekaran. "Natural and social system metaphors for distributed probelm solving: introduction to the issue". *IEEE Trans. Sys. Man and Cybernetics*, vol 11, pag 1-5, 1981.

[Cho57] Chomsky, N. "Syntatic Structures". The Hage, Mounton, 1957.

[Cho65] Chomsky, N. "Aspects of theTheory of Syntax". MIT Press, Cambridge, 1965.

[CKLM92] S. E. Conry, S.E., K. Kuwarabara, V. R. Lesser e R. A. Meyer. "Multistage negotiation for distributed contraint satisfaction". *IEEE Trans. Sys. Man and Cybernetics*, vol 21, pag 1462-1477, 1992.

[Cor95] J. F. S. Corrêa. "Sistema Inteligente para Aplicações de Soluções ao Bombeamento Mecânico de Petróleo", *Tese de Mestrado*, Unicamp, dezembro, 1995.

[DaSm83] R. Davis e R. G. Smith. "Negotiation as a metaphor for distributed problem solving". *Artificial Intelligence*, vol 20, pag 63-109, 1983.

[DuMo91] E. H. Durfee e T. A. Montgomery. "Coordination as distributed search in a hierarchical behavior space". *IEEE Trans. Sys. Man and Cybernetics*, vol 21, pag 1363-1378, 1991.

[Ede87] G. M. Edelman. "Neural Darwinism: the theory of neuronal group selection". *Basic Books*, N. York, 1987.

[Fox81] M. S. Fox. "An organizational biew of distributed systems". *IEEE Trans. Sys. Man and Cybernetics*, vol 11, pag 70-80, 1981.

[Gal88] S. I. Gallant. "Connectionist expert systems". *Communications of ACM*, vol. 31, pag 152-169, 1988.

[HeIn91] C. Hewitt e J. Inman. "DAI betwixt and between: From 'Intelligent Agents' to Open System Science". *IEEE Trans. Sys. Man and Cybernetics*, vol 21, pag 1409-1419, 1991.

[Hew77] C. Hewitt. "Viewing control structure as patterns of passing messages". *Artificial Intelligence*, vol 8, pag 323-364, 1977.

[Hin89] G. E. Hinton. "Connectionist learning procedures". *Artificial Intelligence*, vol. 40, pag. 185-234, 1989.

[Kac86] J. Kacprzyck. "Group decision making with a fuzzy linguistic majority". *Fuzzy Sets and Systems*, vol. 18, pag. 105-118, 1986.

[Kir91] D. Kirsh. "Foundations of AI: the big issues". *Artificial Intelligence*, vol 47, pag 3-30, 1991.

[KoHe81] Kornfield, W. e C. E. Hewitt. "The scientific community metaphor". *IEEE Trans. Sys. Man and Cybernetics*, vol. 11, pag. 24-42, 1981.

[LeCo81] V. R. Lesser e D. D. Corkill. "Functionally Accurate, Cooperative distributed systems". *IEEE Trans. Sys. Man and Cybernetics*, vol 11, pag 81-96, 1981.

[Les91] V. R. Lesser. "A retrospective view of FA/C distributed problem solving". *IEEE Trans. Sys. Man and Cybernetics*, vol 21, pag 1347-1362, 1991.

[Lip95] R. J. Lipton. "DNA Solution of Hard Computational Problems". *Science*, vol. 268, abril, 1995.

[McPi49] W. S. McCulloch e W. Pitts. "Bull. Math". Biophys. 5, pag 115-133, 1949.

[McSh95] H. H. McAdams e L. Shapiro. "Circuit Simulation of Genetic Networks". *Science*, vol. 269, pag. 650-656, agosto, 1995

[MiPa69] M. Minsky e S. Papert. "Perceptron: an introduction to computational geometry". MIT Press, Cambridge, 1969

[Miz73] M. Mizumoto et al. "N-Fold Grammar", *Information Sciences*, vol 5, pag 22-32, 1973.

[NeRa75] C. V. Negoita e D. A. Ralescu. "Applicattions of Fuzzy Sets to Systems Analysis", John Wiley & Sons, New York, 1975.

[Nil91] N. J. Nilsson. "Logic and artificial intelligence". *Artificial Intelligence*, vol 31, pag 31-56, 1991.

[Pat96] A. R. Patricio. "Estudo de um Sistema Inteligente para Elevação de Poços e Controle de Processos Petrolíferos", *Tese de Doutorado*, Unicamp, novembro, 1996.

[PeRo93] W. Pedrycz e A. F. Rocha. "Fuzzy -set based models of neurons". *IEE Trans. Fuzzy Systems*, vol 1, pag 254-263, 1993.

[PLR95] W. Pedrycz, P. C. F. Lam e A. F. Rocha. "Distributed Fuzzy System Modelling". *IEEE Trans. Sys. Man Cyb*, vol 25/5, pag 769-780, 1995.

[ReRo95] M. P. F. Rebello e A. F. da Rocha. "KARDS: na Intelligent Neural Environment". *Proceedings of IFSA 95*, IFSA, pag. 697-699, São Paulo, Brasil, 1995.

[RFB80] A. F. Rocha, E. Françoso e M. A. Balduíno. "Neural Languages". *Fuzzy Sets and Systems*, 3/1, pag. 11-35, 1980.

[Roc92] A. F. Rocha. "Neural Nets: A theory for brains and machines". *Lectures Notes in Artificial Intelligence*, Springer-Verlag, Berlim, 1992, vol. 638.

[Ros58] F. Rosemblatt. "The perceptron: a probabilistic model for information storage and organization in the brain". Psychological Rev., vol 65, pag 386-408, 1958.

[RoSe94] A. F. Rocha e A. B. S. Serapião. "Neurofuzzy Symbolic Systems and Pattern Recognition". *Proceedings of NAFIPS 94*, NAFIPS, San Antonio, Texas, 1994

[RSR96] A. F. Rocha, A. B. S. Serapião. "Neurofuzzy Symbolic Systems and Pattern Recognition". *Submetido.*

[RTMK92] A. F. Rocha, M. Theoto, A. M. K Myiadahira, M. S. Koizumi e I. R. Guilherme. "A neural net for extracting knowledge from natural language data base". *IEEE Transactions on neural nets*, 2/5, pag. 115-123, 1992.

[Sea93] D. B. Searls. "The computational linguistcs of biological scientists in Artificial Intelligence and Molecular Biology", *AAAI Press*, L. Henter, pag. 47-120, 1993.

[Ser96] A. B. Serapião, "SENSOR: Um Sistema Sensorial Inteligente Distribuído", *Tese de Mestrado*, Unicamp, 1996.

[SRRP96] A. B. S. Serapião, A. F. Rocha, M. P. F. Rebello e W. Pedrycz. "Toward a Theory of Genetic Systems". Em: *Genetics Algorithms and Soft Computing*. F. Herrera e J. L. Verdegay (Eds). Physica Verlag, Heidelberg.

[Zad73] L. A. Zadeh. "The concept of a linguistic variable and its application to approximate reasoning". *IEEE Transactions on systems, man and cybernetics*, SMC-3, pag. 28-44, 1973

Apêndice A

A seguir a lista dos agentes primitivos do Kards por ordem alfabética e uma breve descrição de cada um.

_A	'E' lógico
_AI	Comparação maior ou igual
_D	Divisão numérica
_EDITA_FICHA	Edita a ficha atual
_EI	Comparação menor ou igual
_EXEC_ACAO	Executa uma ação do menu de ações do Kards
_EXEC_ACAO2	Executa uma ação pelo nome
_FICHA	Ativa o módulo FICHA do Kards
_IG	Compara se as entradas são iguais
_M	Multiplicação numérica
_MA	Compara se a primeira entrada é maior que a segunda
_ME	Compara se a primeira entrada é menor que a segunda
_MENU_FICHA	Mostra o menu do módulo FICHA do Kards
_MENU_GERAR	Vai para o menu GERAR do Kards
_MENU_MAIN	Retorna ao Menu Principal do Kards
_MM	Comparação maior ou menor (diferente)
_O	'OU' lógico
_P	Potenciação
_QUEST	Ativa o módulo QUEST
_S	Faz adição numérica e concatenação simbólica
_U	Subtração numérica
ABS	Devolve o valor absoluto do número
AJUSTA_GRAF	Ajusta a tela gráfica
AJUSTA_GRAF_SIMP	Ajusta tela gráfica
ALOCA_ARM	Aloca uma variável do tipo armário
ALTERA_CAD	Altera o conteúdo de uma cadeia
ALTERA_LISTA	Altera o valor de um item de uma lista de confiança
ANTES_DELI	Retorna com o conteúdo antes de um delimitador
ANTES_DELI2	Retorna com o conteúdo antes do delimitador
APOS_DELI	Retorna com o conteúdo da string após o identificador
ARCTAN	Calcula o Arco Tangente do Ângulo
ARM_ATU	Retorna com o índice da variável de armário atual
ASC	Retorna o valor do código do caracter
ATIVA_TIMER	Ativa o timer para contrôle de fluxo
BOTOES	Ajusta botões Ok/Cancel/Próximo para executar uma função
CALC	Calcula a expressão fornecida
CALC_BACK	Calcula macro em background
CALC_COR	Calcula a cor dos agentes a serem mostrados
CAMPO	Atribui valor ao Campo

CAMPO_ATUAL	Retorna o número do Campo Atual
CAMPO_COL	Atribui valor a coluna
CAMPOS	Atribui valores a todos os campos da ficha atual
CHR	Retorna o caracter do código especificado
CLIPBOARD	Le valor do clipboard
CLS	Limpa a tela de apresentação
CM_DIALOG	Cria uma caixa de diálogo padrão
CODIF	Método de codificação
COL	Retorna o valor de uma coluna em ficha planilha
COMPRIME_XY	Extrai pontos significativos
CONC	Concatena cadeias
CONF_LIS	Consulta lista para codificar informação
CONF_NUM	Retorna o valor de Y dado X, através de interpolação linear
CONV_GRID_IMP	Converte controle de impressão para ficha grid
CONV_MESA	Converte uma lista de leitura via mesa digitalizadora
CONVERTE_ARMA	Converte um armário do formato antigo para o novo
CONVERTE_FIC	Converte uma ficha do formato antigo para o novo
COPIA	Copia um arquivo
COS	Calcula o Coseno do Ângulo
CRIA_ARMA	Cria um armário
CRIA_IND	Cria um cadeia de indexação
CURVA	Retorna a cadeia da curva atual na tela gráfica
CURVA_PARAM	Retorna parâmetros de uma curva já mostrada
CURVA_STR	Manuseio da Curva Atual
DATA_ATUAL	Retorna a data atual
DELETA_ARQ	Apaga um arquivo
DESALOCA_ARM	Desaloca uma variável do tipo armário
DESATIVA_TIMER	Desativa o timer
DIM	Cria uma variável local
DO_EVENTS	Checar ocorrências no Windows
EDITA_LAYOUT_	Edita o layout de uma rede em forma de ficha normal/grid
EDITA_METO	Edita os métodos de um nó de uma rede
EDITA_TEXTO	Função para uso do Sistema Edita_Textos
EMI	Retorna com a lista dos nós para quem o nó envia dados
ESCOLHE_LISTA	Apresenta relação de dados passados por parâmetro para escolha
ESPERA	Espera mudança de status
EXEC	Executa um programa em modo Shell
EXEC_FILHOS	Executa um método em todos os filhos daquele nó
EXEC_METO	Executa um método específico da rede de ficha atual
EXEC_METO_COL	Executa um método em uma coluna específica
EXEC_SER2	Executa função no servidor
EXIST	Checa a existencia de um arquivo
EXP	Calcula a Exponencial do Número
EXTRAI_COLUNAS	Extrai colunas de um array

F_FIG	Mostra figura permitindo a inserção de símbolos.
F_FIG_STATUS	Retorna status de um grupo de botões de status de F_SIMBO
F_SIMBO	Mostra símbolos em figura exibida por F_FIG
FACT	Retorna o Fatorial de um Número
FICHA_ORD	Retorna o número da linha da ficha atual na pasta
FILELEN	Retorna o número de bytes de um arquivo
FILENAME_FILTER	Filtra os um símbolo para que possa ser nome de arquivo
FILTRA_LISTA	Filtra uma lista
FIND_LIST	Procura por um elemento em uma lista
FOR	Executa expressão n vezes
FORMAT	Formatação de cadeias
FORMAT_ARRAY	Formata um array de números
FORMAT0	Formatação de cadeias
FRASE	Agente que executa o JARGÃO
FREQ_DIR	Calcula a frequencia de direcoes
FUNDO_GRAF	Superpõe Curvas
GERA_CURVA	Gera um vetor XY aleatório
GET_EDITEX	Obtém texto do objeto Edita_Texto
GET_MARCA	Obtém palavra marcada na tela EDITA_TEXTOS
GRAPH_CAPTION	Captura gráfico
GRAPH_DRAW	Apresenta o gráfico na tela
GRAPH_HIDE	Oculta o gráfico
GRAPH_LABEL	Coloca o nome nos eixos do gráfico
GRAPH_LEGEND	Legenda do gráfico
GRAPH_PRINT	Imprime o gráfico
GRAPH_SHOW	Mostra a janela de gráficos
GRAPH_STYLE	Estilo do gráfico
GRAPH_TITLE	Título do gráfico
GRAPH_TYPE	Tipo de gráfico
GRAVA_ARQ	Grava dados em arquivos
GRAVA_FICHA	Grava um aficha
GRAVA_IND	Grava o índice do usuário atual
GRID_POS	Lê a posição do cursor na planilha
GRID_SEL	Marca células da planilha
HELP	Mostra o HELP de uma função
HEX	Calcula o valor hexadecimal do número
HIST	Põe mensagem no quadro Histórico
IF	Implementa controle IF THEN ELSE
IMPORTA_CAMPO	Importa dados de um armário
INDICE	Retorna o nome do Índice em Uso
INFO_SERVER	Fornece informações de um servidor do paralelismo
INPUT	Obtém dado na tela de Entrada de Dados
INPUT_BOX	Pede um valor em uma dialog box
INPUT_SENHA	Lê uma senha do teclado
INT	Devolve a raiz inteira do número
ITEM	Retorna um item especificado de uma lista

L_TRIM	Corta brancos à esquerda
LE_ARQ	Le dados de arquivo
LE_KARDIC_DEF	Lê um arquivo de definições.
LE_LISTA	Seleciona item em uma lista
LE_MESA	Lê uma curva via mesa digitalizdora
LE_REDES	Le um registro com uma sentença tipo rede
LEFT	Retorna a parte esquerda de uma cadeia
LEN	Devolve o comprimento de uma cadeia
LET	Define uma variável
LET_LISTAV	Atribui valores a uma lista de variáveis
LET_NO	Atribui um valor a um nó do grafo atual
LET_NO_F	Atribui um valor a um método do grafo de armário atual
LETG	Atribui um valor a uma variável global
LETR	Atribui um valor a uma variável
LIKE	Compara duas strings aceitando "*" e "?"
LN	Calcula o Logarítmo Natural
LOG	Calcula o Logarítmo Decimal
LPRINT	Imprime textos e controles
LPRINT_TABLE	Imprime uma tabela
MAIUSCULA	Transforma cadeia em maiúsculas
MEMO	Altera/lê o valor das memórias de ClipBoard do Kards
MENS	Retorna uma mensagem da lista de mensagens
MENU	Cria um menu flutuante e espera uma opção ser selecionada
MERGE_LISTA	Combina Dicionários
METODO_FIC	Retorna o método em uso pela ficha
METODO_FIN	Metodo Fina
METODO_INI	Define metodo inicial
METODO_INT	Define método intermediário
MID	Retorna uma subcadeia da cadeia
MKDIR	Cria um diretório
MONTA_GRID	Apresenta uma planilha na tela
MOSTRA_BOTAO	Programação de macros em botões da Tela de Manuseio
MOSTRA_FICHA	Mostra ficha na tela
MOSTRA_FICHA_NORMAL	Método padrão para mostrar fichas
MOSTRA_FICHA_P	Mostra ficha P
MOSTRA_GRAFO	Mostra o grafo da ficha atual
MOSTRA_GRAFO_F	Mostra o grafo da estrutura de armário atual
MOSTRA_PIC	Mostra figura BMP
MOSTRA_REDE	Mostra a rede de uma expressão da calculadora
MOSTRA_REDE_ANT	Mostra rede de uma expressão de forma hierarquica
MSGBOX	Manuseio do MSGBOX
NFIND_LIST	Procura por um elemento em uma lista
NO	Retorna com o conteúdo de um método da rede
NO_F	Retorna com o valor de um método da estrutura de armário
NO_F_ARM	Retorna com o valor de um nó de um armário

	específico
NOME_CAMPO	Retorna o nome do campo
NOME_DIR	Extrai o nome do diretório.
NOME_SEM_DIR	Extrai somente o nome do arquivo do path completo
NOMES_CAMPOS	Retorna uma lista com os nomes dos campos da ficha atual
NORMAL_ARQ	Normaliza dados de um arquivo
NORMALIZA	Normaliza uma cadeia
OCORRENCIAS	Retorna o número de ocorrencias de uma subcadeia em um cadeia
OEM_TO_ANSI	Converte uma string com caracteres de DOS para caracteres de WINDOWS
ON_OFF	Threshold de uma imagem
OPER	Lê/altera o valor da variável interna OPER
OPER_NADA	Seta a variável OPER para que não faça nada.
ORD_ATUAL	Retorna com o conteúdo do .ORD atual
ORD_FICHA	Retorna uma linha da pasta atual
ORD_USUARIO	Retorna o número de usuário
ORDENA	Ordena uma lista
ORDENA_LISTA	Ordena uma lista a partir do termo fornecido como delimitador
PATH_ARM	Mostra nome do armário do calcpath n do armário atual
PATH_NUM_ORD	Número de .ORDs do calcpath do armário atual
PATH_ORD	Mostra .ORD do calcpath n do armário atual
PAUSA	Interrompe o processamento e espera
PEDE_ARM	Seleciona um Armário
PEDE_FIC	Seleciona Ficha
PEDE_GAV	Seleciona Gaveta
PEDE_ITEM	Pede dados para um item
PEDE_MASCARA	Seleciona máscara para ficha de máscaras
PEDE_USU	Seleciona Usuário
PEGA_CURVA	Volta vetor XY de um curva
PERCEPTRON	Calcula a ativação de um Perceptron
PLOTA_CURVA	Plota uma curva definida um vetor
POE_LOG	Poe uma mensagem na janela de LOG
POSICAO	Retorna a posição de uma subcadeia em um cadeia
POSICIONA_TEXTO	Marca uma seleção na tela de Texto do KARDS
PREENCHE_MASCARA	Manuseia dados em ficha máscara
PREENCHIDOS	Retorna com número de células preenchidas
PRINT	Escreve na Tela de Apresentação
PRINT_	Imprime dados na tela de texto do Kards
PRINT2	Escreve na Tela de Entrada de Dados
PRINTB	Mostra uma string na linha de status
PROCESSA_TEXTOS	Processa formulários de programação
PROCURA_ITEM	Procura por um item em uma lista e retorna o item
PROXIMA_GAVETA	Seleciona a próxima gaveta
PROXIMO_USU	Identifica o próximo usuário com mesmo nome
PUSH_ARM	Executa uma função sem alterações no armário atual

QUEST	Aciona uma rede QUEST
R_TRIM	Corta caracteres em branco à direita
REC	Retorna com a lista dos nós de quem o nó recebe dados
RETINA	Imita o funcionamento da retina humana
RETIRA_ITEM	Devolve a n-ésima ocorrência de um campo de uma lista que contém uma subcadeia especificada
RIGHT	Retorna a porção direita de uma cadeia
RND	Fornece um número aleatório
SAY	Ativa o sintetizador de voz da Sound Blaster
SELE_ARM	Seleciona Armário
SELE_DIR	Muda o diretório atual
SELE_FIC	Seleciona Ficha
SELE_GAV	Seleciona Gaveta
SELE_IND	Seleciona Índice Secundário
SELE_LISTA	Seleciona itens de uma lista
SELE_USU	Seleciona o usuário
SEND_KEYS	Envia um conjunto de teclas a janela ativa
SET_MARCA	Modifica texto marcado na tela EDITA_TEXTOS
SETA_CLIPBOARD	Poe valor no clipboard
SETA_FUZZY	Ajusta formulário FUZZY atual.
SETA_HIST	Controle do quadro de histórico
SETA_STATUS	Define status
SGN	Retorna o sinal do número
SHOW_FUZZY	Cria formulário de Funções Nebulosas
SIN	Calcula o Seno do Ângulo
SOMA	Somatória em ficha planilha
SOMA_ARRAY	Soma dois arrays linha a linha
SQR	Calcula o quadrado do número
SQRT	Calcula a raiz quadrada
STATUS	Altera Status em Blackboard
STATUS_INFO	Retorna o campo de informação referente a tarefa especificada
STR	Transforma sequencial de caractéres em cadeias
STR_DATE	Corvete um simbolo de data qualquer para dd/mm/aaaa
STR_SUBST	Substitui subcadeias em uma cadeia
TELA_FICHA_GRID	Mostra a tela de planilha do Kards
TELA_FICHA_TEXT	Mostra a tela de Texto do Kards
TEXTO_ALT	Retorna texto modificado em Ficha Textos
TIME	Retorna a hora
TIRA_BOTOES	Tira todos os botões colocados com MOSTRA_BOTAO.
TIRA_COMENTARIOS	Tira comentários de um texto
TIRA_CURVA	Tira a janela do Plota_Curva
TIRA_EDITEX	Desativa o objeto EDITA_TEXTOS
TIRA_F_FIG	Tira figura mostrada por F_FIG()
TIRA_PIC	Tira uma figura da tela
TIRA_REP_FIM	Tira caracteres repetidos do final da string
TIRA_REPETIDOS	Tira sequência de caracteres repetidos

TIRA_ULTIMO_BOTAO	Manuseio de botões na tela de Manuseio
TRANSPOSTA	Retorna com a transposta de uma matriz
TREINA_P	Treina Perceptron
TXT_QUEST	Formatação da ficha QUEST
ULT_LINHA_PREENCHIDA	Retorna o número da última linha preenchida
USUARIO_ORD	Retorna a cadeia de indexação do usuário atual
VAL	Retorna o valor de uma cadeia
VALOR_CAMPO	Retorna o valor do campo
VALOR_TEXTO	Retorna o texto da ficha atual
VALOR_TEXTO_SC	Retorna o texto da ficha atual sem comentários
VAR_GLOBAIS	Retorna uma lista com todas as variáveis globais definidas
VAR_W	Acesso à variável W
VAR_Z	Acesso a variável Z
VG	Retorna o valor de uma variável global
WHILE	Executa expressão enquanto condição se mantém
ZERA_OPER	Zera a variável Oper

A seguir a lista das redes de processamento (em ordem alfabética) utilizadas pelo Kards para realizar seu processamento básico (redes de processamento definidas para o Kards, e não para as aplicações implementadas no Kards).

_BOLD	Retorna sequência de impressão para BOLD
_CENTER	Retorna comando de impressão para centralizar
_FONT_SIZE	Ajusta o tamanho da fonte para impressão
_LEFT	Retorna comando de impressão para alinhamento à esquerda
_NORMAL	Retorna sequência de impressão para NORMAL
_TABLE	Devolve comando de impressão para imprimir tabela
_UNDERLINE	Ajusta impressão para sublinhado
ABRE_ARM_B	ABRE_ARMÁRIO PARA BUSCAS
ABRE_ARMARIO	Abre armário KARDS
ABRE_FICHA	Abre uma Ficha da Pasta Selecionada
ABRE_GAVETA	Abre uma Gaveta do Armário Kards
ABRE_GAVETA_QUEST	Seleciona uma gaveta para o QUEST
ABRE_PASTA	Abre uma Pasta na Gaveta KARDS
ABRE_PASTA_QUEST	Seleciona um usuário para o QUEST
ABRE_QUEST	Ativa o QUEST
ADCIONA_NODO	Adiciona um agente à lista de agentes executados
APAGA_FICHA	Apaga o conteúdo de uma ficha.
APAGA_LIN_PLAN	Apaga as linha selecionadas de uma planilha
ARMARIOCOPIA	Define Armário para Cópia
BLACKBOARD	Mostra o conteúdo do blackboard
BOTAO	Define as macros das operações da Macro Salva_Ficha_L
BUSC_MASC	Busca Máscara para Macro ENCAD_TEXT

BUSCA_INFO	Busca hipótese em um Blackboard
BUSCAS	Usada para Edição de Buscas
C_AD	Lê dados do conversor AD com a configuração em CAD.DEF
C_DA	Escreve dados no conversor DA com a configuração em CDA.DEF
C_DI	Le dados da entrada digital do conversor AD
C_DO	Escreve dados na saída digital do conversor DA
CALC_FICH	Macro utilizada para ativar os cálculos no Manus_Ficha_L
CALC_IDADE	Calcula quantos anos e meses uma data tem
CALCPLAN	Executa um cáculo em ficha planilha
CAMPO_ATUAL	Devolve o valor do campo atual para Exec_Meto
CAMPO_LISTA	Edita um campo tipo lista
CASO	Aciona o módulo CBR
CBR	Raciocínio por casos
CHAMA_QUEST	Aciona o módulo QUEST
CHAMA_SIMUL	Mostra simulação
CHECA_SENHA	Pede senha se necessário
CLASSE_SIMB	Classifica Curva por método simbólico
CLICK_GRID	Seleciona e apaga linhas de uma planilha com o mouse
CONV_CLA_GRID	Converte linha CLA para linha Planilha
CONV_DIC_GRID	Converte formato do dicionário do jargão para um grid.
CONV_GRID_DIC	Converte formato de grid para dicionário do Jargão
CONVERTE_DOC	Converte arquivos .DOC do Kardic para DOS
COPIA_FICHA	Copia uma Ficha da Pasta Atual para Armário Especificado
COPIA_PASTA	Copia a Pasta do Usuário Atual para Armário Especificado
COPY_FICHA	Copia fichas em uma gaveta
CORTA_TEXTO	Corta textos selecionado do Edita_Texto e coloca no Clipboard - igual a Get_Marca_Editex
CRIA_ARMARIO	Pede nome e cria novo armário
CRIA_IND_ARM	Cria índice .IND buscando um armário inteiro
DECIDE	Organiza a decisão de um raciocínio
DIRE_CUR	Calcula as direções entre pontos de uma curva.
DO_PAUSA	Do_Events parando quando o botão pausa da simulação estiver press.
EDITA_ARMARIO	Edita a lista de armários do kards
EDITA_ARQ_GRID	Edita um arquivo texto por um grid
EDITA_ARQ_TEXTO	Edita um arquivo texto.
EDITA_CAMPO_GRID	Edita campo de grid sem limite de linhas
EDITA_CAMPO_NORMAL	Edita uma campo normal de uma ficha
EDITA_FICHA	Macro para editar ficha no modo gerar
EDITA_FN	Edita Funções Nebulosas
EDITA_FUZZY	Editor de Funções Nebulosas
EDITA_GAVETA	Macro para GERAR GAVETA
EDITA_LAYOUT	Edita Lay-Out de um ficha tipo REDE de ficha.

EDITA_MET	Edita métodos em uma planilha de rede
EDITA_PASTA	Macro para GERAR PASTA
EDITA_REDES	Define as redes de raciocínio de um armárioi
EDITA_SIMULADOR	Edita uma planta de simulação
EDITA_TEXTO_GRID	Edita um texto como um grid
EDITA_TIPO_FICHA	Edita tipo de ficha.
ENCAD_TEXTS	Encadeia textos processados a partir de máscaras
END_DOC	Fim de documento para LPRINT
ESPECIALIZA	Cria Armários PADR e ACOE no diret
EXEC_FILHOS_COL	Executa métodos coluna dos filhos
EXEC_SERVER	Executa macro no Kards servidor
EXEC_SERVER_2	EXEC_SERVER sem dar mensagem.
EXPORTA_FICHA	Exporta o conteúdo da ficha do usuário atual para o arquivo FICHA.TMP
FAZ_BUSCA	Chamada pelo formulário de buscas para realizar a busca
FIC_ANTERIOR	Vai para a ficha anterior
FIC_SEGUINTE	Vai para a ficha seguinte
FICHA_ATUAL	Devolve uma string com Armário, Gaveta, Ficha e Usuário
FINALIZA_IND	Grava os dados do campo para o respectivo arquivo .IND.
FN	Le função nebulosa do armario corrente
FORMATA_COL	Formata coluna de planilha para modo grafico
FORMATA_T_COL	Formata coluna para gráfico
G_TEXT	Grava o texto na ficha
GATILHO	Escreve, no Blackboard, a lista de hipóteses a testar
GERME_TEND_DIR	Tendencia de direções
GET_MARCA_EDITEX	Volta texto marcado em Edita Textos
GRAF_COL	Ajusta uma coluna de uma sentença para gráficos
GRAF_T_COL	Mostra grafico em planilha texto
GRAFIC	Aciona o Editor Gráfico de Redes QUEST
GRAVA_CLIP_FIC	Grava conteúdo do ClipBoard na Ficha Atual
GRAVA_DADO	Grava dado em ficha da pasta atual
GRAVA_DADOS_2	Macro para guardar dados no Manus_Fic_L
GRAVA_DICIONARIO	Grava dicionário para Jargão a partir do armário Dicionários
HELP_BAS	Procura por uma função nos fontes em BASIC
HELP_BAS_ARQ	Procura por uma função em um arquivo BASIC
HELP_FUN	Implementa Help Contexto Sensitivo
HELP_MOS	Macro utilizada pelo primeiro botão de HELP
HELP_MOS_FUN	Utilizado pelo Help para mostrar função escolhida por Tópicos
HELP_MOS_FUN_P	Utilizado pelo Help para mostrar função a procurar
HELP_MOS_TEXTO	Macro básica para mostrar HELP
HELP_ROTEIRO	Aciona Help Local de Usuário - ainda não implementada
HIPOT	Atualiza hipótese no Blackboard
IMP_CLASSE_JAR	Importa a lista das palavras de uma classe do

	dicionário Jargão para a ficha atual
IMPORTA_ARQUIVO	Importa Arquivo como Conteúdo de Ficha
IMPORTA_DADO	Le dado de uma ficha da pasta atual
IMPORTA_FICHA	Ativa o módulo FICHA do Kards
IMPORTA_TABELA	Importa uma tabela texto para um armário do Kards
IMPRIME_FICHA_GRID	Imprime uma ficha grid
INC_ESTAT	Usado para fazer estatística.
INCORP_JAR_FIC	Incorpora arquivo do Jargão em ficha Kards
INDICEJ	Manuseia Indice Jargão para um Armário Kards
INDICES	Manuseia um Indice para o Armário Kards
INICIA_IND	Guarda valor inicial do campo para criação de .IND.
INICIA_LISTA	Zera a lista de agentes executados
INPUT_CAMPO	Pede um campo ao usuário
INPUT_COND	Aceita dado satisfazendo condição especificada
INPUT_LISTA	Força entrada ser elemento de uma lista
JARGAO	Ativa o JARGAO.
LE_CAMPO	Método básico para leitura de campo
LE_CAMPO_GRID	Edita uma ficha tipo planilha texto
LE_CLASSE_JAR	Cria uma lista com as palavras de um dicionário Jargão que pertençam a uma classe determinada
LE_COL_FICHA	Le os dados de uma ficha do armario atual
LE_FAIXA	Aceita dado dentro de faixa expecificada de valores
LE_MASC	Le ORD das Macros de Buscas e Textos a Processar
LIMPA_QUEST	Inicializa os nodos do QUEST
M_HELP1	Utilizada pelo Sistema HELP para mostrar Texto de Ajuda
M_HELP2	Utilizada pelo Sistema Help para Mostra Explicação sobre Função
M_HELP3	Utilizada pelo Sistema Help para Busca Função e Explicar
MANUS_ARMARIO	Manuseio de armário
MANUS_FIC	Manuseio de ficha
MANUS_FICHA_L	Manuseia Ficha em Formato Lista
MANUS_GAV	Manuseio de gaveta
MAX	Calcula o máximo de dois números
MENSAGEM	Le uma mensagem em gaveta especificada do armário de mensagens
MENU_ORD	Mostra uma lista para seleção com uma sentença .ORD
METODOS_DO_NO	Retorna com os possíveis métodos para o nó
MIN	Calcula mínimo
MOD_FICHA	Macro para entrar no manuseio de fichas
MOD_GERAR	Entra no modo Gerar
MONTA_DEF	Atualiza arquivo .DEF do armário atual com os tamanhos de ficha
MONTA_FIC_LIS	Para montar ficha tipo lista
MONTA_LIS	Escreve dados de uma ficha em uma lista
MOSTRA_CONF_NUM	Mostra uma variável lingüística
MOSTRA_ESCOLHE_LISTA	Mostra escolhe_lista com os dados da ficha selecionada.

MOSTRA_FIC_BUS	Mostra ficha para a função Procura
MOSTRA_FICHA_LP	Mostra uma ficha tipo Lista e/ou Planilha
MOSTRA_FICHA_P	Método Padrão para exibição de ficha
MOSTRA_FICHA_P	Mostra uma ficha na forma padrão
MOSTRA_FUZZY	Mostra funções fuzzy e coloca escolhida no ClipBoard
MOSTRA_SIMU	Mostra BMP do simulador com controles e dados
MOSTRAPIC	Mostra Figura BMP
MOVE_FICHA	Desloca as fichas de uma gaveta
MOVE_FICHA_1	Desloca as fichas de uma gaveta
MUDA_TAM_FIC	Muda o tamanho de uma ficha, atualizando gavetas
NOME_GAV	Retorna o nome da gaveta atual
NOME_GAVE	Retorna o nome de uma gaveta dada sua sigla
NOME_USUARIO	Fornece o nome do usuário da pasta atual
NOVO_USU	Cria novo usuário
OBSERV	Macro utilizada para editar observações no Manus_Fich_L
P_MASCS	Macro Utilizada para programar o Botão Clip
P_TEXT	MACRO ASSOCIADA AO BOTÃO P_TEXTO
PEDE_DADOS	Pede dados para uma ficha
PEG_MASC_TEXT	Le um formulário para a macro Encad_Text
PEGA_AND	Para inserir E no formulário de buscas
PEGA_OR	PARA INSERIR AND EM BUSCA
PERCORRE_FIC	Percorre todas as fichas de uma gaveta
PERCORRE_GAV	Percorre as gavetas de um armário, executando uma expressão
PERCORRE_GAVETAS	Percorre Gavetas do Armário Atual
PERCORRE_PASTA	Percorre as Fichas de uma Pasta Executando uma Macro
PERCORRE_USU	Percorre os usuário de uma gaveta, executando uma expressão
PREPARA_TEXTO	Prepara Textos a Partir de Máscaras
PROCURA	Procura ficha por campo ou valor
PROCURA_CAMPO	Procura por um campo em todas as fichas
PROCURA_VALOR	Procura por uma ficha que tenha um determinado valor
PROXIMO_CAMPO	Vai para o próximo campo de edição
PROXIMO_ITEM	Seleciona o próximo item da ficha a manusear
REAL_BUS	Realiza uma Busca em Gaveta Especificada
REPROCESSA	Reprocessa métodos Ini e Fin de uma ficha para o armário inteiro
RETORNA_DO_FICHA	Retorna do menu FICHA para o menu principal
RETORNA_DO_GERAR	Retorna do menu GERAR para o menu principal
RODA	Usada para incorporar dados em fichas de dados temporarios
ROT	Implementa uma ficha de dadoss temporários
SALVA_FICHA_L	Salva uma ficha
SELE_HELP_FIC	Seleciona uma Ficha de Help - Funções HELP
SELE_USU2	Seleciona usuário e edita ficha 1 se usuário novo.
SELECIONA_LISTA	Seleciona itens de uma lista

SENHA	Pede senha
SETA_FICHA	Ajusta o armário, gaveta, ficha e usuário
SETA_VAR_GLOB	Seta um conjunto de variáveis globais
STR_MANUS	Para definir as macros e títulos do formulário de Manus_Ficha_L
SUPERV	Gerencia a chamada de Redes para teste de hipóteses
TREINA_PERCEPTRON	Treina rede de perceptrons
USU_ANTERIOR	Vai para o próximo anterior
USU_SEGUINTE	Vai para o usuário anterior
VALOR_COL	Devolve o valor de uma coluna, de um dado campo
VALOR_T_COL	Retorna o valor de uma coluna