REDES NEURAIS ARTIFICIAIS EM SISTEMAS CRÍTICOS QUANTO À SEGURANÇA: UMA ABORDAGEM PARA CERTIFICAÇÃO

Cesar Augusto Comerlato

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO

Aprovada por:

Prof. Luis Alfredo Vidal de Carvalho

Prof. Roberto Schirru, D.Sc.

Prof. Aquilino Senra Martinez, D.Sc.

Prof. Luis Pereira Calôba, Dr.Ing.

Prof. Felix Mora Camino, Dr.Ing.

Dr. Marco Antônio Bayout Alvarenga, D.Sc.

RIO DE JANEIRO, RJ - BRASIL

MARÇO DE 2002

COMERLATO, CESAR AUGUSTO

Redes Neurais Artificiais em Sistemas Críticos quanto à Segurança: Uma Abordagem para Certificação [Rio de Janeiro] 2002

IX, 248 p. 29,7 cm (COPPE/UFRJ, D.Sc., Engenharia de Sistemas e Computação, 2002)

Tese - Universidade Federal do Rio de Janeiro, COPPE

1 - Sistemas Críticos quanto à Segurança

2 - Redes Neurais Artificiais

I. COPPE/UFRJ II. Título (série)

Para a minha querida Lela

Agradecimentos

Aos meus orientadores Prof. Luis Alfredo e Prof. Schirru pela amizade, paciência, incentivo e apoio ao longo deste trabalho.

Aos membros da banca examinadora por sua participação e contribuições.

Aos meus colegas do Laboratório de Monitoração de Processos, pelo apoio, incentivo e compreensão para a realização deste trabalho.

À minha família, especialmente às minhas irmãs Lota e Viki, pela paciência e suporte ao longo deste trabalho.

Ao Prof. Felix Mora Camino por ter me recebido em Toulouse.

Ao Carlão pela sua disponibilidade em discutir idéias e trocar experiências.

Aos colegas da Coppe Sistemas sempre prestativos em atender as minhas solicitações com presteza e eficiência.

À CAPES pela concessão do doutorado sanduíche.

A todos, que de uma ou outra forma, me auxiliaram na conclusão deste trabalho.

A Deus, Criador de tudo, pela minha energia, disposição e saúde.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Doutor em Ciências (D.Sc.)

REDES NEURAIS ARTIFICIAIS EM SISTEMAS CRÍTICOS QUANTO À SEGURANÇA: UMA ABORDAGEM PARA CERTIFICAÇÃO

Cesar Augusto Comerlato

Março/2002

Orientadores: Luis Alfredo Vidal de Carvalho

Roberto Schirru

Programa: Engenharia de Sistemas e Computação

As Redes Neurais Artificiais (RNA) são sistemas indutivos motivados biologicamente que têm sido usados, com bons resultados, em vários domínios de aplicação. Sistemas Críticos quanto à Segurança (SCS) são sistemas nos quais falhas no seu funcionamento podem trazer graves consequências. A abordagem de problemas através de Redes Neurais Artificiais é particularmente apropriada para aplicações muito complexas, onde é impossível ou impraticável o uso de soluções algorítmicas. O emprego de RNA em SCS é ainda uma área pouco explorada em virtude da falta de uma abordagem para a certificação de tais sistemas. Apresenta-se neste trabalho um conjunto de diretivas básicas que objetivam permitir a utilização de RNA em SCS, avaliando os aspectos de segurança de software envolvidos neste processo, de forma a permitir uma base para a certificação de um SCS.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Doctor of Science (D.Sc.)

ARTIFICIAL NEURAL NETWORKS IN SAFETY CRITICAL SYSTEMS: A CERTIFICATION APPROACH

Cesar Augusto Comerlato

March/2002

Advisors: Luis Alfredo Vidal de Carvalho

Roberto Schirru

Department: Computer Systems Engineering

Artificial Neural Networks (ANN) are biologically motivated inductive systems that have been successfully applied in many fields. Safety Critical Systems (SCS) are systems in which failures may cause catastrophic consequences. The use of Artificial Neural Networks is particularly appropriated in complex applications, as it would be impossible or impractical to employ algorithmic solutions. The application of ANN in SCS is currently quite unexplored because of the lack of an approach to certify such systems. In this work, some basic directives aiming the use of ANN in SCS are introduced, taking into account the software safety aspects involved in order to give a background for a SCS certification.

ÍNDICE

Capítulo 1

1 Introdução

A tendência ao uso de módulos de software que empregam técnicas de Inteligência Artificial (IA) em Sistemas Críticos quanto à Segurança (SCS) tem aumentado nos últimos anos, pois determinados tipos de problemas, principalmente os que apresentam alta complexidade, só podem ser abordados, na prática, através dessas técnicas. O avanço das tecnologias de software e hardware (capacidade de processamento) são outros fatores contribuintes dessa tendência. Sistemas críticos quanto à segurança são sistemas nos quais falhas no seu funcionamento podem trazer graves consequências, tais como a perda de vidas humanas, danos ao meio ambiente ou grandes prejuízos econômicos. Sistemas das áreas de geração de energia nuclear e aviação, civil e militar, são alguns exemplos de domínios de aplicação desses SCS.

A preocupação com a segurança de um SCS é diretamente proporcional às consequências que um mau funcionamento deste pode ocasionar. Quando estes sistemas tem funções críticas executadas por software, este também é classificado como crítico quanto à segurança. *Funções críticas quanto à segurança* são aquelas funções do sistema onde a operação correta, a operação incorreta (incluindo a operação correta no tempo errado) ou a falta de operação podem contribuir para colocar o sistema num estado perigoso. Assim, *funções de software críticas quanto à segurança* são aquelas funções de software que podem, direta ou indiretamente, associadas ao comportamento de outros componentes do sistema ou condições ambientais, contribuir para a colocar o sistema num estado perigoso (Leveson, 1995). *Estado perigoso* (*hazard state*), de um sistema é a situação na qual esse sistema torna-se um perigo real para os seres humanos

ou meio ambiente. Portanto, *segurança de sistema de software* implica na execução do software dentro de um contexto do sistema sem que esse software contribua para colocar o sistema em estados perigosos. Assim, *software crítico quanto à segurança* é todo e qualquer software que pode, direta ou indiretamente, contribuir para a ocorrência de um estado perigoso do sistema.

Para que esses SCS possam entrar em operação eles devem ser submetidos ao escrutínio de um processo de certificação (licenciamento, homologação) de um órgão independente, geralmente governamental. O processo de certificação baseia-se, em geral, em algum tipo de padrão (norma ou conjunto de diretrizes) que estabelece as atividades mínimas que devem ser executadas ao longo do processo de desenvolvimento do sistema de software. Para SCS de áreas como a nuclear esses padrões são compulsórios e, maiores são as exigências quanto maior for o nível de criticalidade do sistema, ou seja, quanto maior a possibilidade de danos que uma falha do sistema acarrete, maiores as exigências estabelecidas pelo padrão. As atividades que devem ser executadas englobam, além da confecção do próprio sistema de software, outras tais como Verificação e Validação (V&V), Análise de Perigos, Análise de Segurança, Gerência da Configuração e Controle da Qualidade. Estas atividades são desenvolvidas por grupos que trabalham sob gerências independentes e em paralelo com o desenvolvimento do sistema de software. Durante o processo de certificação todos os resultados dessas atividades vão ser utilizados pelo órgão certificador para avaliar a conformidade segundo o padrão, ou conjunto de padrões, determinado. Mas, a conformidade não é, por si só, condição suficiente para garantir a certificabilidade do sistema. A avaliação do órgão certificador vai muito além dessa conformidade com o padrão recomendado. Vários outros aspectos do próprio sistema são também

meticulosamente avaliados, tais como as soluções empregadas para resolver problemas apontados na análise de segurança e as suas respectivas argumentações justificadas.

Para o software convencional este processo está bem consolidado, e tem sido empregado com relativo sucesso na maioria dos sistemas desenvolvidos atualmente, principalmente nas áreas nuclear e de aviação, dispondo de várias, e diferentes, metodologias e ferramentas.

Por outro lado, para o *software não convencional*, não existe ainda uma infra-estrutura de desenvolvimento e certificação semelhante aquela para software convencional. Por software não convencional estamos nos referindo neste trabalho às técnicas empregadas na programação de software baseadas em diferentes paradigmas da Inteligência Artificial, técnicas essas que procuram reproduzir a inteligência humana de alguma forma. Sistemas que empregam redes neurais artificiais, lógica nebulosa, algoritmos genéticos, bases de conhecimento com regras de produção (chamados de sistemas especialistas), quadros (*frames*) ou combinações destas, são exemplos dessas técnicas de IA. O software que emprega essas técnicas não dispõe de normas e práticas recomendadas na literatura e na indústria, apenas algumas poucas propostas. Uma exceção a esta colocação diz respeito aos sistemas especialistas que apresentam uma extensa bibliografia sobre o assunto na literatura. No tocante a aplicação de redes neurais artificiais (RNA) em SCS, enfoque básico deste trabalho, a certificação de sistemas que empregam essa técnica é atualmente um extenso e potencial campo de pesquisa, sendo que pouco existe na literatura que trate desse problema.

Portanto, para que módulos de software que empregam técnicas de IA possam ser usados em SCS é necessário que eles sejam certificados. Para que esta certificação seja possível eles precisam ter um processo de desenvolvimento sistemático e bem definido, de forma que as outras atividades paralelas e necessárias possam também ser

estabelecidas. Tal estrutura de desenvolvimento permitirá então que esses módulos de software sejam submetidos a um processo de avaliação para sua certificação. A determinação de um ciclo de vida apropriado para este tipo de software, identificado segundo uma estrutura de diretrizes permitirá a aplicação de um processo de V&V desenvolvido segundo as necessidades de desempenho e segurança da aplicação, seguindo aquela estrutura de diretrizes. Isto permitirá que estes módulos de software contendo RNA possam ser depois incorporados ao processo de certificação global do SCS.

O excelente trabalho desenvolvido em conjunto pela USNRC e pelo EPRI *Guidelines for the Verification and Validation of Expert System and Conventional Software* (NUREG/CR-6316, 1995a) abordou este problema, mas concentrou-se principalmente no estudo do problema da V&V de sistemas especialistas que utilizam regras de produção e bases de conhecimento. Além desse tipo de sistema especialista, do qual trata a maior parte do trabalho citado, foram abordados também, ainda que muito superficialmente, os paradigmas de orientação a objetos e o de quadros (*frames*). Com relação à lógica nebulosa, RNA e algoritmos genéticos (AG) praticamente nada foi tratado, limitando-se a considerar que para estas duas últimas técnicas (RNA e AG) os mesmos princípios eram aplicáveis por extensão. Na nossa opinião esta colocação não é verdadeira, como veremos nos próximos capítulos. Os autores do trabalho supracitado identificaram que eram necessárias técnicas de análise (e ferramentas) particulares para avaliar as bases de conhecimento de sistemas especialistas baseados em regras de produção. Estas bases de conhecimento são composições de software não convencional necessitando portanto de técnicas específicas para a sua avaliação.

Redes neurais artificiais, ou computação neural, são um tipo de programação indutiva onde uma tarefa é executada através de um modelo que é treinado usando

dados que representam esta tarefa. As RNA dispõem de um ambiente de processamento paralelo capaz de aprender para resolver problemas de certos domínios de aplicação onde são impossíveis, ou impraticáveis, o uso de soluções algorítmicas eficientes. Tais aplicações são tipicamente complexas, pouco compreensíveis e imprecisas. O reconhecimento de padrões e, a modelagem e controle de sistemas não lineares são domínios onde as RNA e outras técnicas estatísticas superam os métodos algorítmicos. Portanto, a computação neural representa uma abordagem muito diferente da abordagem de desenvolvimento do software convencional, onde soluções algorítmicas podem ser especificadas antes de se escrever o software, pois no caso das RNA o desempenho e a precisão do modelo não podem ser predeterminados. A arquitetura de uma RNA consiste de várias unidades (*nodos*) de processamento possuindo capacidade computacional muito simples, cujos valores de entrada podem ser valores escalares discretos ou contínuos. A conectividade entre estas unidades é unidirecional e geralmente extremamente complexa. O modelo implementado pelas RNA é singular devido a uma característica particular e única: Apesar das RNA serem implementadas através de software convencional o conhecimento contido nelas não é diretamente explicitável, este conhecimento está implícito na topologia das interconexões e nos pesos das conexões da rede. Isto se deve ao fato de como as RNA são treinadas pois os parâmetros do modelo são a única parte do modelo específicas da aplicação. A interpretação humana desses parâmetros é difícil e imprecisa quando comparada à interpretação de código fonte algorítmico de alto nível do software convencional. Portanto, a singularidade das RNA, em relação ao processo de avaliação convencional, reside no fato de que a programação indutiva, ou seja o aprendizado da rede, não envolve, a princípio, nenhum desenvolvimento de software, já que o software para executar e treinar o modelo é independente da aplicação e dos dados desta.

Tal como para as bases de conhecimento de sistemas especialistas estudadas no trabalho citado (NUREG/CR-6316, 1995a), as redes neurais artificiais também apresentam características particulares. Mas, tais características são próprias das RNA e não podem ser avaliadas a partir da extensão dos princípios de análise das técnicas empregadas para as bases de conhecimento de sistemas especialistas, já que a natureza dos dados dessas duas técnicas de IA é radicalmente diferente. Portanto, é necessário que sejam identificadas e compreendidas as peculiaridades das RNA, consideradas como software não convencional no seu ciclo de vida, de forma a permitir que estas particularidades sejam adequadamente abordadas e tratadas. Dessa forma o desenvolvimento de módulos de software que empregam técnicas de RNA são suscetíveis a um processo de análise sistemática permitindo assim que uma avaliação possa ser promovida pelo órgão certificador. Apesar de haver pouco desenvolvimento de software na confecção de uma RNA propriamente dita, existem outras tarefas pertinentes a este processo de confecção que devem ser executadas e que não tem equivalência real na engenharia de software convencional:

- A coleta de dados é extremamente importante pois a quantidade e a qualidade (relevância) destes é fundamental.
- Um pré-processamento dos dados é geralmente necessário pois dificilmente o melhor desempenho será obtido com os dados usados na sua forma original.
- O treinamento da rede deve ser cuidadosamente acompanhado e monitorado, garantindo com isso, que uma boa solução seja alcançada.
- O foco da avaliação do desempenho deve recair sobre a capacidade de generalização para dados novos, dados estes não vistos no treinamento, ao invés daqueles para os quais a rede foi treinada.

Como veremos, devido às características particulares dos SCS nem todos os tipos de módulos de RNA podem ser empregadas já que aqueles impõem algumas restrições ao uso destas. Por exemplo, módulos de software adaptativo, não podem, a princípio ser utilizados em SCS pois eles se auto modificam impedindo com isso que seja aplicado um processo de V&V.

Como esses módulos de RNA serão incorporados a SCS é necessário também que eles sejam submetidos a uma análise de vulnerabilidade (perigo). A análise de vulnerabilidade é a investigação metódica do sistema, em parte ou como um todo, para encontrar vulnerabilidades potenciais. Uma vulnerabilidade é um estado ou conjunto de condições em um sistema que em conjunto com outras condições ambientais do sistema, vai levar a um acidente. As vulnerabilidades de sistema relacionadas ao software são denominados vulnerabilidades de software. A negação dessas vulnerabilidades de software são exigências de segurança de software que estão relacionadas à segurança dos requisitos de funcionalidade. As restrições de segurança do software (Isaksen, 1996) estão relacionadas aos limites de operação do sistema. Nesta análise são identificados quais e como podem falhar os módulos que geram as entradas para os módulos de RNA, no nosso caso. Além disso, as possíveis consequências desses sinais errados no desempenho dos módulos de RNA devem também ser analisados no contexto geral do sistema. A análise de vulnerabilidade é uma das tarefas da análise de segurança, onde esta vai estabelecer como devem ser tratados os perigos potenciais do sistema e no caso das vulnerabilidades de software como estas podem ser evitadas ou mitigadas, e qual o seu impacto na segurança do sistema como um todo. Portanto, é importante que uma avaliação da confiabilidade funcional da RNA possa ser estabelecida, a partir de um processo de avaliação, já que a operação de aplicações

potenciais terá que ser garantida durante algum período do seu tempo de vida. Isto é especialmente verdade para sistemas críticos quanto à segurança.

Nossa proposta consiste na identificação de um ciclo de vida e de um processo de V&V, abrangendo aspectos gerenciais e técnicos, e principalmente de segurança, envolvidos na confecção de módulos de RNA para SCS. Serão utilizados como base os trabalhos estudados, as recomendações do trabalho (NUREG/CR-6316, 1995a), e a experiência adquirida em um caso de estudo de RNA para SCS. Em resumo, uma compilação dos conceitos técnicos que formam o substrato das atividades envolvidas no processo de desenvolvimento, V&V e análise de vulnerabilidade, e todos os possíveis procedimentos relacionados que devem garantir os comportamentos e o desempenho das funções especificadas nos requisitos para o sistema sejam atendidos durante o desenvolvimento (verificação) e a implementação final (validação) de uma RNA para SCS.

No capítulo 2 são apresentados os conceitos básicos dos tópicos de interesse para análise introduzindo as Redes Neurais Artificiais (RNA), definições e conceitos sobre Sistemas Críticos quanto à Segurança (SCS) e os conceitos relacionados à certificação e o seu processo de execução.

No capítulo 3 são estudados e discutidos os trabalhos encontrados na literatura que tratam da problemática envolvida no emprego de RNA para sistemas com alto nível de integridade relacionados à segurança.

No capítulo 4 são compilados conceito básicos acerca dos trabalhos estudados, das recomendações em (NUREG/CR-6316, 1995a), da experiência adquirida em um estudo de caso para a proposta de um ciclo de vida evolucionário de segurança - CIVES - para o desenvolvimento de RNA para SCS. São também apresentados conceitos

básicos para a aplicação do CIVES em cada uma das etapas de desenvolvimento do ciclo de vida para RNA.

No capítulo 5 são estudados vários métodos de análise de segurança de software propostos na literatura objetivando sua caracterização de forma a identificar os mais apropriados para o emprego nas etapas de análise de segurança e vulnerabilidade do CIVES.

No capítulo 6 são apresentadas as conclusões e as possíveis perspectivas de continuação dos trabalhos e sugestões para trabalhos futuros.

Capítulo 2

2 Redes Neurais Artificiais e Sistemas Críticos quanto à Segurança

As redes neurais artificiais (RNA) (Bishop, 1995b, Fu, 1994, Haykin, 1994, Rojas, 1996) dispõem de um ambiente de processamento paralelo capaz de aprender para resolver problemas de certos domínios de aplicação tais como o reconhecimento de padrões e o controle de sistemas dinâmicos. Entretanto, elas não são adequadas para problemas normalmente associados com a lógica convencional baseada em sistemas computacionais, tais como a rápida execução de operações matemáticas.

A arquitetura de uma RNA consiste de várias unidades (*nodos*) de processamento possuindo capacidade computacional muito simples, cujos valores de entrada podem ser valores escalares discretos ou contínuos. A conectividade entre estas unidades é unidirecional e geralmente extremamente complexa. A sua natureza permite uma taxonomia de redes neurais artificiais conforme é mostrado na Figura 2.0. Nas RNA *feedforward*, a saída de uma unidade não tem efeito direto ou indireto na sua operação (desconsiderando-se a retroalimentação da RNA no seu contexto de operação), i. e. não existem *loops*. Quando não existe tal restrição as RNA são classificadas como *feedback* ou *recorrentes*. Associado a cada conexão entre duas unidades existe um valor numérico chamado de *peso* que modifica o valor escalar de saída introduzido na unidade de entrada. Em geral esses pesos são os únicos parâmetros que podem ser modificados em uma rede neural para determinar a sua operação. Entretanto, a natureza básica da sua operação é influenciada pela sequência na qual as unidades são atualizadas, i. e. quando elas emitem novos valores com base nas suas

entradas correntes. Três regras de atualização podem ser identificadas: *síncrona*, *sequencial* e *assíncrona.*

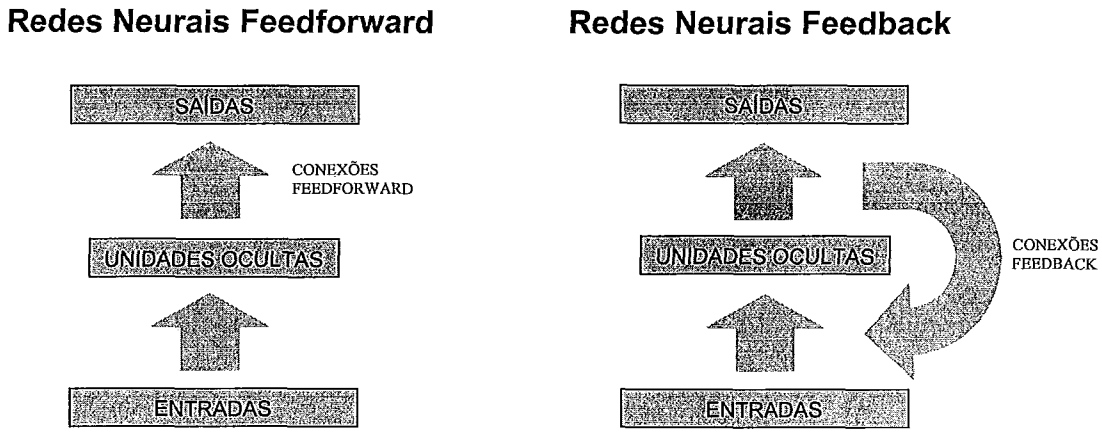

Figura 2.0 - Conectividade em redes neurais artificiais.

Nas RNA síncronas, todas as unidades são atualizadas simultaneamente. Em uma RNA sequencial o processo é similar, exceto que as unidades são atualizadas uma de cada vez numa ordem fixa. E finalmente, se as unidades são atualizadas em uma base individual, sem uma ordem fixa, então sua operação é dita assíncrona. A função computada por uma unidade individual pode ser definida como:

$$saída = f_{ativação}\,[g(\underline{w},\underline{x})]$$

onde os componentes do vetor $\underline{w}$ são os pesos das suas conexões de entrada, e os componentes de $\underline{x}$ são os valores de entrada aplicados a cada conexão (Figura 2.1). Repare que estas poderiam ser as saídas de outras unidades de processamento da RNA. A função $f_{ativação}$ que modifica o resultado da interação conjunta das entradas g e dos pesos é conhecida por *função de ativação* ou *função limítrofe*. Esses termos referem-se

ao seu papel de limitar a magnitude absoluta de saída de uma unidade. A *ativação* de uma unidade é o resultado de *g*.

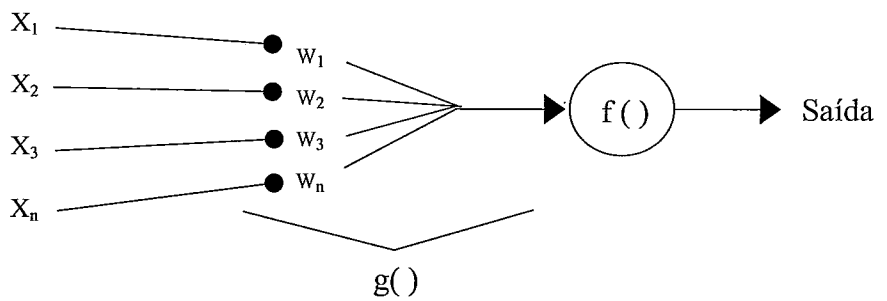

Figura 2.1 – Diagrama funcional de uma unidade (nodo) em uma rede neural artificial.

Aprendizado é o processo através do qual os parâmetros livres de uma RNA são escolhidos de forma que a operação da RNA resolva o problema desejado. A funcionalidade de uma RNA não é programada, ao invés disso um *algoritmo de aprendizado* modifica o seu comportamento segundo o seu ambiente e qualquer orientação externa que possa ser fornecida. Três estilos diferentes de algoritmos de aprendizado podem ser identificados dependendo do tipo de informação disponível.

- No aprendizado *supervisionado*, a saída correta é conhecida para um conjunto de entradas.
- Algoritmos de aprendizado de *reforço* só tem acesso a valores escalares indicando o grau de correção de saída da RNA.
- Se nenhuma orientação externa é fornecida, o aprendizado é chamado de *não supervisionado*.

Duas importantes propriedades das RNA são a *generalização* e a *distribuição*:

- A *generalização* refere-se à capacidade da RNA de produzir uma resposta razoável (saída) para entradas que ela não conheceu durante o treinamento. Por exemplo, se uma RNA é treinada para reconhecer uma assinatura e recebe uma entrada distorcida por ruído ela deve ainda ser capaz de lembrar ou achar a saída correta.

- Durante o aprendizado, a apresentação de qualquer entrada pode resultar em uma modificação potencial de qualquer parâmetro da RNA. Geralmente isto é conhecido por *distribuição da informação*. Durante a operação, todos os elementos em uma RNA estão envolvidos no processamento de uma entrada, e isso tem sido descrito como *distribuição do processamento*.

O enfoque principal do nosso trabalho são as redes neurais artificiais do tipo *feedforward* com múltiplas camadas com aprendizado supervisionado usando a regra de retropropagação do erro (*backpropagation*), também chamadas de *redes backpropagation*. Trata-se de um método de gradiente descendente cujo objetivo é minimizar o erro quadrático total da saída computado pela rede. A natureza genérica do método de treinamento por retropropagação implica que uma rede *backpropagation* pode ser usada para resolver problemas de várias áreas. São por exemplo, aplicações complexas que não dispõem de soluções algorítmicas para a solução de problemas que requerem o mapeamento de um dado conjunto de entradas em um dado conjunto de respostas de saída (ou seja que usam treinamento supervisionado) tem nesse método uma solução.

Um importante teorema que iluminou a capacidade das redes neurais de múltiplas camadas é o teorema de Kolmogorov (Hecht-Nielsen, 1990, Haykin, 1996,

Rojas, 1996, Nascimento, 2000). Kolmogorov provou que funções contínuas de n argumentos podem sempre ser representadas usando-se uma composição finita de soma de funções de um único argumento. É sem dúvida um teorema surpreendente, já que implicitamente diz que a adição é única função com mais de um argumento necessária para representar funções contínuas com *qualquer* número de argumentos. Do ponto de vista de redes de funções, o teorema de Kolmogorov pode ser interpretado como a declaração de que qualquer função contínua de n variáveis pode ser representada por um rede finita de funções de uma única variável, onde a adição é usada como a única função com vários argumentos. O teorema de Kolmogorov é importante para a área das RNA pois ele estabelece que qualquer função contínua pode ser reproduzida exatamente por uma rede finita de unidades computacionais, para a qual as funções primitivas necessárias para cada nodo *existem*. Mas ele não nos diz como encontrar estas funções (Fu, 1994). Cibenko, Funahashi e Hornik em Haykin (1996) apresentam uma abordagem para a identificação do número de nodos ocultos que seriam necessários para uma rede *MLP* (*Multi Layer Perceptron*) aproximar de maneira uniforme. O mesmo foi feito para redes *RBF* (*Radial Basis Function*) por Hartman e Park & Sandberg em Haykin (1996). Entretanto, se queremos apenas aproximar funções, não exigindo reproduções exatas, mas somente um erro de aproximação limitado, procuramos então pela melhor aproximação para uma dada função f. Esta é exatamente a abordagem que empregamos com as *redes backpropagation* e qualquer outro tipo de rede de mapeamento.

Apresentamos no Apêndice A a teoria das redes neurais Perceptron de Múltiplas Camadas, também conhecidas pela sigla *MLP* (*Multi Layer Perceptron*). Este tipo de rede, e suas variações, é empregado na maioria dos problemas de modelagem de reconhecimento de padrões e, monitoração e controle de sistemas.

2.1 Lógica Indutiva e Lógica Dedutiva

A natureza da programação do software convencional através da construção de uma estrutura elaborada de símbolos em uma linguagem formalmente definida é muito similar à prova matemática. Portanto, a dedução é a base subjacente da estrutura de produção do software convencional. Assim, dedução, numa definição simples significa raciocinar do geral para o particular e neste contexto específico, referimo-nos à prática de gerar implementações especificas a partir de especificações gerais, i. e. a noção clássica resumida de "programação" (Partridge, 1996). A lógica indutiva, ou indução, ao contrário da dedução parte do particular para o geral. Um exemplo simples comparando as lógicas dedutiva e indutiva é mostrado na Tabela 2.0 abaixo.

Lógica Dedutiva	**Lógica Indutiva**
premissa: *todos os cisnes são brancos* fato: *X é um cisne*	fato: *P é branco e P é um cisne* fato: *Q é branco e Q é um cisne* fato: *R é branco e R é um cisne*
conclusão: *X é branco*	conclusão: *todos os cisnes são brancos*
Se o axioma e o fato são verdadeiros então a conclusão é verdadeira	Se todos os fatos são verdadeiros então a conclusão pode ainda assim ser falsa

Tabela 2.0 – Contraste entre as lógicas dedutiva e indutiva.

Apesar de não serem totalmente precisas, existe uma série de técnicas baseadas na indução para a produção de implementações que computam quantidades úteis. Implementações geradas indutivamente podem ser, por exemplo, árvores de decisão ou redes neurais. Não se tratam de programas no senso clássico de uma lista de instruções, ainda que uma árvore de decisão possa facilmente ser transformada numa lista de regras condição-ação (regras de produção), a maioria das redes neurais não admite tal

transformação. Uma rede neural pode implementar uma função mas se parece muito pouco com a noção clássica de programa.

O uso destas técnicas de programação indutiva acontece por diversas razões e elas são encontradas sob uma variedade de rótulos tais com "regra de indução", "aprendizado baseado em casos" e "conexionismo". Sob o emblema de "Aprendizado de Máquina", uma das linhas da área de Inteligência Artificial, uma boa quantidade dessas técnicas pode ser encontrada (Russell, 1995).

Comentamos na introdução sobre o excelente trabalho desenvolvido pela USNRC-EPRI (NUREG/CR-6316, 1995a) acerca da verificação e validação de sistemas especialistas e software convencional, onde é dito (páginas 9 e 10) que os resultados são aplicáveis às RNA por extensão (NUREG/CR-6316, 1995e)[1]. Observamos nessa altura que tal afirmação não era verdadeira pois, como vimos acima, a natureza dos sistemas especialistas (e do software convencional em geral) e das redes neurais é radicalmente diferente. Os sistemas especialistas baseiam-se na lógica dedutiva enquanto que as redes neurais tem por base a lógica indutiva. Um exemplo simples dessa diferença pode ser observado em relação a confecção da especificação funcional dos sistemas. Para o software convencional a primeira atividade do ciclo de desenvolvimento é a confecção de uma especificação funcional completa do sistema, enquanto que para as redes neurais (como veremos), este é um processo incremental que só termina ao final do ciclo de desenvolvimento. Podemos ver portanto a diferença entre os princípios básicos dessas duas abordagens de desenvolvimento de software.

Não estamos querendo dizer com isso que nada do que foi desenvolvido pelo trabalho citado se aplica às redes neurais, ao contrário, muito do que foi desenvolvido pode ser aplicado para o caso das redes neurais. Além disso, uma outra observação

acerca dos tipos de dados de RNA e algoritmos genéticos no trabalho (NUREG/CR-6316, 1995a)[2] à página 74, está de acordo com a nossa visão sobre as diferenças existentes entre sistemas especialistas e redes neurais.

Se considerarmos que um módulo de RNA tem uma interface de entrada e uma interface de saída, ou mais ainda, um pré e um pós tratamentos (Figura 2.2), estes com certeza serão implementados através de software convencional. Sob o ponto de vista da segurança, estas interfaces podem funcionar também como barreiras de segurança, i. e. analisando os dados que chegam e que saem evitando que dados desconhecidos (fora do envelope de operação da rede) sejam inseridos na rede gerando respostas incorretas (desconhecidas). Isto poderia contribuir para a ocorrência de estados perigosos que poderiam levar o SCS, onde a RNA está inserida, a um acidente.

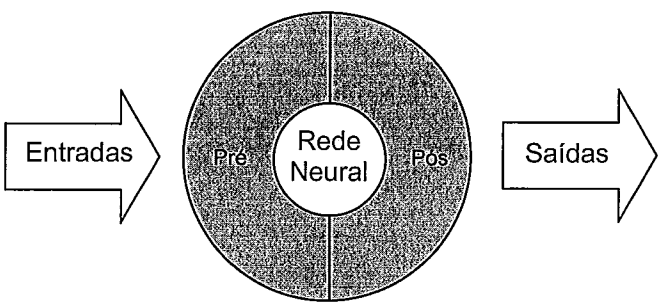

Figura 2.2 – Uma rede neural com pré e pós processamento.

A detecção de novidades (dados não vistos durante o treinamento) é um dos aspectos que devem ser considerados em um pré-tratamento dos dados de entrada da rede neural. Módulos de software que antecedem a RNA na sequência de

[1] *"To the extent Neural Nets or Genetic Algorithms involve declarative representation of applications knowledge, corresponding to an AI System knowledge base, these guidelines apply more directly."*
[2] *"However, these guidelines do not specifically provide for testing the data structures and subprocesses unique to these types of systems." (referindo-se às redes neurais e algoritmos genéticos)*

processamento podem gerar dados a partir de estados atípicos do sistema, que podem ser desconhecidos para a RNA.

A detecção de novidades testa a completeza do modelo de IA, e com testes estatísticos a completeza do modelo pode ser garantida para os dados de projeto, e a novidade de dados futuros pode ser testada em relação a esse mesmos dados, resultando em um problema bem definido suscetível a soluções teoricamente satisfatórias (Lisboa, 2001). Mas, estas soluções satisfatórias, dependendo do nível de integridade de segurança estabelecido para o sistema, podem requerer algum tipo de medida de confiabilidade, ou seja, uma garantia de correção da resposta.

Tal situação deixa transparecer a necessidade de se ter algum mecanismo que permita identificar entradas novas (desconhecidas, não vistas durante o treinamento) que possam contribuir para colocar o sistema como um todo numa condição ou estado perigoso. Portanto, uma análise de vulnerabilidade de software deverá identificar as vulnerabilidades (*hazards*) que possam apresentar entradas incorretas para a RNA. Além disso, a possibilidade de vulnerabilidades resultantes de saídas incorretas da rede neural devem também ser analisadas de forma a se identificar quais seriam as consequências dessas saídas, ou de que forma elas contribuiriam, no contexto de segurança global do sistema.

Uma dificuldade ainda não totalmente solucionada, em relação às RNA, é até que ponto os dados de treinamento disponíveis são representativos do espectro de comportamentos esperados do sistema, ou módulo (Lisboa, 2001).

A intrínseca falta de transparência das RNA, aliadas a sua inerente propensão ao super-ajuste (*overfitting*) caso o treinamento não seja cuidadosamente monitorado, são aspectos que merecem grande atenção quando se considera o emprego de RNA em SCS. Estes pontos requerem grande atenção em função do seu aspecto crítico ao longo de

todo o processo de desenvolvimento da RNA. Aspectos como esse devem ser considerados no desenvolvimento do ciclo de vida de segurança, paralelo ao ciclo de vida de desenvolvimento, por se tratarem de pontos fundamentais para a certificação do módulo de RNA.

Podemos observar portanto, que existem vários aspectos que devem ser considerados no emprego de RNA em SCS. Alguns desses aspectos são peculiares ao tipo da aplicação mas, os aspectos relacionados a segurança são comuns a todos os tipos de aplicações em SCS e deve-se portanto prever os meios e medidas necessárias para o tratamento dos requisitos de segurança.

2.2 Sistemas Críticos quanto à Segurança

Segurança (*safety*) é a propriedade que um sistema possui de que ele não vai colocar em perigo a vida humana ou o meio ambiente. Assim, um *sistema relacionado à segurança* (*safety-related system*) pode ser definido como sendo aquele no qual a segurança do equipamento, ou planta, está garantida (Gardiner, 1999). Portanto, podemos definir *sistemas críticos quanto à segurança* (*safety-critical systems*) como sistemas nos quais possíveis falhas de funcionamento podem trazer graves consequências, tais como a perda de vidas humanas, danos ao meio ambiente ou prejuízos econômicos (Moura, 1996). Quando estes sistemas tem funções críticas executadas por software, esse também é classificado como crítico quanto à segurança. *Funções críticas quanto à segurança* (*safety-critical functions*) são aquelas funções do sistema onde a operação correta, a operação incorreta (incluindo a operação correta no tempo errado) ou a falta de operação podem contribuir para colocar o sistema num

estado perigoso. *Funções de software críticas quanto à segurança* (*safety-critical software functions*) são aquelas funções de software que podem, direta ou indiretamente, associadas ao comportamento de outros componentes do sistema ou condições ambientais, contribuir para a colocar o sistema num estado perigoso (Leveson, 1995). *Periculosidade*, ou *estado perigoso* (*hazard*), de um sistema é a situação na qual esse sistema torna-se um perigo real, ou potencial, para as pessoas ou para o ambiente (Storey, 1996). Portanto, *segurança de sistema de software* (*software system safety*) implica na execução do software dentro de um contexto do sistema sem que esse software contribua para colocar o sistema em estados perigosos. Concluindo, *software crítico quanto à segurança* (*safety-critical software*) é todo e qualquer software que pode, direta ou indiretamente, contribuir para a ocorrência de um estado perigoso do sistema.

Em sua grande maioria os sistemas relacionados à segurança são *sistemas de controle* ou *sistemas de proteção*. Sistemas de controle são usados para determinar a operação de algum tipo de equipamento ou planta. Em alguns casos as funções de controle não estão relacionadas à segurança, significando que o sistema controlado não é capaz de ações perigosas, ou porque a segurança está sob a gerência de outro sistema ou subsistema. Em algumas aplicações o sistema de controle pode executar as duas funções: controlar o equipamento e desempenhar as funções de segurança.

Sistemas de proteção utilizam sensores para detectar condições de falha ou anormalidade e gerar saídas para tratamento desses eventos. Estas saídas podem ser informações aos operadores ou sinais que vão comandar equipamentos de proteção. Em muitos casos a ação do sistema de proteção é desligar o equipamento em questão, sendo então conhecidos como sistemas de desligamento (*shutdown systems*).

No caso do sistema ser de controle ou proteção a configuração geral do sistema é similar à mostrada na Figura 2.3(a). O equipamento, ou sistema, com o qual a aplicação está preocupada é chamado de *equipamento sob controle*. Ele vai ter entradas e saídas do ambiente através das quais vai executar as funções implementadas. O equipamento sob controle pode ser um processo completo ou uma linha de produção, tal como uma estação de potência ou planta química, ou ele pode ser um pequeno equipamento como um marca-passo cardíaco artificial ou componente automotivo. O sistema de controle, ou proteção, interage com o equipamento sob controle através de sensores e atuadores que são usados para monitorar e controlar certos equipamentos. A operação do sistema é determinada pelas funções e algoritmos nele implementados. Os sistemas de controle nem sempre estão relacionados à segurança mas os sistemas de proteção, pela sua própria natureza, estão.

Nem todos os sistemas de controle ou proteção são baseados em computadores ou utilizam computadores, pois existem situações onde é mais vantajosa a utilização de sistemas menos complexos e mais baratos.

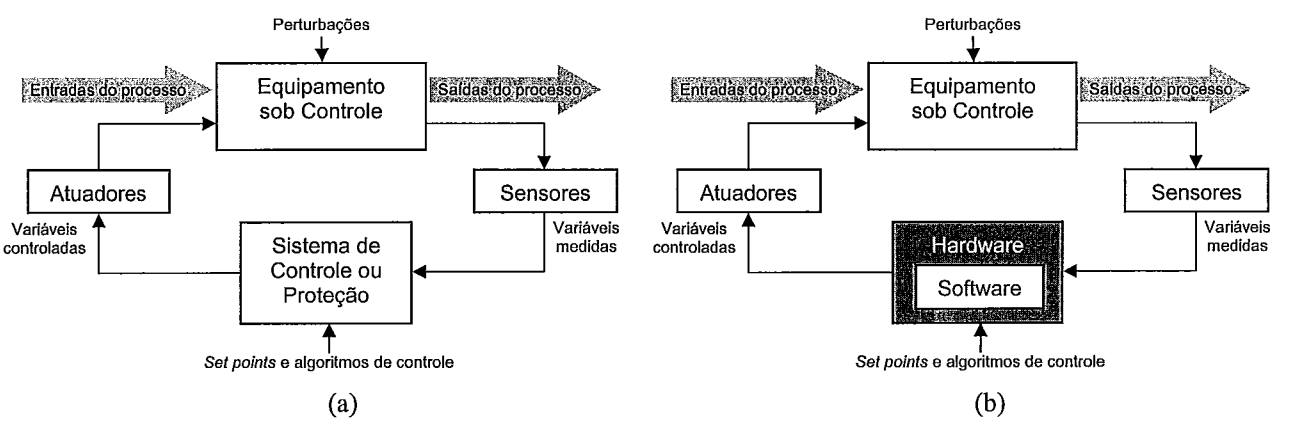

Figura 2.3 - Sistema de controle ou proteção: (a) configuração típica; (b) baseado em computador.

Neste trabalho nós estamos interessados em sistemas que utilizam computadores e onde a sua utilização apresente vantagens sobre aqueles sistemas, ou que a sua

utilização seja essencial, para se obter os requisitos de controle e segurança necessários. Assim quando um sistema baseado em computador é adotado, o sistema tem dois componentes principais: hardware e software. Ambos são essenciais, e ambos afetam diretamente a segurança do sistema, Figura 2.3(b).

Podemos classificar os SCS em dois grandes grupos, segundo a sua mobilidade: *fixos* e *móveis*. SCS fixos são aqueles sistemas geograficamente fixos que residem em computadores desde os mais compactos até os de grande porte (*mainframes*), cujo código pode ser desde um simples código embutido (*firmware*) até sistemas com milhares de linhas de código fonte. Alguns exemplos desses sistemas são os sistemas de controle de tráfego aéreo (Bachelier, 1990, Kahne, 1996), sistemas de desligamento automático de usinas atômicas (Gerhart, 1994, Harauz, 1997), sistemas de monitorização e controle de plantas químicas e sistemas médicos (Blyth, 1997).

SCS móveis são aqueles sistemas que residem em computadores cujo substrato é geograficamente móvel por meios próprios. Em geral são computadores onde prevalecem limitações do tipo: volume físico do computador e seus periféricos, tamanho da memória, capacidade de endereçamento, velocidade de processamento (MIPS[3]), volume de entrada e saída de dados (Moura, 1995) entre outras. É evidente que estas limitações dependem do tipo do sistema e da sua aplicação.

Portanto, são SCS móveis sistemas embarcados em aviões (Hagelauer, 1998, Potocki, 1993, Sweet, 1995), foguetes lançadores (Lion, 1996), mísseis (Littlewood, 1992), satélites (Neumann, 1995), helicópteros (Moura, 1997), navios, trens (Gerhart, 1994) e humanos (Mojdehbakhsh, 1994), e.g. marca-passo cardíaco artificial. Assim, *software embarcado* é todo o software contido em SCS móveis por meios próprios.

[3] Milhões de instruções por segundo (MIPS).

Algumas das principais propriedades de um SCS, com algumas variações em função do tipo da aplicação, são:

- ***confiabilidade*** é a probabilidade de que um componente, ou sistema, funcione corretamente durante um dado período de tempo sob um determinado conjunto de condições operacionais.
- ***disponibilidade*** de um sistema é a probabilidade de que o sistema estará funcionando corretamente em um dado período de tempo.
- ***integridade de sistema*** é a capacidade que um sistema tem de identificar falhas na sua própria operação e informar um operador humano.
- ***integridade de dados***, é a capacidade de um sistema de prevenir danos ao seu próprio banco de dados e de detectar, e possivelmente corrigir, erros que ocorram.

Estas propriedades são identificadas e definidas na especificação de requisitos do SCS. Além destes requisitos funcionais existem ainda os requisitos não funcionais (facilidade de manutenção, tamanho, custo, etc.) e os requisitos de segurança que estabelecem as necessidades do sistema para garantir os níveis de segurança adequados.

Os requisitos de segurança de sistema são determinados através de várias etapas:

- Identificação e classificação das vulnerabilidades associadas ao sistema;
- Determinação de métodos para tratamento dessas vulnerabilidades;
- Atribuição dos requisitos de confiabilidade e disponibilidade apropriados;
- Determinação de um nível de integridade de segurança apropriado;
- Especificação dos métodos de desenvolvimento exigidos segundo o nível de integridade determinado.

A segurança de um SCS deve ser avaliada dentro de um contexto abrangente onde nenhum elemento deve ser tratado isoladamente, pois a segurança é relativa ao sistema como um todo. Trata-se de uma propriedade emergente do sistema e não uma propriedade dos seus componentes. A ocorrência de falhas isoladas do software, ou em conjunção com outros fatores, pode colocar o sistema numa condição vulnerável (*hazard*), ou seja, aquela em que o sistema está exposto a um acidente (Camargo, 1997). O software de um SCS não deve ser tratado de forma que a segurança seja considerada como uma propriedade em separado; a segurança deve estar presente desde o início do projeto, ou seja, é algo que deve ser desenvolvido junto com o sistema e não algo que possa ser agregado no final do processo de desenvolvimento. Uma das principais responsabilidades da análise de segurança é avaliar as interfaces entre os componentes do sistema e determinar os efeitos da interação desses componentes, sendo que esse conjunto de componentes inclui seres humanos, máquinas e ambiente. Parnas observa que não se deve tentar separar código fonte crítico quanto à segurança de outros código fonte em um SCS. Isto porque o software apresenta um comportamento de *elo frágil*, ou seja, mesmo uma falha numa parte "pouco importante" pode gerar efeitos imprevistos em outras partes do sistema (Parnas, 1990). Rushby (1995) acrescenta que nem todos os erros produzem falhas imediatas, pois um erro em um estado pode culminar numa falha do sistema em um estado posterior. A falha é uma propriedade do comportamento externo de um sistema - o qual por sua vez é uma manifestação dos estados internos e das transições desses estados.

Segundo Leveson (1995), o software pode afetar a segurança de um sistema de duas formas:

1. Pode exibir um comportamento em termos de valores de saída e temporização que contribuam para o sistema atingir um estado vulnerável, ou

2. Pode falhar no reconhecimento, ou tratamento, de erros de hardware necessário para o controle ou resposta do sistema.

A preocupação com a segurança de um SCS é diretamente proporcional às consequências que um mau funcionamento deste pode ocasionar. O nível de complexidade e o grau de integridade requerido são os fatores determinantes do nível de criticalidade do sistema. A medida que a complexidade do sistema aumenta, o potencial de falhas tende a aumentar. O nível de integridade almejado vai determinar o quanto se está interessado em evitar a ocorrência de tais falhas de funcionamento. Apesar de significativa a segurança é sempre de importância relativa ao risco envolvido em diferentes situações. Dessa forma, requisitos diferentes para sistemas de segurança levam ao conceito de níveis de integridade para SCS (Gardiner, 1999). Neste contexto a palavra "integridade" é relativa a "integridade da segurança" que pode ser definida como:

Integridade de segurança é probabilidade de que um sistema relacionado à segurança execute as funções de segurança sob todas as condições declaradas durante um período de tempo estabelecido.

Apesar de se poder expressar a integridade de segurança quantitativamente, é mais comum alocarmos a um sistema um nível de integridade dentre vários disponíveis.

A confecção de um SCS pode ser considerada como um problema de gerenciamento de falhas. As falhas podem ser aleatórias como é o caso das falhas de hardware e, sistemáticas como é o caso das falhas de projeto de software e hardware. Portanto, na determinação do nível de integridade de um sistema é necessário levar em consideração o seu desempenho com respeito a estas duas áreas:

> ***Integridade de Hardware*** é a parte da integridade de segurança relacionada a falhas aleatórias perigosas de hardware.

> ***Integridade Sistemática*** é a parte da integridade de segurança relacionada a falhas sistemáticas perigosas.

Em alguns casos é necessário analisar o software de forma isolada (módulos de RNA, por exemplo) que leva a uma terceira classificação:

> ***Integridade de Software*** é a parte da integridade de segurança relacionada a falhas de software perigosas.

O risco é uma combinação da probabilidade de um acidente e da severidade potencial das consequências deste. As Tabelas 2.1 e 2.2 apresentam uma classificação dos riscos e suas respectivas interpretações segundo a proposta de norma IEC 1508 (IEC, 1995).

	Consequências			
Frequência	*Catastrófica*	*Crítica*	*Marginal*	*Negligenciável*
Frequente	I	I	I	II
Provável	I	I	II	III
Ocasional	I	II	III	III
Remota	II	III	III	IV
Improvável	III	III	IV	IV
Muito improvável	IV	IV	IV	IV

Tabela 2.1 – Classificação dos riscos segundo a proposta de norma IEC 1508 (IEC, 1995).

O risco aumenta a medida que aumenta a probabilidade de um acidente ou a magnitude das perdas (desde que a outra componente não diminua proporcionalmente). Alguns dos fatores que podem afetar estas duas componentes de risco são: complexidade, centralização e automação.

Classes de risco	*Interpretação*
I	Risco intolerável
II	Risco indesejável, e tolerável somente se a redução de risco é impraticável ou se os custos são muito desproporcionais às melhorias obtidas
III	Risco tolerável se o custo da redução de risco vai exceder as melhorias obtidas
IV	Risco negligenciável

Tabela 2.2 – Interpretação das classes de risco segundo a norma IEC 1508 (IEC, 1995).

O risco é definido pelos engenheiros de segurança como sendo uma função da:

1) Probabilidade de ocorrer um estado vulnerável;
2) Probabilidade de que o estado vulnerável conduza o sistema a um acidente;e
3) Pior caso possível de perdas e danos causados pelo acidente.

A diminuição de qualquer um desses fatores de risco, ou de todos, reduz o risco associado ao sistema (Leveson, 1995). Bell e Reinert (1993), observam que a tarefa de

se construir um SCS pode ser considerada como um processo de gerenciamento dos riscos, ou a redução destes.

A utilização de software em SCS apresenta uma questão acerca da confiabilidade do software, pois se este é tão pouco confiável porque não se continua usando circuitos digitais e analógicos? Parnas (1990) apresenta as três principais vantagens da utilização de software no lugar do hardware:

1. O software viabiliza a introdução de mais *lógica* no sistema. Sistemas controlados através de software podem distinguir um maior número de situações e atender cada uma delas apropriadamente. Sistemas de hardware não conseguiriam obter tal comportamento sem uma quantidade proibitiva de hardware. Sistemas controlados por computador dispõem de recursos adicionais, tal como a execução periódica de programas de verificação de hardware aumentando assim a confiabilidade.

2. A lógica implementada em software é, teoricamente, mais fácil de se alterar do que a mesma lógica implementada em hardware. Muitas alterações podem ser feitas sem a adição de novos componentes de hardware (flexibilidade). É muito mais fácil fazer alterações no software do que no hardware, quando um sistema é replicado ou está instalado numa posição física de acesso difícil, ou inacessível, e. g. o caso das sondas espaciais.

3. A tecnologia dos computadores e a flexibilidade do software tornou possível o fornecimento de uma maior número de informações aos operadores num formato muito mais apropriado. O operador de um sistema moderno controlado através de software pode ser provido com informações que seriam inimagináveis em sistemas

de hardware puro. Tudo isso num espaço físico menor, e com menos energia elétrica do que seria necessária a um sistema não computadorizado.

A principal vantagem dos sistemas baseados em computador é o seu enorme poder de processamento, que permite executar funções de controle complexas, tarefa impossível por outros meios. A sua operação é caracterizada por alta velocidade, baixo consumo de energia e reduzido tamanho físico. Em contrapartida, a principal desvantagem desses sistemas relacionados à segurança é a sua inerente complexidade, o principal obstáculo da segurança (Gardiner, 1999). Apesar dos problemas, a principal razão para a utilização dos computadores é que eles fornecem uma enorme capacidade e velocidade de processamento, e além disso são cada vez menores, mais leves e mais poderosos.

Finalizando, podemos concluir que a singularidade e o poder do computador digital sobre as outras máquinas advém do fato de pela primeira vez na história dispormos de uma máquina de propósito geral (Figura 2.4). Por exemplo, não é mais preciso construir um piloto automático a partir do zero, é preciso somente "escrever o projeto" do piloto automático na forma de instruções ou passos para obter os objetivos desejados. Esses passos são então carregados no computador, o qual, enquanto executa as instruções, torna-se a máquina de propósito especial (o piloto automático). Se são necessárias alterações, as instruções podem ser alteradas ao invés de se construir uma nova máquina.

Figura 2.4 - O software mais uma máquina de propósito geral criam uma nova máquina de propósito especial.

2.2.1 Tecnologias Empregadas nos Sistemas Críticos quanto à Segurança

O objetivo desta seção é apresentar um panorama das diferentes características empregadas nos diferentes tipos de Sistemas Críticos quanto à Segurança (SCS) computadorizados que existem atualmente. São sistemas que empregam tecnologia de software convencional com um considerável nível de consolidação. A exposição característica dessas tecnologias permite-nos também visualizar as áreas potenciais para o emprego de Redes Neurais Artificiais (RNA) em SCS.

Em geral os SCS empregam uma, ou mais, tecnologias apresentadas a seguir, dependendo do tipo e das exigências da aplicação. Assim um SCS pode ser de tempo real ou não, pode ter algum tipo de controle sobre o processo ou não, ser ou não tolerante a falhas ou ser ainda uma combinação desses tipos. O principal enfoque abordado é a certificação do sistema, concentrando-se em aspectos específicos do software desses SCS. Sistemas que empregam estas tecnologias são sistema de vários domínios de aplicação (geração de energia nuclear, sistemas aeroespaciais, sistemas ferroviários, sistemas médicos, sistemas náuticos, etc.) cujos respectivos organismos responsáveis da área são responsáveis pela sua regulamentação.

2.2.1.1 Sistemas de Controle

Um processo controlado consiste numa malha de componentes interligados onde o sinal de saída da malha depende do sinal de entrada desta, numa configuração que vai permitir que o sistema responda da forma desejada. Estas entrada e saída representam o relacionamento causa e efeito do processo, onde o sinal de entrada é processado de

forma a gerar um sinal de saída variável. Um controle de *malha aberta* (*open loop*) utiliza um controlador para obter a resposta desejada.

Um sistema de *malha fechada* (*closed loop*) utiliza uma medida adicional da saída atual para comparar com a resposta de saída desejada. Esta medida é chamada de sinal de retroalimentação. Um sistema retroalimentado é um sistema que procura manter um relacionamento prescrito de uma variável do sistema com outra através da comparação da função dessas variáveis, usando a diferença entre elas como meio de controle. Sistemas retroalimentados geralmente usam uma função que relaciona a saída com a entrada como referência para controlar o processo.

O controle de um processo industrial por meios automáticos, ao invés de meios manuais, é chamado de *automação*. Automação é a operação automática, ou controle, de um processo, dispositivo ou sistema. As principais características desejáveis desses sistemas de controle são precisão, confiabilidade e robustez , e outras específicas ao tipo da aplicação.

Dois casos exemplos são a Boeing Company e o consórcio europeu Airbus. Essas duas construtoras de aviões decidiram utilizar a tecnologia *fly-by-wire* (no Boeing 777 e no Airbus A320) não por um aumento de segurança, redução de peso ou custo menor - fatores que também pesaram na decisão - mas, principalmente por um aumento da confiabilidade e pela facilidade de manutenção já que fios e componentes eletrônicos são mais leves, mais fáceis de instalar e manter do que seus correspondentes mecânicos (Aplin, 1997, Peterson, 1996).

Em sistemas de controle retroalimentados o computador é usado como dispositivo controlador. A diferença do controle convencional (analógico) para o computadorizado (digital) reside nos diferentes tipos de informação usados. O controle convencional utiliza sinais analógicos contínuos e o digital trabalha com amostragens

periódicas de sinais. Portanto, é importante ressaltar aqui que os métodos para análise e projeto desses dois tipos de controle são distintos.

A utilização de computadores digitais como dispositivo controlador aumenta a cada dia, em função dos melhorias dos dispositivos digitais no que se refere a confiabilidade, versatilidade e preço, entre outras características.

O computador digital trabalha com sinais digitais (discretos), diferente dos sistemas analógicos (contínuos). Um sistema de controle digital utiliza sinais digitais e um computador digital para controlar um processo. Os dados medidos são convertidos do formato analógico para o formato digital através do conversor analógico-digital. Após o processamento das entradas, o computador digital fornece uma saída no formato digital. Esta saída é então convertida para o formato analógico por um conversor digital-analógico.

O número de sistemas que usam um computador na indústria cresceu muito ao longo das últimas décadas. Existem atualmente em torno de vinte milhões de sistemas de controle em uso ao redor do mundo, considerando-se aí somente sistemas de controle de natureza complexa, tais como controle de processos químicos ou controle de aeronaves (Dorf, 1995).

2.2.1.2 Sistemas de Tempo Real

Em geral, sistemas de tempo real geram algum tipo de ação em resposta a eventos externos (ou internos), através da aquisição de dados e controle, sob severas condições de tempo e confiabilidade. Um sistema de tempo real pode ser definido como um sistema cuja principal medida de desempenho é se ele atende, ou não, os prazos

preestabelecidos das *tarefas*[4]. O tempo entre a identificação de um evento e a ação de resposta varia em geral de milésimos de segundo a minutos, dependendo do tipo da aplicação. São classificados em duas categorias: Sistems de *tempo real rígido* (*hard real-time*) e sistemas de *tempo real não rígido* (*soft real-time*) (Krishna, 1994). Esses sistemas são empregados principalmente em processos onde a complexidade e a dinâmica impedem que sejam tratados através de controle humano direto. Sistemas de controle em tempo real são usados em aplicações onde o computador é um dos componentes da malha fechada do processo. O tempo de resposta do sistema é o tempo dentro do qual o sistema deve detectar um evento e responder com uma ação específica. Caso o tempo de tratamento desses eventos seja excedido o sistema pode falhar. Aplicações onde a falha do sistema em atender os prazos de execução, resultam em mortes ou grandes prejuízos são classificados de sistemas de *tempo real rígido*. Aplicações onde os prazos preestabelecidos não sejam, porventura, cumpridos não resultando com isso em falhas com consequências catastróficas são classificados como sistemas de *tempo real não rígido*.

Uma aplicação de tempo real é composta, em geral, por um conjunto de tarefas cooperativas. Essas tarefas são executadas em intervalos regulares e possuem prazos dentro dos quais devem completar suas respectivas funções. Cada vez que uma tarefa é invocada ela avalia o estado do sistema, efetua certas computações, e se necessário, envia comandos para modificar ou mostrar o estado do sistema. Estas tarefas são conhecidas como *tarefas periódicas*. Uma característica particular das tarefas periódicas é que elas são críticas em relação ao tempo, no senso de que o sistema não pode funcionar sem que elas terminem sua execução dentro dos prazos preestabelecidos. Portanto, é muito importante para o sistema garantir que os prazos dessas tarefas

[4] Os programas executáveis que rodam em sistemas de tempo real são normalmente chamados de tarefas

periódicas críticas sejam cumpridos independentemente de outras condições do sistema. Nem todas as tarefas de tempo real são ativadas em intervalos regulares. Certas tarefas são ativadas somente quando certos eventos ocorrem e são classificadas como *tarefas aperiódicas*. Caso ocorra evento crítico em relação ao tempo então a tarefa correspondente terá um prazo dentro do qual ela deverá terminar sua execução. Por outro lado, se o evento não é crítico em relação ao tempo então a tarefa não terá nenhum prazo, mas ela deve ser executada o mais rápido possível sem prejudicar os prazos das outras tarefas (Shin, 1994).

Os sistemas de tempo real na maioria das vezes estão sujeitos a um ambiente operacional hostil. Sistemas de propósito geral estão normalmente localizados em escritórios ou centros de processamento de dados que tem faixas de variação de temperatura e umidade estreitas e controladas. A maioria dos sistemas de tempo real operam em ambientes sujeitos às condições do ambiente (grandes variações de temperaturas, poluição, estresse mecânico, interferência de radiação eletromagnética, etc.), criando assim a necessidades de requisitos de alta confiabilidade, disponibilidade e reatividade para o hardware da plataforma computacional, através de modelos conhecidos e estabelecidos. Mas, modelos de confiabilidade semelhantes para o software é ainda um campo novo e pouco conhecido (Krishna, 1994). Um sistema de tempo real é caracterizado pelos seguintes pontos:

- O tempo é o recurso mais precioso a gerenciar. As tarefas devem ser ativadas e escalonadas para que completem suas execuções antes do fim dos seus prazos, nas suas respectivas prioridade e sincronismo. O funcionamento correto não depende

(do inglês, *tasks*) já que devem concluir sua execução dentro de um determinado prazo (*deadline*).

somente da correção lógica mas também do tempo no qual os resultados são produzidos.

- A confiabilidade, disponibilidade e reatividade são cruciais, já que a falha de um sistema de tempo real rígido (SCS) pode ter consequências catastróficas.
- O ambiente onde o computador opera é um componente ativo do sistema, ou seja, não faz sentido considerar somente os computadores, em um sistema *fly-by-wire*[5] e não considerar o avião onde o sistema está embarcado (Shin, 1994).

A expressão *tempo real* tem um significado diferente de *interativo* e *tempo compartilhado* (*time-sharing*). Sistemas de tempo real devem responder dentro de limites de tempo restritos. O tempo de resposta de sistemas interativos ou de tempo compartilhado pode, em geral, ser excedido sem consequências graves. Outra característica importante das aplicações de tempo real é a de que o comportamento do sistema de tempo real controlador deve ser determinístico (previsível), ou seja, deve ser possível mostrar durante a modelagem que todas as restrições de tempo da aplicação serão atendidas desde que certas condições assumidas para o sistema tenham sido satisfeitas. Isto requer que se saiba as características exatas de todas as tarefas *a priori*. Portanto, seria necessário se conhecer *a priori* o número de tarefas e os recursos necessários para todas as tarefas o tempo todo. Além disso, seria necessário saber também todas as possíveis mudanças no ambiente o tempo todo já que o ambiente pode influenciar significativamente o comportamento do sistema. É fácil constatar que essas informações não estão disponíveis durante a modelagem do sistema, expondo as dificuldades de modelagem desses sistemas.

[5] Sistema no qual são utilizados meios elétricos, digitais ou analógicos, em substituição aos meios mecânicos ou hidráulicos, para o comando das superfícies aerodinâmicas empregadas nas manobras de vôo (Moura, 1997).

A sincronização das tarefas é comandada pelo sistema operacional através de funções específicas para aplicações de tempo real. Através destas funções e do esquema de prioridade de execução predefinido, o esforço para sincronizar as tarefas de um determinado sistema diminui bastante, ficando o trabalho maior para o supervisor do sistema, uma tarefa que faz a troca de tarefa que está executando em um instante por outra, que está pronta executar e tem prioridade maior. Neste ponto, uma interrupção de hardware pode ser tratada como um semáforo, obrigando a uma revisão pelo supervisor da fila de tarefas prontas para executar. Este tipo de sistema, em uma tarefa de maior prioridade é executada primeiro, é chamado de preemptivo. Não consideramos aqui sistemas multiprocessados.

2.2.1.3 Sistemas Tolerantes a Falhas

O objetivo da tolerância a falhas é projetar sistemas de forma que ocorrência de defeitos não causem a falha do sistema. Os métodos de tolerância a falhas estão baseados em algum tipo de *redundância*, resultando em sistemas mais complexos para executar a tarefa desejada. Existem dois tipos de redundância: *Espacial* e *Temporal*. Redundância Espacial refere-se a duplicação (ou triplicação) das funções dos grupos de componentes físicos dos sistema. A Redundância Temporal envolve a solução de um subproblema várias vezes usando os resultados para gerar uma solução final com base em algum tipo de média. Segundo Laprie (1991) um sistema confiável (*dependable system*) é um sistema para o qual a confiança pode justificadamente ser colocada sob certos aspectos da qualidade do serviço que ele entrega. A qualidade do serviço inclui tanto a sua correção (conformidade com os requisitos, especificações e expectativas)

como a continuidade da entrega do serviço (Laprie, 1991). Falhas são atribuídas a causas subjacentes chamadas de defeitos. Defeitos podem incluir erros na especificação ou projeto (*bugs*), falha de componentes, operação imprópria e anomalias do ambiente. Nem todas os defeitos resultam em falhas imediatas: uma falha é uma propriedade do comportamento externo de um sistema - o qual por sua vez é uma manifestação dos estados internos e das transições de estado. Um erro é um *erro latente* quando ele persiste através de várias transições de estado, tornando-se um *erro efetivo* (num determinado estado) quando afeta a entrega do serviço causando a falha do sistema. Portanto, a tolerância a falhas está baseada na detecção de erros latentes, antes que eles se tornem efetivos, e na substituição do componente errôneo do estado por uma versão livre de erro (Rushby, 1993). Enquanto os sistemas tolerantes a falhas estão preocupados em reduzir a incidência de falhas, através da detecção e correção de erros *latentes* antes que eles se tornem *efetivos*, a segurança preocupa-se com a ocorrência de acidentes: eventos não planejados causadores de grandes prejuízos. Enquanto falhas de sistema são definidas em termos de serviços do sistema, a segurança é definida em termos de consequências externas.

SCS projetados na década de setenta utilizavam configurações de sistemas redundantes (duplos e as vezes triplos), onde o ênfase estava na tolerância a falhas aleatórias de hardware, conhecidas como falhas operacionais, que presumia-se ocorrerem independentemente em cópias de hardware redundante. A experiência com esses primeiros sistemas mostrou que a redundância oferece uma alternativa efetiva para esse tipo de prevenção de falhas, mas também complicava substancialmente a tarefa de validação em função do aumento da complexidade. Um fator contribuinte para isto foi a abordagem *ad-hoc* na gerência de redundância, empregada muitas vezes sob a suposição simplista de que redundância era equivalente a tolerância a falhas.

Propagação de falhas, propagação de erros, sincronização entre elementos redundantes e outros itens referentes a gerência de redundância foram subestimados. Para um computador ser considerado adequadamente confiável para aplicação em SCS, ele deve ser capaz de sobreviver a um número específico de falhas aleatórias de componentes. Esta abordagem tem tido sucesso ao ponto da causa dominante da falha de um computador (corretamente projetado com resiliência Bizantina) ser resultado de falhas de modo comum (Lala, 1994). Uma *falha de modo comum* ocorre quando múltiplas cópias de um sistema redundante falham praticamente ao mesmo tempo, geralmente devido ao mesmo motivo. Sistemas resilientes a falhas Bizantinas são sistemas que suportam a ocorrência de combinações complexas de falhas (modos de falha complexos) . Sistemas *resilientes* são sistemas capazes de retornar rapidamente a uma boa condição de funcionamento. A falta de uma teoria unificadora que trate as falhas de modo comum de forma segura faz com que tomemos os seguintes cuidados com os princípios básicos abaixo relacionados:

- Técnicas que evitem a introdução de defeitos (*fault avoidance*) principalmente nas fases de especificação, projeto e implementação;
- Técnicas de remoção de defeitos (*fault removal*) aplicadas principalmente durante as fases de teste e validação do software e hardware;
- Técnicas de detecção de defeitos (*fault detection*) durante o funcionamento do sistema de forma a minimizar os seus efeitos;
- Técnicas de tolerância a falhas (*fault tolerance*) são projetadas para permitir que o sistema funcione corretamente na presença de falhas.

A diferente natureza entre hardware e software não permite que as mesmas técnicas de tolerância a falhas empregadas para o hardware sejam usadas para o software com o mesmo propósito. A utilização de elementos de hardware redundantes em SCS provou ser efetiva na detecção e tolerância de defeitos físicos. Mas falhas de projeto, causadas por erros humanos ou ferramentas de projeto erráticas, são reproduzidas quando cópias redundantes são produzidas, assim a simples replicação de elementos de software ou hardware é insuficiente. Em vez disso, a tolerância a falhas de projeto depende da aplicação da diversidade de projeto, que cria componentes diversos a partir de um requisito comum. *Diversidade* significa uma abordagem na qual componentes para computações redundantes não são cópias, são componentes projetados independentemente para atender os requisitos de sistema. A redundância de software não implica necessariamente em replicar o software por completo. O objetivo da redundância de software é torná-lo robusto quanto ao aspecto da segurança e da confiabilidade, ou seja, tornar o modo de falha mascarável e detectável sendo que, num aspecto limítrofe, garantir a falha num modo seguro (Camargo, 1997). As técnicas de redundância de software mais utilizadas, segundo (Kelly, 1991) são :

- **Verificação de consistência** onde várias versões do mesmo processo são executadas - em paralelo ou sequencialmente - e ao final é eleita a resposta mais "votada" segundo as regras de consenso estabelecidas. Essa versões podem utilizar diferentes algoritmos e heurísticas para resolução do mesmo problema. É claro que no caso da execução sequencial é necessário que exista tempo hábil para a votação, mas esse tipo de configuração não é muito aconselhável pois uma falha de hardware fará o sistema falhar, ao menos temporariamente.

- **Múltiplas versões de software** exige que sejam desenvolvidas várias versões do mesmo software, em diferentes linguagens para diferentes processadores (no caso de haver diversidade de hardware também) por equipes independentes e sem contato. Nessa abordagem todas as versões podem ser desenvolvidas a partir de uma mesma especificação de requisitos ou a partir de especificações independentes. O desenvolvimento a partir de uma mesma especificação não é aconselhável uma vez que um erro não detectado na especificação comprometerá todas as versões criando uma situação perigosa que poderá levar a uma falha de modo comum do sistema. Esta técnica utiliza um método de recuperação de erro com *encadeamento direto* (*forward*) para construir um novo estado sem erros a partir das informações redundantes existentes.

- **Blocos de recuperação** é uma técnica que também está baseada em múltiplas versões de software e no conceito de diversidade de projeto. Esta técnica utiliza uma abordagem de recuperação de erro com *encadeamento reverso* (*backward*) que tenta trazer o sistema de volta a um estado anterior sem erros a partir do qual a execução pode ser tentada novamente.

Os resultados com a pesquisa de software com múltiplas versões não foram muito promissores, pois sistemas desenvolvidos independentemente tendem a apresentar problemas nos mesmos lugares, pois as pontos difíceis do projeto do sistema são difíceis para todos os envolvidos (Joch, 1995). O desenvolvimento do computador de controle de vôo do Boeing 777, por exemplo, começou sendo projetado com uma diversidade de projeto nunca antes tentada na prática ou mesmo num laboratório de pesquisa. A concepção inicial baseava-se em três sistemas redundantes com quatro

processadores diferentes com quatro versões diferentes de software (Lala, 1994). O projeto de diversidade de software foi simplificado ao longo do desenvolvimento e ao invés das três linguagens de programação inicialmente selecionadas (*Ada, C* e *PL/M*) passou a ser usada somente uma linguagem de programação (Ada usando três compiladores diferentes) aparentemente devido a problemas de sincronismo entre os sistemas paralelos, e o hardware passou a trabalhar com três processadores diferentes ao invés dos quatro do projeto original (Aplin, 1997). As equipes de desenvolvimento de software do Boeing 777 foram então unificadas, pois segundo a Boeing profissionais qualificados devem trabalhar juntos e não em grupos separados, já que as partes difíceis são difíceis para todos (Joch, 1995).

Este problema foi constatado na prática por Parnas (Storey, 1996) durante o processo de reengenharia para certificação dos sistemas de desligamento automático dos reatores nucleares da estação de geração de energia elétrica de Darlington (Ontario, Canada). Cada um dos quatro reatores da estação tem dois sistemas de desligamento independentes de filosofias distintas do ponto de vista nuclear (inserção de barras e injeção de veneno no moderador). Durante a execução de uma análise informal dos sistemas de desligamento, Parnas identificou um erro conceitual de projeto comum às duas implementações, apesar das aplicações terem sido desenvolvidas por equipes independentes, em diferentes linguagens de programação (FORTRAN e Pascal).

Uma última e importante consideração é a de que os métodos de tolerância a falhas não resolvem o problema de requisitos errados.

2.3 Certificação e Normas

A *certificação* (homologação, licenciamento) consiste no processo de emitir um certificado para indicar a conformidade com uma norma, um padrão, um conjunto de diretrizes ou documento similar. Qualquer organização ou indivíduo pode emitir um *certificado*, cuja importância vai variar em função da sua natureza e do órgão emissor. Em alguns casos pode existir a necessidade de um certificado relacionado a aspectos legais: por exemplo, um certificado de credenciamento aéreo é necessário para que uma aeronave tenha autorização para voar. Em tais circunstâncias o certificado desempenha o papel de uma licença da autoridade legal. A necessidade de tais licenças para sistemas críticos quanto à segurança varia muito entre países.

A certificação é executada (e o certificado atribuído), na maioria das vezes, por órgãos governamentais ou por organizações de renome nacional ou internacional, para indicar a aceitabilidade em relação a critérios específicos. Várias áreas da indústria tem autoridades reguladoras que governam projetos específicos desses setores. Alguns exemplos, no Brasil, são o IFI (*Instituto de Fomento e Coordenação Industrial*) dentro do CTA (*Centro Técnico Aeroespacial*) que é o responsável pela homologação de empresas e produtos aeroespaciais (Silva, 1995), e a CNEN (*Comissão Nacional de Energia Nuclear*) para as aplicações nucleares. Nos EUA alguns órgãos certificadores são o FAA (*Federal Aviation Administration*) para a aviação civil; a NRC (*Nuclear Regulatory Commission*) para a área nuclear e a NASA (*National Aeronautics and Space Administration*) para a área espacial. Na Europa podemos tomar como exemplo a JAA (*Joint Airworthiness Authority*) composta pelas autoridades de aviação civil individuais da Alemanha, França, Holanda e Reino Unido (Rushby, 1993).

A certificação de sistemas críticos quanto à segurança tem vários objetivos: 1) aperfeiçoar a segurança do sistema; 2) aumentar a consciência das implicações do desempenho do sistema sobre a segurança; 3) observar um padrão mínimo de projeto e manufatura dentro da indústria relevante; e 4) estabelecer uma estrutura de responsabilidade profissional. A certificação pode ser aplicada a: 1) organizações e indivíduos; 2) métodos e ferramentas; e 3) sistemas e produtos. A certificação implica em avaliações particulares para cada um desses três grupos sendo que cada uma delas é regida por normas e padrões específicos segundo o país e o domínio de aplicação em questão. Desses três grupos o último é talvez o mais importante e é a forma de certificação que nos interessa neste trabalho.

Para obter a certificação, o desenvolvedor de um sistema crítico quanto à segurança deve convencer o órgão regulador da área específica de que o sistema é seguro, mas devido a natureza da segurança, esta é uma tarefa difícil. O desenvolvedor deve ser capaz de mostrar que todas as principais situações de risco foram identificadas e tratadas, e que a integridade do sistema é apropriada para o domínio da aplicação. O processo de certificação segue uma norma, ou conjunto de normas particulares, mas esta conformidade não é, por si só, prova suficiente da adequabilidade do projeto (Pfleeger,1994). O desenvolvedor precisa fornecer evidências de que os métodos e técnicas empregados ao longo do processo de desenvolvimento são adequados e que os testes executados para investigação do comportamento do sistema satisfazem às exigências de desempenho. Além disso, será necessário que o desenvolvedor gere rigorosos argumentos para suportar a afirmação de que o sistema é suficientemente seguro, e que assim vai permanecer ao longo de toda sua vida. Este trabalho de obtenção da certificação é considerável e requer um planejamento cuidadoso, que envolve especialistas de diferentes áreas.

2.3.1 Certificação de Organizações e Indivíduos

Uma organização pode procurar certificação de uma autoridade reguladora como um meio de estabelecer a sua competência em áreas específicas de atividade. A certificação tem por objetivo garantir que as organizações autorizadas atendem a certos níveis de proficiência e que satisfaçam as normas, critérios ou padrões estabelecidos. Mas este tipo de abordagem não é muito apropriado às áreas de interesse da segurança, pois é fácil investigar os procedimentos usados dentro de uma empresa, mas é difícil medir a competência com a qual eles são executados. Por isso a certificação é aplicada com mais frequência em áreas tais como garantia da qualidade e testes, do que em atividades como modelagem e projeto.

A certificação é aplicada também a indivíduos. Engenheiros que trabalham no desenvolvimento de SCS devem ter quantidades consideráveis de treinamento, experiência e habilidade nessa área, mas muito poucos setores da indústria possuem algum tipo de certificação para tais indivíduos (Neumann, 1995). A certificação de indivíduos não garante a sua competência para um papel específico, mas permite que uma indústria imponha certos padrões mínimos. Como a maioria dos setores não opera segundo este princípio, esses padrões são impostos pelas próprias empresas através da seleção dos membros da equipe envolvidos no desenvolvimento de SCS.

Algumas empresas credenciam indivíduos para executar tarefas particulares, ou para trabalhar em projetos específicos. A FAA, por exemplo, delega várias atividades de certificação a alguns funcionários das empresas de fabricação. Esses engenheiros representantes designados (*DER - Designated Engineering Representative*) atuam como agentes da FAA e executam várias tarefas de certificação dentro da própria empresa.

Nos projetos de aeronaves da Boeing e McDonnell Douglas, entre 90 e 95% de todas as atividades de certificação são executadas pelos DER (Rushby, 1993)

Algumas normas exigem e enumeram explicitamente a qualificação mínima dos profissionais de engenharia de segurança envolvidos no processo de desenvolvimento de SCS (MIL-STD-882D, 1997).

Ainda que várias normas relacionadas a SCS que envolvem software, estipularem responsabilidades e exigirem qualificações mínimas dos profissionais envolvidos com a segurança e com a engenharia, nenhuma norma (às quais o autor desse trabalho teve acesso) traz explicita qualquer exigência de formação, experiência ou certificação no com respeito a software.

2.3.2 Certificação de Ferramentas e Métodos

As ferramentas e métodos de desenvolvimento utilizados na produção de SCS tem um papel importante na determinação do desempenho do sistema. Por isso, várias normas impõem restrições às ferramentas e métodos que devem ser utilizados na sua confecção. Algumas normas (MoD 00-55, 1997) enumeram uma série de técnicas de desenvolvimento obrigatórias dentro do ciclo de vida e estabelecem os requisitos de integridade para todas as ferramentas de suporte. Outras normas (IEC, 1995) fornecem uma orientação detalhada sobre os métodos e ferramentas apropriados para as diferentes fases de um projeto de sistema segundo os vários níveis de integridade. Ao passo que outras(RTCA/DO-178B,EUROCAE/ED-12B),por exemplo,não definem explicitamente as ferramentas de desenvolvimento a serem usadas, mas fornecem detalhes do processo de qualificação de ferramentas necessário para que estas sejam aceitas. Talvez a forma

de certificação de ferramentas de maior sucesso e mais amplamente utilizada é o processo de validação aplicado aos compiladores de linguagens de programação, quando uma instituição de renome certifica que um compilador particular está de acordo com o padrão internacional da linguagem e que esta não apresenta extensões.

2.3.3 Certificação de Sistemas e Produtos

Os fabricantes de sistemas críticos devem obter a certificação de seus produtos porque este requisito é obrigatório dentro da sua área. Os requisitos para a certificação variam amplamente entre os diferentes domínios de aplicação e entre países, sendo voluntários em alguns casos e compulsórios em outros para as mesmas áreas. Em domínios de aplicação tais como o de aviação civil e militar, e o de geração de energia nuclear, a certificação é sempre compulsória. A certificação, ao final do seu processo de aplicação, é sempre concedida ao sistema como um todo, mas em alguns setores existem casos onde certificação é aplicada a componentes individuais.

A certificação de produtos contendo software apresenta problemas singulares para desenvolvedores e certificadores. Devido às dificuldades relativas a confiabilidade dos testes de tais sistemas, a certificação deve ser baseada no processo de desenvolvimento e na demonstração de desempenho do próprio sistema. Isto requer não só um julgamento dos métodos usados, mas também uma avaliação da competência da equipe envolvida. Em virtude de tais problemas, quando se está tratando de aplicações críticas, os organismos certificadores assumem uma postura conservadora recomendado cautela quanto à adoção de novas tecnologias não compravadas (Moura, 1997). Em geral estes organismos estabelecem parâmetros rígidos exigindo que, na impossibilidade

de testes exaustivos, que o desenvolvedor demonstre ao menos, ter seguido um processo sistemático segundo as práticas de engenharia, presume-se, conduza a um sistema conforme o requerido.

2.3.4 O Processo de Certificação

O processo de certificação vária principalmente em função do domínio de aplicação (agência reguladora), em função do nível de integridade de segurança do sistema e do país onde está sendo aplicado. Nosso objetivo aqui é apresentar uma visão geral do que consiste tal processo e das suas principais atividades. O processo apresentado está baseado nos princípios do sistema Inglês para SCS grandes com elevado nível de integridade de segurança.

Apesar da fase de certificação de um projeto acontecer no final da etapa de desenvolvimento, o planejamento desse trabalho (tal como o das atividades de V&V) deve ser executado nas etapas iniciais. Devido ao fato da certificação implicar na persuasão de um órgão externo acerca da segurança do sistema, é essencial que se mantenham discussões com esse órgão para se saber o que será necessário. Esta *ligação* estabelece uma comunicação, encoraja o entendimento entre as partes e auxilia no desenvolvimento da certificação. As discussões iniciais entre o desenvolvedor e o corpo regulador enfocam a natureza global do sistema e a abordagem de desenvolvimento a ser escolhida. Padrões (normas) específicos ou diretrizes são compulsórias em algumas indústrias e a certificação implicará na aderência a esses documentos. Em outras indústrias não é exigida nenhuma norma em particular, apesar de que a adoção de uma norma adequada vai muitas vezes facilitar o processo (Hesselink, 1995). Em alguns

setores da indústria não existem padrões relevantes e os desenvolvedores podem decidir modelar o seu trabalho segundo uma norma genérica tal como a IEC 1508 (*Functional Safety: Safety-Related Systems*) (IEC, 1995) com as adequações necessárias a sua aplicação.

Uma vez estabelecidos os métodos globais a serem usados, o desenvolvedor vai gerar um plano de verificação para aprovação da autoridade reguladora. Este plano vai fornecer detalhes sobre o sistema proposto, sobre os métodos de desenvolvimento a serem usados e sobre a documentação a ser fornecida. Onde for adotada uma norma particular, o plano vai indicar as técnicas propostas para obter conformidade com aquela norma. O plano vai listar também quaisquer áreas nas quais o desenvolvedor planeja se desviar da norma, com a justificativa adequada. Normalmente a submissão do plano de verificação será seguida de discussões entre as duas organizações para resolver quaisquer desentendimentos ou mal entendidos. Ao final desse processo espera-se que o desenvolvedor receba o "de acordo" do regulador sobre a adequação do esquema de desenvolvimento proposto. Caso contrário, o desenvolvedor terá que revisar o esquema proposto fazendo as modificações necessárias segundo o órgão regulador.

Durante o projeto, o aumento do conhecimento dos problemas envolvidos pode implicar em alterações nos métodos utilizados. Se tais modificações tiverem alguma influência sobre o plano de verificação, deve ser obtida então uma aprovação junto ao órgão certificador, antes que as modificações sejam efetuadas, garantindo assim que a certificação não será afetada. Assim esse processo de *ligação* da certificação continua ao longo de todos os estágios do projeto. A medida que o trabalho progride o desenvolvedor vai suprindo o órgão regulador com a documentação adequada demonstrando que as exigências do plano de certificação estão sendo satisfeitas. O

desenvolvedor deve também fornecer dados gerados nos vários estágios do projeto, para embasamento das suas colocações.

Em grandes projetos a documentação necessária é imensa e representa um grande investimento de tempo e esforço. A parte principal dessa submissão será o plano de segurança, que detalha o tratamento dos tópicos de segurança ao longo do processo de desenvolvimento. No julgamento do material, a autoridade reguladora vai manter uma série de revisões nas quais o material será discutido com a equipe de desenvolvimento e quaisquer subcontratados envolvidos. Se o órgão certificador está satisfeito com os termos em que foi concluído o plano de certificação e que todas as suas exigências foram satisfeitas, então um certificado, ou licença, será emitido. Em alguns casos esta certificação pode ser condicional, impondo certas restrições operacionais ou temporais.

2.3.5 O Caso de Segurança

O caso de segurança (*safety case*) é um registro de todas as atividades de segurança associadas ao sistema, ao longo da sua vida. Ele é criado cedo no processo de desenvolvimento, sendo expandido a partir de então para incluir detalhes de todos os aspectos do trabalho de desenvolvimento que são relevantes para a segurança. Seguindo o desenvolvimento o caso de segurança deve ser mantido através da fase operacional, para documentar quaisquer modificações do sistema ou na sua utilização. A medida que os requisitos mudam, ou o sistema é modificado, será necessário justificar tais alterações nos termos das suas implicações para a segurança do sistema.

Uma das funções mais importantes do caso de segurança é dar o suporte necessário à aplicação para a sua certificação. Aqui a autoridade reguladora vai estar

procurando evidências de que todas as possíveis vulnerabilidades (perigos) potenciais tenham sido identificadas e que as devidas providencias foram tomadas para trata-las. O caso de segurança deve demonstrar também que os métodos de desenvolvimento apropriados foram adotados e que eles foram executados corretamente. A Tabela 2.3 lista os itens que um caso de segurança deve conter e é visível o grande volume de informações que ele contém e que a sua produção requer uma quantidade de esforço. A maior parte dos casos de segurança são desenvolvidos usando-se ferramentas tais como processadores de texto e bancos de dados, e dependem de revisões manuais para garantir a sua consistência e completeza. Em sistemas muito complexos esta tarefa torna-se extremamente difícil, e ferramentas baseadas em computador são de grande auxílio nesse trabalho. Estas ferramentas não substituem a capacidade necessária na preparação do caso de segurança, mas auxiliam na *gerência* do processo.

Um dos problemas associados à produção de um caso de segurança é que os tópicos pertinentes são sempre multidisciplinares. Pode ser apropriado, ou necessário,

Conteúdo do Caso de Segurança
▪ Uma descrição do SCS
▪ Evidências da competência do pessoal envolvido em qualquer atividade de segurança
▪ Uma especificação dos requisitos de segurança
▪ Os resultados da análise de vulnerabilidade e da análise de riscos
▪ Detalhes das técnicas de redução de riscos empregadas
▪ Os resultados da análise de projeto mostrando que o projeto do sistema atende todos os objetivos de segurança necessários
▪ A estratégia de verificação e validação (V&V)
▪ Os resultados de todas as atividades de V&V
▪ Registro das revisões de segurança
▪ Registro de quaisquer acidentes que ocorreram ao longo da vida do sistema
▪ Registro de todas as modificações feitas no sistema e a justificava de continuidade da sua segurança

Tabela 2.3 – Conteúdo de um caso de segurança (*safety case*).

envolver equipes com especialidade de áreas diversas, tais como: software, hardware, eletrônica analógica, engenharia elétrica, engenharia mecânica, pneumática, hidráulica, fatores humanos e psicologia. Serão também necessárias informações de especialistas relacionados às áreas de aplicação tais como engenheiros aeronáuticos em projetos aeroespaciais ou engenheiros biomédicos em projetos médicos. A interação entre essas disciplinas diversas é extremamente complexa, e a identificação de tópicos de segurança relevantes é muito difícil.

O caso de segurança deve incluir uma argumentação rigorosa acerca da segurança do sistema e deve demonstrar que ele satisfaz todos os seus requisitos de segurança. Isto vai envolver numerosos passos que de alguma forma lembram os componentes de uma prova matemática. Cada estágio dessa "prova" deve justificado cuidadosamente e qualquer assunção (ato ou efeito de assumir) feita explicitamente. Qualquer assunção não justificada representa um defeito no argumento de segurança e consequentemente no caso de segurança. Infelizmente, a identificação de assunções não justificadas em projetos multidisciplinares requer capacidade e dedicação. Por esta razão, a produção do caso de segurança representa um dos aspectos mais difíceis e que demandam mais atenção na confecção de SCS.

2.4 Conclusão

Com o objetivo de apresentar um panorama geral do contexto para aplicação de Redes Neurais Artificiais (RNA) em Sistemas Críticos quanto à Segurança (SCS), definimos e caracterizamos inicialmente as RNA. Procuramos então apresentar o amplo

espectro de domínios nos quais os SCS são utilizados segundo os diferentes tipos e tecnologias empregados, enfocando principalmente aspectos relacionados ao software de forma a expor o contexto para a aplicação de RNA nesses sistemas computadorizados. Finalizamos definindo e caracterizando a certificação, procurando apresentar uma visão do processo de como esta é executada e os problemas envolvidos na sua aplicação.

Capítulo 3

3 Redes Neurais Artificiais e a Certificação

Parece existir um consenso generalizado de que as Redes Neurais Artificiais (RNA) não são adequadas para uso em sistemas que exigem um alto nível de integridade ou em Sistemas Críticos quanto à Segurança (SCS). Bishop (1995a) apresenta e discute esta visão, onde explica que este consenso não causa surpresa, pelo modo como as RNA são geralmente apresentadas. Ele coloca, por exemplo, que a maioria dos textos introdutórios sobre RNA faz analogias históricas entre sistemas de processamento de informação e sistemas biológicos como motivação para o desenvolvimento de arquiteturas de redes e algoritmos. Dessa forma, criam uma expectativa e atraem o leitor com a perspectiva da possibilidade de criação de sistemas de Inteligência Artificial (IA) com capacidade semelhante àquelas do cérebro humano, mas fornecem pouca ou nenhuma orientação sobre como chegar a algoritmos ótimos e como verificar o desempenho desejado da RNA. Ainda segundo o autor, estes problemas são exacerbados em função dos vários pacotes de software de RNA disponíveis no mercado, que permitem criar, treinar e até gerar aplicações de RNA, tudo isto com um mínimo conhecimento das bases teóricas da modelagem desenvolvida.

De uma perspectiva de reconhecimento de padrões Bishop (1995b) coloca que as RNA podem ser consideradas como uma extensão de várias técnicas desenvolvidas nas últimas décadas e que a falta de conhecimento dos princípios básicos da estatística de reconhecimento de padrões está no âmago de vários erros na aplicação de RNA. Segundo o autor, o estigma de "caixa-preta" das RNA é injustificado já que existe um

conhecimento considerável acerca de como as RNA operam e de como usa-las efetivamente.

Corroborando estas colocações de Bishop (1995a), constatamos que não existem muitos trabalhos na literatura sobre redes neurais artificiais no contexto da certificação de Sistemas Críticos quanto à Segurança ou que tenham alguma relação direta com o tema. Encontramos menos de dez trabalhos sobre o assunto, com um enfoque mais direto, cujas abordagens concentraram-se no problema da certificação de módulos de software que empregam RNA do ponto de vista do desenvolvedor, do ponto de vista do certificador e outros.

Nosso objetivo neste capítulo é discutir os trabalhos encontrados na literatura sobre o tema de certificação de RNA no contexto dos SCS. O interesse está voltado principalmente para os processos de desenvolvimento empregados e como são avaliadas as RNA de forma que possa identificar uma estrutura sobre a qual possa ser aplicado um processo de V&V.

As RNA estudadas neste trabalho são implementadas através de software em computadores digitais convencionais, não consideramos implementações em hardware. Concentramos nossa discussão sobre os aspectos particulares próprios das RNA, ou seja, nas diferenças que elas apresentam no processo de desenvolvimento em relação ao software convencional. Excluímos desta discussão a certificação de redes neurais que aprendem on-line (se auto modificam) já que este tipo de RNA apresenta respostas diferentes para um mesmo conjunto de dados de entrada em tempos diferentes. Esta característica é incompatível com os SCS pois estes tem que ser determinísticos (previsíveis). Isto significa que deve se saber de antemão como o SCS vai se comportar numa determinada situação. Não tratamos também da certificação de sistemas de IA baseados na extração de regras a partir de redes neurais (Wen & Callahan, 1996), nem

de redes neurais geradas a partir de sistemas especialistas (Austin, 1991) por considerarmos tópicos fora do escopo do nosso trabalho.

Como vimos a computação neural é uma forma de programação indutiva, onde uma tarefa é executada por meio de um modelo que foi treinado com os dados que representam esta tarefa. Tal abordagem é particularmente apropriada para aplicações muito complexas, pouco compreensíveis e que comportam um determinado grau de incerteza. É a abordagem ideal (as vezes a única disponível) para aplicações onde é impossível ou impraticável o uso de soluções algorítmicas.

Peterson (1993) descreve algumas das características dos domínios para os quais as RNA são uma abordagem viável. Para redes Perceptron de Múltiplas Camadas com treinamento pela regra *backpropagation*, estas características são:

1) Dados adequados estão disponíveis. Treinar uma RNA geralmente requer casos de treinamento acurados na ordem das centenas e idealmente uma quantidade semelhante de dados para teste. Os dados de treinamento devem ser representativos do universo inteiro de experiência que a rede espera tratar durante a sua operação. Para cada caso de treinamento deve existir uma resposta correta.
2) O problema é do tipo que requer adaptação a situações que se modificam. As RNA podem ser treinadas para responder bem a várias combinações diferentes de entradas. Portanto, uma rede devidamente treinada pode responder adequadamente em tempo real a entradas que se modificam.
3) O problema é do tipo de reconhecimento de padrões. Estes problemas compreendem tarefas pequenas como o reconhecimento de letras escritas a mão e problemas mais complexos tais como a detecção e classificação de aviões inimigos. Um problema

que requer tanto o reconhecimento quanto a adaptação é um grande candidato para ser resolvido por uma RNA.

4) O Problema é difícil de modelar. A RNA pode ser capaz de construir um modelo adequado. Problemas que são difíceis de modelar são geralmente não lineares e podem envolver estatística não Gaussiana.
5) Não é necessário aprendizado em tempo real. Em geral o tempo para o treinamento de uma RNA usando *backpropagation* é medido em minutos ou horas. Isto é incompatível com a operação em tempo real.

Fazendo uma comparação entre a computação neural com a abordagem para o desenvolvimento de software convencional através de um exemplo concreto (desenvolver um modelo para um motor a combustão) podemos apreciar melhor as diferenças entre estas duas abordagens. Numa abordagem convencional seria necessário determinar os processos físicos que governam o motor, analisando-os para gerar as equações matemáticas que representam o sistema para então programar o software para simular estas equações e as suas soluções. Em princípio este método poderia ser usado sem o uso de um motor físico. Uma abordagem indutiva envolveria a coleta de dados de um motor e o treinamento de um modelo (uma RNA) para reproduzir os mesmos relacionamentos tais como os presentes nos dados e determinar uma boa aproximação da função subjacente que gerou os dados.

Os autores, dos trabalhos estudados, são unânimes na constatação de que os métodos atuais de verificação e validação para software convencional não se aplicam diretamente à uma estrutura de desenvolvimento para as RNA, em função da sua natureza distinta e das atividades adicionais que elas requerem. É importante observar aqui o porquê desse grande interesse sobre o processo de V&V. Da mesma forma que a

análise de segurança, o processo de V&V é uma das atividades mais importante sob a ótica da certificação da aplicação. São duas atividades críticas, se não as mais críticas, do ponto de vista do órgão certificador. Isto porque a verificação preocupa-se com o processo de desenvolvimento (o sistema foi desenvolvido dentro dos melhores princípios da engenharia?) e a validação com o produto final (o sistema desenvolvido era o que se queria desenvolver?). Da mesma forma a segurança é avaliada procurando garantir que o sistema (ou módulo) não vai executar, ou contribuir para, determinadas ações que ponham em risco a vida humana ou o meio ambiente. Estas duas atividades (V&V e análise de segurança) são iniciadas junto com o desenvolvimento e se desenrolam em paralelo à confecção do sistema. Tal como outras importantes atividades envolvidas na confecção de um SCS os atributos do sistema não podem ser agregados a este *a posteriori*, i. e. depois de pronto.

Peterson (1993), apresenta um trabalho abrangente sobre os principais aspectos envolvidos na construção e avaliação de uma RNA. Apesar de abordar superficialmente os aspectos de segurança relacionados a aplicação de RNA em SCS e não tratar explicitamente do processo de certificação conforme as normas e padrões aplicáveis, apresenta uma visão geral desse processo, cujo objetivo é delinear os fundamentos para um desenvolvimento futuro de uma metodologia completa de V&V para redes neurais. Segundo o autor a maior parte do trabalho se aplica a todo tipo de RNA mas, alguns aspectos são específicos à mais comum das redes: a rede Perceptron de Múltiplas Camadas (*MLP*) treinada com a regra de retropropagação (*backpropagation*).

A primeira tarefa, quando se cogita uma implementação usando RNA, é a especificação de requisitos. Não se trata aqui de uma especificação de requisitos do estilo para software convencional, trata-se de um documento que expõe o problema, descreve as metas a serem alcançadas, as possíveis restrições conhecidas, como pode

ser medido o sucesso da implementação e qualquer outro tipo de informação que seja considerada pertinente. Podemos considerar este documento como uma especificação de alto nível já que pouco pode ser detalhado (Bedford, 1996). Este documento é a chave para uma posterior avaliação consistente do desempenho do sistema (Nabney, 1997) e será a base para uma validação final do sistema. É claro que tal documento deverá ser atualizado ao longo do desenvolvimento complementando as informações iniciais. Peterson (1993) observa que este documento é a base da especificação funcional, cuja versão final será escrita ao final do desenvolvimento, e não precisa ser necessariamente um conjunto completo de requisitos. Segundo o autor, este documento pode conter, além de metas a serem alcançadas, uma especificação de comportamento, uma metodologia padrão ou mesmo propriedades que o sistema deverá ter, tais como corretude e acurácia. A dificuldade desta tarefa reside no fato de que não é possível se conhecer de antemão o comportamento preciso de uma rede neural, já que esta pode ser considerada como um tipo de software parcialmente não determinístico.

Na construção de software convencional, o comportamento exato do produto implementado pode (ao menos em teoria) ser especificado antecipadamente em um documento de requisitos. Os requisitos e o comportamento são explícitos: uma dada entrada de dados resulta em um saída precisamente especificada. Por outro lado, para as RNA, pode restar um certo grau de incerteza sobre qual será a resposta para um dado conjunto de dados de entrada. Esta incerteza está presente mesmo quando o produto está em uso, e existe uma incerteza ainda maior antes que o produto seja construído.

Alguns requisitos, ou mais precisamente, restrições, são conhecidos, mas os requisitos primários do usuário (as metas) são objetivos cujo sucesso em alcançá-los é incerto. A tese de Peterson (1993) é a de que este software parcialmente não determinístico pode ser avaliado, verificado e validado segundo uma base que contém

outras características além de um conjunto completo de requisitos. Segundo ele quanto maior a flexibilidade, menor a necessidade de requisitos. A construção de uma rede neural é bastante flexível – somente uma pequena quantidade de esforço precisa ser despendido para mudar a estrutura da rede para uma nova execução dos casos de treino. Dessa forma não é difícil de se proceder à construção de uma RNA com um conjunto mínimo de requisitos. Por outro lado, a análise dos resultados requer um esforço considerável, aspecto não observado por Peterson (1993). Considerando a natureza e a fonte das possíveis bases para avaliação de uma RNA, Peterson (1993) aponta os seguintes:

1. Requisitos, ou restrições, que possam ser especificados. Por exemplo, pode ser possível especificar precisamente propriedades não desejadas (propriedades de segurança) e também especificar precisamente a fonte e o formato dos dados de entrada.
2. O conjunto de metas dos usuários ou desenvolvedores. Se estas metas podem ser quantificadas, numa declaração contendo probabilidades, por exemplo, então a extensão na qual estas são alcançadas pode certamente ser uma base para avaliação. Mesmo metas não quantificáveis que sejam claras e não ambíguas podem ser usadas como base.
3. Uma metodologia de desenvolvimento padrão adotada pelos desenvolvedores. A avaliação compararia o método de desenvolvimento real com o padrão (norma). De forma similar, numa avaliação de um banco de dados de casos de treino, uma base possível seria uma propriedade tal como acurácia ou completeza.

4. Uma especificação do que o sistema faz, escrita após a construção do sistema. Esta especificação deve quantificar a incerteza final e delinear as fronteiras de comportamento aceitável e inaceitável. E ao final do desenvolvimento uma validação compararia o produto contra sua especificação final.

Bedford et al. (1996) discute uma série de aspectos particulares das RNA para a sua aplicação em SCS, e propõem uma série de requisitos genéricos apresentando uma proposta de norma segundo uma série de considerações mínimas para o desenvolvimento de módulos de software contendo RNA para emprego em SCS. Tais requisitos poderiam então ser adaptados para criar um processo de certificação apropriado a uma norma particular para Redes Neurais Artificiais Críticas quanto a Segurança (RNACS). Segundo os autores, a perspectiva para a certificação de RNA treinadas é melhor do que para os sistemas de software convencional, devido as propriedades de interpolação das RNA, já que estariam menos sujeitas a conter os erros, do tipo discreto, encontrados nos sistemas de software convencional. Para identificar os requisitos de uma norma de certificação para RNA eles colocam as seguintes questões: Quais atributos de uma RNA tornariam a seu processo de certificação diferente da certificação do software crítico convencional? Qual é a forma mais apropriada de certificar um processo de confecção de uma RNA? O que significaria verificar e validar uma RNA?

Morgan et al. (1995) aponta que a principal diferença entre as RNA e as implementações convencionais de software é que as RNA são treinadas através de exemplos, tomados de um conjunto de dados. Para a certificação de uma implementação convencional é necessário que a especificação não contenha ambigüidades e seja, dentro

de certos limites, completa. Uma especificação convencional descreve o comportamento do sistema para circunstâncias dentro das quais ele deve operar. Garantir que um sistema atenda às suas especificações é o método principal de avaliação dos procedimentos convencionais de certificação. Para Bedford et al. (1996), caso se preserve esta abordagem para a certificação de RNA, a natureza da especificação deve ser alterada radicalmente. Com base nas propriedades de um processo de desenvolvimento para uma RNA, eles descrevem alguns requisitos que a norma deve especificar:

- Como os objetivos, ou requisitos, de alto-nível para um módulo de RNA devem ser obtidos;
- O quê deve ser feito para garantir que os dados de treinamento representam adequadamente a aderência aos objetivos de alto-nível;
- Que tipo de redes podem ser usadas e como cada tipo deve ser designado de forma não ambígua (o tipo da rede deve especificar a sua arquitetura, número de camadas, conectividade), a regra de aprendizado e outros aspectos particulares.

Os princípios básicos do desenvolvimento de software convencional, quando aplicáveis podem, e devem, ser utilizados no processo de desenvolvimento de uma RNA. Por exemplo, Nabney (1997) observa que os erros devem ser reconhecidos, diagnosticados e tratados o mais cedo possível ao longo do processo de desenvolvimento. E no caso específico das RNA deve-se procurar antecipar os tipos de erros que provavelmente ocorrerão e incorporar controles práticos para monitorá-los e minimizá-los.

Peterson (1993) propõe uma terminologia especifica para o desenvolvimento de RNA, pois segundo ele a terminologia existente foi desenvolvida para fornecer uma base conceitual para o software convencional, mas esta base não é ampla o suficiente para suportar a avaliação de redes neurais. Segundo ele *avaliação* é o processo de:

1. Desenvolver um exame que pode ser a base para julgamentos corretos,
2. Administrar o exame, e
3. Usar os resultados para fazer um julgamento sobre o quê fazer a seguir.

Se existem "pontos fracos" na modelagem, então o propósito da avaliação é encontrá-los. Caso contrário, o propósito da avaliação é confirmar que não existem "pontos fracos". Este termo *avaliação* tem um conceito amplo, na concepção de Peterson (1993) englobando as atividades de testes, *walkthroughs*, auditorias, inspeções, verificação e validação. O aspecto que requer um esforço maior de criação durante uma avaliação é o desenvolvimento dos testes apropriados. O principal problema é que os testes tem que ser compreensíveis. Deve existir um número suficiente de casos de testes óbvios, medianos e de fronteira.

(Nabney, 1997, Morgan, 1995, Bedford, 1996) observam que o aprendizado indutivo não envolve nenhum desenvolvimento de software propriamente dito, já que o software para executar e treinar o modelo é independente da aplicação e dos dados. Sob certos aspectos este software é similar a um compilador no desenvolvimento de software convencional, com o processo de treinamento similar a compilação e os parâmetros do modelo treinado comparável ao código de máquina. Este software é totalmente algorítmico, e portanto pode ser certificado para uso em aplicações relacionadas à segurança com os métodos de certificação existentes. Métodos formais

matemáticos tais como Z, VDM e PVS (NASA-GB-001-97, 1997) poderiam ser usados para escrever a especificação deste código.

3.1 Dados para uma RNA

Na construção de uma RNA os dados são usados para determinar a rede. É muito importante que estes dados formem um conjunto representativo de dados reais que serão encontrados no campo.

Nabney (1997), observa em relação aos dados que a distinção entre o software convencional e o não convencional não é muito clara. Quando modelamos qualquer sistema complexo do mundo real, são necessários alguns dados, nem que sejam somente para calibração e validação. A maioria dos tópicos discutidos nos trabalho estudados são relevantes sempre que dados do mundo real são usados para tal propósito. Da mesma forma, a computação neural pode representar somente parte da solução de um problema, por exemplo, a programação convencional pode ser necessária para pré-processar os dados antes de ajustar um modelo. Nabney (1997) chama a atenção para o fato de que até 40% do esforço total do desenvolvimento de uma RNA são despendidos na coleta, análise e pré-processamento dos dados.

Peterson (1993) faz uma série de observações acerca dos dados. Os dados precisam ser *compreensíveis* no senso de que devem existir casos de treino para cada um dos aspectos que devem ser reconhecidos pela RNA. Além disso, a amostragem dos dados deve ser mais frequente em áreas do espaço de entrada que se modificam mais rapidamente. Os dados devem ser *corretos* já que erros de treinamento irão mapear erros correspondentes no software. Deve-se procurar evitar a existência de contradições

e inconsistências nos dados. Se os dados de entrada variam com o tempo eles devem ser o mais atuais possíveis. Algumas dessas características são difíceis de identificar e quantificar. Todos os dados devem ser documentados de forma que seja possível identificar a sua origem, o seu formato, seu escopo, o porquê de serem considerados acurados e os detalhes de qualquer pré-processamento que tenha sido aplicado. Isto é muito importante para o caso de se querer repetir um resultado.

A coleta de dados amostrados do sistema a ser modelado deve seguir um procedimento previamente estabelecido, medidas de custo devem ser estabelecidas e outras informações anteriores, por exemplo, o quão suave a função deve ser, restrições operacionais das variáveis.

Uma análise preliminar dos dados coletados pode ter vários objetivos tais como a visualização dos dados para compreende-los, extração de características, pré-processamento de dados ausentes e com ruído (Nabney, 1997).

Para a coleta, análise e seleção dos dados (Peterson, 1993) sugere que, além do questionamento e escrutínio, sejam usada técnicas adicionais. Por exemplo, plotar os valores de cada característica dos dados e procurar anormalidades, tais como pontos fora da curva (*outliers*) e descontinuidades. Outros problemas adicionais dos dados de treinamento podem ser diagnosticados através de técnicas estatísticas. Usando estas técnicas é possível identificar *outliers*, i. e. pontos muito afastados das concentrações principais; subconjuntos influentes (*influential subsets*), i. e. pontos ou subconjuntos de dados que podem fornecer mais do que a sua parte de influência nas saídas do processo de treinamento; colinearidades, i. e. situações nas quais os valores de uma característica são uma combinação linear dos valores de outras características. No caso de identificação desses problemas deve-se estudar como eles afetam os dados de treinamento e, se for o caso, tomar as medidas necessárias para remediar esses

problemas, tal com excluir um *outlier* ou adicionar dados de forma a remover uma colinearidade.

Bedford (1996) observa que os dados de treinamento devem ter algum elo de ligação (rastreabilidade) com a especificação de alto-nível de forma que se possa estabelecer algum critério que defina a conformidade desses dados com estes requisitos de alto-nível.

Lisboa (2001), observa que a dificuldade em distinguir informações úteis de informações redundantes é comum a todos os sistemas de inferência e também que o pré-processamento dos dados é um dos pontos chave na capacidade de generalização da rede.

O pré-processamento dos dados pode ser desejado de forma a tornar os padrões procurados mais "visíveis", através da diminuição da quantidade de informação. Apesar de existir uma certa perda de informações, os benefícios da diminuição da dimensão das entradas compensa segundo Bishop (1995b).

Bedford (1996) considera que numa aplicação típica o módulo de RNA receberá dados de outros módulos e no caso de SCS seria imperativo que o pré e o pós processamentos fossem registrados.

Segundo (Nabney, 1997) a computação neural, particularmente em uma abordagem Bayesiana, permite quantificar e formalizar algum conhecimento anterior. Isto torna mais fácil testar a validade das assunções, o que também é uma parte importante da verificação do desempenho da RNA treinada.

Bishop (1995b) apresenta uma abordagem segundo a qual as RNA são vistas como algoritmos para o reconhecimento estatístico de padrões com base em uma estrutura teórica bem fundamentada. Somente assim as RNA podem ser usadas com sucesso no tratamento de problemas não triviais e os aspectos relacionados à

Verificação e Validação (V&V) de sistemas contendo RNA podem ser abordados. Segundo o autor, no contexto dos problemas de classificação a maioria das aplicações de RNA é usada na forma de *discriminante não linear*, i.e. a própria rede é usada para decidir como classificar uma nova entrada. Mas, existe uma interpretação muito mais poderosa para o uso das RNA nesse contexto. O aspecto chave é fazer a distinção entre os dois estágios no processo de classificação: *inferência* e *decisão*. No estágio de inferência o objetivo é determinar as probabilidades de pertinência posteriores de uma dada entrada em cada uma das classes. Estas probabilidades podem ser subseqüentemente usadas na tomada de decisões, tal como a associação das entradas às classes. Nesse caso, o papel da RNA é predizer as probabilidades, com o subsequente processo de decisão executado separadamente. Através de um arranjo para o qual as saídas da rede aproximam as probabilidades posteriores pode-se explorar uma série de resultados, muitos dos quais não estão disponíveis caso a RNA seja usada como um simples discriminador não linear.

3.2 O Processo de Desenvolvimento da RNA

Segundo a maioria dos trabalhos estudados (Lisboa, 2001, Partridge, 1995, Peterson, 1993, Nabney, 1997) as RNA aplicáveis a SCS são do tipo *feedforward MLP* e *RBF* (Função de Base Radial) treinadas com a regra *backpropagation*. Nabney (1997) observa que cerca de 70% a 80%, das aplicações de monitoração e controle processos emprega redes desses tipos.

O processo de treinamento de uma RNA consiste no ajuste de parâmetros do modelo que usa os dados. O número de camadas e o número de nodos da RNA devem

ser obtidos empiricamente, já que não existe uma regra para isso (Russell, 1995). O número de nodos tem uma relacionamento íntimo com o super-ajuste (*overfitting*), uma escolha errada pode levar a um baixo desempenho. Se escolhe-se uma rede muito pequena o modelo será incapaz de representar a função. Se escolhe-se uma rede muito grande ela será capaz de memorizar todos os exemplos mas não terá uma boa capacidade de generalização para entradas desconhecidas.

Sabe-se (teorema de Kolmogorov) que uma RNA *feedforward* com uma camada intermediaria é capaz de aproximar qualquer função contínua a partir de suas entradas e uma RNA com duas camadas intermediarias pode aproximar qualquer função. Mas o número de unidades em cada uma das camadas intermediárias pode crescer exponencialmente em relação ao número de entradas. Uma das estratégias possíveis é a utilização de algoritmos genéticos (Russell, 1995) para otimizar a busca no espaço de estruturas de rede. É mais comum o uso de técnicas de *hill-climbing* que modificam seletivamente a estrutura de uma rede. Existem duas formas para se fazer isto: começar com uma rede grande e ir diminuindo, i. e. removendo conexões e nodos (*pruning*) (Bishop, 1995b) , ou começar com uma rede pequena e ir aumentando o seu tamanho, i. e. número de nodos na camada intermediária (*growing*) (Bishop, 1995b). Outras técnicas de otimização existentes são a *optimal brain damage* e a *tiling algorithm* (Russell, 1995). As técnicas de validação-cruzada (*cross-validation*) são úteis para auxiliar na decisão de quando se obtém uma rede do tamanho certo.

Nabney (1997) comenta que uma consequência do modo como as RNA são treinadas é que os parâmetros do modelo são a única parte específica de uma aplicação. A interpretação humana desse parâmetros é difícil e menos precisa do que a interpretação do código fonte algorítmico de alto nível do software convencional. Isto implica que a avaliação de sistemas de RNA é necessariamente estatística por natureza.

Como os casos de segurança (*safety cases*) são tipicamente escritos em termos de limitações das probabilidades de falha, isto pode ser uma vantagem real.

Partridge (1996) afirma que na computação neural a natureza da versão implementada, i. e. a rede treinada, é determinada pelas condições iniciais – o conjunto de treino, a configuração da rede e os valores de uma variedade de parâmetros. Assim, explorando os efeitos de diversidade dos elementos das condições iniciais, o potencial de geração da diversidade das várias opções pode ser determinado. O resultado é que conjuntos de versões de redes podem ser sistematicamente organizados para serem diversos. A Figura 3.0 ilustra a metodologia de Partridge (1996). Após decidir que um dado problema é potencialmente implementável através de uma RNA (bloco 1, Figura 3.0), os dados disponíveis são organizados em conjuntos de treinamento, aceitação e validação e os tipos de rede adequados são identificados (bloco 2). Experimentos de prototipação são usados para determinar as arquiteturas de rede mais efetivas e os níveis individuais de confiabilidade e diversidade esperados (bloco 3). Dada está informação básica, um sistema de versões múltiplas é modelado para suportar a confiabilidade de sistema necessária - conjuntos de versões, características de diversidade e estratégia de decisão tal como o voto da maioria (bloco 4). Segundo (Partridge, 1996), uma estratégia produtiva para sistematizar a diversidade é "super-produzir e escolher" – i. e. gerar um número de diversas redes treinadas maior do que o necessário para o sistema (bloco 5), e então selecionar subconjuntos diversos maximizados para o sistema de software multiversão final (bloco 6), usando por exemplo, um procedimento heurístico. O sistema finalizado é então validado e uma especificação *de facto* é confeccionada (bloco 7).

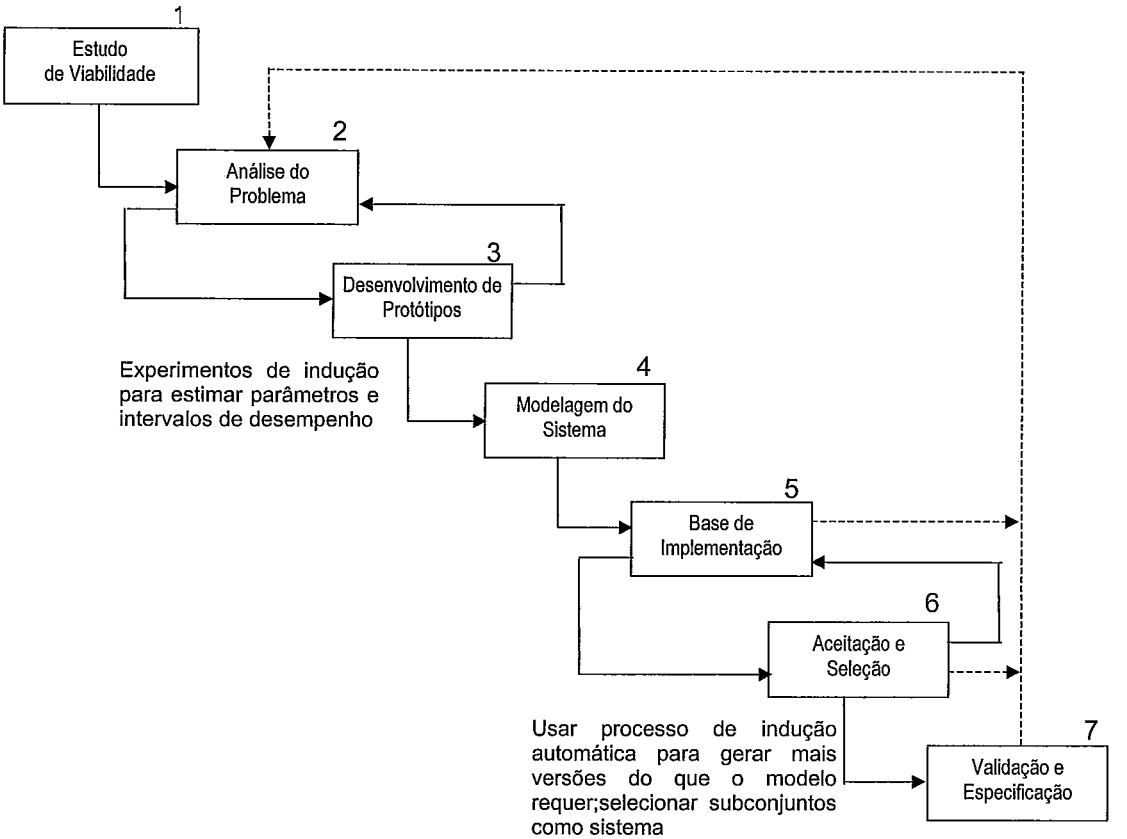

Figura 3.0 – Metodologia de Partridge usando o princípio da diversidade (Partridge, 1996).

Um exemplo é ilustrado na Figura 3.1. Os experimentos de prototipação (bloco 3) indicam que redes Perceptron com Múltiplas Camadas (MLP) podem ser treinadas para alcançar 98% de correção enquanto que redes de Função de Base Radial (RBF) alcançam somente 85% de correção. Mas, o experimento indica que existe uma alta diversidade em uma implementação com esses dois tipos de RNA. Consequentemente, o modelo do sistema (bloco 4) irá usar os dois tipos de RNA juntas com uma estratégia de "concordância máxima" para obter um sistema ótimo. Variações nas arquiteturas individuais das redes, nos conjuntos de treinamento usados e nas iniciações aleatórias dos pesos das redes resultam em um conjunto de redes treinadas MLP e RBF (bloco 5). Uma heurística é então usada para selecionar deste conjunto seis redes individuais que são altamente diversas (bloco 6). As seis versões individuais selecionadas, cinco redes MLP e uma rede RBF, apresentam desempenhos individuais entre 84.79% (a rede RBF) a 98% como ilustrado na Figura 3.1 abaixo. Devido a alta diversidade entre estas seis

versões (0.85 numa escala de 0 a 1), os vários resultados de "máxima concordância" são significativamente melhores do que qualquer versão individual pode fornecer (Partridge, 1996).

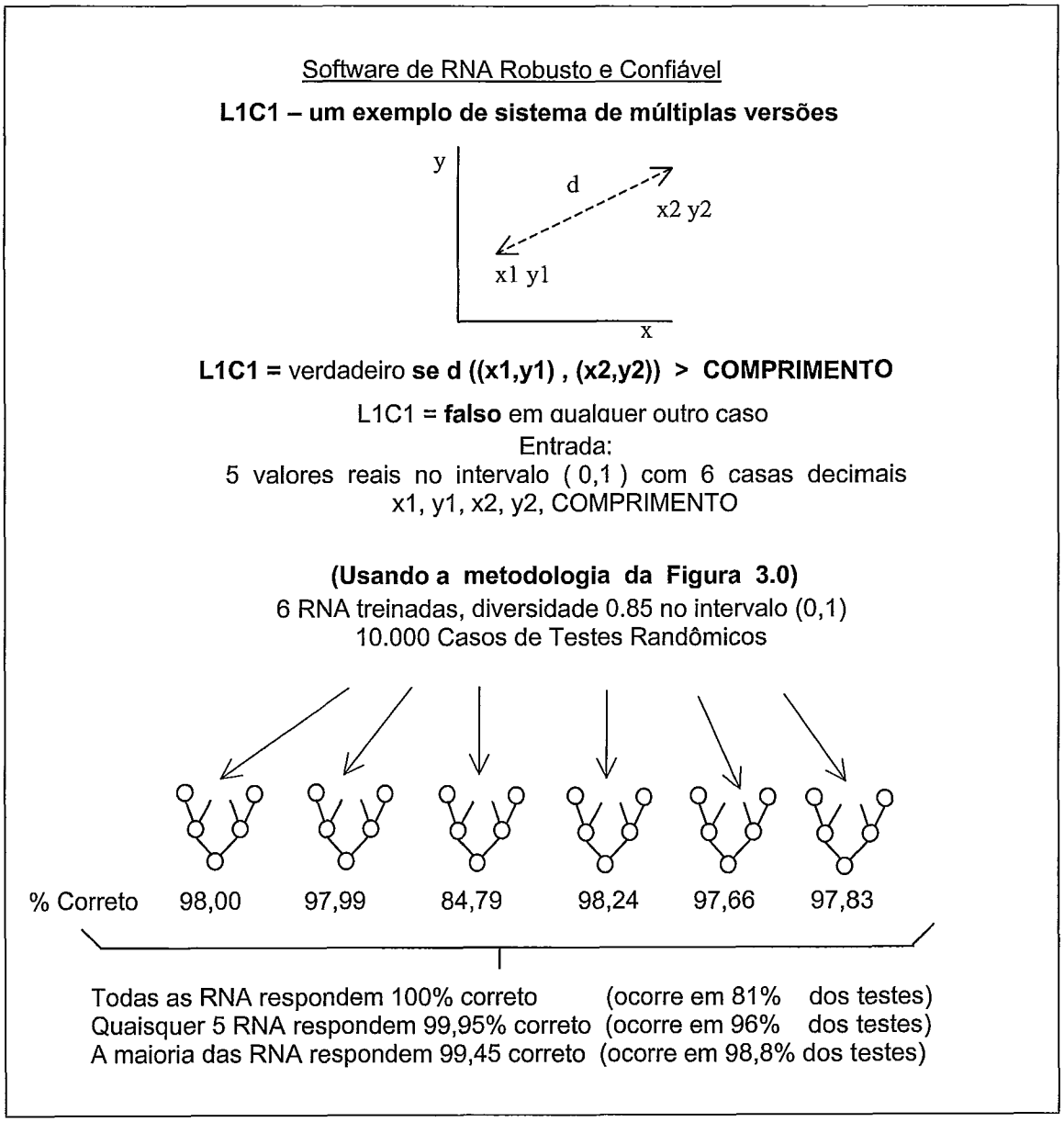

Figura 3.1 – Metodologia de Partridge usando o princípio da diversidade (Partridge, 1996).

3.3 Ciclo de Vida

O papel da especificação é fornecer uma descrição do comportamento do sistema que será usada como referência no seu projeto e implementação. No contexto do software convencional são usadas técnicas formais e informais. Independentemente da abordagem (formal ou informal) parte-se da identificação de um conjunto primário de requisitos que vão sendo refinados e decompostos ciclicamente até se alcançar um conjunto de elementos que podem ser implementados. Esta estrutura de desenvolvimento é conhecida por ciclo de vida, onde cada uma das etapas depende da etapa predecessora. Este ciclo pode ser percorrido várias vezes dependendo da necessidade, já que a cada ciclo percorrido as etapas são refinadas e os documentos pertinentes são complementados. O ciclo de vida deve ser estabelecido no planejamento do desenvolvimento do sistema pois ele é uma exigência no caso dos SCS e também porque é sobre ele que será aplicado o processo de Verificação e Validação (V&V) um dos pilares de sustentação do processo de certificação.

Esta sequência global de desenvolvimento e manutenção do software é chamada de ciclo de vida. O ciclo de vida consiste em um número de passos, que podem ser sequenciais ou iterativos, que começa com os requisitos de software, V&V do software, instalação e uso, modificações de engenharia, e termina tipicamente anos mais tarde quando o software é retirado de operação. A verificação ocorre durante o desenvolvimento do software conferindo cada etapa do desenvolvimento contra a versão do estágio anterior. A validação testa e avalia o sistema de software para garantir que ele executa de forma correta as funções pretendidas. A verificação acontece durante as fases de desenvolvimento do ciclo de vida do software enquanto que a validação é executada após o término do desenvolvimento do software.

A maioria das fases de um ciclo de vida de desenvolvimento de uma aplicação de computação neural é semelhante àquelas usadas pela engenharia de software convencional. As principais diferenças surgem em função das seguintes características particulares das RNA:

- Não é possível confeccionar uma especificação funcional precisa no início do desenvolvimento.
- O uso intensivo de dados significa que outras tarefas devem ser executadas.
- O desenvolvimento é obrigatoriamente iterativo (ciclo de vida cíclico).

Nabney (1997) constata que apesar de não existir um modelo de ciclo de vida definitivo, tal como para a engenharia de software convencional, a seguinte abordagem de princípios tem sido usada com sucesso na prática:

1. Desenvolvimento do modelo: o desenho do modelo baseia-se no conhecimento prévio e na análise dos dados. Vários modelos devem ser treinados, e. g. modelos simples para testes comparativos e modelos mais complexos para tentar capturar mais características do problema e aperfeiçoar a acurácia.
2. Integração: interface com outros software.
3. Operação e manutenção: validação e retreinamento contínuos do modelo.

Nabney (1997) apresenta o modelo de ciclo de vida da Figura 3.2 e comenta que este é bastante genérico e similar a outros na literatura. Contudo existem dois pontos importantes a observar acerca do ciclo de vida apresentado que frequentemente são omitidos:

1. Os dados do sistema a ser modelado não são simplesmente coletados. Informações existentes podem e devem ser usadas para estruturar e restringir a solução.

2. Durante a operação, o modelo deve ser validado e retreinado quando necessário. Técnicas desenvolvidas permitem medir, e monitorar, a qualidade do desempenho da RNA durante a operação. Por exemplo, intervalos de confiança (Bishop, 1995b) permitem associar uma medida de confiança, no prognóstico da RNA.

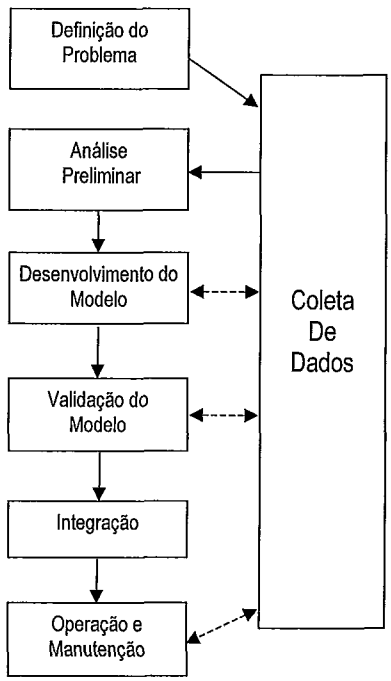

Figura 3.2 – Modelo de ciclo de vida. As linhas sólidas denotam o caminho principal, e as linhas tracejadas os pontos onde mais dados podem ser necessários e o desenvolvimento deve retornar a um estágio anterior (Nabney, 1997).

O ciclo de vida de desenvolvimento apresentado na Figura 3.3 abaixo é proposto por Peterson (1993).

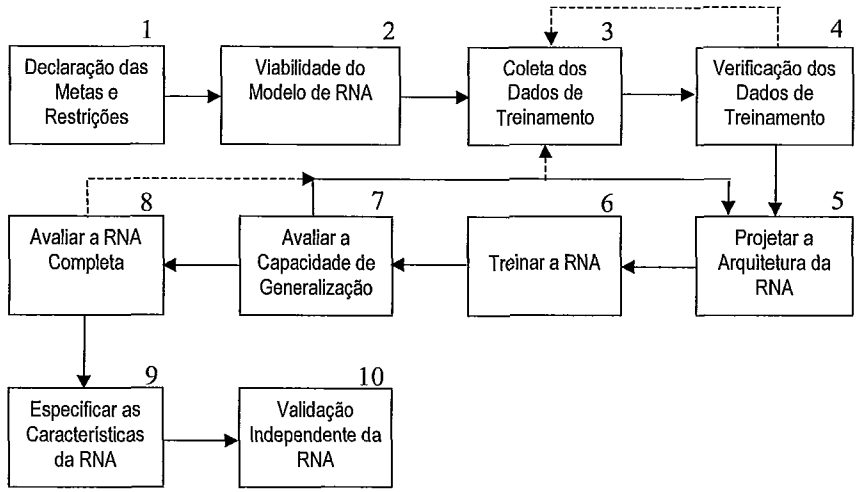

Figura 3.3 – Fases no desenvolvimento de uma rede neural artificial (Peterson, 1993).

Os passos de avaliação são mostrados nos blocos 2, 4, 7, 8 e 10. Note que um passo de avaliação é sempre um ponto de bifurcação, algumas vezes a bifurcação está implícita, porque um julgamento sobre o quê fazer a seguir está sendo feito.

O desenvolvimento de uma RNA começa com o objetivo de atingir uma determinada meta. Esta meta deve ser declarada explicitamente de maneira a fornecer uma base inicial para a atividade de avaliação. Além disso, qualquer restrição que seja conhecida, tal como, o formato dos dados de entrada, ou o ambiente de programação, deve também ser especificada. Estas atividades são mostradas no Bloco 1. Na atividade 2 deve-se examinar as metas e restrições e fazer um julgamento se a RNA é o tipo certo de tecnologia para esta tarefa. Se o problema é considerado adequado para o emprego de RNA então os dados são coletados e examinados (3 e 4). A verificação consistirá na avaliação da compreensibilidade e acurácia dos dados e no julgamento da sua adequação para o treinamento da rede. Pode também ser preciso neste estágio decidir

sobre uma estratégia de pré-processamento para os dados de treinamento. Na atividade 5, a tarefa de modelagem da arquitetura da rede consiste em decidir o tipo de rede a usar, o número de nodos da camada intermediária, o número de camadas e o cálculo dos pesos. Nas atividades restantes a RNA é avaliada segundo a sua capacidade de generalização e são especificadas as suas características finais para um validação independente.

Em relação a confecção de uma RNA Bedford (1996) considera que o processo de construção e treinamento devem ser subdivididos e gerenciados. Esta seria a forma mais construtiva e apropriada para certificar o processo de desenvolvimento de uma RNA. Assim, devido às várias tarefas associadas ao desenvolvimento de uma RNA, o processo de confecção deve ser gerenciado de forma competente. Da mesma forma, o pessoal envolvido na coleta dos dados de treinamento deve estar consciente da importância do pré-processamento dos dados coletados. Portanto a norma deve especificar: A extensão do conhecimento necessário sobre redes neurais artificiais necessário à gerência e a equipe de desenvolvimento. Eles antecipam que todo o pessoal envolvido não precisa ter a mesma competência, sobre redes neurais, que a gerência mas devem entender: 1) a área pela qual são responsáveis; 2) as áreas que dependem deles; e 3) as áreas das quais eles dependem. Eles colocam também a seguinte questão para a gerência: Que modelo de desenvolvimento deve ser usado para o módulo de RNA? Isto implica em tomar as sub-tarefas envolvidas no desenvolvimento de uma RNA, delinea-las, indicar suas interdependências, ou independência e, possivelmente, identificar indivíduos ou equipes responsáveis pela sua implementação e testes. Eles antecipam (Bedford, 1996) que uma norma de certificação exigirá que a RNA forneça algum tipo de estimativa de confiança do seu próprio desempenho. Isto poderia ser uma estimativa de confiança direta gerada pela rede ou algum outro tipo de medida de

confiança calculada. Colocam as seguintes questões acerca da construção do software necessário à implementação da RNA:

- Se devem ser usados métodos formais de base matemática, ou argumentação rigorosa para desenvolver o software que implementa a RNA;
- Quais métodos de garantia da qualidade devem ser empregados no treinamento da rede;

A garantia da qualidade da rede treinada vai depender principalmente dos métodos usados e do cuidado tomado na coleta dos dados de treinamento. Exemplos de fatores individuais que podem contribuir e que devem ser considerados por uma norma são, por exemplo: intervalo de operação de sensor, linearidade (ou não linearidade) e valores de falhas.

3.4 Treinamento e Generalização

Aprender para uma RNA significa ajustar os parâmetros, normalmente chamados de pesos, para aproximar uma função desconhecida com base em um conjunto de dados amostrados daquela função. Os pesos são ajustados durante o processo de treinamento através da minimização de uma função de erro, ou custo. Esta função erro é uma medida global da discrepância entre os valores alvo e os valores prognosticados pela RNA.

A função erro geralmente é muito complexa (uma vez que para as RNA ela depende dos pesos de uma forma altamente não linear). A maioria dos algoritmos baseiam-se no fato de que o mínimo global de uma função é um ponto onde todas as

derivadas parciais da função são iguais a zero. Nabney (1997) observa que esta é uma condição necessária mas não suficiente. Estes pontos podem ser mínimos locais (i. e. neste ponto o valor da função é um mínimo para uma pequena região ao redor do ponto) ou uma região plana. Ficar preso numa região de mínimos locais "ruim" implica que a solução não é ótima, o que vai resultar numa não conformidade da especificação. A maioria dos algoritmos de otimização encontram um mínimo local próximo ao seu ponto de partida, e portanto, é importante fazer várias execuções de treinamentos múltiplos com pontos de partida aleatórios. Todos esses dados devem ser devidamente registrados.

Com frequência, os dados usados no treinamento estão contaminados com ruído, e. g. devido a imprecisão dos instrumentos de medida. O objetivo é evitar aprender o ruído mas aprender a estrutura subjacente de forma que o modelo seja capaz de *generalizar* quando receber entradas não vistas no treinamento. Assim se uma RNA aprende os dados de ensino e os ajusta perfeitamente, ela é dita "super-ajustada" (*overfitted*). Isto normalmente leva a um pobre desempenho de generalização, que pode ser constatado ao se testar o modelo com um conjunto de dados independente.

Geralmente o "super-ajuste" está associado a um modelo complexo para o qual a função computada pode variar muito entre os pontos dos dados de treinamento. Normalmente, espera-se que a função varie de uma forma relativamente suave e isto pode ser incorporado no treinamento do modelo por meio de técnicas de *regularização* (Bishop, 1995b) o que normalmente melhora a generalização do modelo treinado.

A verificação desse processo de treino é uma atividade completamente diferente da verificação de um software convencional, de forma que os métodos de verificação existentes não são diretamente aplicáveis (Peterson, 1993). A avaliação dos dados de treinamento usará como base os próprios dados de treinamento, a documentação

contendo as informações desses dados e as características. Os testes iniciais serão através de perguntas e escrutínio.

Um sistema de software, antes de ser usado, deve ser avaliado em relação a vários aspectos, entre o quais, a validação do desempenho, é um dos mais significativos. É frequente a confusão entre Verificação e Validação. A validação refere-se a determinar se o sistema pode ser executado dentro de um nível aceitável de desempenho em termos de acurácia e eficiência. A verificação refere-se a determinar se o sistema foi implementado corretamente. Um sistema precisa ser verificado antes, para só então poder ser validado.

Um certo número de técnicas foi estabelecido para a validação do desempenho de sistemas. A validação de um aprendizado do sistema difere daquela de um desempenho do sistema. Por exemplo nós avaliamos a capacidade de generalização em um aprendizado de sistema mas não em um desempenho do sistema. Uma RNA pode ser tanto um sistema de aprendizado como um sistema de desempenho. Na fase de treinamento, é um sistema de aprendizado, e após o treinamento torna-se um sistema de desempenho.

Um aspecto chave do desenvolvimento das RNA com um risco mínimo de uso é o processo de avaliação contínuo como uma parte integral das atividades de construção. Esta avaliação mede o erro da RNA quando usada para dados novos (não vistos no treinamento). Este é o erro verdadeiro da RNA, em oposição ao erro aparente que é medido nos casos de treinamento. Se o erro verdadeiro pode ser estimado, então as decisões de modelagem como o número de entradas, o número de pesos ou o tempo de treinamento podem ser obtidos usando-se um processo de "tentativa e erro". Segundo este processo escolhe-se um valor inicial arbitrário, que meça o valor do erro verdadeiro, e se segue na direção do erro verdadeiro decrescente até que um erro

mínimo tenha sido alcançado. O erro verdadeiro é uma medida da capacidade de generalização da rede e o risco verdadeiro em usá-la. Portanto o objetivo da modelagem e construção de uma RNA é minimizar o risco minimizando o erro verdadeiro (Peterson, 1993)

Uma forma de estimar o erro verdadeiro é dividir os casos de treino em duas partes, uma para treinamento e outra para testes. O erro nos casos de teste serão uma estimativa do erro verdadeiro da rede. Com um número limitado de casos, uma função de aproximação melhor será encontrada se todos os casos forem usados no treinamento. Nessse caso usamos a técnica conhecida por validação cruzada (*cross-validation*). Nesse caso deixamos de fora alguns casos durante o treinamento e usamos os deixados de fora para estimar o erro. Esse processo é repetido várias vezes deixando-se de fora diferentes casos de treino. A média de todos erros estimados é uma estimativa do erro verdadeiro quando a rede for treinada com todos os casos. Uma última forma de estimar o erro verdadeiro é usar uma estimativa algébrica tal como o erro final de predição (*FPE - final prediction error*) (Peterson, 1993) definido por FPE = MSE(1+2S/N), onde MSE é o erro médio quadrático dos casos de treinamento, S é o número de pesos do modelo e N é o número de casos de treino. N e S são valores conhecidos e o MSE é um cálculo simples usando os dados de treino e a rede treinada.

Peterson (1993) observa que esta técnica de tentativa-e-erro pode ser usada para otimizar várias características da arquitetura da rede neural e pode ser usada para otimizar o tempo de treinamento da rede. Para uma rede de uma camada intermediaria, o número de unidades da camada pode ser otimizado começando-se com uma unidade e ir aumentando de um em um até que o FPE, ou outra estimativa de erro, comece a aumentar. O número de entradas pode ser reduzido através da sua ordenação em relação a sensibilidade da saída às várias entradas, eliminando uma por vez enquanto o erro

verdadeiro não aumente. De forma similar, os pesos podem ser ordenados através da magnitude ou pelo efeito sobre o erro por uma mudança no peso. Pode-se então zerar um-por-um (*optimal brain damage* eliminando a conexão) até que o erro comece a crescer. Finalmente o tempo de treino pode ser otimizado através de treinamento contínuo enquanto uma estimativa de erro estiver decrescendo.

3.5 Interpolação, Extrapolação e Densidade de Dados

Entender a distinção entre interpolação e extrapolação é fundamental para a confiabilidade da RNA. Tipicamente as RNA são muito mais precisas quando interpolam do que quando extrapolam. A definição usual desses termos (que os novos pontos de dados estão no interior ou exterior respectivamente da região do espaço de entrada contendo os dados de treinamento) não é precisa e não são fáceis de medir em mais de uma dimensão (Nabney, 1997). Em vez disso, nós dizemos que áreas do espaço de entrada onde a densidade dos dados de treinamento é alta são regiões de interpolação, enquanto áreas do espaço de entrada onde a densidade dos dados de treinamento é baixa são regiões de extrapolação. Intuitivamente a idéia é a de que áreas onde o modelo tem muita informação o seu comportamento é restrito, enquanto que nas áreas onde tem pouca ou nenhuma informação a sua funcionalidade é irrestrita e portanto pouco confiável.

Outra forma de medir a confiabilidade dos prognósticos de uma RNA é gerar intervalos de incerteza (*error bars*) que fornecem um intervalo de valores prováveis que levam em consideração possíveis fontes de variação em torno do valor prognosticado de saída. Existem diferentes tipos de intervalos de incerteza correspondendo a diferentes

fontes de variação: ruído de entrada, ruído de saída e incerteza de parâmetros. Quanto mais amplos são estes intervalos, menor a certeza da RNA sobre a sua saída (Bishop, 1995b).

Foi mostrado em (Bishop, 1995b) que barras de erro Bayesianas, que levam em consideração a incerteza dos pesos da rede devido às amostras de dados no treinamento, estão relacionadas a densidade de dados de entrada, que associam esses intervalos de incerteza a detecção de novidades.

Para Nabney (1997) e Bishop (1995a) nas aplicações críticas de RNA, a novidade nos dados de entrada deve ser monitorada, de forma que as saídas que possam não ter a confiabilidade necessária sejam identificadas. Isto poderia ser parte do sistema de monitoração e poderia ser uma forma de avaliar quantitativamente o desempenho, e testar as assertivas feitas durante o desenvolvimento. Além disso, a maioria dos sistemas de RNA assumem que o gerador de dados é *estacionário*, i. e. que não se modifica com o tempo. Esta assunção pode ser testada através da monitoração das novidades das entradas de dados, avaliando continuamente a precisão das saídas da RNA.

Roberts et al. (1996), propõe um índice de validação para as RNA. Segundo a proposta dos autores este índice é obtido através de uma abordagem Bayesiana para problemas de classificação. Eles criam uma região de entradas adicional (além das identificadas para o problema) de forma que toda entrada nova que não faça parte das regiões de entrada das classes identificadas pertence a esta classe adicional. Assim eles podem monitorar as entradas novas da rede, podendo com isso detectar a ocorrência de problemas, por exemplo.

Os princípios básicos para avaliação de uma RNA, segundo Nabney (1997), são os mesmos do software convencional:

- Verificar que a função RNA foi desenvolvida de uma forma controlada e planejada (i. e. a qualidade do processo e da metodologia usadas.)
- Verificar que a função RNA atendeu aos testes e atende à sua especificação.

Nenhum desses dois aspectos, se aplicado sozinho, é suficiente para avaliar uma RNA: os dois são complementares afirmam os autores. Apesar de serem válidos e aplicáveis os princípios dos métodos para software convencional, a sua aplicação prática pode diferir. Por exemplo, repetitibilidade significa que os dados aleatórios iniciais devem ser registrados de forma que os resultados dos experimentos possam ser confirmados (repetidos) num estágio posterior.

O fato das RNA estarem baseadas em dados tem várias implicações para o desenvolvimento e avaliação. Os dados tornam-se parte do "programa", e portanto precisam ser submetidos ao mesmo controle que os outros documentos de projeto. Os dados precisam ser de boa qualidade e representativos do problema. Isto pode ser difícil de testar, e portanto avaliar, já que existem poucos testes estatísticos objetivos para estas propriedades. Tal como em qualquer sistema, certas assunções são feitas durante o desenvolvimento e uma das vantagens das RNA, em relação ao software convencional, é que é possível quantificar muitas dessas assunções e testa-las antes da implementação (Bishop, 1996b).

Nabney (1997) e Lisboa (2001) observam que a avaliação deve empregar métodos diversos para avaliar a estabilidade, por exemplo, e que nenhum método deve ser estabelecido como preferencial. E também não deve ser assumido que resultados similares serão obtidos de métodos diferentes para uma dada avaliação.

Depois que o sistema está pronto e foi exaustivamente testado e avaliado, e todos os problemas corrigidos, deve ser preparada uma especificação final da rede. A

especificação final deve incorporar todas as metas alcançadas e a declaração de requisitos iniciais. O propósito da especificação final é descrever completamente o escopo e o poder da RNA para identificar o que o sistema pode, e o que não pode, fazer. Esta especificação será a base primária para a sua validação final. Devem ser especificadas neste documento a fronteira entre o comportamento aceitável e inaceitável, e as fronteiras das entradas para garantir um nível de acurácia especificado. As medidas de desempenho especificadas são provavelmente de natureza estatística. Por exemplo, a taxa de erro sob várias circunstâncias, ou o erro médio quadrático sobre uma certa região do espaço de entrada poderiam ser especificados.

Segundo Peterson (1993) a tarefa final é demonstrar aos clientes que o sistema trabalha bem o suficiente para que eles possam se beneficiar com ele. Infelizmente, dado o presente *estado da arte*, é difícil explicar o processo de raciocínio que é usado em uma rede neural. Este fato dificulta a validação de uma RNA comparativamente à validação de um sistema especialista ou um software convencional. O único caminho que pode ser usado para convencer os clientes do valor da rede neural são os testes extensíveis e compreensíveis. O teste mais convincente do ponto de vista do usuário será uma demonstração compreensível das capacidades da RNA com dados novos, talvez fornecidos pelo próprio cliente. Uma validação independente do sistema é importante de forma que possam ser avaliadas as capacidades do sistema por meio de testes independentes (princípio da V&VI). O principal aspecto relacionado à avaliação das RNA é o da aceitação por parte do usuário (a validação) e posteriormente a certificação por um órgão competente.

Tal como para o software convencional, fornecer uma prova formal de correção de uma aplicação RNA geralmente é impraticável. Nabney (1997) e Peterson (1993), observam que se uma RNA é desenvolvida de forma controlada e sistemática, e

devidamente testada, então deve ser possível verificar e validar a aplicação com uma efetividade comparável à do software convencional. Nos casos estudados em (Nabney, 1997) os autores colocam que a tecnologia foi bem aplicada seguindo os princípios de boa prática da engenharia. Eles observam que a avaliação de tais aplicações requer uma boa compreensão das propriedades básicas das RNA. Os casos estudados, mostraram que durante o desenvolvimento de uma aplicação de RNA três tópicos geralmente são negligenciados: 1) documentação, 2) especificação e 3) testes das assunções. Por exemplo, nas aplicações avaliadas nenhum objetivo quantitativo foi definido na especificação e um modelo padrão de ruído foi usado sem nenhum exame do quão apropriado ele era para os dados das aplicações. Segundo os autores, as principais razões para esta negligência provavelmente são:

1. A natureza iterativa do ciclo de vida de desenvolvimento e o fato do desempenho do sistema não poder ser predito com antecipação, significa que na maioria dos casos não é apropriado definir uma especificação concreta no início do projeto. Baixos limites de desempenho podem ser derivados dos argumentos de segurança e da análise de custo/benefício. Após o estágio do estudo de viabilidade de execução a especificação tem que ser revisada e atualizada, de forma a torna-la mais precisa, mas isto não acontece na maioria dos casos.

2. Como as RNA são uma forma nova de encarar o software, não existem normas ou padrões para o seu desenvolvimento. Por exemplo, não existe uma definição clara do que deve constar na especificação de uma RNA.

3. Apesar da tecnologia das RNA estar atualmente se movendo da pesquisa para os produtos, muitas aplicações de RNA são desenvolvidas por acadêmicos. Apesar deles terem uma ótima compreensão da área e conhecimento da tecnologia, geralmente eles não tem os mesmos objetivos e experiência no desenvolvimento de software das empresas de software.

Nabney (1997) conclui dizendo acreditar que esses problemas sejam temporários devido a relativa juventude desta tecnologia. Além disso, existem várias aplicações de software convencional que são mal especificadas e documentadas, e portanto estes problemas não são exclusivos das aplicações de RNA. Segundo os autores, a motivação para uma padronização do desenvolvimento de RNA inclui os seguintes aspectos:

1. Prover diretrizes para o desenvolvimento de aplicações de RNA de sucesso tornam a tecnologia mais acessível.

2. O desenvolvimento das aplicações é mais fácil de controlar se feito de uma forma sistemática. Além disso, um melhor controle sobre o desenvolvimento é normalmente sinônimo de maior confiabilidade.

3. Um método de desenvolvimento padronizado permite também a padronização da verificação e validação. Isto não é completamente alcançável, tal como para o software convencional, mas é um objetivo a ser perseguido.

4. Resultados quantitativos. Para algumas regras de boa prática não existem técnicas padrão para testar corretamente as aplicações de forma quantitativa. Assim, para alguns aspectos das diretrizes correntes, o seu julgamento envolve a garantia de que o bom senso e a boa prática foram empregados. Isto não é desejável para aplicações que requerem os maiores níveis de integridade.

5. Qualidade e caracterização dos dados. Isto é essencial para o sucesso das aplicações, mas existem poucos testes úteis para determinar os pontos fracos desses aspectos.

6. Níveis de integridade de segurança. Em princípio, como as RNA são modelos estatísticos que fazem predições probabilísticas, elas devem se incorporar bem nos casos de segurança em função da sua natureza.

7. Controladores neurais. Algumas aplicações usam RNA como parte de um *loop* fechado de sistema de controle. Treinar essas RNA para esta tarefa é uma abordagem consideravelmente diferente da forma usual do regime de treinamento supervisionado que consideramos até agora, o que levanta também questões de estabilidade.

As bases para avaliação do sistema incorporam os critérios que serão usados para julgar a RNA declara Peterson (1993). Estas bases incluem documentos que foram criados durante a construção do sistema, características da RNA e os testadores humanos do sistema. Especificamente estas bases são:

1. A declaração de metas e requisitos (especificação final),
2. As amostras de teste e a sua documentação,
3. A descrição de como foi construída a rede neural,
4. Características ideais que são genéricas para quase todos os sistemas,
5. Especialistas humanos com a especialidade que as redes estão tentando emular, e
6. Usuários potenciais do sistema.

O espectro das técnicas para avaliação de redes neurais é limitado em virtude da sua forma singular de "programação". Existe portanto, uma grande dependência em relação aos testes e assim estes ser efetuados sob severa monitoração. As avaliações do sistema são mais abrangentes no escopo do que as avaliações executadas durante a construção da rede. Durante a fase de construção a meta é otimizar o desempenho do sistema. Na fase de avaliação do sistema, o objetivo é avaliar se os requisitos declarados, as metas e as necessidades desejadas foram alcançadas.

Um estudo acerca dos efeitos de falhas sobre a confiabilidade na operação das RNA de Bolt (1992) mostra que as RNA tem um potencial inerente para a tolerância a falhas. Tal potencial deve levar em conta a qualidade e quantidade dos casos de treinamento. Dessa forma, se a RNA teve um bom treinamento e apresenta um bom desempenho, a rede é capaz de responder adequadamente ainda que perca um certo número de dados de entrada (Bolt, 1992). Neste trabalho (Bolt, 1992), a funcionalidade foi visualizada num nível abstrato de forma que a tolerância a falhas resultante da natureza computacional das RNA pudesse ser analisada. O autor define uma metodologia através da qual os efeitos das técnicas de tolerância a falhas nas RNA podem ser avaliadas. O autor avalia também o conflito acerca da tolerância a falhas das

RNA na literatura. Ele explica que este conflito é resultado da falta de distinção entre o paradigma da computação neural e das RNA treinadas. Dadas as sua assunções implícitas ambas visões estão corretas na sua essência. As RNA tem potencial inerente para a tolerância a falhas dado um algoritmo de treinamento apropriado. Entretanto, os algoritmos atuais, tal como o de retropropagação, não desenvolvem configurações de pesos adequadas para isto. Segundo o autor (Bolt, 1992) a distribuição dos pesos, após a aplicação do algoritmo de retropropagação, não é uniforme. Dessa forma a perda de determinados dados de entrada pode ter uma influência maior no desempenho da RNA, diminuindo assim o seu potencial de tolerância a falhas.

Bedford (1996) afirma que uma norma deve certificar uma RNA segundo um alto padrão de segurança mas, não deve impor demandas impraticáveis aos desenvolvedores. Do ponto de vista da verificação e validação, uma das mais importantes tarefas avaliadas durante a certificação de um SCS, os autores declaram "*Felizmente, existem motivos para por mais peso nos testes de uma RNACS (Rede Neural Artificial Crítica quanto à Segurança) do que no software convencional de um SCS*". Isto porque o software convencional crítico se processa através de estados digitais discretos e uma certificação completa do produto de software não é possível devido ao número astronômico de estados possíveis. Em contraste, as RNA executam através de somatórios e ativações das entradas. Portanto existe uma melhor perspectiva de certificação desse produto, se o espaço de entrada de dados puder ser adequadamente coberto, já que existe uma melhor fundamentação para assumir a interpolação entre os casos de teste das RNA, do que para sistemas convencionais.

Alguns dos requisitos para o V&V não são afetados na mudança de software convencional para a computação neural. Por exemplo, o V&V deve ser executado por

uma equipe independente da equipe de desenvolvimento: Verificação e Validação Independente (V&VI).

Ainda segundo Bedford (1996), nos sistemas convencionais a verificação consiste na conferência do comportamento de parte do sistema ou módulo em relação a sua especificação. Segundo os autores, a especificação de desempenho para uma RNA não está escrita, ela está implícita nos dados de treinamento. Para conferir se uma RNA aprendeu os princípios contidos no conjunto de dados de treinamento, a capacidade de generalização da rede deve ser testada. A validação em sistemas convencionais envolve um teste compreensivo do módulo todo no seu ambiente operacional. Isto pode ser preservado na formulação da computação neural, mas os autores propõem que fatores particulares devem ser enfatizados e assim, que a equipe de V&V deve validar a RNACS através da investigação do comportamento da rede sobre todo o espaço de entrada .

Para investigar de forma completa e profunda o espaço de entrada, a equipe de V&V deve considerar como os módulos que geram entradas para a RNA podem falhar e que valores a rede vai "ver" no caso de tais falhas e os efeitos subsequentes sobre outros sistemas ou sub-sistemas. Do ponto de vista do julgamento de falhas, perigos e riscos os autores propõem como as RNACS devem ser integras nos procedimentos existentes de avaliação dentro deste contexto. São sugeridos pelos autores, sem nenhuma justificativa, os métodos FMEA e HAZOP para avaliação da RNACS. Segundo eles o risco é proporcional ao produto da criticalidade de um evento e da probabilidade com a qual ele é estimado ocorrer. Assim, a norma para uma RNACS deve especificar quais as possíveis abordagens para uma avaliação da segurança do módulo. A FMEA determinaria o efeito de falhas nos módulos que enviam dados para a RNA e as possíveis consequências, através da rede, de tais entradas. Tal análise poderia ser

comparada depois com os testes feitos pela equipe de V&V para tais eventos. A FMEA poderia subsequentemente ser divida em tipos de risco, por exemplo, de segurança, *status* da missão, seguridade, etc. Para tanto os desenvolvedores devem identificar os possíveis modos de falha do próprio módulo de RNA e as suas consequências. Uma Análise de Vulnerabilidade identificaria os perigos e os níveis de criticalidade destes. As saídas da rede que poderiam, ou podem, levar a tais eventos seriam estabelecidas. Dessa forma a operação da RNA seria investigada para estabelecer quais entradas iriam gerar tal saída. A técnica HAZOP é geralmente executada por uma equipe e a norma de certificação deveria fornecer diretrizes sobre o forma da equipe e das reuniões da equipe de HAZOP, e também as palavras guia para o seu uso. Normalmente, o objetivo do comitê HAZOP é investigar como o sistema pode desviar das desejadas intenções de projeto. A norma deveria indicar quais membros são responsáveis pela investigação de quais aspectos de desenvolvimento da RNA. Palavras guia poderiam ser organizadas na forma, em uma forma estruturada de listas de conferência, ou procedimentos de revisão.

A FMEA e a HAZOP de redes neurais não poderiam ser possíveis sem as informações que caracterizam a operação da rede. Portanto, os desenvolvedores tem que construir a rede de forma que todos os dados necessários estejam disponíveis para que se possa executar uma análise. Isto requer que os requisitos de segurança sejam considerados desde o início do processo de desenvolvimento da RNA.

3.6 Conclusão

Procuramos neste capítulo, através da discussão dos trabalhos estudados, criar um panorama abrangente de forma a cobrir os principais aspectos dos métodos e

técnicas atualmente empregados no desenvolvimento e avaliação de Redes Neurais Artificiais (RNA) para uso em Sistemas Críticos quanto à Segurança (SCS). Nosso enfoque concentrou-se principalmente nas características particulares das RNA em relação às aplicações de software convencional.

Pudemos constatar que não existe um conjunto mínimo de diretrizes que englobe os principais aspectos envolvidos no ciclo de desenvolvimentos e avaliação (Verificação e Validação Independente) das RNA, que permitam a confecção de uma estrutura sobre a qual possa ser aplicado um processo, com um mínimo de exigências, para a certificação dessas redes. Portanto, uma compilação de diretrizes obtidas a partir da literatura existente e das recomendações do trabalho conjunto da USNRC-EPRI (NUREG/CR-6316, 1995a) seria de interesse para aplicações críticas que necessitem empregar módulos de software contendo RNA.

Capitulo 4

4 Ciclo de Vida de Desenvolvimento de RNA para SCS

Vamos agora estabelecer a interseção entre a Engenharia de Software e o desenvolvimento de Redes Neurais Artificiais (RNA) para Sistemas Críticos quanto à Segurança (SCS). Como já comentamos, o desenvolvimento de software convencional já está relativamente bem consolidado através de várias metodologias existentes aplicadas com sucesso. O nosso objetivo principal é identificar uma série de diretrizes segundo as necessidades que as RNA impõem no seu desenvolvimento devido às suas características particulares. Estas diretrizes serão identificadas a partir da consolidação dos trabalhos estudados nos capítulos anteriores e dentre as estruturas levantadas no trabalho da USNRC-EPRI (NUREG/CR-6316, 1995a). Naturalmente, a identificação destas diretrizes levará também em conta de uma forma geral a nossa experiência no desenvolvimento de SCS (Martinez, 1986, 1988, Machado, 1996, Comerlato, 2000) e em particular a experiência adquirida no desenvolvimento de um caso estudo de SCS baseado em RNA.

Já observamos que o processo de certificação de um SCS tem uma grande dependência do processo de V&V aplicado ao sistema. Nosso principal interesse recaí sobre os aspectos relacionados ao software de SCS.

A confusão entre Verificação e Validação é frequente, cabendo portanto relembrar a definição desses termos. A validação refere-se a determinar se o sistema pode ser executado dentro de um nível aceitável de desempenho em termos de acurácia e eficiência. A verificação refere-se a determinar se o sistema foi implementado corretamente. Um sistema precisa ser verificado antes, para só então poder ser validado.

Existem várias definições para os termos verificação e validação nas normas e padrões de software existentes. A norma (IEEE-1012, 1998) para V&V de software, apresenta as seguintes definições:

- ***Verificação*** é o processo de determinação se os produtos de uma dada fase do ciclo de desenvolvimento de software atende aos requisitos estabelecidos durante a fase anterior, ou não.
- ***Validação*** é o processo de avaliação do software ao final do processo de desenvolvimento de software para garantir a conformidade com os requisitos de software.

Estas definições de V&V são válidas para o desenvolvimento de software convencional e, a princípio, não precisariam ser modificadas para serem aplicadas às RNA na nossa opinião, ainda que existam algumas diferenças entre as filosofias de implementação de software convencional e a de redes neurais. Apesar de demandarem uma série de outras atividades extras em relação ao software convencional acreditamos, tal como os autores dos trabalhos estudados (Peterson, 1993, Nabney, 1997, Partridge, 1995), que as RNA possam ser submetidas a um processo de V&V, desde que exista um ciclo de vida de desenvolvimento adaptado às necessidades das RNA.

Como vimos no capítulo anterior nenhum dos autores apresenta, explicitamente, em suas propostas uma estrutura ou conjunto de diretrizes para a aplicação de um processo de Verificação e Validação (V&V), ou que contemple aspectos de segurança no ciclo de vida de desenvolvimento de uma RNA. Esta preocupação tem fundamento pois, para que uma RNA possa ser empregada em um SCS é imprescindível que seja submetida a um processo de V&V, e dependendo do nível de integridade de segurança

este processo de V&V deve ser executado por uma equipe independente. A Verificação e Validação Independente (V&VI), segundo a norma (IEEE-1012, 1998) é o processo de V&V executado por uma organização com graus específicos de independência técnica, gerencial e financeira da organização responsável pelo desenvolvimento do software. O relacionamento organizacional da V&V com as outras entidades responsáveis de um projeto é mostrado na Figura.4.0.

O nosso trabalho visa aspectos técnicos das atividades de V&V mas, não podemos deixar de observar que apesar das RNA serem um tipo diferente de software, não deixam de ser software e este deve ser gerenciado. O sucesso ou fracasso de qualquer projeto de software depende fundamentalmente de como ele é gerenciado (Presmann, 1997).

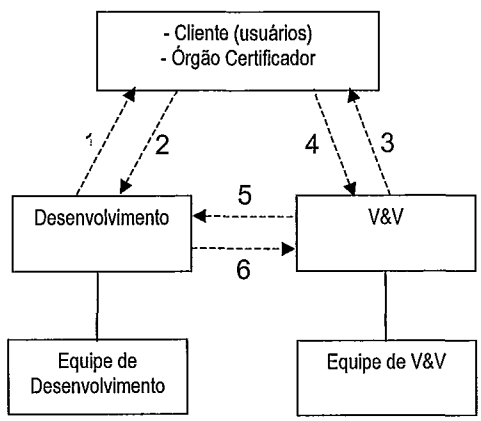

Figura 4.0 – Exemplo do relacionamento organizacional da V&V com outras responsabilidades do projeto (IEEE-1012, 1998).

Existe um consenso tácito generalizado acerca do processo de V&V segundo o qual a V&V é altamente antipatizada. É importante observar que uma V&V adequada tem um custo alto, muitas vezes equivalente ao custo do desenvolvimento, mas tradicionalmente é uma das primeiras atividades a ser eliminada quando um projeto enfrenta dificuldades de orçamento. A necessidade da V&V, especialmente pelos seus benefícios de redução de manutenção, raramente é compreendida e aceita pela gerência

de um projeto. Mas apesar disso, existe uma concordância geral entre os analistas de software de que a V&V sempre apresentará um custo-benefício positivo sobre a vida de um sistema (NUREG/CR-6316, 1995a).

A utilização dos métodos e técnicas, e da filosofia de V&V convencional para sistemas de Inteligência Artificial (IA) apresenta a vantagem de terem uma maior aceitação, em relação às técnicas novas desenvolvidas somente para sistemas de IA, em função da sua comprovada consolidação.

O desenvolvimento de um sistema computadorizado deve seguir um plano de desenvolvimento baseado em um ciclo de vida onde as fases desse ciclo de vida são bem definidas segundo os produtos de entrada e saída para cada uma dessas fases, tais como a especificação de requisitos e código para sistemas convencionais. No caso de SCS existem várias normas, práticas recomendadas e diretrizes para os diferentes domínios de aplicação. Para que esses SCS possam ser usados na sua atividade fim é necessário que um órgão independente faça a certificação do sistema. Esse processo de licenciamento é suportado pelas três principais atividades complementares que acompanham o desenvolvimento do SCS, do início à sua entrada em funcionamento e na sua manutenção, ao longo do seu ciclo de vida: Verificação e Validação (V&V), Análise de Segurança e Controle da Qualidade. Dessas três atividades as duas primeiras são as mais importantes já que a terceira está diretamente relacionada a aplicação das duas primeiras.

Portanto, inicialmente nossa intenção é identificar um tipo de ciclo de vida adequado ao desenvolvimento das RNA que suporte as atividades de um processo de V&V, abrangendo os aspectos técnicos envolvidos, principalmente os de segurança, na confecção desses módulos. Em resumo, estamos interessados em um ciclo de vida de desenvolvimento específico para RNA, de forma que esta possa ser incorporada a um

SCS, segundo às exigências impostas a este tipo de sistema. Tal ciclo de vida de software refere-se somente ao módulo de RNA, com relativa independência do ciclo de vida global do SCS como um todo. É óbvio que alguns aspectos, tais como os relacionados à segurança, estão diretamente relacionados, o que não impede um certo grau de independência, desde que sejam considerados os aspectos de segurança envolvidos. Dessa forma, abordaremos os aspectos técnicos que permitam estabelecer um substrato para as atividades de V&V, de forma que esta possa averiguar todos os comportamentos e desempenhos das funções específicas da RNA durante o desenvolvimento (verificação) e a implementação final (validação).

4.1 Ciclo de Vida de Desenvolvimento

O termo ciclo de vida refere-se às fases contidas entre o início e o fim do desenvolvimento de um sistema de software. Em geral, o ciclo de vida padrão para o desenvolvimento de software convencional engloba as seguintes fases: especificação de requisitos, modelagem, implementação, integração, instalação e manutenção. O ciclo de vida de software fornece uma abordagem sistemática para o desenvolvimento e manutenção de um sistema de software. Um ciclo de vida bem definido e bem implementado é fundamental para o sucesso na aplicação das técnicas de V&V.

Existem várias propostas de ciclo de vida para software, mas os dois tipos básicos de modelo de ciclo de vida são: o modelo sequencial e o modelo iterativo. O modelo sequencial é composto por uma sequência de atividades que não fornece nenhuma informação retrospectiva das fases posteriores para as fases anteriores. O modelo iterativo por sua vez, envolve uma repetição cíclica de execução das fases do

ciclo de vida. Como já constatamos, o desenvolvimento de uma RNA é cíclico por natureza e portanto o modelo iterativo é o que melhor se adapta à confecção de RNA. Devemos observar mais uma vez, que estamos considerando aqui a RNA como um "sistema" independente, ou seja, mesmo que sua aplicação final seja como um módulo de software no contexto de um sistema de software convencional maior nós estamos considerando a RNA como um sistema independente do ponto de vista do desenvolvimento.

O modelo de ciclo de vida iterativo é apropriado quando os requisitos não estão bem estabelecidos ou não são bem conhecidos. Além disso, podem existir aspectos e questões técnicas sobre qual a melhor forma de implementação do software de forma a atingir os objetivos desejados. Um modelo iterativo permite refinamentos sucessivos dos requisitos e melhorias na implementação através de uma série de desenvolvimentos. O problema de requisitos desconhecidos, ou pouco claros, é encontrado no desenvolvimento de aplicações de software convencional de domínios conhecidos e também no domínio de aplicações de novas tecnologias como é caso de sistemas que empregam técnicas de IA.

Existem várias propostas de ciclos de vida iterativos (McConnell, 1996), mas o três tipos principais são: Espiral (Pressman, 1997), Incremental (NUREG/CR-6316, 1995b) e Evolucionário (IEEE/EIA-12207, 1997). Estes modelos são recomendados para o desenvolvimento de software convencional e para software de IA (sistemas especialistas), onde é exigida uma alta confiabilidade de sistemas complexos (NUREG/CR-6316, 1995b).

A abordagem desses três modelos difere segundo as características empregadas nas iterações da implementação. O modelo espiral é utilizado quando são desenvolvidos iterativamente vários protótipos, onde geralmente estes protótipos são completamente

diferentes uns dos outros. O modelo incremental é usado quando vão sendo acrescentadas "camadas" de software ao módulo inicial de forma incremental a cada ciclo. O modelo evolucionário é semelhante ao modelo espiral, no que se refere a requisitos parcialmente estabelecidos no início do desenvolvimento, mas no modelo evolucionário os requisitos iniciais vão sendo refinados e complementados a cada ciclo do desenvolvimento, sem mudanças radicais na modelagem do software. A Tabela 4.0 abaixo apresenta as estratégias de desenvolvimento empregadas nos ciclos de vida discutidos.

Estratégia de Desenvolvimento	Define todos os Requisitos no Início?	Múltiplos Ciclos de Desenvolvimento?	Distribui Software Provisório?
Sequencial	Sim	Não	Não
Incremental	Sim	Sim	Depende da aplicação
Espiral	Não	Sim	Depende da aplicação
Evolucionário	Não	Sim	Depende da aplicação

Tabela 4.0 – Aspectos chave das estratégias de desenvolvimento empregadas em diferentes ciclos de vida (IEEE/EIA-12207, 1997).

Podemos constatar, pelo que já vimos até agora, que o ciclo de vida que melhor se adapta as necessidades de desenvolvimento de uma RNA, dentre os apresentados, é o ciclo de vida evolucionário. Uma característica particular das RNA pode ser considerada incremental no que diz respeito aos dados, já que a medida que o desenvolvimento avança a adição de novos dados pode ser necessária para o aprimoramento de determinadas características da RNA. Assim, ao invés de se acrescentar novas "camadas" de software como no ciclo de vida incremental para o desenvolvimento de software convencional, pode ser necessária a adição de novas "camadas" de dados no ciclo de vida evolucionário de desenvolvimento para as RNA. Observamos aqui que estas possíveis novas "camadas" de dados, no caso das RNA,

significam na realidade novas "camadas" de conhecimento sobre a aplicação para a qual a RNA está sendo desenvolvida. Esta estratégia de abordagem através de um ciclo de vida evolucionário, para o desenvolvimento de RNA, está de acordo com os trabalhos estudados na literatura.

Como já vimos o processo de desenvolvimento de RNA não se adapta aos processos de desenvolvimento de software como implementado no corpo das normas de software atuais. Apesar de haver pouco desenvolvimento de software, propriamente dito, na confecção de uma RNA uma série de outras atividades importantes deve ser executada para as quais não existe equivalência na engenharia de software convencional. Algumas dessas atividades particulares foram citadas por (Nabney, 1997):

- Devem ser obtidas quantidades adequadas de dados relevantes, que devem ser analisados e tratados, já que raramente o melhor desempenho é obtido com os dados no seu estado original.
- A rede neural deve ser treinada e monitorada de forma a garantir que o desempenho desejado seja obtido.
- A avaliação do desempenho permitirá avaliar a capacidade de generalização da rede para dados desconhecidos, i.e. não vistos no treinamento.

Apesar de uma RNA não ser um tipo de software programado da forma convencional, uma RNA não deixa de ser software, ainda que "programado" de outra forma (indutiva). É fácil notar, tanto pelo exposto nos capítulos anteriores, como no enfoque aos dados realçados nas observações de Nabney (1997) acima, que o conhecimento acerca da aplicação é refletido pelos dados uma vez que o software que

implementa e treina a RNA é independente da aplicação e dos próprios dados. Portanto, todo o conhecimento sobre a aplicação está contido nos próprios dados. Se pretendemos usar esse dados para gerar uma RNA para SCS, a segurança deve obrigatoriamente ser uma propriedade intrínseca desses dados.

Os dados para a confecção de uma RNA existem na forma de pares ordenados de conjuntos, onde cada conjunto de entrada corresponde a um determinado conjunto de saída. Portanto a relação de entrada e saída desses conjuntos representam os estímulos e respostas da RNA. Os diferentes conjuntos de dados de entrada representam o domínio operacional válido da RNA e os conjuntos de dados de saída da RNA representam o domínio operacional de respostas da RNA. Assim, qualquer conjunto de dados de entrada deve estar contido no domínio operacional de entrada da RNA para que esta funcione corretamente. Como devem então ser avaliados os aspectos de segurança relacionados à uma RNA? Numa primeira abordagem é necessário identificar os possíveis problemas que podem resultar em dados de entrada fora do domínio operacional de entrada da RNA.

Para tratarmos deste problema é necessário a introdução de alguns conceitos relativos à segurança de sistemas críticos. Como vimos a segurança é uma propriedade que deve ser considerada desde o início do desenvolvimento, não é um atributo a ser agregado ao sistema após a sua confecção. As várias funções de software de um SCS devem ser consideradas como subsistemas em um estudo da segurança de um SCS. Vários estudos aliados à experiência da indústria (Gardiner, 1999) indicam que a maior fonte de erros e falhas relacionados ao software de SCS são introduzidos na fase de especificação de requisitos e na interação do software com o seu hardware, e o ambiente operacional do sistema. Assim, a noção de falha de software só pode ser considerada, na maioria das vezes, no contexto operacional do sistema. Este é particularmente o caso da

segurança, onde as vulnerabilidades (perigos) manifestam-se de forma física no contexto do sistema. Consequentemente, qualquer análise de falhas de um modelo depende da capacidade do processo de modelagem de incorporar a perspectiva de contexto do sistema. Assim, uma análise dos mecanismos de falha na análise de vulnerabilidade deve ser estendida ao software a partir de uma perspectiva do sistema.

Como estamos abordando a RNA na forma de um sistema independente, seria portanto necessário uma análise de vulnerabilidade dos módulos ou sistemas que antecedem a RNA na sequência de execução do sistema. Mas, como na fase inicial de desenvolvimento não temos ainda uma especificação completa da RNA teríamos que aplicar uma análise de vulnerabilidade preliminar (AVP) a partir das vulnerabilidades identificadas do sistema como um todo. A AVP é usada nos primeiros estágios do ciclo de vida na fase de exploração conceitual de forma que sejam incluídas, nos estudos de viabilidade e nas alternativas de projeto, considerações quanto à segurança para identificar as funções críticas e vulnerabilidades do sistema. Estas vulnerabilidades ou perigos são julgados e priorizados, e são então estabelecidos os critérios e os requisitos de segurança para a modelagem. O processo é iterativo com a AVP sendo atualizada a medida que mais informações sobre o modelo são obtidas e modificações são efetuadas. Os resultados servem como linhas de básicas para posterior análise e são usadas no desenvolvimento de requisitos de segurança e na preparação da especificação de requisitos e metas da RNA, agora no contexto não só da aplicação mas também no de segurança da mesma.

As saídas da RNA também devem ser estudadas de forma a identificar possíveis saídas que estejam fora do domínio operacional de respostas da rede e as possíveis consequências dessas saídas ao longo da continuação na sequência de execução do sistema global. Dessa forma são identificados possíveis estados perigosos

(vulnerabilidades) que podem ser geradas para outros módulos de software, ou hardware, na sequência de execução.

Do ponto de vista de níveis de integridade de segurança, as RNA apresentam uma vantagem em relação ao software convencional já que elas permitem o estabelecimento de medidas de probabilidade de acertos (ou erros). Dessa forma pode-se estabelecer medidas de taxas de falhas para as RNA. Mas para isso é necessário que seja feito um estudo estatístico que leve em consideração aspectos anteriores e posteriores (estatística Bayesiana) e o domínio dos casos de treinamento a partir do qual a taxa é estabelecida. A norma IEC 1508 (IEC, 1995) por exemplo, estabelece taxas de falhas admissíveis para cada um dos níveis de integridade de segurança. Ou seja, a norma define o número máximo de vezes que se espera que, um sistema construído segundo um nível de integridade particular, falhe durante um dado período de tempo. A norma define ainda duas classes de sistemas que são usados de forma diferente. A primeira classe cobre sistemas que operam em "modo contínuo", cujas falhas são expressas em *falhas por ano*. A segunda classe cobre os sistemas que operam no chamado "modo de demanda". As taxas de falhas para este tipo de sistemas é expressa em termos de *falhas por demanda*, referindo-se a probabilidade de que o sistema vai falhar na operação quando solicitado para tal. A Tabela 4.1 apresenta as taxas de falhas para estas duas classes de sistemas segundo os níveis de integridade de segurança da norma IEC-1508 (IEC, 1995).

Nível de Integridade de Segurança	**Operação em Modo Contínuo** (probabilidade de uma falha perigosa por ano)	**Modo de Operação em Demanda** (probabilidade de falha na execução da sua função em demanda)
4	$\geq 10^{-5}$ a $<10^{-4}$	$\geq 10^{-5}$ a $<10^{-4}$
3	$\geq 10^{-4}$ a $<10^{-3}$	$\geq 10^{-4}$ a $<10^{-3}$
2	$\geq 10^{-3}$ a $<10^{-2}$	$\geq 10^{-3}$ a $<10^{-2}$
1	$\geq 10^{-2}$ a $<10^{-1}$	$\geq 10^{-2}$ a $<10^{-1}$

Tabela 4.1– Taxas de falhas para os níveis de integridade de segurança da norma IEC-1508 (IEC, 1995).

Podemos agora facilmente notar a necessidade de um ciclo de vida para as RNA que comporte os aspectos de segurança. A inclusão de elementos de segurança no ciclo de vida de desenvolvimento de software não é necessariamente um assunto novo já que a maioria das normas e padrões para SCS fazem esta recomendação (Storey, 1996, Gardiner, 1999).

Neste ponto antes de apresentarmos uma proposta para o ciclo de vida de RNA faremos, na seção seguinte, uma breve descrição de um estudo de caso. Tal estudo tem como objetivo levantar uma série de pontos críticos no desenvolvimento de RNA que devem também ser considerados em um ciclo de vida para RNA.

4.2 Estudo de Caso

O processo de identificação de radionuclídeos a partir de emissões gama implica no reconhecimento do espectro de radiação gama do radionuclídeo a ser identificado (Tsoulfanidis, 1983). Em geral o espectro de radiação gama é obtido por meio de um processo de detecção e contagem de emissões gama, por meio de sistemas contendo detetores, amplificadores e um multicanal. O envelhecimento e variações nas condições operacionais, entre outros, desses equipamentos eletrônicos usados no processo, causam distorções e variações no espectro gerado. A maioria desses sistemas de identificação de radionuclídeos só é capaz de identificar os respectivos espectros quando devidamente calibrados, necessitando assim de uma calibração periódica dos equipamentos.

O processo de calibração consiste em ajustar o equipamento de modo que o espectro de um radionuclídeo conhecido, usado como padrão, seja gerado corretamente

com seus picos de energia devidamente posicionados nos canais de energia correspondentes ao elemento.

No caso de um sistema computadorizado automatizado desse tipo, responsável pela monitoração de efluentes radiativos, e relacionado à segurança, a incapacidade de uma calibração automática compromete a disponibilidade e a robustez do sistema, podendo implicar em consequências perigosas para o processo que tal sistema monitora.

Com o objetivo de resolver um problema de calibração do Sistema de Monitoração Isotópica da Chaminé (SMIC) (Comerlato, 2000) da usina nuclear Angra 1 optamos pela aplicação de uma RNA para identificação de radionuclídeos, mais especificamente do radionuclídeo usado como padrão para a calibração (Eu-152). Este estudo foi abordado nos trabalhos (Canedo, 2002, Comerlato, 2002).

O SMIC é um sistema que monitora continuamente os efluentes da chaminé da usina. Trata-se de um sistema computadorizado com mais de quinze anos, cujos principais componentes são um minicomputador, um analisador multicanal, três detetores de Germânio, amplificadores, conversores analógicos-digitais e um sistema de tubulações controlado por um CLP (controlador lógico programável). É um sistema com alto grau de automatização necessitando pouca intervenção dos operadores. A função do SMIC é colher diversas amostras (particulados, iodetos e gases nobres) durante um período de tempo preestabelecido (24 a 48 horas) e depois analisar estas amostras através de uma aplicação de software para averiguar se o nível das emissões está abaixo dos máximos tolerados.

Em virtude da idade do equipamento, principalmente dos módulos eletrônicos do detetor multicanal o sistema apresenta problemas de calibração. Isto significa que o sistema não é capaz de efetuar uma calibração automática a partir do radionuclídeo

padrão e é assim incapaz de entrar em funcionamento, requerendo frequentemente a intervenção de um operador humano para uma calibração manual.

O detalhamento do trabalho desenvolvido para solucionar este problema através de RNA é apresentado no Apêndice B (Canedo, 2002), e apresentamos a seguir somente os aspectos de maior interesse para o desenvolvimento de RNA em SCS.

Optamos inicialmente, por ser a visão da aplicação do caso ideal (sem falhas), por experimentar a identificação de espectros através de uma RNA, partindo do trabalho de Keller (1994). Implementamos uma RNA que utiliza uma rede do tipo matriz linear hetero-associativa (OLAM) (Keller, 1994), tal modelo de rede permite identificar um espectro rapidamente em situações onde a quantificação é menos importante. Esta rede OLAM é útil na determinação da composição de uma amostra desconhecida quando o espectro é uma superposição linear de espectros conhecidos. Uma característica desta técnica é que ela utiliza todo o espectro ao invés de somente os picos individuais. Por esta razão, é potencialmente mais interessante para o processamento de dados de espectrômetros de baixa resolução (Germânio e Sódio).

Usamos os programas *Cspectr* e *Vspectr v. 1.0 beta* (ATOM, 1999) desenvolvidos pelo *V. G. Khlopin Radium Institute (KRI)* em colaboração com o *U.S. Department of Energy (U.S.DOE)* para gerar espectros de radio isótopos conhecidos para treinamento da rede OLAM que foi capaz de identificar os espectros individuais como também foi capaz de identificar os espectros conhecidos de vários "coquetéis" de amostras compostas por vários elementos. Observamos que para situações ideais a aplicação responde certo 100% das vezes. Neste ponto deveria ser executada uma análise de vulnerabilidade sobre as entradas e saídas da rede a fim de identificar possíveis falhas, mas já sabíamos de antemão que existiam problemas de deformação do espectro gerado pelos módulos eletrônicos, problemas estes que naturalmente surgiriam

nas respectivas análises. Partimos então para a deformação dos espectros para averiguar como a rede respondia.

Os espectros podem apresentar dois tipos de deformação: *translação* e *distorção*, e ainda uma combinação destas. A *translação* consiste no deslocamento de todo o "bloco" do espectro em relação à origem, ponto de energia 0. A *distorção* provoca o alongamento ou compressão da curva do espectro em relação ao eixo das energias, o eixo x, mantendo sempre constante a área sob a curva do espectro. Neste trabalho não foi considerado o tratamento da distorção do espectro.

Descrevemos a seguir os resultados obtidos no nosso experimento. Constatamos que um pequeno deslocamento do espectro causava problemas para a rede, que não era capaz de identificá-lo. Portanto, falhas nos módulos eletrônicos do sistema multicanal invalidam totalmente o ótimo desempenho obtido com RNA.

Tentamos então um prétratamento dos dados através da aplicação da transformada discreta de Fourier (Antoniou, 1979, Bracewell, 1965) ao espectro, passando a treinar a rede OLAM segundo as características de fase e amplitude do espectro. Obtivemos sucesso, pois identificamos uma característica invariável dos dados através de um préprocessamento, segundo o qual a posição do espectro deixava de ser importante para a sua identificação.

Para o caso de um espectro conhecido exposto ao detetor durante um tempo previamente estabelecido, que é o caso de um elemento padrão para calibração, podemos considerar outra característica invariante do espectro que é a área sob a curva do espectro que se mantém constante independentemente do tipo de distorção que o espectro esteja submetido.

Através da identificação de duas potenciais vulnerabilidades às quais os dados de um espectro podem estar sujeitos no caso de uma RNA identificamos um requisito de

segurança através da aplicação de um préprocessamento dos dados de entrada da rede de forma que a rede é capaz de reconhecer o espectro independentemente do seu posicionamento em relação à origem.

Tal estudo, além de expor a necessidade de um tratamento de dados no treinamento de uma RNA, evidencia fortemente a dependência, não só dos dados de treinamento que devem conter o conhecimento sobre o tratamento de falhas das entradas da rede, mas principalmente evidencia a dependência do próprio modelo da RNA em relação às falhas das entradas e as vulnerabilidades das saídas da RNA.

4.3 CIVES - Ciclo de Vida Evolucionário de Segurança para RNA

Como vimos nas seções anteriores, se desejamos RNA seguras para SCS, o próprio conceito de segurança deve estar embutido nos dados de treinamento e por conseguinte no próprio conhecimento que a rede irá adquirir sobre o modelo da aplicação. Dessa forma não é possível desassociar o ciclo de vida de software da análise de segurança.

Portanto, nossa proposta será inicialmente baseada em um Ciclo de Vida Evolucionário de Segurança - CIVES - para o desenvolvimento de RNA. Neste ciclo de vida estamos considerando as atividade de V&V, e também os aspectos de segurança relacionados à RNA, como mostrado na Figura 4.1.

Apesar de se tratar de um ciclo de vida de desenvolvimento para RNA, este é necessário para que se possa estabelecer um processo de V&V, a partir do ciclo de vida de desenvolvimento. Daremos enfoque para as etapas de especificação de requisitos,

para as que se relacionam aos dados, e para as etapas de testes, por estas serem as etapas significativas do ciclo de vida.

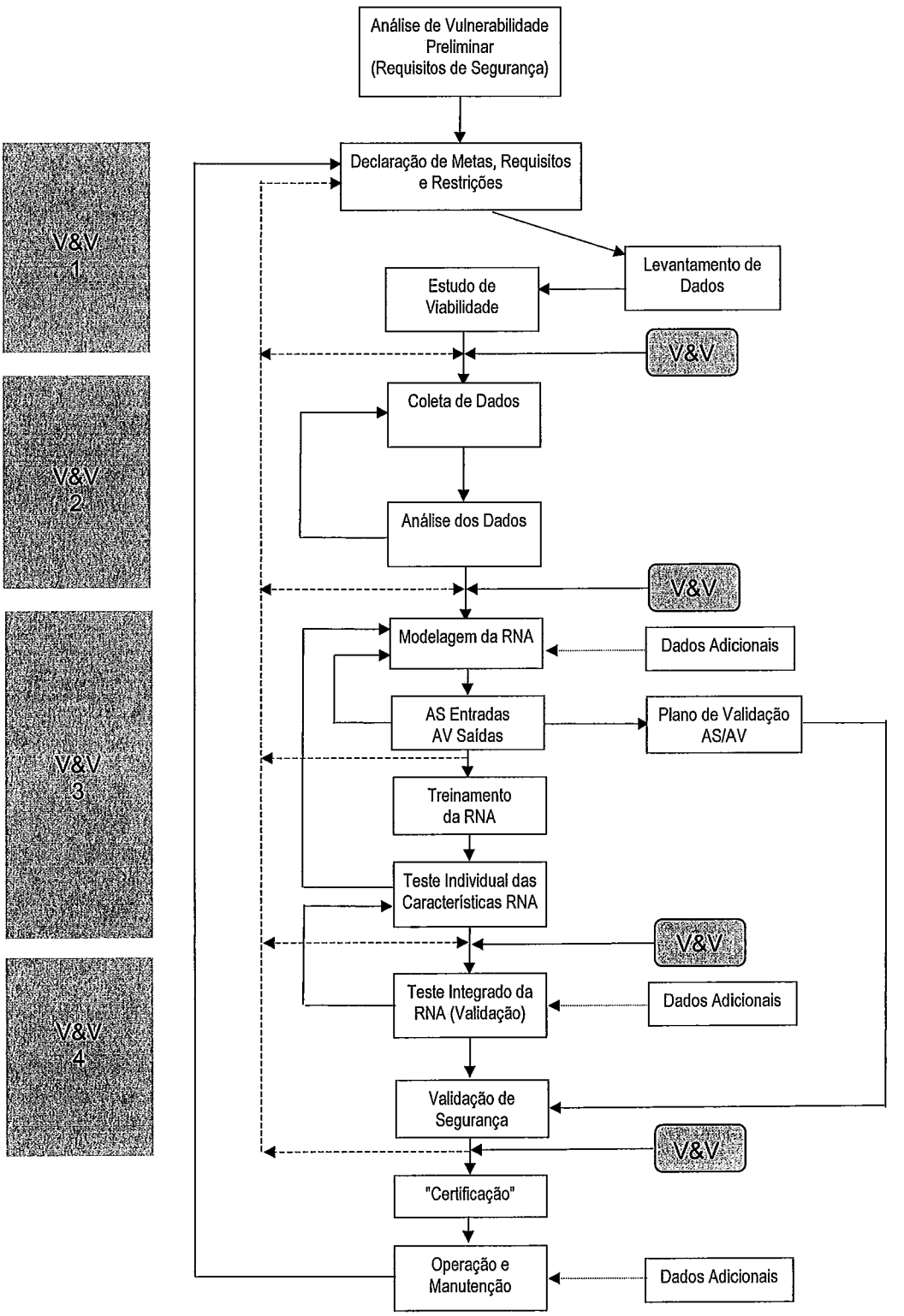

Figura 4.1 – Ciclo de vida evolucionário de segurança (CIVES) para o desenvolvimento de RNA para SCS.

Uma vez que as etapas que dependem das análises de segurança e vulnerabilidade são cruciais para a certificação da RNA, dedicaremos o capítulo 5 para o delineamento e seleção das metodologias apropriadas para as RNA. No momento já adiantamos que ao contrário de algumas sugestões de outros trabalhos, nosso foco concentra-se sobre técnicas e métodos de análise de segurança de software (Comerlato, 1998), e não hardware. As RNA que estamos interessados são mecanismos implementados a nível de software, sugerindo portanto uma aderência muito mais natural à metodologia abordada do que as oriundas especificamente do hardware.

Fazemos a seguir uma descrição de cada uma das etapas do nosso ciclo de vida proposto, apesar das fases de desenvolvimento serem praticamente auto explicativas pelo que já vimos até agora.

1. ***Análise de Vulnerabilidade Preliminar*** - Esta análise permite identificar as possíveis vulnerabilidades geradas por sistemas ou módulos que antecedem a execução da RNA e que podem gerar, ou contribuir para, dados de entrada fora do domínio operacional da RNA.
2. ***Declaração de Metas, Requisitos e Restrições*** – Esta é a especificação de requisitos inicial da RNA, a qual deverá ser complementada e verificada ao longo do desenvolvimento. Na realidade esta especificação de requisitos já deve incorporar os objetivos e metas básicos de segurança, falhas e vulnerabilidades que a RNA deve implementar.
3. ***Levantamentos de Dados*** – Deve ser feito um levantamento de dados sobre o problema que se quer resolver e dos dados que serão necessário para a sua solução.
4. ***Estudo de Viabilidade*** – O estudo deve avaliar se o emprego de RNA é viável para problema em questão e justificar esta escolha para a solução. Deve também

apresentar possíveis alternativas de modelagem, tais como, diferentes tipos de redes neurais, arquiteturas, etc.

5. ***Coleta de Dados*** – Deve ser coletado um conjunto de dados inicial que será considerado para o treinamento e teste da RNA, tanto no senso aplicativo como no senso de segurança.
6. ***Análise dos Dados*** – O conjunto de dados inicial deve ser analisado e estudado de forma a se obter um melhor conhecimento do problema. Devem também ser analisados possíveis préprocessamentos, que permitam extrair dos dados características particulares objetivando diminuir a dimensionalidade da rede, e possivelmente aumentando sua velocidade de execução.
7. ***Modelagem da RNA*** – Deve-se aqui avaliar os tipos de redes neurais mais apropriados para o problema, conforme o estudo de viabilidade com as alternativas de projeto inicialmente identificadas, segundo as características identificadas dos dados. O modelo pode ser modificado caso os requisitos de segurança assim requeiram.
8. ***AS Entradas / AV Saídas*** - Análise de segurança das entradas e análise de vulnerabilidade das saídas que podem exigir a mudança do modelo (7) ou das metas definidas na especificação (2).
9. ***Treinamento da RNA*** – Treinamento propriamente dito da RNA segundo as regras de treinamento aplicáveis identificadas na etapa anterior de modelagem da RNA.
10. ***Teste Individual das Características da RNA*** – Início dos testes das características individuais que se quer identificar. Deve existir pelo menos um teste para cada uma dessas características.

11. ***Teste Integrado da RNA (Validação)*** – Nesta etapa todas as características da RNA são testadas em conjunto e são avaliadas segundo a última versão atualizada da especificação de requisitos
12. ***Validação de Segurança*** - Aqui são testados os requisitos de segurança segundo critérios estabelecidos na especificação funcional e no plano de validação AS/AV (aqui é feita a validação final pela equipe de V&V). Deve ser gerado um relatório fornecendo detalhes sobre cada uma das caraterísticas de segurança testadas, ferramentas e equipamentos usados, os resultados de cada teste e as discrepâncias entre os resultados esperados e os obtidos.
13. ***Certificação*** – Neste ponto a RNA é submetida ao processo de certificação do órgão responsável de forma a obter a certificação.
14. ***Operação e Manutenção*** – Quaisquer modificações necessárias para correção, melhorias ou manutenção feitas após a certificação implicarão na necessidade de um novo ciclo desde a primeira fase até a uma nova certificação da RNA, e a execução de um novo processo de V&V ao longo dessas fases.

É importante observar que os pontos indicados pelas atividades de V&V ao longo do ciclo de vida podem implicar em modificações caso sejam identificados problemas que assim o exijam. Estas possíveis "reciclagens" são identificadas pelas linhas tracejadas para as atividades de V&V e pelas linhas cheias para as atividades de desenvolvimento, na Figura 4.1, que podem implicar em mudanças na RNA. Os principais aspectos avaliados são sempre os relacionados aos requisitos de segurança.

No ciclo de vida proposto nós consideramos quatro etapas distintas ao longo das fases de desenvolvimento, mostradas como as faixas cinza à direita do ciclo de vida, numeradas V&V 1 a V&V 4, na Figura 4.1. Ao final de cada una destas etapas deve ser

executada uma verificação antes que o desenvolvimento possa avançar para a próxima atividade do ciclo de vida de desenvolvimento. Ao final de cada uma das quatro etapas, antes do início da tarefa de verificação, a especificação de requisitos deve ser atualizada pela equipe de desenvolvimento, caso existam mudanças ou atualizações. A verificação vai então considerar esta nova versão da especificação na tarefa de verificação.

O rigor e a frequência das tarefas de V&V pode variar em função do nível de integridade necessário para o software, ou segundo outros acertos considerados em conjunto durante o estabelecimento do plano de desenvolvimento do SCS.

4.3.1 Conceitos Básicos da Etapa de Especificação de Requisitos

O papel da especificação de requisitos é fornecer uma descrição do comportamento do sistema que será usada como referência no seu projeto e implementação. No contexto do software convencional são usadas técnicas formais e informais. Independentemente da abordagem (formal ou informal) parte-se da identificação de um conjunto primário de requisitos que vão sendo refinados e decompostos ciclicamente até se alcançar um conjunto de elementos que podem ser implementados.

Para o tipo de software representado pelas RNA este modelo de especificação de requisitos não é apropriado por se tratar de um modelo indutivo, onde não se é capaz de gerar uma especificação de requisitos completa no início do projeto.

A especificação de requisitos inicial para uma RNA deve conter o máximo de informações disponíveis no início do desenvolvimento. A especificação deve conter, por exemplo, uma descrição do problema, os requisitos disponíveis, as restrições

conhecidas, os requisitos de segurança, o tipo e o formato dos dados de entrada. A especificação de requisitos deve ser atualizada a cada ciclo sempre que novas informações sejam obtidas e o processo de V&V deve verificar esta especificação a cada ciclo do processo de V&V como indicado no ciclo de vida proposto. Esta especificação será a base para a execução do processo de certificação do módulo pelo organismo responsável.

Dos trabalhos estudados foram levantadas as informações, apresentadas na Tabela 4.2 acerca do conteúdo de uma especificação de requisitos para uma RNA.

Tópicos acerca do conteúdo de uma Especificação de Requisitos para uma Rede Neural Artificial (RNA)
▪ Qualificação da equipe de desenvolvimento (funções, responsabilidades, qualificações)
▪ Descrição detalhada do problema a ser tratado pela RNA
▪ Metas do cliente (quantificáveis, sempre que possível) claras e não ambíguas
▪ Metas do desenvolvedor (quantificáveis, sempre que possível) claras e não ambíguas
▪ Restrições conhecidas (propriedades de segurança, por exemplo)
▪ Métricas quantitativas para avaliação de características da implementação
▪ Requisitos conhecidos quantificáveis (corretude, completeza, acurácia, etc.)
▪ Requisitos conhecidos de alto nível (claros e não ambíguos)
▪ Ambiente de desenvolvimento ("programação")
▪ Origem dos dados
▪ Tipo e formato dos dados de entrada, e outras informações pertinentes
▪ Tipo e formato dos dados de saída e outras informações pertinentes
▪ Dados invariantes; permitem a extração de características segundo conhecimento prévio
▪ Préprocessamento dos dados de entrada
▪ Pósprocessamento dos dados de saída
▪ Critério de distinção entre uma saída correta e uma saída incorreta
▪ Descrição ou especificação de comportamento
▪ Padrão de metodologia de desenvolvimento
▪ Critério para o estabelecimento do nível de integridade de software
▪ Critério para determinação da capacidade de generalização da RNA
▪ Fronteiras de decisão
▪ Fronteiras do domínio operacional
▪ Tipo da RNA
▪ Número de camadas

Tabela 4.2 – Informações que devem constar de uma especificação de requisitos para uma RNA (ao longo do processo de desenvolvimento).

Tópicos acerca do conteúdo de uma Especificação de Requisitos para uma Rede Neural Artificial (RNA) (cont.)
▪ Função de ativação ▪ Função de erro ▪ Função de ruído adequada selecionada após análise ▪ Regra de treinamento ▪ Número de casos de treinamento ▪ Número de casos de teste ▪ Documentação de referência

Tabela 4.2 (cont.) – Informações que devem constar de uma especificação de requisitos para uma RNA (ao longo do processo de desenvolvimento).

Os tópicos apresentados na Tabela 4.2 podem, e devem, ser usados para o esforço de desenvolvimento e também para o esforço da verificação e validação. O objetivo dos tópicos apresentados é realçar aspectos importantes da especificação, principalmente aqueles específicos às RNA. Todas as informações constantes da especificação de requisitos devem ser acompanhadas das respectivas justificativas sempre que possível, e principalmente quando existem diferentes opções para um determinado aspecto. Por exemplo, justificar porque a função de ruído utilizada foi a escolhida para o caso específico da aplicação em questão, ou justificar porque determinada meta de segurança foi definida em função da AVP utilizada.

Acreditamos ser relevante neste ponto abordar com maior profundidade o conceito de fronteira dos dados de treinamento. Do nosso ponto de vista seria prioritariamente recomendável que qualquer especificação de requisitos para a RNA de SCS, e por consequência os próprios dados de treinamento e o modelo da RNA contenha um novo conceito, que aqui será denominado de auto-proteção de fronteira.

O conceito de auto-proteção de fronteiras pode ser entendido como a especificação de mecanismos no modelo ou nos dados de uma RNA que possibilitem a

própria rede "responder corretamente" a um conjunto de dados de entrada que encontra-se fora de uma determinada fronteira operacional inicialmente especificada. Neste caso específico entenda-se que "responder corretamente" implica que para os conjuntos de dados de entrada que resultariam em saídas questionáveis, i. e. fora de uma fronteira estabelecida, sejam classificados de forma mais conservadora, ou segura, possível para a aplicação, como por exemplo, em problemas de identificação/diagnóstico a rede deverá ser capaz de responder "não-sei" a todo conjunto de entrada para o qual ela não foi treinada, protegendo assim o SCS de saídas além da fronteira especificada para a RNA.

Na realidade devemos conceituar o requisito de auto-proteção de fronteira em função do tipo da aplicação do SCS, ou melhor, em função do tipo de saída de uma RNA, contínua ou discreta.

No caso de uma RNA em que a saída é esperada como sendo uma função contínua de um determinado processo, como nos casos de RNA para a simulação ou controle de processos, a métrica que pode ser estabelecida para auto-proteger a fronteira dos dados é o próprio erro percentual das saídas, uma vez que estabelecido na especificação de requisitos o erro aceitável na saída, que pode ser entendido como precisão da resposta da RNA, o mesmo pode ser controlado por casos teste no domínio, inclusive nas fronteiras, dos dados operacionais de aplicações. É importante realçar, que ao contrário do software convencional, no caso das RNA com saídas contínuas a expectativa de comportamento correto entre dois casos teste é bastante razoável em função da natureza interpoladora da mesma, e portanto pressupõe-se que caso a RNA tenha "aprendido" o modelo do processo em questão o erro apresentado em regiões em torno dos casos teste, ou seja a precisão da resposta, seja equivalente ao do próprio caso teste.

Por outro lado, no caso em que a saída esperada para uma RNA seja discreta, caso típico dos problemas de classificação em geral, como os de reconhecimento, identificação e diagnóstico, a especificação de requisitos deve estabelecer uma métrica, e baseada nesta métrica um critério de distância, de modo a permitir que conjuntos de entrada que não tenham um determinado grau de certeza para a sua classificação sejam classificados como "não-sei" forçando deste modo a RNA a auto-proteger as suas fronteiras do aprendizado.

A idéia de distância é baseada no fato de que para qualquer problema de classificação, de uma forma direta ou indireta, a RNA através dos dados de treinamento produz centróides das classes aprendidas em função da concentração de semelhança entre os conjuntos de entrada/saída. Portanto a distância de uma determinada amostra de entrada aos centróides das classes treinadas para a RNA pode ser traduzida como o grau de pertinência da amostra à classe, como esquematizado na Figura 4.2. A especificação de uma distância máxima permite estabelecer que amostras de entrada que estejam além dessa distância para os centróides sejam consideradas como pertencentes a uma classe não treinada (aprendida) e implicando em uma resposta do tipo "não-sei" da RNA.

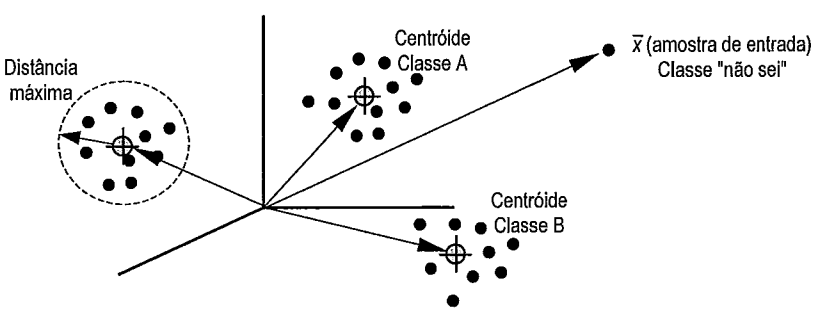

Figura 4.2 – Centróide das classes e a classe "não sei".

Observa-se que este conceito de auto-proteção de fronteira para problemas de classificação, apesar de contextos diferentes, possui a mesma base que o índice de Roberts (1994) no contexto de erros de saída para sistemas de diagnóstico médico.

Ressaltamos, que não necessariamente o modelo de RNA, ou a própria aplicação, produzam centróides de classes como os apresentados na Figura 4.2, mas necessariamente sempre deverá existir uma métrica, dos dados ou do próprio modelo de rede que permitirá isolar (auto-proteger) conjuntos de entrada/saída não treinados. Um exemplo simples é o das RNA que geram saídas que podem ser interpretadas como percentuais de acerto (grau de certeza) para a classificação como é o caso do estudo desenvolvido para o SMIC onde pela especificação de requisitos pode-se estabelecer um percentual, 50% por exemplo, que se abaixo a identificação não pode ser realizada.

Entendemos também que o estabelecimento de uma métrica, assim como a definição de uma distância máxima, para este tipo de problema de RNA, não é tarefa fácil, mas acreditamos que este objetivo deve sempre ser perseguido quando tratamos de aplicações do tipo SCS pois o retorno obtido em auto-proteger a fronteira a partir dos dados de treinamento resultará em um aumento da confiabilidade e da segurança.

Na área nuclear diversas pesquisas com RNA, principalmente para identificação e classificação de acidentes em reatores nucleares, estão adotando ao longo do tempo este tipo de critério como pode ser observado nos trabalhos (Alvarenga, 1997, Alves, 1993a, 1993b, Bartal, 1995, Pereira, 1998). Portanto nos SCS a especificação de requisitos deve, de modo incremental, definir uma métrica, e consequentemente uma distância limiar, de forma a estabelecer a fronteira de dados a ser protegida.

O ciclo de vida para uma RNA é necessariamente iterativo o que obriga a um acompanhamento rígido do desenvolvimento da especificação. Este é um aspecto importante no caso das RNA pois, como vimos (Gardiner, 1999, Leveson, 1995) a

maior parte dos problemas de segurança (requisitos não especificados) no desenvolvimento de SCS podem ser retrorastreados até a sua origem na especificação de requisitos. As constatações de Nabney (1997) acerca dos trabalhos estudados também apontam para o problema da não atualização da especificação de requisitos das RNA.

A Tabela 4.3 apresenta uma lista de conferência de possíveis falhas em uma especificação de requisitos para software, extraída do trabalho desenvolvido pela USNRC (*United States Nuclear Regulatory Commission*) em conjunto com o EPRI (*Electric Power Research Institute*) (NUREG/CR-6316, 1995a). Os tópicos apresentados são válidos também para uma especificação de requisitos de uma RNA, por exemplo, durante a execução de um processo de V&V. Deve-se observar entretanto que atributos como a completeza, por exemplo, não podem ser considerados integralmente no início do desenvolvimento devido a impossibilidade de se ter uma especificação completa para uma RNA, como já vimos.

Falha em relação à	Possíveis Falhas em uma Especificação de Requisitos de Software
Clareza	▪ Ambiguidade no escopo de qualificadores ▪ Ambiguidade no significado de palavras ▪ Ambiguidade nas referências ▪ Frases imprecisas ▪ Qualificadores abstratos
Completeza	▪ Seções incompletas ▪ Termos não definidos ▪ Lógica não articulada totalmente ▪ Requisito detalhado não totalmente decomposto e portanto ambíguo ▪ Metas do sistema não declaradas integralmente ▪ Função necessária faltando ▪ Especificação abstrata não decomposta adequadamente
Consistência	▪ Uma declaração contradiz diretamente uma outra ▪ Uma declaração é indiretamente inconsistente com outra ▪ Requisitos de sistemas cooperativos são inconsistentes ▪ Uma declaração é inconsistente com aspectos externos ao sistema conhecidos

Tabela 4.3 – Possíveis falhas na especificação de requisitos para software (NUREG/CR-6316, 1995a).

Falha em relação à	Possíveis Falhas em uma Especificação de Requisitos de Software (cont.)
Correção	▪ Falha em especificar o estado inicial quando não é igual a zero ▪ Requisitos não expressam as necessidades dos usuários - Erros objetivos - Referências incorretas a Normas, Padrões ou outros trabalhos - Erros de lógica ▪ Um valor ou variável errado ou faltando ▪ Os requisitos violam os princípios aceitos de engenharia ▪ Os requisitos superestimam ou subestimam os recursos computacionais associados à especificação ▪ Entradas ou saídas do sistema não descritas integralmente ▪ Os requisitos não incorporam o ambiente operacional ▪ Os requisitos não são possíveis devido a outros fatores do sistema (e. g. memória disponível) ▪ Os requisitos são excessivos para as necessidades operacionais
Explicitação	▪ Alguma coisa é implicitamente declarada por uma sentença mas não é explicitamente declarada ▪ Alguma coisa é logicamente implícita mas não é declarada
estabilidade	▪ Uma condição para uma meta é declarada mas não nenhuma medida é declarada ou implica em como determinar se a meta foi atingida ▪ Um requisito é logicamente não testável
Não identificável separadamente	▪ Sentenças compostas contendo dois ou mais requisitos ▪ Informações parciais sobre um único requisito distribuída em vários lugares

Tabela 4.3 (cont.) – Possíveis falhas na especificação de requisitos para software (NUREG/CR-6316, 1995a).

4.3.2 Conceito Básicos para as Etapas de Levantamento, Coleta e Análise de Dados, e Dados para Treinamento

Os conceitos básicos aplicáveis aos dados de treinamento para a confecção de uma RNA foram extensamente abordados no capítulo 3 de uma forma geral e com o enfoque de segurança nas seções prévias deste capítulo, portanto a partir dos trabalhos estudados levantamos e apresentamos na Tabela 4.4 apenas uma série de tópicos relevantes, que foram previamente discutidos.

Tópicos relativos aos Dados de Treinamento para o desenvolvimento de uma Rede Neural Artificial (RNA)
▪ Os procedimentos para coleta de dados devem ser especificados ▪ Qualificação do pessoal envolvido na coleta do dados (funções, responsabilidades, qualificações) ▪ Devem ser estabelecidas medidas de custo para obtenção dos dados ▪ Devem ser compreensíveis, ter significado ▪ Devem existir dados específicos para cada uma das características que a rede deve identificar ▪ Amostragem maior nos intervalos onde os dados se modificam mais rapidamente ▪ Devem ser corretos ▪ Devem evitar contradições e inconsistências ▪ Devem ser atuais ▪ Atenção às características difíceis de identificar e quantificar ▪ Devem estar documentados (origem, formato, escopo, justificativa do porquê são considerados representativos) ▪ Explicitar detalhes de qualquer préprocessamento e posprocessamento ▪ Restrições operacionais ▪ Precisão dos dados ▪ Análise preliminar (plotar os dados de cada característica, por exemplo) ▪ Análise de possíveis anomalias dos dados (descontinuidades, subconjuntos influentes, colinearidades, etc.) ▪ Medidas para tratamento de problemas encontrados nos dados ▪ Extração de características dos dados (possibilidade de diminuição da quantidade de informações) ▪ Tratamento de dados ausentes ▪ Tratamento de dados com ruído ▪ Rastreabilidade ▪ Intervalos de operação ▪ Valores de falha ▪ Identificação dos casos de treinamento ▪ Identificação dos casos de teste ▪ Número de casos de treinamento e número de casos de teste ▪ Função custo, ou erro, a ser minimizada ▪ Fazer várias execuções de treinamento com pontos de partida aleatórios para evitar mínimos locais e registrar os valores iniciais usados ▪ Evitar aprender o ruído, procurando aprender a estrutura subjacente dos dados (melhora a generalização) ▪ Evitar a repetição de casos de treinamento iguais ou muito semelhantes ▪ O treinamento deve ser rigorosamente monitorado e acompanhado ▪ Usar estimativas estatísticas e estimativas algébricas ▪ Usar Intervalos de incerteza para ruído de entrada, ruído de saída, incerteza de parâmetros ▪ Utilizar técnicas de validação cruzada para avaliar o desempenho da RNA ▪ Documentação de referência

Tabela 4.4 – Tópicos relativos aos dados para confecção de uma RNA.

4.3.3 Conceitos Básicos para as Etapas de Teste

A partir dos trabalhos estudados fizemos um levantamento dos tópicos relacionados aos testes dinâmicos de uma RNA, segundo as suas características particulares. Como já vimos as RNA tem uma grande dependência dos testes já que esta é praticamente a única forma de validação possível para uma RNA. Os aspectos referentes aos tópicos relacionados na Tabela 4.5 a seguir devem ser considerados tanto no desenvolvimento quanto na aplicação do processo de V&V.

Tópicos relativos aos Testes de uma Rede Neural Artificial (RNA)
▪ Devem ser compreensíveis, ter significado e estabelecidos segundo um plano de testes
▪ A qualificação da equipe de testes deve ser definida no plano de testes (funções, responsabilidades, qualificações)
▪ Um procedimento de teste deve constar do plano de testes
▪ Todos os testes devem ser documentados segundo um padrão estabelecido no plano de testes
▪ O plano de teste deve conter testes específicos para cada uma das características que se quer identificar
▪ Devem existir casos de teste para cada uma das características que a rede deve identificar
▪ Deve existir um número suficiente de casos de testes válidos e inválidos
▪ Devem ser considerados casos de teste com falta de dados
▪ Devem ser usados casos de teste reais sempre que possível
▪ Simuladores de dados podem ser usados para a geração de casos de teste o mais verossímeis possível
▪ Atenção especial deve dada às características difíceis de testar
▪ O teste por meio de especialistas humanos deve ser considerado sempre que possível
▪ Diferentes métodos para testar a estabilidade devem ser usados
▪ Devem ser testadas as fronteiras entre comportamento aceitável e não aceitável sempre que possível
▪ A taxa de erro deve ser testada sob várias circunstâncias
▪ As assunções da especificação de requisitos tem que ser testadas
▪ As restrições operacionais devem ser testadas
▪ Testes de dados com nível de ruído variáveis devem ser executados de forma a estabelecer os níveis de ruído suportáveis
▪ Testes de intervalos de operação
▪ Testes para valores de falha
▪ Testes relativos aos aspectos de segurança
▪ Número de casos de teste
▪ Documentação de referência

Tabela 4.5 – Tópicos relativos aos testes a serem aplicados na RNA desenvolvida.

Peterson (1993) sugere que as técnicas descritas a seguir sejam empregadas ao longo do desenvolvimento RNA, e também devem ser empregadas nas tarefas de V&V.

Inspeções. Os vários documentos devem ser inspecionados em relação a completeza e acurácia. Uma inspeção em particular deve ser feita procurando uma declaração específica da diferença entre uma saída válida e uma saída inválida, já que os testes são totalmente dependentes desta informação. O processo de treinamento é avaliado através do exame da documentação que foi preparada durante esta fase, garantindo que a rede foi treinada de um maneira efetiva. Após os testes terem sido completados, existe uma inspeção final cujo objetivo é a comparação dos resultados dos testes com as declarações iniciais das metas e requisitos.

Desenvolvimento de casos de teste. O principal problema é o desenvolvimento de um número suficiente de casos de teste. De forma a testar a rede adequadamente deve existir um amplo espectro de casos de teste válidos e inválidos, especialmente de casos próximos a fronteira de comportamento aceitável. Deve haver uma cuidadosa amostragem do espaço de entrada para se obter estimativas estatísticas confiáveis do erro verdadeiro. Se existe um conhecimento da distribuição das entradas que o sistema vai confrontar, então os casos de teste devem seguir a mesma distribuição. Se existem regiões do espaço de entrada para as quais o sistema tem dificuldades, então estas regiões devem ser enfatizadas no conjunto de testes. A amostragem deve ser mais frequente nas áreas onde o espaço de entrada muda com maior rapidez. Se o propósito da rede neural é o reconhecimento de imagens, então devem existir testes tanto para a imagem presente quanto para a imagem ausente de forma a garantir que a imagem é reconhecida quando está presente e não é reconhecida quando não está presente.

Quando dados reais podem ser obtidos para teste, então eles devem ser usados. Mas, geralmente o problema é a disponibilidade de dados reais para teste a um custo razoável. Nesta circunstância, considera-se a possibilidade de aumentar a compreensibilidade do conjunto de testes aumentado-o com casos de teste simulados.

Uso de simulações. O uso de simuladores para produzir entradas representativas para testar a rede construída é um método efetivo de avaliação do sistema. Se um simulador for usado, ele deve espelhar, o melhor possível, o ambiente no qual a rede será usada. A dificuldade com as simulações é o custo para se criar representações verossímeis do mundo real. Se a simulação de um ambiente inteiro não é possível, então cenários representativos do mundo real podem ser considerados para simulação.

Teste por especialistas. Frequentemente a tarefa de uma rede neural é uma na qual pelo menos alguns humanos a superam. Se for este o caso, então as capacidades da rede podem ser comparadas com aquelas do especialista humano. Por exemplo, medidas de desempenho, tais como percentagens dos padrões reconhecidos podem ser usadas para comparar a máquina com o especialista.

Teste por usuários. O último juíz será o usuário eventual do sistema. É essencial, portanto que alguns usuários façam parte da equipe de avaliação. Descobrir se o sistema atende os seus padrões de usabilidade. Descubra se eles aperfeiçoam suas decisões através do uso do sistema.

Teste para as características ideais. Uma abordagem inteligente é considerar cada uma das características ideais de uma rede neural e decidir como testá-la. As três características seguintes são ideais.

Teste para segurança. Existe alguma possibilidade de saída da rede que possa resultar em danos a pessoas ou equipamento? Se a resposta for sim, considere a captura dessas saídas fornecendo então uma resposta segura. Se a captura e uma resposta alternativa não são possíveis, e o sistema está operando num ambiente inerentemente inseguro, então devem ser executados testes extremamente rígidos de forma a garantir que a rede seja o mais segura possível.

Testes para encontrar a fronteira de comportamento aceitável. Para a maioria das redes neurais será difícil encontrar a fronteira entre comportamento aceitável e inaceitável. Se esta fronteira pode ser encontrada, ainda que de forma difusa, então a confiança da rede pode ser significativamente aumentada. Em geral a fronteira é encontrada fornecendo um amplo espectro de casos no conjunto de testes e usando estes testes para separar os casos de teste em categorias de válidos e inválidos. Deve-se então tentar encontrar uma forma simples de descrever a categoria de dados de teste válidos e inválidos. Por exemplo, se a tarefa é reconhecer um objeto, testes exaustivos podem revelar que o objeto pode ser reconhecido 98% do tempo a luz do dia se estiver obstruído menos de 10%.

Teste para robustez. De forma a testar a robustez, os dados podem ser modificados e as respostas conferidas. Por exemplo, nós podemos descobrir o que a rede faz se lhe for dado um *outlier* ("um ponto fora da curva") forçando uma extrapolação, ou nós

podemos executar a rede com alguns dos dados de entrada faltando, para ver como e a RNA se comporta.

Teste de campo. A melhor forma de testar qualquer sistema de software é levá-lo para onde ele será realmente usado e deixar os usuários reais testá-lo no seu ambiente real. Se for possível deve ser feito. Se forem feitos testes de campo, deve ser considerado o uso de arquivos para registro das atividades (*log*), ou outros métodos para coletar automaticamente dados sobre o uso do sistema.

Debriefing dos usuários. Um conjunto selecionado de perguntas para os usuários e especialistas que testaram o sistema pode ajudar na avaliação do sistema.

Peterson (1993) também aponta uma série de características que devem ser verificadas nos módulos de RNA, quando aplicáveis, que são mostradas na Tabela 4.6 abaixo.

Segurança	Em nenhuma circunstância o módulo vai gerar saídas que possam causar, ou contribuir pa danos sérios a humanos ou equipamento.
Erro verdadeiro	O erro verdadeiro foi cuidadosamente estimado e é aceitável para o cliente.
A saída é adequadamente sensível às entradas	Cada entrada tem um efeito observável (notável) na saída, mas uma pequena modificação entrada não deve causar uma grande alteração na saída.
As fronteiras de desempenho aceitável são conhecidas	Existe uma forma de medir que as entradas estão dentro das fronteiras onde um desempe aceitável foi verificado.
Degradação suave na fronteira da especialidade	Próximo a fronteira de desempenho aceitável, a queda de acurácia é gradual.
Robusto	O sistema continua a fornecer resultados úteis mesmo que algumas das entradas este ausentes ou sejam estimadas.
Mantenível	Se as entradas do sistema variam com o tempo então, o retreinamento com dados atuais revalidação do desempenho podem facilmente ser executadas pelos usuários do sistema.
Modificável	Se espera-se que o sistema seja usado em aplicações ligeiramente diferentes no futuro, entã capacidade para executar as modificações estão implícitas no sistema e as instruções pa seu uso são fornecidas para o usuário.

Tabela 4.6 – Características ideais de uma RNA segundo Peterson (1993).

4.4 Conclusão

Apresentamos neste capítulo um panorama dos principais aspectos envolvidos ao longo do desenvolvimento de uma RNA para SCS, principalmente aqueles relacionados à segurança. Mostramos que os requisitos de segurança não podem ser desassociados dos dados de uma RNA. Definimos um ciclo de vida evolucionário agregado a elementos de segurança que denominamos de CIVES. Apresentamos as características relevantes das principais etapas deste ciclo de vida, agrupando uma série de recomendações extraídas dos trabalhos estudados, e incorporamos as atividades de verificação e validação ao mesmo. Introduzimos o conceito de auto-proteção de fronteira como requisito de RNA para SCS. Foram também definidas as etapas críticas do ciclo de vida baseando-as em análise de segurança para as entradas e análise de vulnerabilidade para as saídas.

No capítulo seguinte faremos um estudo do estado da arte sobre as técnicas e métodos para análise de segurança de software de forma a selecionar os mais adequados para as necessidades de RNA aplicadas a SCS.

Capítulo 5

5 Análise de Segurança

A análise de segurança é um processo sistemático que requer uma análise estática e dinâmica do sistema do ponto de vista da segurança. Esta análise exige que olhemos tanto para o que o sistema deve fazer como para o que ele não deve fazer. A análise de segurança de um sistema tem início quando o projeto é concebido e continua ao longo do ciclo de vida do sistema. Ela não pode ser feita de forma independente, quer dizer, sem o conhecimento dos detalhes físicos e lógicos do sistema, da natureza e do tipo de aplicação desse sistema. Muitas vezes as tarefas de segurança de sistema e segurança de software são definidas separadamente, mas elas não podem realmente ser separadas: O processo de análise de segurança de software é um subconjunto do processo global de segurança do sistema (Leveson, 1995). Segurança é um atributo do sistema e portanto não se pode determinar a segurança de um componente do sistema de forma isolada.

Um sistema não é seguro quando ele alcança um estado vulnerável, que é um estado que pode levar a um acidente. Um acidente é um evento, ou sequência de eventos, não desejado de graves consequências. Um sistema pode chegar a um estado vulnerável tanto pela falha no controle de componentes críticos quanto pelo fornecimento de informações incorretas aos operadores que tomam decisões com consequências de potencial crítico.

A análise de vulnerabilidade (*hazard analysis*) é umas das tarefas do processo de análise de segurança. Existem várias metodologias e procedimentos disponíveis para a análise de vulnerabilidade na indústria de processos, mas a segurança de sistemas computadorizados ainda não está bem estabelecida. Do ponto de vista da segurança a

abordagem mais realista é a de minimizar os riscos que concentra a atenção nos pontos fracos do processo de forma a torná-los mais seguros (MoD, 1996a).

A falha de um sistema pode ser definida como a incapacidade de desempenhar os seus requisitos operacionais. Falhas sistemáticas fazem o sistema falhar sob determinadas combinações particulares de entrada ou sob determinadas condições particulares do ambiente. O outro tipo de falhas são aquelas que seguem um modelo estocástico. Exemplos dessas falhas são o desgaste de componentes mecânicos e a falha aleatória de componentes eletrônicos. Uma falha de software é sempre uma falha sistemática. Outras falhas sistemáticas são aquelas causadas por erros ou enganos na especificação, projeto, construção, operação ou manutenção. A ocorrência de uma falha não causa diretamente um acidente. Em geral é necessária a ocorrência de um determinado estado do processo, ou de uma combinação de falhas, para criar uma situação de vulnerabilidade e levar a um acidente.

O sistema de automação em geral é seguro, a vulnerabilidade resulta do processo controlado pela automação. A segurança de um sistema ou planta de processo está baseada no desenho do processo que define o número de possíveis estados inseguros e na probabilidade de ocorrência desses estados. Entretanto, vários aspectos de segurança estão sob a responsabilidade do desenho de automação. Numa situação de vulnerabilidade o objetivo do sistema de controle é prevenir a ocorrência de um acidente mantendo o processo num estado seguro ou trazer o processo de volta para um estado seguro. Algumas vezes não é possível manter o sistema num estado seguro e nesses casos o sistema de controle deve minimizar as consequências.

Um dos objetivos de segurança mais importantes no projeto de automação de um processo é o de prevenir a criação de situações de vulnerabilidade por parte da automação. Na prática, a análise de segurança consiste na identificação de possíveis

perturbações do processo que venham a criar situações vulneráveis e desencadear acidentes.

5.1 Análise de Segurança de Software

Segurança é um atributo do sistema e não pode ser determinada examinando-se a segurança dos componentes do sistema de forma isolada. Assim, como o software é um componente do sistema, a segurança do software só pode ser avaliada quando considerada como parte do todo, o sistema. Um sistema não é seguro quando alcança um estado vulnerável, ou seja, um estado que pode conduzir a um acidente. Um acidente é um evento ou uma seqüência de eventos não desejados de graves consequências. Um sistema pode alcançar um estado vulnerável tanto pela incapacidade de controlar componentes perigosos como pelo fornecimento de informações incorretas aos operadores que tomam decisões com conseqüências potencialmente perigosas. Por esta razão, o software só pode ser considerado não seguro somente no contexto de um sistema particular e de seu ambiente operacional.

Segundo Leveson (Leveson, 1995), segurança de software significa que o software executará dentro de um contexto do sistema sem contribuir para a criação de estados vulneráveis do sistema. Uma condição ou um estado vulnerável (do sistema) é um estado que pode levar à ocorrência de um acidente. Assim, o software de sistema pode afetar a segurança do sistema de duas formas:

1. o software pode exibir um comportamento em termos de valores de saída e de sincronismo que contribuam para que o sistema alcance um estado vulnerável, ou

2. o software pode falhar no reconhecimento, ou no tratamento, de falhas de hardware que ele deve controlar ou responder a elas de alguma forma.

Podemos considerar uma falha como o efeito de um erro de serviço do sistema e, um erro como a manifestação de uma falha no sistema (Laprie, 1991). É importante observar que nem todos os componentes de software em um SCS são necessariamente críticos em relação à segurança. Software crítico em relação à segurança é todo o software que pode direta, ou indiretamente, conduzir o sistema a um estado inseguro, ou seja a um estado vulnerável.

A segurança de um sistema é determinada segundo a compreensão que se tem das vulnerabilidades potenciais do sistema, ou seja, o potencial para causar acidentes que o sistema possui. Uma vez que essas vulnerabilidades são identificadas e compreendidas, o sistema pode ser analisado em termos da vulnerabilidade dos seus componentes, conduzindo a uma hierarquia das especificações de segurança dos componentes do sistema (Place, 1993). Estas especificações de segurança identificam quais são as funções (componentes) críticas em relação à segurança do sistema. Funções críticas de segurança são aquelas cuja operação correta ou incorreta (incluindo a operação correta no tempo errado), ou a falta da operação podem contribuir para um estado vulnerável do sistema. Entre estas funções críticas de segurança estão aquelas funções de software (*safety critical software functions*) que podem direta ou indiretamente, em conjunto com o comportamento, ou circunstâncias ambientais, de outros componentes do sistema contribuir para criação de um estado vulnerável do sistema (Leveson, 1995). Estas vulnerabilidades de sistema relacionadas ao software são denominados vulnerabilidades de software. A negação dessas vulnerabilidades de software são exigências de segurança de software e as restrições de segurança do

software (Isaksen, 1996), onde as primeiras estão relacionadas à segurança dos requisitos de funcionalidade e as últimas estão relacionadas aos limites de operação do sistema. Portanto, o objetivo das tarefas de segurança de software em sistemas críticos deve assegurar que:

1. o software não vai causar ou contribuir para que o sistema alcance um estado vulnerável;
2. o software não vai falhar na detecção de um estado vulnerável, ou na tomada de ações corretivas na ocorrência de tais estados;
3. o software não vai falhar na mitigação dos danos na ocorrência de um acidente (NASA-GB-1740.13-96, 1996).

5.1.1 Requisitos Conflitantes

Frequentemente aspectos da segurança entram em conflito com aspectos de desempenho e custo de desenvolvimento. Por exemplo, segurança não é sinônimo de confiabilidade, apesar de terem alguns aspectos comuns. *Confiabilidade* é a medida do quão bem o sistema pode executar sua missão pretendida. *Segurança* é a medida do quão bem o sistema evita aquilo o que não pretende fazer (Mojdehbakhsh, 1994). Confiabilidade e segurança podem as vezes estar em conflito. Por exemplo, está implícito na definição de confiabilidade que o software deve cumprir sua missão mesmo que isto crie situações perigosas (possibilidade de ocorrência de acidentes). Está implícito na definição de segurança que o software seja seguro mesmo no caso de comprometimento da missão. Pode-se ver que confiabilidade é definida em termos de

serviços de sistema, e a segurança é definida em termos de conseqüências externas. O software do Therac-25 (uma máquina de radioterapia controlada por computador), por exemplo, era altamente confiável, trabalhou milhares de vezes sem problemas antes de aplicar superdoses de radiação em alguns poucos pacientes (Leveson, 1995). Um outro exemplo de diferentes aspectos do software refere-se segurança (*safety*) e a seguridade (*security*) (Veríssimo & de Lemos, 1989, Veríssimo, 1996) que têm atributos diferentes: a segurança preocupa-se com as conseqüências que ameaçam o mundo exterior; a seguridade preocupa-se com a integridade dos dados contidos no sistema.

5.1.2 Hardware e Software

Em muitos sistemas críticos de segurança o hardware responsável pela segurança está sendo substituído por software. Esta substituição introduz novas modalidades de falha, que não podem ainda ser analisadas pelas técnicas tradicionais de engenharia de segurança. Estas novas modalidades de falhas acontecem devido às diferentes naturezas das falhas de software e de hardware. Todas as falhas de software são sistemáticas, isto é, são falhas de concepção e de projeto; o software não sofre desgaste, nem envelhece como o hardware. Falhas sistemáticas não são aleatórias, mas as falhas de sistema devido às falhas de projeto estão associadas a probabilidade de ocorrência das seqüências de entrada que as podem desencadear. De certa forma o software é o próprio projeto.

Hardware de segurança nunca deve ser substituído por software. Certamente, o desenvolvimento atual de SCS preocupa-se com os componentes de segurança do software e do hardware. Assim, os componentes responsáveis na prevenção de acidentes

são duplicados no software e no hardware. Por exemplo, erros do software que mataram e feriram pessoas no caso Therac-25 eram simples incômodos na operação da máquina do sistema predecessor, o Therac-20 (Leveson, 1995). Isto porque o Therac-20 tinha circuitos protetores de hardware independente para monitorar a aplicação do feixe de elétrons. Os circuitos protetores não permitiam que o feixe fosse aplicado a menos que o escudo protetor estivesse na posição correta, não permitindo assim a ocorrência de um estado perigoso onde o paciente seria exposto a uma dose excessiva de radiação. Estas proteções de hardware foram removidas no Therac-25, e o software com erros foi reutilizado (o software do equipamento não foi incluído na análise de segurança do Therac-25). Assim, quando no Therac-20 um estado perigoso (vulnerabilidade) causado por erros do software ocorria, os fusíveis da máquina queimavam, desligando a unidade. A mesma situação no Therac-25 expôs os pacientes a doses excessivas de radiação. Enquanto no Therac-20 existiam bloqueios mecânicos para evitar comportamentos errôneos da máquina, no Therac-25 esses bloqueios estavam a cargo do software. Este caso mostra que o software não precisa ser perfeito para ser seguro, desde que existam proteções adicionais (intertravemento de hardware ou técnicas de software tolerante a falhas, por exemplo) consideradas na concepção do sistema de tal forma que a segurança não recaia somente sobre o software.

5.1.3 Reutilização de Software

A reutilização de software em um sistema de segurança não significa que o novo sistema será seguro também, porque como já citado a segurança é um atributo do sistema e não um atributo do software de um componente isolado (Fenelon, 1995). O

caso Therac-25 apresentado é um exemplo de reutilização de software de forma incorreta em um sistema crítico. Um exemplo recente foi o acidente do primeiro vôo do foguete lançador de satélites Ariane 5, onde o software reutilizado foi usado diversas vezes nos vôos do Ariane 4 e considerado de alta confiabilidade. Infelizmente, o Ariane 5 tem parâmetros de vôo diferentes e o software do sistema inercial reutilizado não foi capaz de tratá-los (Lion, 1996). Assim, sob ponto de vista de segurança, a reutilização de software não significa muito, se não for executada uma nova análise de segurança no novo sistema onde o software está sendo reutilizado (Fenelon, 1995, Stalhane, 1998).

5.1.4 Software Crítico e Software Não Crítico

Finalmente, não existe consenso sobre a praticabilidade na separação do software crítico do software não crítico. Para Leveson (1995) a estratégia é identificar as partes do software que controlam operações críticas e concentrar os esforços de análise de segurança naquelas funções e no trajeto crítico que conduz a sua execução. Para Parnas et al. (1990), o software exibe um comportamento de elo frágil, ou seja, falhas em partes não críticas do código podem ter repercussões inesperados em outras partes. Assim, deve ser considerada a contaminação do software crítico pelo software descontrolado (por exemplo, o software não crítico poderia travar o computador ou escrever em áreas de memória de software crítico quando software crítico e não crítico compartilham um mesmo processador). Parnas et al. (1990) são a favor da separação do software crítico do software não crítico em computadores diferentes, alegando que a segurança é uma propriedade que não pode ser separada da confiabilidade.

5.2 Análise de Vulnerabilidade

Dado um sistema e um ambiente operacional, a análise de vulnerabilidade é a investigação metódica do sistema, em parte ou como um todo, para encontrar vulnerabilidades potenciais. A análise de vulnerabilidade é o núcleo da análise de segurança. Uma vulnerabilidade é um estado ou conjunto de condições em um sistema que em conjunto com outras condições ambientais do sistema, vai levar a um acidente (Leveson, 1995). Consequentemente, uma vulnerabilidade é um estado das circunstâncias existentes para as quais um acidente torna-se inevitável. Portanto, a descrição de uma vulnerabilidade deve conter o estado do sistema e deve ser definida com respeito às condições reais do seu ambiente operacional.

Considere, por exemplo, um sistema de piloto automático de uma aeronave. O piloto automático (PA) tem controle somente sobre a aeronave, nada externo a ela. Assim os limites da aeronave definem os limites do PA. O ambiente operacional é tudo mais que a aeronave encontra (e.g. outra aeronave, as condições atmosféricas, o solo, sinais de rádio, etc.). Uma vulnerabilidade potencial é um problema de desvio para baixo no PA da aeronave, que sob determinadas circunstâncias (e.g. baixa visibilidade sobre um terreno montanhoso) vai causar um acidente. É importante observar que não existe vulnerabilidade quando a aeronave está no chão e parada.

A análise de vulnerabilidade é um conjunto de técnicas, cada uma delas fornecendo um *insight* diferente das características do sistema sob investigação. Alguns métodos evoluíram dentro de setores particulares da indústria e tem uso limitado em outras áreas. Outros métodos, apesar de terem suas origens dentro de domínios de aplicações específicas, tem encontrado ampla aceitação através de vários setores da indústria. A Figura 5.0 mostra as tarefas típicas de uma análise de vulnerabilidade no

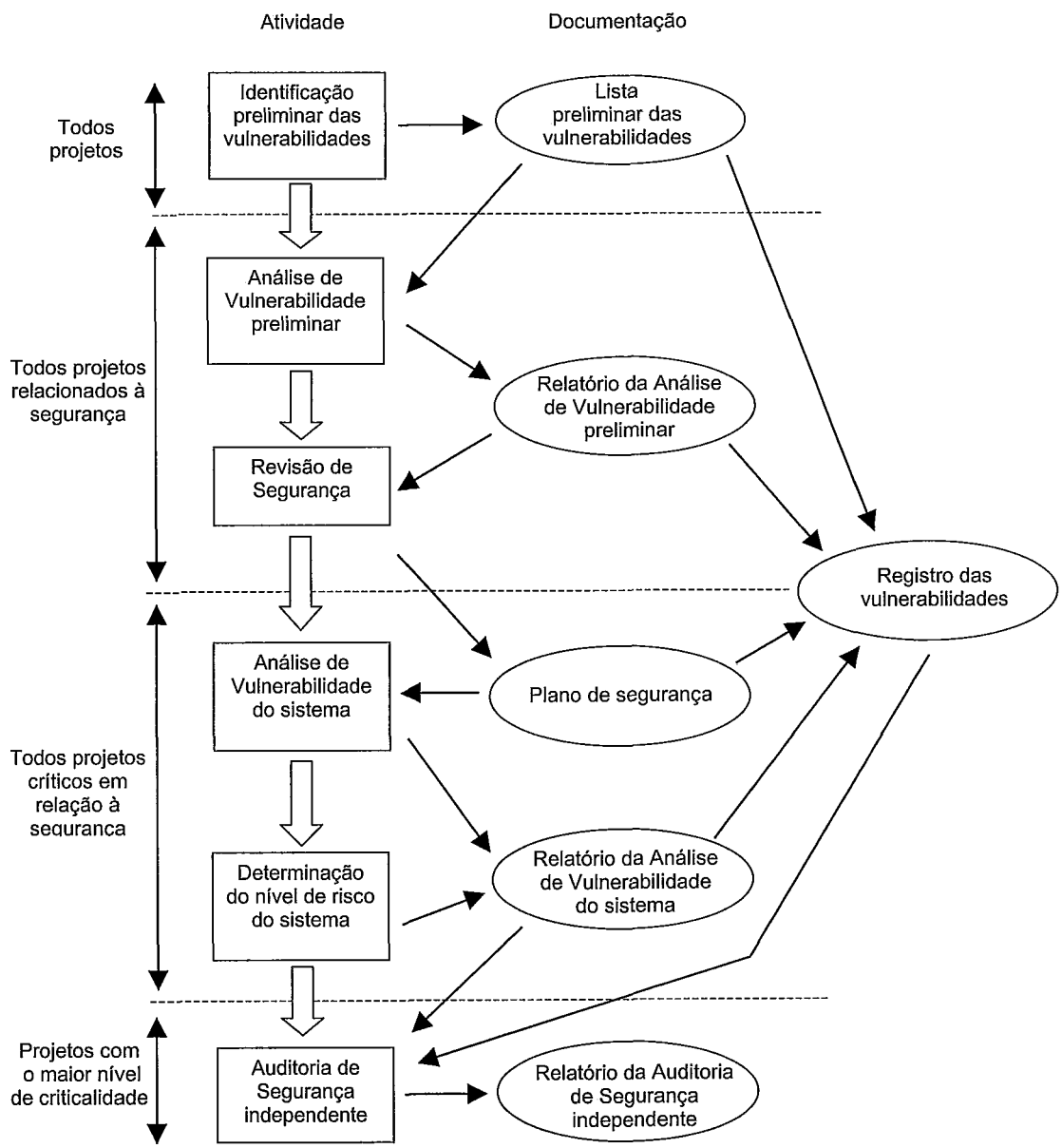

Figura 5.0 – Tarefas típicas da análise de vulnerabilidade no desenvolvimento de um SCS (Storey, 1996).

desenvolvimento de um SCS. A análise de vulnerabilidade pode ser executada em qualquer estágio do desenvolvimento do sistema, mas existem evidências que maiores benefícios podem ser obtidos caso a análise de vulnerabilidade comece nos estágios iniciais de desenvolvimento. Tais evidências, como apontadas por diversos autores (Gardiner, 1999, Laprie, 1996, Leveson, 1997, Lutz, 1996a), são de que acidentes envolvendo SCS baseados em computadores podem normalmente ser rastreadas até a

especificação de requisitos de software incompletas ou com outro tipo de problema. Este é o caso dos erros de software relacionados à segurança, que persistem até os estágios finais do desenvolvimento. É importante observar que requisitos não intencionais (requisitos que não foram intencionalmente especificados) são a principal causa dos erros funcionais relacionados à segurança.

Erros na especificação de requisitos resultam normalmente de duas origens: desejos mal interpretados do "cliente" ou requisições do "cliente" pobremente concebidas. Do ponto de vista da segurança isto implica que os requisitos devem ser analisados tanto para os comportamentos desejáveis, como para os indesejáveis. Quanto mais cedo forem iniciados os trabalhos de análise de vulnerabilidade no processo de desenvolvimento de um sistema, maiores serão os benefícios. Do ponto de vista da pesquisa atual, um aspecto de grande apelo é melhorar a análise de vulnerabilidade nos estágios iniciais da concepção (Stuparu, 1996). Outro aspecto crítico nos SCS baseados em computador é a interface entre sistemas. A interação entre componentes de um sistema é feita através de interfaces, de onde podem resultar estados perigosos (McDermid, 1994). A análise de vulnerabilidade demandada por este aspecto envolve um grande compromisso de esforço e atenção. Como apontado por (Sweet, 1995) por exemplo, diferentes fabricantes de aviônicos, aliados a razões legais e éticas, criam uma compartimentalização no desenvolvimento dos aviônicos resultando num *feedback* deficiente entre os especificadores, projetistas e integradores de sistema. Um bom exemplo de problema de interfaceamento é, outra vez, o acidente do Ariane 5, no qual o sistema inercial de referência (IRS) e o software de vôo foram testados separadamente usando diferentes especificações para os parâmetros de vôo (o SRI usou parâmetros especificados para o Ariane 4 (Jézéquel, 1997)). Apesar dos investigadores do acidente - *Inquiry Board Report* - (Fenelon, 1995) recomendarem que o sistema deve ser testado

como um todo, não em partes, o erro de software responsável pelo acidente estava em um trecho de código em execução não era necessário para o vôo (Jézéquel, 1997).

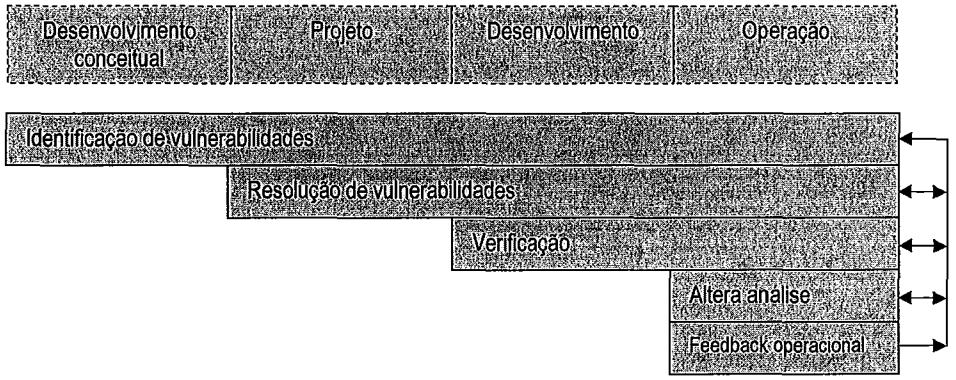

Figura 5.1 – As vulnerabilidades ao longo das fases de desenvolvimento de um sistema (Leveson, 1995).

5.2.1 Análise de Vulnerabilidade Preliminar

A abordagem geral para a análise de vulnerabilidade é primeiro executar uma análise de vulnerabilidade preliminar (*Preliminary Hazard Analysis - PHA*) para identificar as possíveis vulnerabilidades. A PHA é a primeira de uma série de análises de vulnerabilidade de níveis do sistema. A execução de uma PHA geralmente implica em:

- Determinar quais vulnerabilidades podem existir durante a operação do sistema, bem como a sua magnitude relativa;
- Desenvolver diretrizes, especificações, requisitos e restrições de projeto para serem seguidas no desenho do projeto;
- Ações iniciais para controle de vulnerabilidades particulares;

- Identificação das responsabilidades gerencial e técnica para ação e aceitação do risco, e garantia de que controle efetivo é exercido sobre as vulnerabilidades;
- Determinação da magnitude e complexidade dos problemas de segurança no programa, de forma a quantificar os esforços de gerência e de engenharia necessários para minimizar e controlar as vulnerabilidades (Leveson, 1997).

A PHA é a primeira fonte de requisitos de segurança de software específicos uma vez que ela é única para uma arquitetura particular de um sistema. Portanto, a PHA é um prerequisito para executar qualquer análise de segurança de software. O resultado da PHA é uma lista de vulnerabilidades. A Tabela 5.0, por exemplo, apresenta três das seis vulnerabilidades identificadas e os respectivos requisitos de alto nível associados (declarações ***deve***) e restrições de projeto (declarações ***não deve***) para um sistema de controle de tráfego aéreo particular (Leveson, 1997).

VULNERABILIDADES	REQUISITOS – RESTRIÇÕES
1. Um par de aviões controlados viola os padrões mínimos de separação.	1a. O CTA (Controle de Tráfego Aéreo) ***deve*** fornecer orientação para manter uma distância de separação segura entre os aviões. 1b. O CTA ***deve*** fornecer alertas de conflito.
2. Uma aeronave controlada entra numa área de condição atmosférica perigosa (condições de gelo, ventos fortes, células de tempestade, etc.).	2a. O CTA ***não deve*** fornecer orientações que direcionem a aeronave para dentro de áreas com condições atmosféricas perigosas. 2b. O CTA ***deve*** fornecer informações sobre as condições do tempo e alertas para as tripulações das aeronaves. 2c. O CTA ***deve*** avisar as aeronaves que entram numa região de condição atmosférica perigosa.
3. Uma aeronave controlada penetra,sem autorização, numa região de espaço aéreo restrito.	3a. O CTA ***não deve*** emitir orientações que direcionem uma aeronave para dentro de um espaço aéreo restrito. 3b. O CTA ***deve*** fornecer avisos de tempo cronometrado de forma a prevenir a incursão de aeronaves em espaços aéreos restritos.

Tabela 5.0 – Exemplo de algumas vulnerabilidades de um sistema de Controle de Tráfego Aéreo e os requisitos e restrições de segurança associados (Leveson, 1997).

Geralmente o número de vulnerabilidades em um SCS não é grande, mesmo nos sistemas mais complexos. Exemplos de causas e controles de vulnerabilidades são mostrados na Tabela 5.1. O software é crítico em relação à segurança em todos os casos onde ele é uma causa potencial de vulnerabilidade ou é usado como controle de uma vulnerabilidade (NASA-GB-1740.13-96, 1996).

Causa	Controle	Exemplo de Ação de Controle
Hardware	Software	Detecção de falha e ativação da função de segurança.
Software	Hardware	Sensor disparando diretamente uma chave de segurança para sobrepujar um sistema de controle de software.
Software	Hardware	Temporizador de hardware ou lógica de hardware discreto identifica comandos ou dados inválidos.
Software	Software	Dois processadores independentes, um verificando o outro e interferindo se uma falha é detectada. Emulação do desempenho esperado e detecção de desvios.
Software	Operador	O operador vê uma violação de parâmetros de controle e interrompe (termina) o processo.
Operador	Software	Software de validação de um comando perigoso iniciado pelo operador.

Tabela 5.1 – Exemplos de causas de vulnerabilidades e os respectivos controles (NASA-GB-1740.13-96, 1996).

Na norma MoD 00-56 Defense Standard (MoD, 1996a), a abordagem para identificação de vulnerabilidades começa com uma lista padrão contendo diferentes tipos de vulnerabilidades que devem ser consideradas na enumeração das vulnerabilidades preliminares. A abordagem para identificação das vulnerabilidades em (MoD, 1996a) é mostrada na Figura 5.2.

Como proposto em (Leveson, 1997), após a PHA, as análises de vulnerabilidade de sistema e subsistemas subsequentes devem ser executadas para determinar os contribuíntes para a PHA. Com base na PHA, a análise de vulnerabilidade de sistema (*System Hazard Analysis - SHA*) deve estudar as possíveis vulnerabilidades criadas nas

interfaces entre os subsistemas do sistema como um todo. Especialmente a SHA deve examinar todas as interfaces de subsistemas para:

- Conformidade com os critérios de segurança da especificação de requisitos do sistema;
- Degradação da segurança resultando da operação normal do sistema; e
- Possíveis combinações de eventos de vulnerabilidade ou falhas independentes, dependentes e simultâneas, inclusive comportamento errático dos controles e dispositivos de segurança, que podem levar a vulnerabilidades.

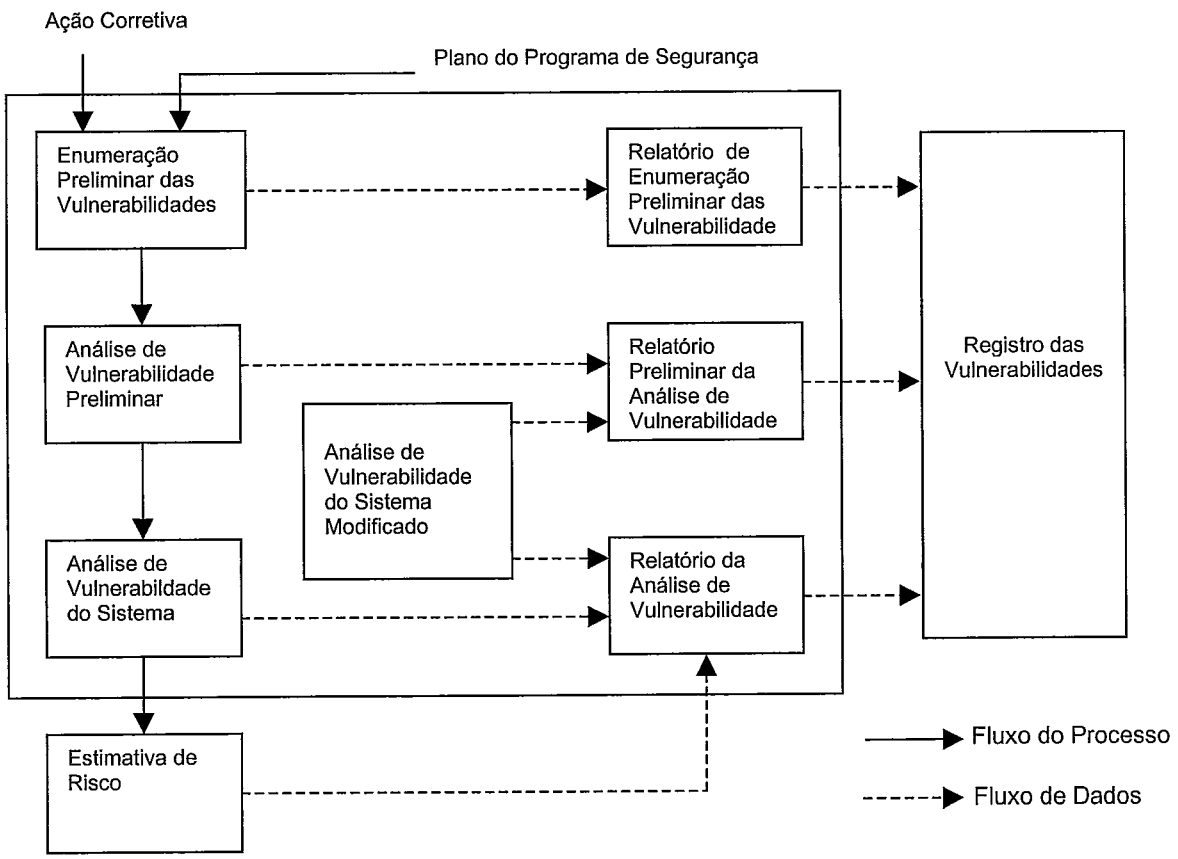

Figura 5.2 – Abordagem evolucionária escalar para a identificação e refino de vulnerabilidades (MoD, 1996a).

A abordagem da SHA na norma (MoD, 1996a) contém algumas diferenças em relação ao exposto acima e é composta das seguintes atividades:

- *Análise funcional* é efetuada para identificar vulnerabilidades associadas com todas funções do sistema na operação correta, incorreta e não operação.
- *Análise zonal* é uma análise da disposição física do sistema e dos seus componentes na sua instalação ou domínio de operação.
- *Análise de falha dos componentes* deve ser efetuada para se considerar todos os modos de falha dos componentes.
- *Análise de vulnerabilidade de suporte e operação* deve ser efetuada para avaliar as vulnerabilidades associadas com as tarefas que podem ser executadas pelo pessoal de operação e suporte.
- *Análise de vulnerabilidade de insalubridade* deve ser efetuada para identificar perigos à saúde e recomendar medidas a serem incluídas no sistema.
- *Análise de vulnerabilidade de modificação do sistema* deve ser feita quando é proposta uma modificação do sistema.

A análise de vulnerabilidade de subsistema (*Subsystem Hazard Analysis - SSHA*) estuda cada subsistema individualmente e identifica os efeitos da sua operação e falha, sob vários aspectos (e.g. operação normal, falha funcional, etc.), nas vulnerabilidades do sistema. A SSHA deve identificar como eliminar ou mitigar o risco das vulnerabilidades identificadas. Portanto, a análise de vulnerabilidade de subsistema de software deve ser executada para qualquer função crítica de segurança envolvendo:

- A causa de uma vulnerabilidade;

- O controle de uma vulnerabilidade;
- O software que emite mensagens sobre decisões de segurança críticas para os operadores ou usadas como meio de detecção de falhas (NASA-GB-1740.13-96, 1996).

Apesar das análises SHA e SSHA serem executas de forma similar, seus objetivos são diferentes. Enquanto a SSHA examina como a operação ou falha de um componente individual afeta a segurança do sistema, a SHA determina como os modos normal e de falha dos componentes operando juntos podem afetar a segurança do sistema.

A maioria dos erros nas análises de vulnerabilidade estão relacionadas com a falha em antever todas as formas em que uma vulnerabilidade pode ocorrer e, segundo Kletz (1997), o tempo é normalmente melhor empregado na procura por todas as fontes de vulnerabilidade do que na quantificação com grande precisão daquelas já identificadas.

5.3 Tipos de Análise de Vulnerabilidade

A maioria das técnicas de análise de vulnerabilidade envolve uma busca por vulnerabilidades no sistema (requisitos, projeto, etc.), procurando por condições que podem levar a acidentes. O tipo da busca pode ser categorizado, com respeito à direção da análise. Buscas para-frente são aquelas onde o analista rastreia a partir da causa para o efeito. Buscas para-trás trabalham a partir dos efeitos para as causas. Estes dois tipos de análise são também conhecidos como indutiva e dedutiva respectivamente.

Métodos indutivos são aplicados para determinar quais estados do sistema são possíveis e métodos dedutivos são aplicados para determinar como um determinado estado pode ocorrer. Relacionamentos típicos envolvidos nesse tipo de buscas estão associados ao tempo, ou à sequência, dos acontecimentos. Assim, o comportamento dinâmico de um sistema pode ser representado como um mapeamento de um estado para o estado subsequente. As definições dinâmicas diferem das definições estruturais pois, naquelas os relacionamentos dependem do tempo, e nestas da estrutura.

Os resultados das buscas para-frente e para-trás não são necessariamente os mesmos (Figura 5.3). Isto acontece porque examinando somente os efeitos de falhas de componentes individuais no comportamento global do sistema perdem-se comportamentos vulneráveis do sistema resultantes de combinações de falhas de subsistemas, ou ainda das interações de comportamento correto entre os vários subsistemas.

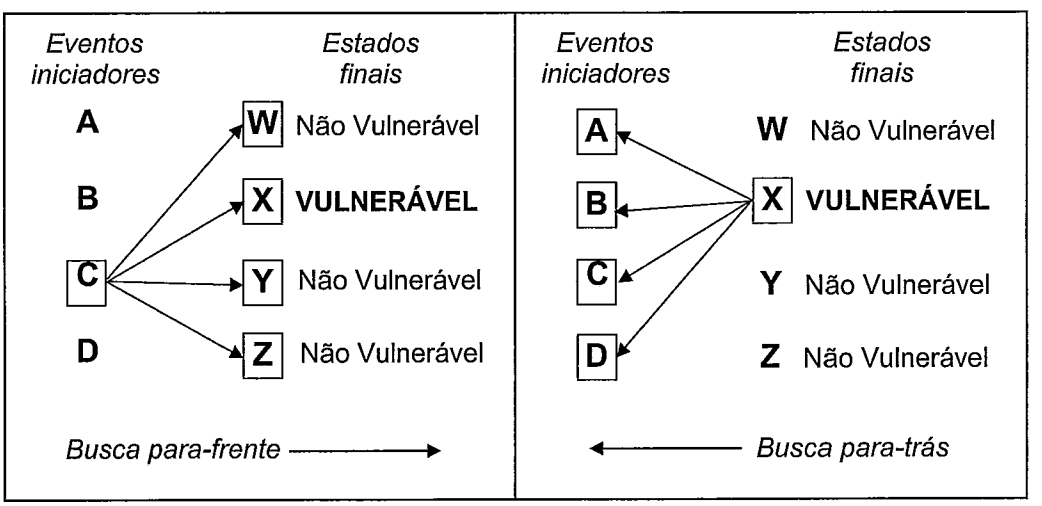

Figura 5.3 – As buscas para-frente e para-trás fornecem tipos diferentes de informação (Leveson, 1997).

Uma segunda categorização dos métodos de busca é *top-down* (de-cima-para-baixo) e *bottom-up* (de-baixo-para-cima) onde é investigado o relacionamento

estrutural. A busca neste caso envolve o refinamento do evento em seus eventos constituintes. Uma definição estrutural representa o sistema como uma hierarquia estática de elementos, definição esta que é invariante em relação ao tempo.

Existem outras técnicas que combinam diferentes estratégias de busca usando, por exemplo, buscas para-frente e para-trás tentando encontrar um elo de ligação entre as vulnerabilidades e suas causas ou efeitos. Métodos que empregam está estratégia podem ser chamados de exploratórios já que as causas e os efeitos das vulnerabilidades não são necessariamente conhecidos de antemão.

Seguindo a taxonomia de análise de vulnerabilidade proposta em (Fenelon, 1994), nós adicionamos informações à Tabela 5.2 para obter uma visão mais clara das abordagens para análise de vulnerabilidade.

	Causa Conhecida	**Causa Desconhecida**
Efeito Conhecido	• Descrição do comportamento	• Dedutivo • Busca para-trás • *Top-down* (e.g., FTA)
Efeito Desconhecido	• Indutivo • Busca para-frente • *Bottom-up* (e.g., FMEA)	• Exploratório (e.g., HAZOP)

Tabela 5.2 – Taxonomia das Análises de Vulnerabilidade (Fenelon, 1994).

A abordagem exploratória é a forma de análise necessária em projetos inovadores, tal como a Estação Espacial Internacional (Anônimo, 1998), onde um novo conceito de projeto deve ser investigado (causas e efeitos não são conhecidos de antemão e devem ser investigados). Na fase de desenho da arquitetura de um novo software, nada se sabe sobre os seus modos de falha, e o conhecimento acerca dos efeitos de falhas estará

geralmente limitado a uma análise de vulnerabilidade preliminar de alto nível. Este é o estágio no qual é maior o benefício para se tomar medidas para aprimorar as características de segurança de um sistema (Fenelon, 1994). Isto porque, durante esta fase, as especificações de projeto podem ser refeitas, evitando assim propostas mais vulneráveis.

Não existe uma análise de segurança ou técnica de avaliação que pode tratar todos os aspectos de um SCS complexo. Um plano de segurança efetivo deve cobrir todas as fases do ciclo de vida do sistema e tratar todas as vulnerabilidades potenciais.

O software é um componente do sistema, geralmente algum tipo de controle ou função de monitoração, e deve ser analisado como um subsistema na análise de vulnerabilidade do sistema. O comportamento do computador e das suas interfaces deve ser avaliado em relação ao resto do sistema no referente ao seu potencial de contribuição para vulnerabilidades do sistema. Quando é identificado um comportamento crítico, ele deve ser rastreado às suas origens no software de forma a identificar as partes do software que requerem características de projeto especiais ou que precisam ser analisadas em profundidade. Cada componente de software, inclusive o software comercial de prateleira (*Commercial Off-The-Shelf - COTS*), precisa ser analisado ao nível necessário para determinar qualquer impacto ou influência sobre as vulnerabilidades do sistema identificadas. Do ponto de vista da segurança de software é interessante examinar a estrutura da documentação de alto nível para o desenvolvimento de software de SCS, baseados em computador (Figura 5.4), a qual é obrigatória segundo a norma MoD 00-55 Defense Standard (MoD, 1997).

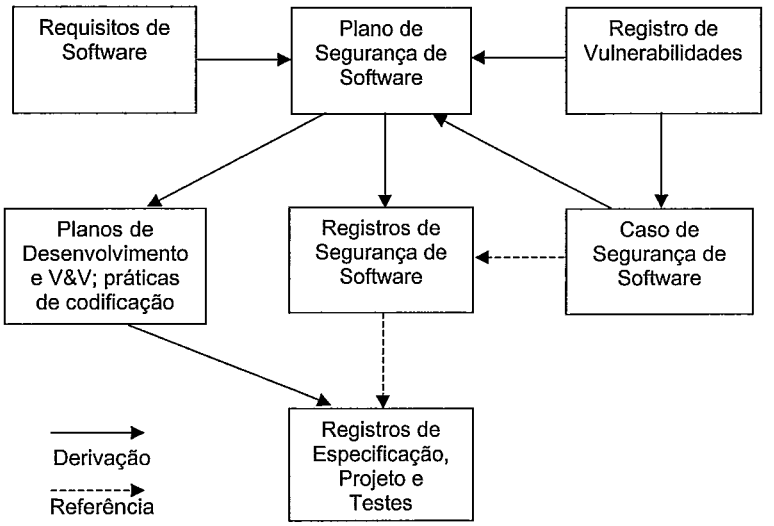

Figura 5.4 – Estrutura da documentação de alto nível para software segundo a norma MoD 00-55 (MoD, 1997).

Apresentamos técnicas de análise de vulnerabilidade consideradas no tratamento de segurança de software. Estes métodos foram extraídos da literatura sobre segurança de software e de sistemas, e de práticas reportadas de vários domínios de aplicação industrial. A intenção é apresentar os métodos e técnicas aplicáveis a software de SCS baseados em computador, uma vez que nem todas as técnicas de análise de vulnerabilidade são aplicáveis a software e, consequentemente, aplicáveis ao tipo de software implementado através de Redes Neurais Artificiais (RNA) em Sistemas Críticos quanto à Segurança (SCS).

As técnicas apresentadas a seguir estão divididas em técnicas de análise indutiva, dedutiva e exploratória.

5.3.1 Técnicas de Análise de Vulnerabilidade Indutivas

As técnicas de análise de vulnerabilidade indutivas partem de uma causa simples e analisam as consequências possíveis, fazendo uma busca para-frente procurando pelos efeitos de uma única falha de um componente de sistema.

Figura 5.5 – Busca para-frente partindo de uma causa para as possíveis consequências.

5.3.1.1 Análise de Modos e Efeitos de Falha de Software (SFMEA)

A Análise de Modos e Efeitos de Falha de Software (*Software Failure Modes and Effects Analysis – SFMEA*) é uma extensão da Análise de Modos e Efeitos de Falha (*Failure Modes and Effects Analysis - FMEA*) para hardware. O procedimento da FMEA foi padronizado (MIL-STD-1629A, 1980), mas não existe norma semelhante para execução da SFMEA. A principal diferença entre essas duas análises é que FMEA geralmente tem um número limitado de modos de falha, enquanto que SFMEA pode ter um número muito maior de modos de falha (Lutz, 1996b). Quando a FMEA leva em consideração a avaliação da severidade dos modos de falha e a probabilidade de suas ocorrências, a análise é chamada de FMECA (*Failure Modes, Effects, & Criticality Analysis - FMECA*). No caso de uma SFMECA (*Software Failure Modes, Effects, & Criticality Analysis - SFMECA*) a criticalidade está relacionada a severidade das consequências dos modos de falha através da associação de níveis de criticalidade às falhas dos componentes de software.

A abordagem da FMEA é sistemática, buscando identificar os modos de falha potenciais do sistema, suas causas e os efeitos dessas ocorrências no funcionamento do sistema. Ela é usada para avaliar a segurança dos componentes do sistema e para identificar as modificações de projeto e as ações corretivas necessárias para mitigar os efeitos das falhas no sistema. Esta técnica está baseada numa abordagem hierárquica e indutiva de forma a determinar como cada modo de falha dos componentes afeta a operação do sistema. A Figura 5.6 mostra o fluxo geral do processo iterativo de uma FMECA.

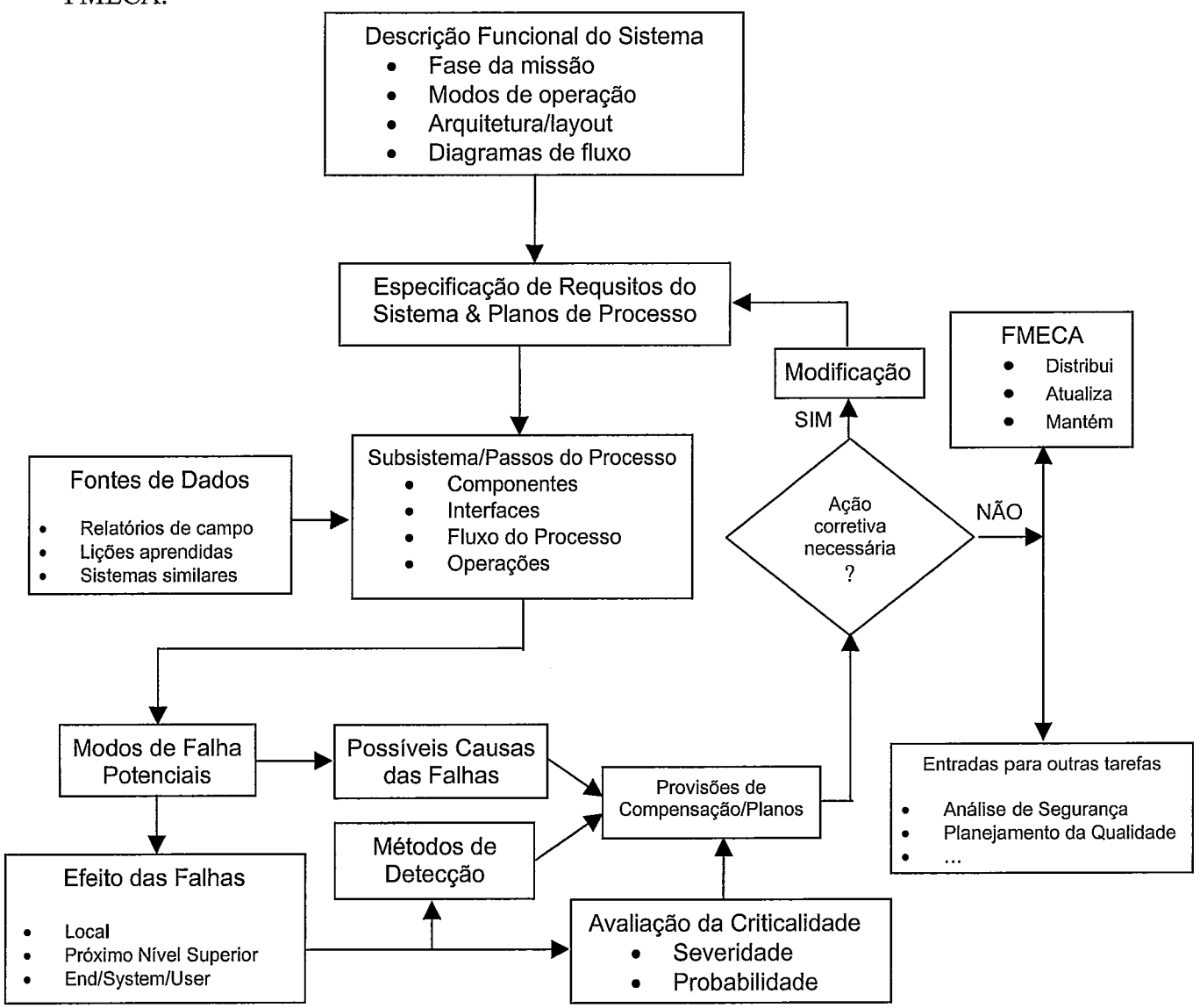

Figura 5.6 – Diagrama dos processos da FMECA.

O efeito de cada componente em cada modo de falha é avaliado através de tabelas desenvolvidas com esse propósito, para cada componente, onde são registrados pelo menos as seguintes informações: 1) os modos de falha, 2) as possíveis causas, 3) os efeitos locais, 4) os efeitos no sistema e 5) as ações de remediação.

A FMEA analisa os possíveis modos de falha de um componente e identifica as consequências daquela falha. Uma vez que nem todas as falhas resultam em acidentes, analisar todos os componentes geralmente é um processo longo e ineficiente para os propósitos de segurança. Entretanto, este não é o caso se a FMEA é usada somente para componentes selecionados sabidamente críticos em relação a segurança. No caso da SFMEA, é possível identificar requisitos fracos ou ausentes e identificar defeitos de software latentes (Rabéjac, 1993). A SFMEA focaliza com particular atenção as dependências ou interações implícitas que podem propagar dados errôneos para outros módulos de software (Lutz, 1996a) e portanto potencialmente interessante para a análise de entradas de uma RNA. O fator de criticalidade, i. e. a severidade do efeito da falha, pode ser usado na determinação de onde aplicar outros recursos de análise e testes.

Apesar da FMEA ter sido originalmente concebida para análise de confiabilidade e segurança, os projetistas descobriram que podem aprender mais sobre o sistema que estão construindo enquanto fazem a análise, transformando a FMEA em uma importante ferramenta de projeto (Ippolito, 1995, Lutz, 1996b). Nas indústrias militar, aeroespacial e nuclear, onde os aspectos de segurança são de importância primária, a FMEA tornou-se um processo de desenho de sistemas desde as primeiras fases de desenvolvimento conceitual até às etapas de projeto e teste. A SFMEA é mais usada durante a análise de projeto, mas pode ser usada efetivamente durante a análise de requisitos se a especificação de requisitos fornecer detalhes suficientes. Como parte do

processo da análise de requisitos ela pode, por exemplo, contribuir para a validação dos requisitos de sistema.

A FMECA pode ser geralmente classificada como uma FMECA de produto ou FMECA de processo, dependendo da sua aplicação (Tabela 5.3). Uma FMECA de produto analisa o projeto do produto através do exame de como os itens modos de falha afetam o projeto do produto. Uma FMECA de processo analisa os processos envolvidos na construção, uso e manutenção de um produto através do exame de como falhas nos processos de manufatura ou serviços afetam a construção do produto.

Tipos de FMECA Funcional & Detalhada					
Produto			Processo		
Hardware	*Software*	*Temporização / Sequenciamento*	*Produção*	*Manutenção*	*Uso*
Elétrico	Programa/Tarefa		Montagem	Operações	Modos
Mecânico	HW Interfaces		Químico	Documentação	Interface Humana
Interfaces	SW Interfaces		Maquinário	Treinamento	Overstress
			Software		Prova do Cliente
					Documentação
					Treinamento

Tabela 5.3 – Tipos de FMECA.

Nosso interesse é uma FMECA produto que possa ser aplicada a produtos de hardware e software, utilizados em SCS que empregam RNA. O software é composto por programas, cujas execuções são responsáveis pela implementação das várias funções do sistema. No software existem também interfaces com o hardware (Kanoun, 1996), entre diferentes programas, e também as interfaces de operação, as IHMs (Interface Homem-Máquina). O software pode estar embutido em um sistema de propósito especial ou ser executado em um computador de propósito geral.

A SFMEA pode ser aplicada ao software em qualquer fase do ciclo de vida de desenvolvimento, i. e. do nível mais geral ao nível mais particular de componente,

dependendo das informações disponíveis e das necessidades do projeto. Quanto mais baixo o nível no qual a análise é executada, maior o nível de detalhe necessário para a análise e maior o número de modos de falha que devem ser considerados. Esta técnica analisa o sistema buscando identificar os efeitos das falhas que podem ocorrer no nível mais elevado do sistema. Estas falhas são então rastreadas nos sucessivos níveis funcionais inferiores até que a função mais básica tenha sido considerada (Toola, 1993). A SFMEA pode também ser considerada uma técnica de confiabilidade, já que ela pode identificar caminhos funcionais para os quais outras técnicas de verificação podem ser aplicadas para julgar a probabilidade que uma tarefa tem de executar sua função pretendida por um período de tempo específico.

Para executar uma SFMEA podem ser usados quaisquer produtos de modelagem (descrições textuais, especificações gráficas, tabelas, etc.) como entradas para o processo. Em um software de proteção de falhas no nível de sistema, por exemplo, para cada componente de software (função, módulo, subsistema) os seguintes passos são executados (Lutz, 1996b):

1. Descrever a operação normal do componente, com base na documentação de desenvolvimento do sistema, e outras informações disponíveis sobre o sistema.
2. Descrever as possíveis falhas funcionais do componente a ser protegido.
3. Descrever a operação normal do software de proteção de falhas na proteção do componente, i. e. como o software de proteção no nível do sistema responde a cada uma das falhas listadas acima.
4. Descrever os possíveis modos de falhas e efeitos do software de proteção de falhas. Este passo é o ponto crucial da SFMEA, uma vez que ele confirma

que o software protege contra falhas conhecidas e também que ele não vai introduzir novas falhas.

5. Classificar cada falha segundo a sua criticalidade/probabilidade. A classificação é um par ordenado onde o primeiro elemento descreve o nível de criticalidade e o segundo elemento a probabilidade de ocorrência da falha.
6. Usando as tabelas geradas nos passos 4 e 5, identificar qualquer ponto preocupante ou área potencialmente vulnerável.

Várias tentativas para automatizar um processo similar à SFMEA não forneceram um substituto para a SFMEA manual. Isto porque, ferramentas automáticas para a análise de falhas requerem a construção de um modelo do sistema, em primeiro lugar, e os resultados tendem a espelhar a compreensão do sistema aos olhos do projetista do modelo perdendo com isso as vantagens da validação independente da SFMEA. Análises mais rigorosas de estados de alcançabilidade, apesar de úteis, requerem um grande esforço e tempo de modelagem do sistema (Lutz, 1996b).

Para Lutz (1996a) a experiência de integração de análise indutiva (SFMEA), seguida de uma análise dedutiva (FTA), aprimorou a efetividade da SFMEA auxiliando na análise dos requisitos de software nas áreas críticas do software das espaçonaves Cassini e Galileo. Os objetivos eram reduzir o número de modos de falha, minimizar os efeitos dos modos de falha restantes e procurar por modos de falhas não previstos. Esta abordagem permitiu a descoberta de modos de falha desconhecidos, anomalias múltiplas ou coincidentes e dependências implícitas entre processos de software. O maior benefício da combinação da SFMEA com uma técnica de busca dedutiva (FTA), para as causas contribuintes, está na descoberta de modos de falhas desconhecidos

durante a análise de requisitos. A Figura 5.7 mostra uma visão geral do processo de análise usado.

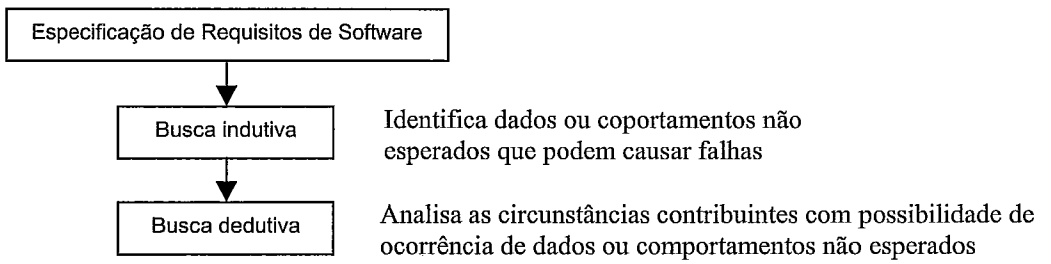

Figura 5.7 – Visão geral do processo de análise de vulnerabilidade usado na análise do software das espaçonaves Cassini e Galileo (Lutz, 1996a).

Como apontado por Lutz (1996b), a SFMEA apresenta diversas vantagens que não são encontradas em outros métodos de validação:

- A simplicidade do método torna fácil para os desenvolvedores revisar os resultados e incorporá-los nas iterações de desenho seguintes. É um processo sistemático.
- A SFMEA permite a validação do desenho independente de outros documentos existentes, i.e. não requer a tradução dos recursos existentes em um diagrama simplificado ou modelo de estados.
- Todos os métodos de validação analisam a correção do caso nominal, a SFMEA postula também a existência de dados ruins e atividades de software incorretas considerando os seus efeitos no software e no sistema.
- A SFMEA analisa a robustez do software na presença de anomalias, bem como a sua correção funcional.
- A SFMEA pode revelar anomalias não previstas.

A relação entre a SFMEA e outros métodos estáticos é mostrada nas Figuras 5.8 e 5.9.

A SFMEA tem algumas limitações e desvantagens. Como a maioria dos métodos de análise de falhas é um processo que consome um tempo considerável e na sua maior parte é um processo tedioso. Depende em grande parte do domínio de conhecimento do analista e da acurácia da documentação. Além disso, diferente da abordagem para hardware, não se pode construir uma lista completa de modos de falha para o software, e ainda, trata-se de um processo manual, não automatizável. Sózinha a SFMEA não contempla falhas múltiplas e/ou simultâneas.

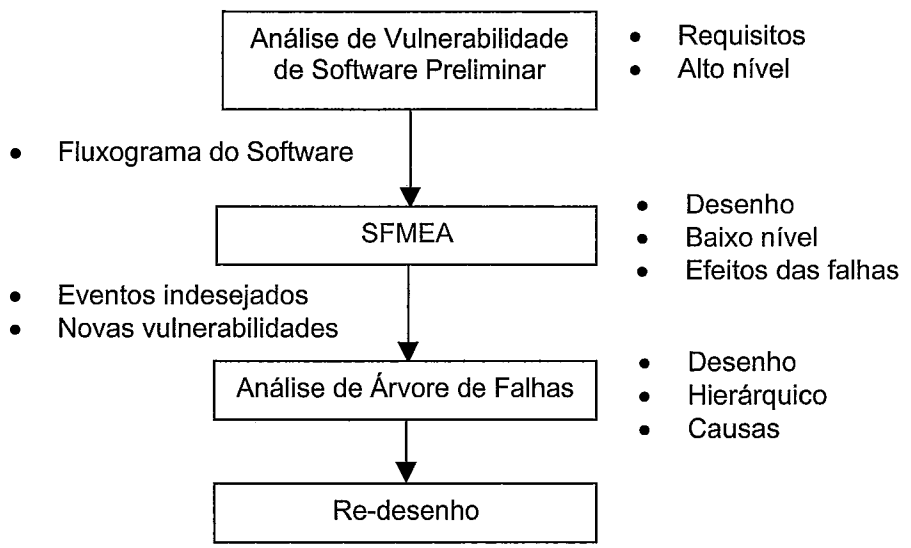

Figura 5.8 – Resultados da análise de vulnerabilidade de software preliminar (Lutz, 1996b).

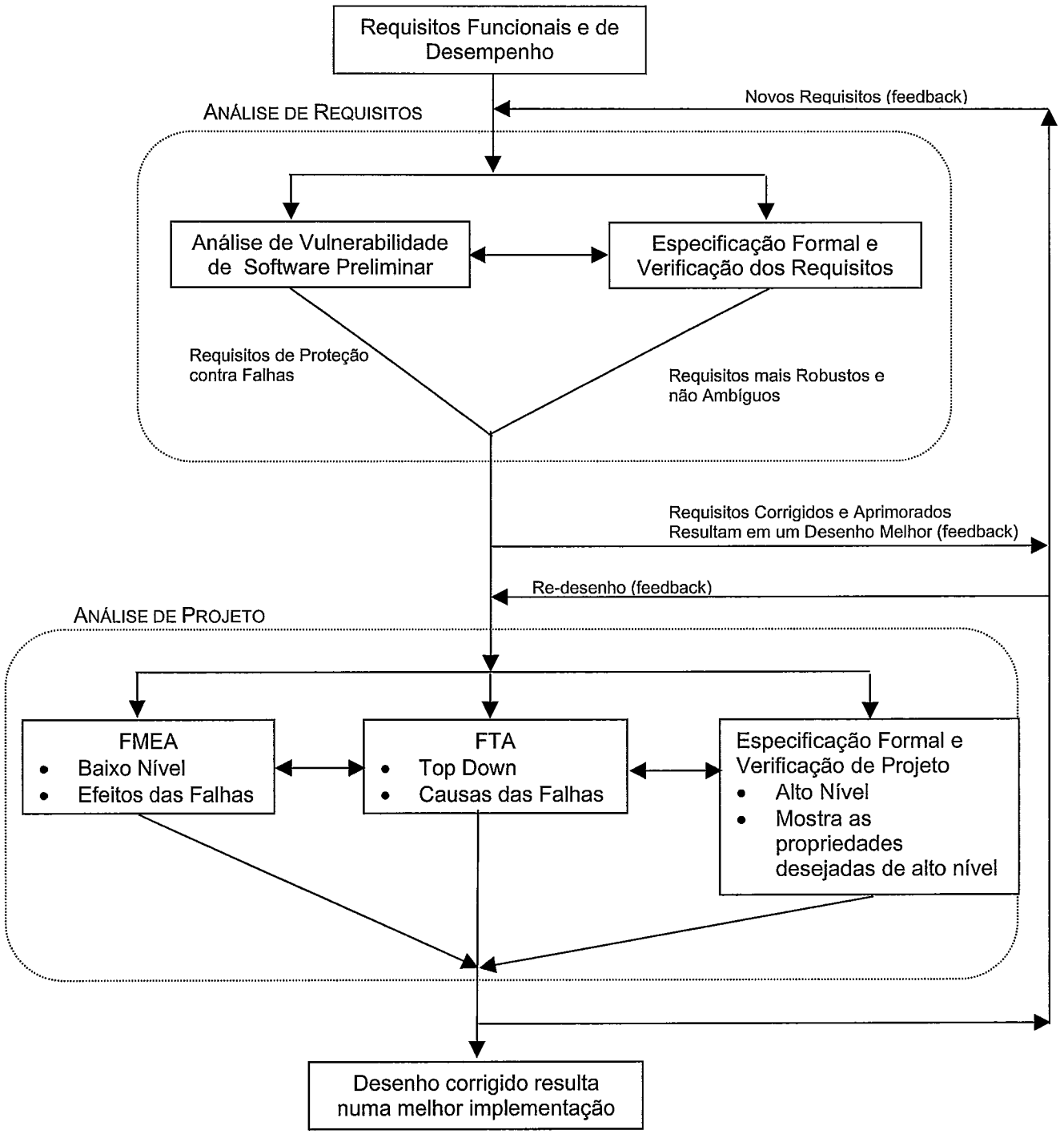

Figura 5.9– SFMEA e as ferramentas relacionadas (Lutz, 1996b).

5.3.1.2 Análise de Árvore de Eventos

A Análise de Árvore de Eventos (*Event Tree Analysis - ETA*) é uma técnica de análise de vulnerabilidade indutiva, desenvolvida para a indústria nuclear (Storey, 1996). A ETA toma como ponto de partida o evento que pode afetar o sistema e rastreia-o para-frente para determinar suas possíveis consequências. Esses eventos incluem tanto aqueles associados com a operação esperada do sistema como aqueles associados com as condições de falha. Para sistemas complexos isto resulta em árvores muito grandes, uma vez que são considerados tanto os eventos normais quanto os eventos de vulnerabilidades.

A estrutura básica da ETA é a árvore de decisão, conhecida como *árvore de eventos*. Cada nó numa árvore de eventos representa a ocorrência ou não de um evento, em geral uma falha. Numa árvore de eventos todos os nós que são equidistantes do nó raiz referenciam o mesmo evento, e o tempo deve prosseguir monotonicamente ao longo da árvore, ou seja, um nó evento não pode ocorrer antes que seu pai. A Figura 5.10 mostra um exemplo de árvore de eventos de um estudo de segurança para um reator nuclear.

Como já exposto a ETA é aplicada usando um sistema binário de estados, no qual cada bifurcação da árvore tem um estado falha e um estado de sucesso. Se for definido um número maior de estados para cada bifurcação, deve-se então adicionar um ramo para cada estado. Um problema é a explosão do número de caminhos possíveis, já que cada evento causa uma bifurcação no diagrama, e uma árvore com N eventos terá 2^N bifurcações. Este número pode ser reduzido se as bifurcações impossíveis forem eliminadas, mas ainda assim pode restar um grande número de caminhos.

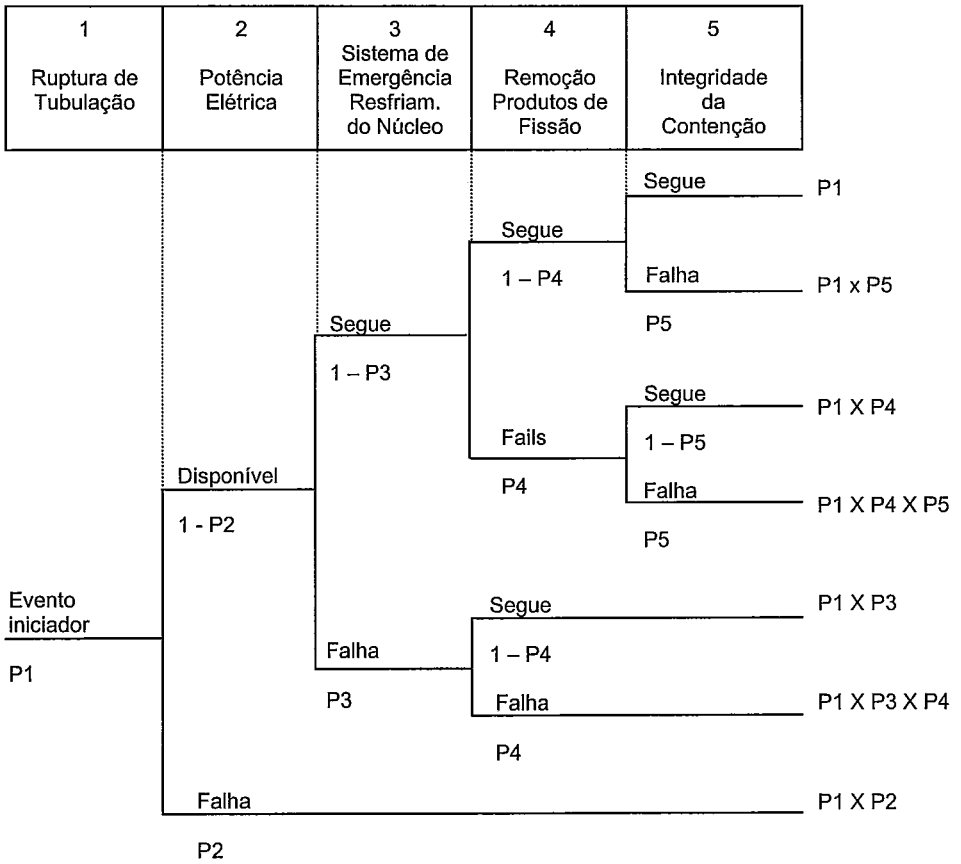

Figura 5.10 – Uma árvore de eventos reduzida para um acidente de perda de refrigerante (LOCA) em um reator nuclear.

Apesar de estarmos preocupados com a identificação de vulnerabilidades em primeiro lugar, deve ser observado que a ETA pode também ser usada para se obter uma medida da probabilidade de ocorrência de uma falha. Isto é feito através da atribuição de probabilidades a cada ramo da árvore correspondendo à probabilidade de ocorrência daquele caminho. Isto permite obter a probabilidade geral de uma falha através da combinação das probabilidades dos vários ramos. A Figure 4.7 mostra as probabilidades (*Pn*) que foram atribuídas a cada ramo. A expressão à direita de cada caminho é a probabilidade para aquele caminho. Devido a probabilidade de falha assumida ser muito pequena, a probabilidade de sucesso é sempre próxima a 1. Portanto, a probabilidade associada com as bifurcações acima (sucesso) na árvore é

assumida como 1. A probabilidade total de um caminho é encontrada através da multiplicação de todas as probabilidades dos ramos no caminho, e o risco total de um acidente é encontrado através da combinação das probabilidades de todos os caminhos que levam ao acidente. O evento inicial é expresso como uma frequência (eventos por ano), enquanto os eventos secundários são probabilidades (Leveson, 1995). A ETA pode ser muito útil na representação detalhada dos múltiplos cenários possíveis resultantes de uma falha. Entretanto, isto pode tornar-se muito caro porque as árvores podem tornar-se muito grandes.

Do ponto de vista do ciclo de vida de desenvolvimento a análise de árvore de falhas é apropriada somente no estágio de projeto detalhado e nos estágios subsequentes de desenvolvimento (Ippolito, 1995). É importante notar que uma análise geral para-frente deste tipo precisa limitar os eventos a serem considerados, devido ao potencial de crescimento da árvore de eventos.

O poder da análise da árvore de eventos é tal que, em arranjos muito complexos, ela permite investigar resultados de eventos em situações onde suas consequências não são fáceis de serem avaliadas (Storey, 1996). As árvores de eventos são melhores que as árvores de falhas no tratamento das noções de continuidade (lógica, temporal e física), porque permitem a introdução direta de fatores de tempo e variáveis aleatórias contínuas. Apesar disso, aspectos temporais podem causar problemas na construção de árvores de eventos, porque em alguns casos a lógica da falha muda dependendo da ordem de acontecimento dos eventos.

Uma desvantagem da ETA é a sua limitação a um único evento iniciador. No caso de análise probabilística ela torna-se consideravelmente complicada se os eventos não são independentes, e não é seguro assumir, ou fácil de provar, que dois eventos são independentes.

5.3.2 Técnicas de Análise de Vulnerabilidade Dedutivas

Técnicas dedutivas de análise de vulnerabilidade enfocam os estados de falhas ou eventos que podem resultar em um acidente através da busca para-trás de forma dedutiva, começando pelo incidente/acidente.

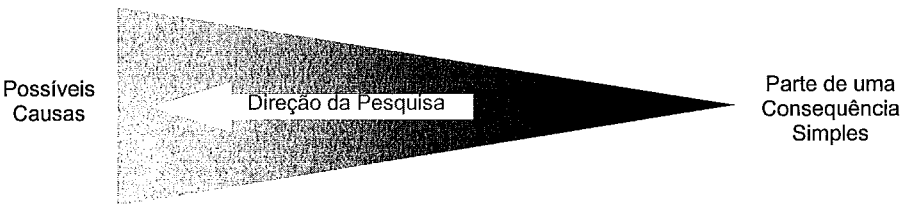

Figura 5.11– Busca para-trás partindo de uma consequência para as possíveis causas.

5.3.2.1 Análise de Árvore de Falhas (FTA)

A análise de árvore de falhas (*Fault Tree Analysis – FTA*) é um método gráfico dedutivo (IEC, 1990) que começa com um evento diretamente relacionado a uma vulnerabilidade, o *evento topo*, e trabalha para-trás para identificar sua(s) causa(s). Trata-se de uma abordagem top-down cuja entrada consiste no conhecimento das funções do sistema bem como dos seus modos de falha e efeitos. O resultado da análise é um conjunto de combinações de falhas de componentes que pode levar a um incidente específico ou acidente (o evento topo). A natureza gráfica da árvore de falhas que usa símbolos padronizados nas suas instruções, simplifica a interpretação. Duas normas (IEC, 1990), (Veseley, 1981) definem esses símbolos. Os símbolos para construção de árvores de falhas da norma (IEC, 1990) são mostrados na Figura 5.12.

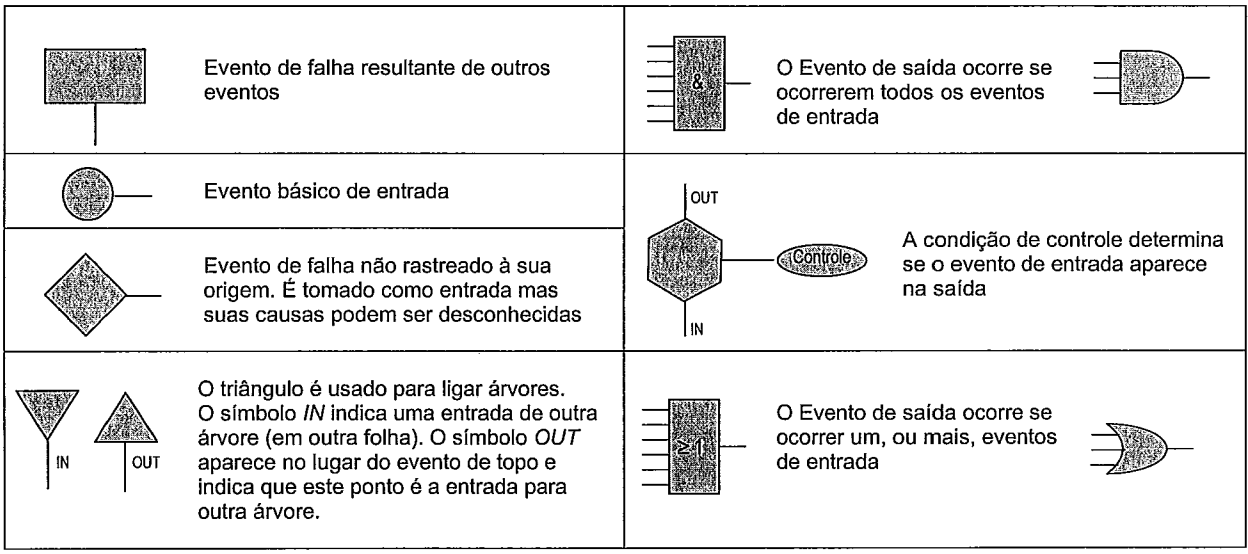

Figura 5.12 – Símbolos da árvore de falhas da norma IEC 1025 (IEC, 1990).

Um sistema é analisado de forma que cada evento topo identificado revele suas possíveis causas, que vão popular os níveis inferiores da árvore de falhas. Cada folha da árvore pode ser encarada como uma outra árvore de falhas correspondente a um sistema (subsistema componente). O nível de abstração escolhido define a extensão da análise, i. e. quais componentes são considerados básicos (Hansen, 1998). Uma árvore de falhas não é um modelo de todas as possíveis causas para falha do sistema, mas dada uma falha particular, ela revela as possíveis combinações de falhas de componentes que podem levar a uma falha específica do sistema. Árvores de falhas podem ser usadas qualitativamente para determinar se, ou como, é possível a ocorrência de um evento de topo particular através da inspeção de conjuntos de corte (*cut sets*). Um conjunto de corte é um conjunto de eventos primários tal que todos os eventos são necessários e suficientes para causar o evento topo (Lapp, 1977). Tradicionalmente a análise de árvore de falhas é uma metodologia probabilística. Pode-se executar uma análise quantitativa se forem fornecidas probabilidades para os eventos primários, apesar dos eventos dependentes serem um fator complicador da análise (Fenelon, 1993).

Entretanto, a validade de métodos probabilísticos tem sido discutida, devido as taxas de falha reais terem excedido os valores calculados em várias ordens de magnitude (Littlewood, 1992). Este é um ponto de grande debate, especialmente no que diz respeito ao software, devido à diferente natureza entre as falhas de software e de hardware. A FTA para software pode ser usada em diferentes estágios do ciclo de vida do software, variando de acordo com o nível de informação disponível. A Figura 5.13 mostra uma árvore de falhas de software de alto nível.

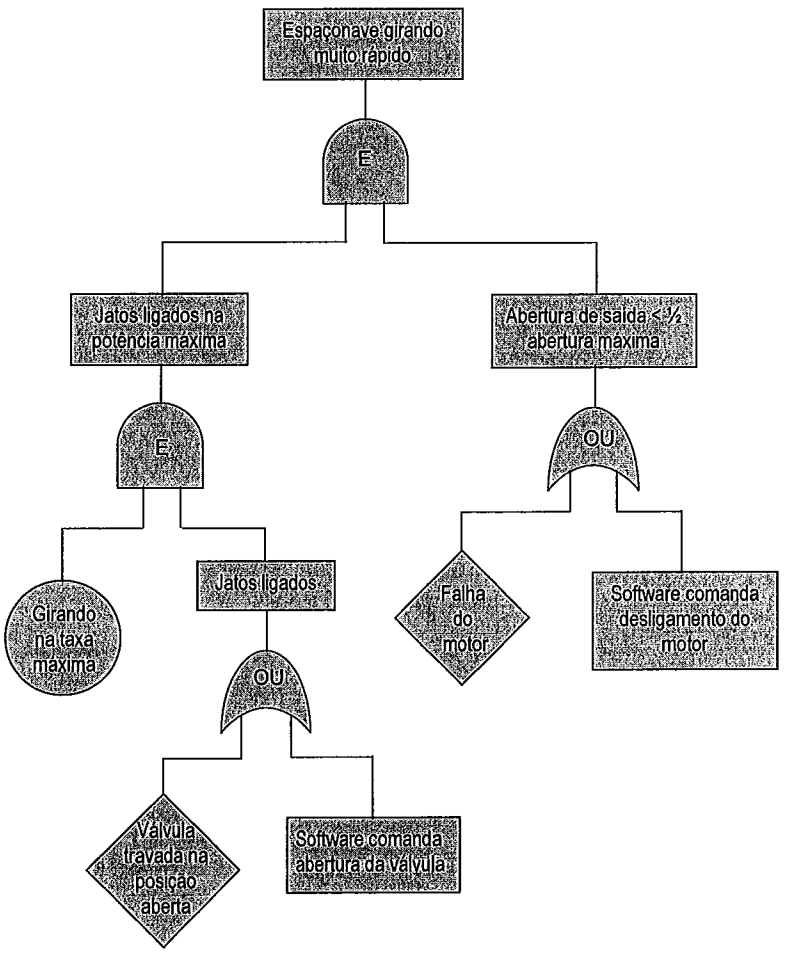

Figura 5.13 - Exemplo de uma árvore de falhas de alto nível para software (NASA-GB-1740, 1996).

Devemos observar que a análise de árvore de falhas de software não é probabilística já que os modos de falha de software são determinísticos (as falhas são sistemáticas). Mas, como observado em (Rushby, 1995), uma falha de projeto está

associada com a probabilidade de ocorrência da sequência de entrada que vai ativá-la. Assim, árvores de falhas de software podem ser interfaceadas com árvores de falhas de sistema quando se considera as interações de software com o resto do sistema (atuadores, sensores, etc.). Portanto, precisamos considerar aspectos probabilísticos de comportamento do sistema e integrar dados de análise de segurança de software com dados derivados do julgamento probabilístico de outras partes do sistema compondo assim árvores de falhas híbridas (software & hardware). A simulação de falhas humanas, tais como erros dos operadores, podem também ser analisadas usando a técnica de análise de árvore de falhas de software (Patterson, 1989).

Deve-se observar que o custo de aplicação da FTA no código de software será várias ordens de magnitude maior do que de uma FTA aplicada na fase de análise da especificação, e não será capaz de identificar erros conceituais relacionados à segurança cometidos na fase de especificação de requisitos. A FTA é usada frequentemente durante a fase de análise de vulnerabilidade preliminar (PHA) para identificar as principais vulnerabilidades de um sistema. A Figura 5.14 mostra uma árvore de falhas de um estudo de vulnerabilidade de um sistema de controle de tráfego aéreo (Leveson, 1997).

Em (Hansen, 1998) uma análise de árvore de falhas é acoplada a especificação de requisitos de tal forma que os requisitos de segurança de software podem ser derivados diretamente dos requisitos de segurança do sistema. Isto significa que se pode demonstrar que o software satisfaz os requisitos de segurança do sistema no desenvolvimento de sistemas críticos de segurança. É demonstrado como árvores de falhas resultantes da análise de segurança podem ser interpretadas diretamente como requisitos. A conexão toma por base um modelo dinâmico do sistema onde os estados são funções contínuas do tempo e o comportamento do sistema é definido por fórmulas

temporais que restringem o comportamento a um intervalo de tempo limitado. Os nós da árvore de falhas representam tais fórmulas enquanto que as portas da árvore de falhas representam fórmulas que expressam a ordenação temporal e causal dos eventos nos nós. Para especificar estas relações temporais eles usam uma lógica de intervalos de tempo real, o cálculo de duração.

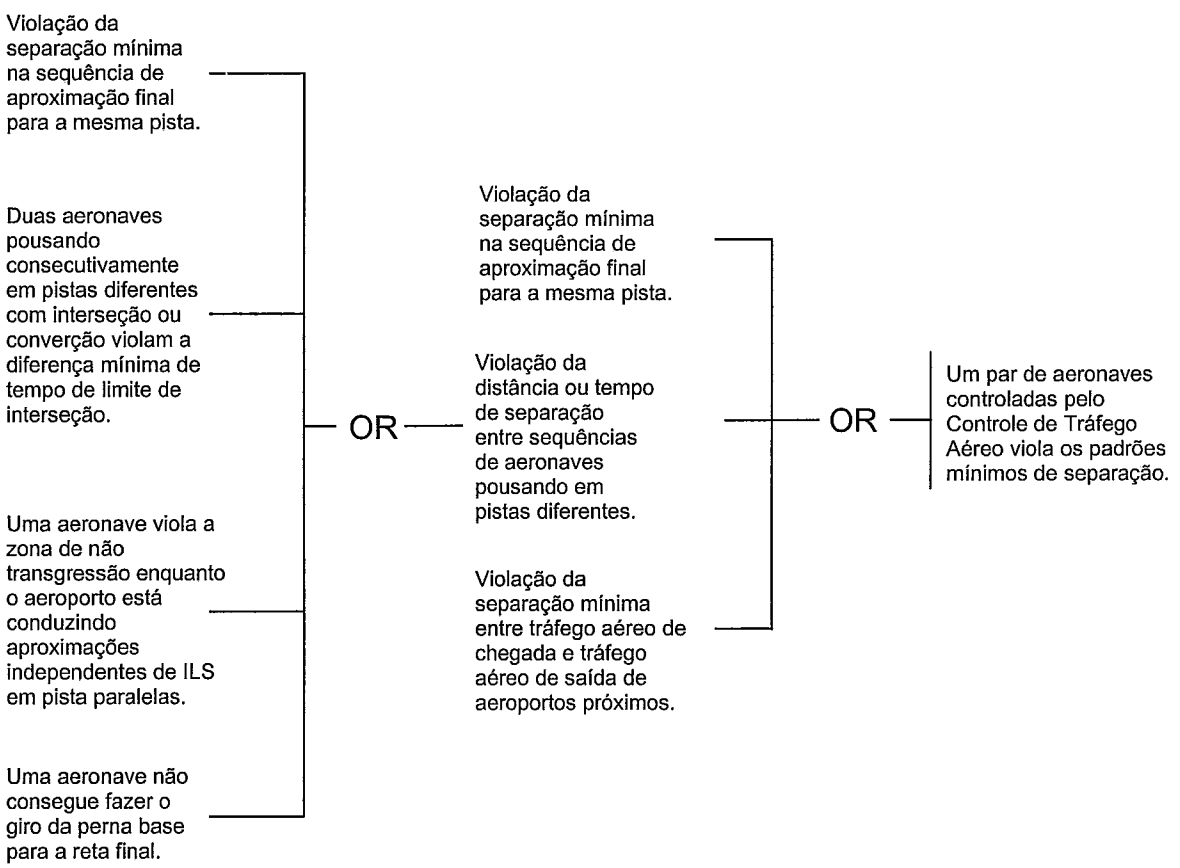

Figura 5.14 – Árvore de falhas de alto nível de uma Análise de Vulnerabilidade Preliminar (PHA), onde as vulnerabilidades consideradas resultam da perda de separação mínima entre duas aeronaves controladas pelo Controle de Tráfego Aéreo (Leveson, 1997).

Apesar de terem comprovado a sua utilidade na prática, as árvores de falhas confeccionadas manualmente tem alguns inconvenientes. A sua construção requer uma grande quantidade de tempo e esforço. Além disso estão sujeitas a erros lógicos e

omissões (Lapp, 1977). Por exemplo, diferentes analistas frequentemente constróem árvores de falhas para um mesmo sistema que são inconsistentes entre si. Isto se deve ao fato de que as portas de uma árvore de falhas podem ser interpretadas de forma ambígua, resultando algumas vezes em várias interpretações diferentes. Existem entretanto, ferramentas automatizadas de software que podem diminuir este esforço e ainda integrar diferentes abordagens às árvores de falhas e outros métodos tais como o HAZOP (Kuo, 1997). Outra desvantagem é a de que eventos com dependências temporais e de gradiente, tais como variações em parâmetros críticos do sistema e comportamento dinâmico, não são facilmente representáveis, nem os intervalos de tempo e a ordenação cronológica de eventos.

Uma vantagem da FTA é a de que ela pode tratar falhas múltiplas e também estudar falhas simultâneas de diferentes tipos, tais como erros humanos em conjunto com falhas de componentes.

5.3.2.2 Digrafos

A análise de árvore de falhas pode ser automatizada através do uso de grafos direcionados, referenciados na literatura como digrafos (Figura 5.15). Os digrafos são usados na análise de falhas para a construção de modelos de propagação de falhas. Os nós em um digrafo representam variáveis de processo, e as arestas direcionadas representam linhas de influência entre as variáveis. O procedimento básico para gerar uma árvore de falhas usando um digrafo é primeiro converter a descrição do sistema em um digrafo e então percorrer o digrafo de trás para a frente (i.e., na direção oposta das

arestas). A conversão da especificação em um digrafo pode ser automatizada se forem usados componentes padrão (Lapp, 1977).

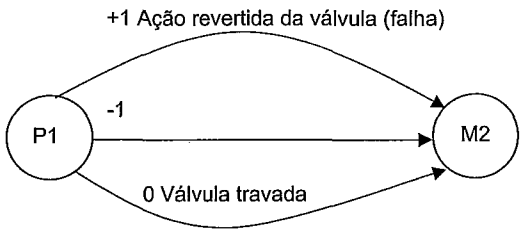

Figura 5.15 – Exemplo de um digrafo simples (Lapp, 1977).

Os digrafos podem ser aumentados através da especificação do tipo de influência que uma variável pode ter sobre outra variável (Iverson, 1995). Um digrafo pode ter arestas múltiplas entre dois nós. Neste caso, todas menos uma das arestas são qualificadas com condições de falha. Um aspecto interessante desta técnica é a possibilidade de se usar valores qualitativos nos desvios de valores de variáveis.

5.3.2.3 Análise de Vulnerabilidade de Máquina de Estados (*SMHA*)

A Análise de Vulnerabilidade de Máquina de Estados (SMHA) é um procedimento que foi desenvolvido para a fase de análise de requisitos de software (Leveson, 1995). A máquina de estados é um modelo dos estados de um sistema e das transições desses estados. O procedimento está baseado em um algoritmo que usa como entrada um modelo do sistema desenvolvido em uma linguagem formal de estados. O procedimento requer que os estados do sistema sejam agrupados em dois conjuntos

disjuntos, i. e. estados a partir dos quais é possível se alcançar estados vulneráveis e estados possivelmente não vulneráveis, e ainda aqueles estados a partir dos quais se pode alcançar somente estados não vulneráveis. É necessário também, definir um tipo particular de estado chamado crítico, que é um estado não vulnerável que pode levar a um estado vulnerável e a um estado não vulnerável. Assim, se um estado vulnerável é alcançável, deve existir um estado crítico no caminho a partir do estado inicial para o estado vulnerável. Esta é uma restrição necessária para execução da análise. Através do uso do modelo de estados inverso (quando as funções de entrada e saída são revertidas), é possível determinar se uma estado vulnerável é alcançável através do uso do estado vulnerável como estado inicial e determinar se o estado inicial original é alcançável. Entretanto, o grafo de alcançabilidade para-trás pode ser tão grande, ou ainda maior, do que o grafo original. É usado um algoritmo que não necessita que o grafo de alcançabilidade reverso seja gerado integralmente.

O algoritmo começa com um conjunto de condições vulneráveis. Para cada elemento deste conjunto são gerados os estados imediatamente anteriores. Cada um desses estados um-passo-para-trás é então examinado para averiguar se é um estado crítico potencial e se pode ser usado para eliminar um caminho para o estado vulnerável. O início não é feito com um conjunto de estados completo, mas somente com estados parciais (i. e. algumas condições nos estados não são importantes) e assim a composição completa dos estados alcançáveis (o conjunto completo de estados a partir do qual se vai começar a análise para-trás) não é conhecido no início do algoritmo. As componentes "tanto faz" (*don't care*) em cada estado são preenchidas com aquelas condições que são possíveis no processo de execução do algoritmo. Finalmente, só é necessário buscar um passo para-frente a partir de cada estado potencialmente crítico de forma a rotular o estado como crítico (i. e. existe um próximo estado que não é

vulnerável). Isto é assim porque se esse caminho também leva a um estado vulnerável, então o algoritmo vai eliminá-lo num passo futuro. Um modelo de rede de Petri inversa foi usado para demonstrar o uso da SMHA (Leveson, 1987).

Uma vez que a SMHA usa um modelo para executar a análise, ela pode ser usada em qualquer estágio do ciclo de vida, após terem sido identificadas algumas vulnerabilidades já que elas serão usadas como entradas. A maior limitação da SMHA é a necessidade de um modelo, que pode demandar um grande esforço, e ainda assim ser impraticável para sistemas grandes e/ou complexos. Outra limitação é a de que a análise é feita sobre o modelo do sistema, ficando assim dependente do quão bem o modelo reflete o sistema real.

5.3.3 Técnicas Exploratórias de Análise de Vulnerabilidade

O objetivo desse tipo de técnica de análise é usar uma combinação de busca para-frente e busca para-trás.

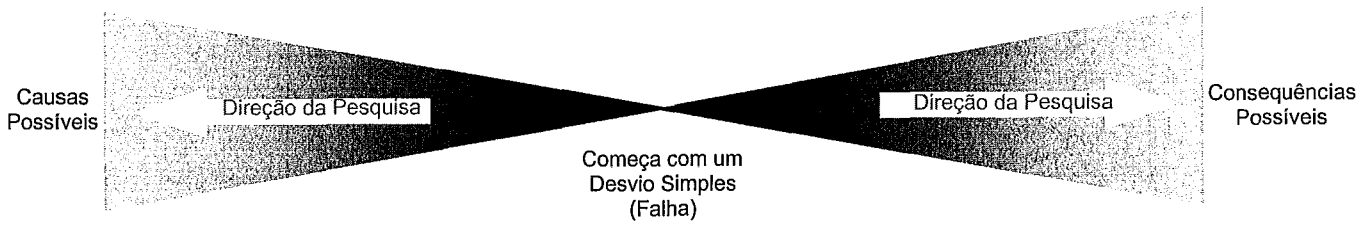

Figura 5.16 – A Técnica Exploratória trabalha em ambas direções, para-frente buscando as possíveis consequências e para-trás buscando as possíveis causas.

5.3.3.1 Estudos de Operabilidade e Vulnerabilidade (HAZOP)

A técnica HAZOP (*HAZard and OPerability studies*) foi desenvolvida inicialmente para a indústria química (Redmill, 1999). A HAZOP se baseia na teoria de que os acidentes são causados por desvios das intenções de projeto, ou de operação. O estudo HAZOP é um processo de identificação de vulnerabilidades desenvolvido por uma equipe através do exame de uma, ou mais, representações do desenho do sistema. Trata-se de um processo estruturado cujos quatro estágios sequenciais principais, são apresentados na Figura 5.17.

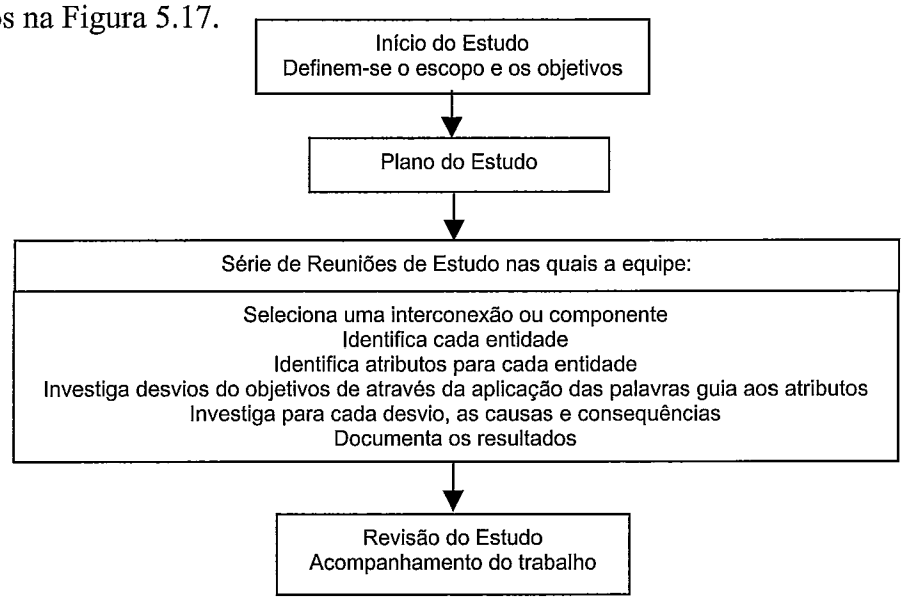

Figura 5.17 – Visão geral do processo de estudo HAZOP (MoD, 1996b).

Como definido em (MoD, 1996b), um estudo HAZOP é um exame sistemático formal, por meio de uma equipe, sob a gerência de um líder especialista, acerca das intenções de projeto de um sistema novo ou existente (ou partes de um sistema). O objetivo é identificar vulnerabilidades, operações inadequadas ou defeitos de entidades individuais no sistema como um todo no seu ambiente operacional. A Figura 5.18 mostra um fluxograma do processo de estudo HAZOP.

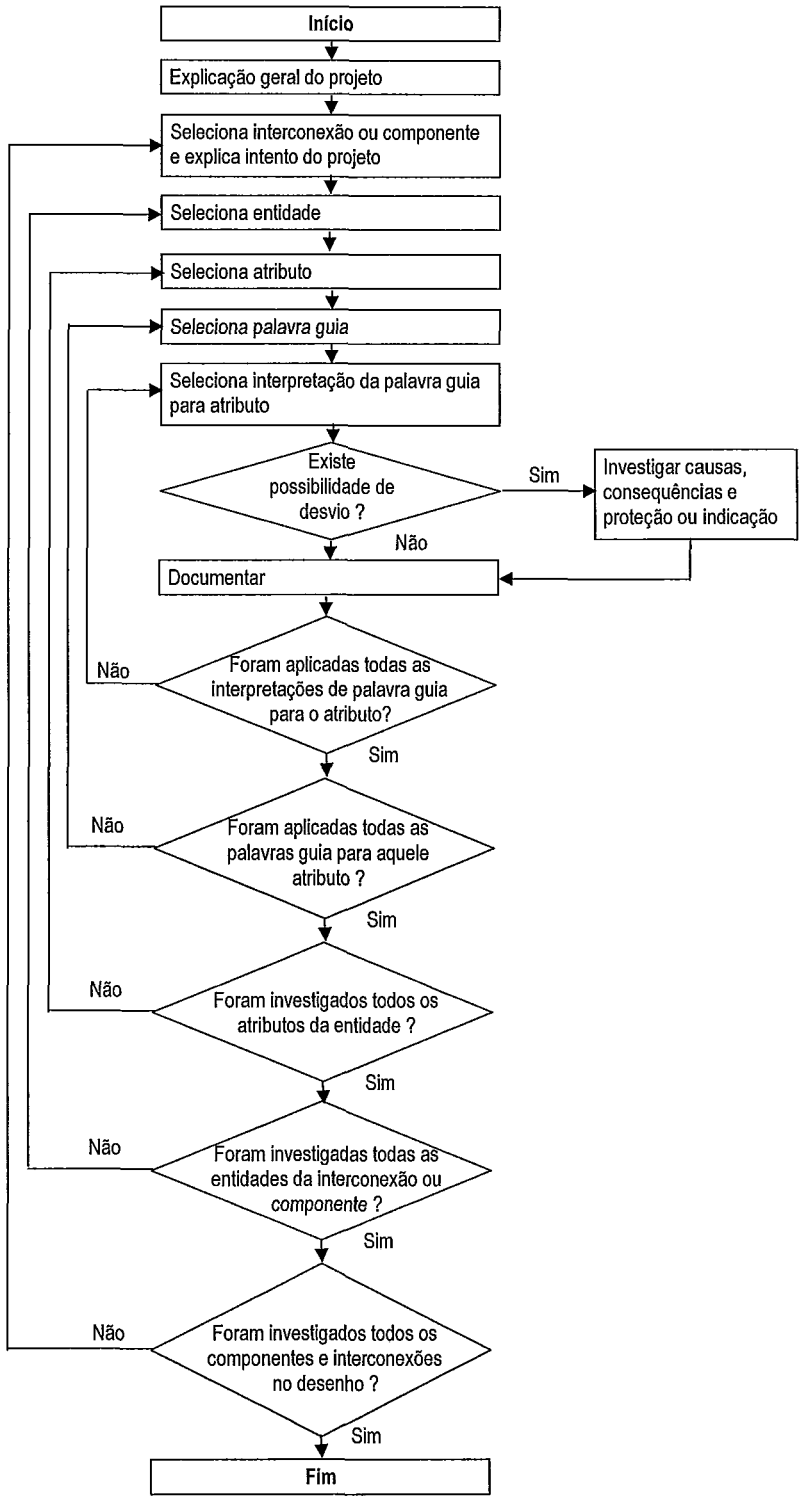

Figura 5.18 – Fluxograma do processo de estudo HAZOP (MoD, 1996b).

Tipicamente a HAZOP comporta várias reuniões de estudo. Nessas reuniões o líder coloca uma bateria de questões aos especialistas na tentativa de elicitar vulnerabilidades potenciais do sistema. Mais precisamente, o líder da reunião HAZOP

coloca situações anormais do sistema e a análise prossegue em ambas direções determinando se e como estas situações são possíveis e quais são os seus efeitos sobre o sistema. A técnica HAZOP é basicamente uma análise exploratória, já que nem falhas potenciais ou vulnerabilidades foram identificadas de antemão (McDermid, 1994).

A HAZOP é uma análise baseada em fluxo que usa diagramas de processos e fluxos. Estes elementos (entidades) tem atributos distintos e é aplicada uma lista de palavras guia a cada entidade para gerar um inventário de desvios do comportamento normal ou esperado. As Tabelas 5.4 e 5.5 mostram exemplos de interpretações de palavras guias para diagramas de transição de estados e temporização, respectivamente.

Atributo	Palavra Guia	Interpretação
Evento	Não	Evento não acontece
	Como também	Acontece outro evento além do esperado
	Outro	Ocorre um evento não esperado ao invés do evento esperado
Ação	Não	Ação não é executada
	Como também	É executada uma ação adicional (indesejada)
	Parte de	É executada uma ação incompleta
	Outra	É executada uma ação incorreta

Tabela 5.4 – Exemplo de interpretação de palavras guia de atributos para diagramas de estados (MoD, 1996b).

Em (MoD, 1996b) são dadas interpretações para palavras guia para representação de diagramas de fluxos de dados para projeto de software, diagramas de transição de estados, e projeto orientado para objetos. São também feitas algumas considerações acerca da escolha de atributos e sobre a interpretação de palavras guia quando estão disponíveis informações sobre temporização. A interpretação de palavras guia deve ser feita no contexto do sistema que está sendo estudado e da sua representação, porque elas devem ser entendidos por todos os membros da equipe que as estão usando.

Um exemplo de desvio é a aplicação da palavra guia MAIS, padrão HAZOP, à pressão de um tubo *P*. As duas perguntas são feitas aos analistas: 1) *Qual é o efeito de uma pressão alta no tubo P?* e, 2) *Como (ou o quê) pode causar aumento excessivo da pressão do tubo P ?*

Atributo	Palavra Guia	Interpretação
Tempo do Evento ou Ação	Não	Evento/Ação nunca ocorre
	Cedo	Evento/Ação ocorre antes do esperado
	Tarde	Evento/Ação ocorre após o esperado
	Antes	Acontece antes de outro evento ou ação que deveria precedê-lo
	Depois	Acontece depois de outro evento ou ação que se esperava vir após ele

Tabela 5.5 – Exemplo de interpretação de palavras guia para atributos de interpretação de temporização (MoD, 1996b).

Em (McDermid, 1994) é proposta uma técnica para adaptação do método HAZOP para um projeto de software. É usado um diagrama MASCOT, que codifica um modelo estrutural do desenho de software, lembrando uma linguagem de fluxo de dados O método fornece uma forma sistemática para executar uma HAZOP manual de diagramas de projeto tendo sido aplicada com sucesso a um sistema aeroespacial de tamanho moderado. O método é aplicável aos primeiros estágios do ciclo de vida de desenvolvimento, fornecendo um conjunto útil de entrada para os estágios de desenvolvimento subsequentes.

Em (Hebbron, 1997) é proposto a aplicação de estudos HAZOP para modelos de requisitos estruturados de tempo real. A abordagem proposta é um método evolutivo para a técnica HAZOP à análise de requisitos expressa numa notação de fluxo de dados de tempo real., usando os modelos comportamental e ambiental de Ward & Mellor. Esta abordagem é composta de dois estudos HAZOP distintos: um Modelo HAZOP Ambiental (*EMHAZOPS*) e um Modelo HAZOP Comportamental (*BMHAZOPS*). Os

estudos EMHAZOPS e BMHAZOPS desenvolvem quatro atividades similares: 1) Desenvolvem *threads* evento-resposta; 2) Preparação de tabelas de graduação; 3) Reuniões HAZOP; e 4) Uma comparação e avaliação dos resultados. A aplicabilidade da HAZOP a modelos de requisitos estruturados de tempo real é demonstrada e alguns problemas são considerados.

Em (Reese, 1997) é apresentada a Análise de Desvio de Software (*SDA*) que, como a HAZOP, está baseada na assertiva básica de que os acidentes são causados por desvios nos parâmetros de sistema. Para a SDA um desvio é diferença entre os valores real e o correto. A SDA pode determinar se um comportamento vulnerável de software pode resultar (geralmente uma saída) de uma classe de desvios de entrada. A SDA permite avaliar como as componentes de sistema vão se comportar em um ambiente não perfeito. A técnica requer como entrada uma especificação formal de requisitos de software, que é automaticamente convertida em um diagrama causal. Este diagrama causal é uma estrutura de dados interna que codifica informações causais entre variáveis de sistema com base na especificação e na semântica da linguagem de especificação. O procedimento aumenta então o diagrama causal com fórmulas de desvio de forma que sejam representados os desvios da variável. Neste ponto o procedimento está pronto para o analista identificar as saídas de software crítico quanto a segurança e pelo menos um desvio no ambiente de entrada. São passados para o programa de busca o diagrama causal aumentado, as saídas de software crítico e os desvios iniciais, que constrói uma árvore de estados. Os desvios iniciais estão na raiz da árvore de busca. As folhas tanto podem ser caminhos sem saída (nas quais o estado não contém qualquer desvio) ou estados contendo desvios críticos de segurança. A busca está baseada na aplicação de matemática qualitativa ao diagrama causal. A matemática qualitativa particiona domínios infinitos em um pequeno conjunto de intervalos e permitindo a execução de

operações matemáticas nesses intervalos. O uso de intervalos fixos simplifica a análise se comparada às iterações necessárias sobre o espaço de estados inteiro. A saída é uma lista de cenários, que são conjuntos de desvios nas entradas de software mais as restrições nos estados de execução do software que são suficientes para levar a um desvio na saída de software crítico.

A HAZOP tem sido usada extensivamente em muitos domínios de aplicação e o seu sucesso pode ser atribuído a vários fatores. A natureza hipotética das questões da HAZOP encoraja a antecipação imaginativa das vulnerabilidades (McDermid, 1994). Além disso, a diversidade do conhecimento e experiência dos membros da equipe HAZOP ajuda a garantir a investigação através de todas as propriedades do sistema (Leveson, 1995). Esta abordagem interdisciplinar ajuda a prevenir a resolução de um problema em uma área e criar novos problemas em outras áreas. A HAZOP pode ser aplicada em todos os estágios do ciclo de vida de desenvolvimento já que ela é usada para garantir uma avaliação sistemática dos aspectos funcionais do sistema (Place, 1993).

Entretanto, a HAZOP tem também várias desvantagens. Ela requer tempo e esforço, em grande parte devido a sua dependência em reuniões da equipe (Leveson, 1995). Como técnica exploratória a HAZOP analisa causas e efeitos relacionados a desvios do comportamento esperado, mas não analisa se o desenho, sob condições normais de operação, produz o comportamento esperado ou se o comportamento esperado é o desejado (Reese, 1997). Como é uma análise baseada em fluxos os desvios que se originam dentro dos componentes não são investigados diretamente pelos estudos HAZOP. Mais precisamente, um desvio de dentro de um componente (tal como falhas de operação ou outras perturbações ambientais) é assumido como sendo uma manifestação de distúrbio de fluxo (Suokas, 1988). Uma explosão combinatória pode

resultar se forem analisados desvios de processos em conjunto com estados de entrada e saída do hardware do computador (Broomfield, 1997).

5.3.3.2 Metodologia de Fluxograma Dinâmico

A Metodologia de Fluxograma Dinâmico (*Dynamic Flowgraph Methodology - DFM*) (Garret, 1995, 1998) foi desenvolvida como uma ferramenta para modelar e analisar sistemas embutidos. O hardware e o software embutido do sistema são representados e analisados segundo uma abordagem integrada. A parte software pode estar na forma de especificação de projeto. Dessa forma a DFM pode ser aplicada para validar o desenho (projeto) do sistema e encontrar erros de projeto antes que seja muito tarde, ou muito caro, para corrigi-los, ou ainda para verificar a implementação do sistema e averiguar se ele atende aos requisitos de projeto. A DFM está baseada na Metodologia Lógica de Fluxogramas (LFM), que é um conceito de análise de sistemas com características dinâmicas limitadas. A LFM foi originalmente desenvolvida como um método de análise e diagnóstico de plantas de processos com malhas de controle com retroalimentação e *feedforward*.

A DFM é também uma metodologia para análise e teste de sistemas críticos embutidos. Está direcionada para o fato específico de que a análise de segurança de sistemas embutidos não pode ser executada de forma efetiva se o processo de análise e qualificação das partes de software e hardware foi executado em isolamento relativo sem uma boa compreensão e representação integradas das funções e interfaces do sistema de forma global. Como se espera que os testes de software sejam um dos pilares de qualquer procedimento de verificação de confiabilidade, esta metodologia é

projetada de forma a prestar assistência ao processo de testes do software. Os principais propósitos da DFM, como ferramenta de análise de sistemas embutidos, são: 1) Identificar como certos eventos (desejáveis ou não) podem ocorrer no sistema; e 2) Identificar uma estratégia de testes apropriada baseada na análise do comportamento do sistema. Modelos de sistema, que expressam a lógica do sistema em termos de relacionamento causal entre variáveis físicas e características temporais da execução dos módulos de software, são analisados para determinar como um certo estado (desejável ou não) pode ser alcançado. Isto é feito através do desenvolvimento de árvores de falhas para um dado evento de topo (traduzido em termos de estados de uma ou mais variáveis do sistema) através de um rastreamento para-trás ao longo do modelo de forma sistemática. Estas árvores de falha temporizadas tomam a forma de combinações lógicas de árvores estáticas relacionando parâmetros do sistema a diferentes pontos do tempo. As informações resultantes relativas aos estados de hardware e software que podem levar a certos eventos de interesse podem então ser usadas para identificar aqueles caminhos de execução de software e as condições associadas de hardware e do ambiente que são potencialmente capazes de levar a falhas sérias no sistema implementado. O esforço de verificação pode então ser focalizado nessas características críticas de segurança do sistema para garantir que tais falhas não possam ocorrer.

A DFM envolve dois passos principais. No primeiro passo, um modelo do sistema embutido é construído expressando a lógica e a dinâmica do sistema em termos dos parâmetros de software e hardware. A Figura 5.19 apresenta os blocos de construção de um modelo DFM. O modelo integra uma "rede causal" que descreve os relacionamentos funcionais entre os parâmetros de software e hardware, uma "rede condicional" que representa comportamentos discretos do software devido a ações de

chaveamento condicional e desempenho descontínuo de hardware devido a falha de componentes, e uma "rede de transações-temporais" que indica a sequência nas quais as subrotinas de software, e as ações de controle, são executadas.

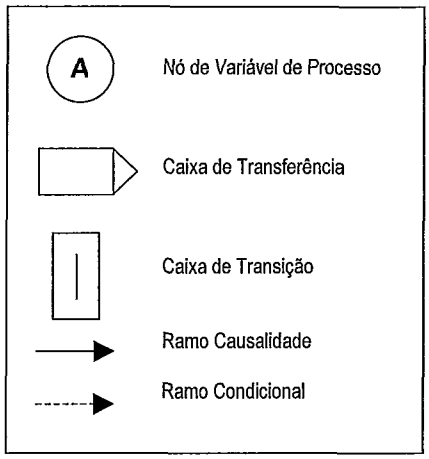

Figura 5.19– Blocos de construção de um modelo DFM.

No segundo passo, o modelo desenvolvido no primeiro passo é analisado para determinar como o sistema pode alcançar um certo estado. Isto é feito através do desenvolvimento de árvores de falhas "temporizadas" para determinados eventos de topo (traduzidos em termos de estados de um ou mais parâmetros de software) através de um rastreamento reverso sistemático ao longo do modelo. Estas árvores de falhas "temporizadas" tomam a forma de uma sequência de árvores estáticas relacionando parâmetros do sistema a diferentes pontos no tempo, essencialmente uma série de *snapshots* da evolução do sistema. Toda a informação necessária para construir as árvores de falha temporizadas estão implicitamente contidas no modelo DFM desenvolvido no primeiro passo. Além disso, a base de conhecimento estabelecida no primeiro passo e a abordagem algorítmica do rastreamento reverso permitem que este processo seja completamente automatizado. Portanto, no segundo passo, podem ser construídas várias árvores de falhas "temporizadas" para analisar diferentes eventos de

topo usando um único modelo DFM. Deve-se observar que os resultados de uma análise DFM são obtidos na forma de árvores de falhas temporizadas, que mostram como os estados do sistema investigado podem evoluir. Assim a DFM compartilha algumas características com a análise de árvore de falhas tradicional na forma final dos resultados fornecidos. Entretanto, as diferenças são que a abordagem da DFM fornece um modelo documentado do comportamento e das interações do sistema, bem como uma estrutura para modelar e analisar comportamentos tempo-dependentes, ambos não fornecidos pela análise de árvore de falhas tradicional (Garret, 1995).

5.3.3.3 Redes de Petri

As redes de Petri (*Petri Nets - PN*) (Valette, 1994) são uma técnica gráfica para descrição formal de sistemas cuja dinâmica é caracterizada pela concorrência, sincronização, exclusão mútua e conflito. Elas podem ser usadas para modelar e analisar sistemas críticos de segurança em relação a propriedades tais como alcançabilidade, recuperabilidade, *deadlock* e tolerância a falhas. As PN permitem identificar relacionamentos entre componentes de sistema tais como software e hardware, interferência ambiental e interação humana ou efeitos em ambos software e hardware. Técnicas de PN de tempo real permitem a construção de modelos dinâmicos que incorporam informações de tempo. Pode-se então monitorar e verificar a sequência e o escalonamento de ações do sistema de estados que podem levar a situações vulneráveis. A modelagem PN é diferente da maioria dos outros métodos de análise pelo fato dela demonstrar de forma clara a progressão dinâmica dos estados de transição. As redes de Petri podem também ser traduzidas em expressões de lógica matemática que podem ser

analisadas por ferramentas automatizadas. Essas informações podem ser extraídas e reformuladas em grafos e tabelas de auxílio à análise relativamente fáceis de entender (e.g. grafos de alcançabilidade, grafos de redes de Petri invertidos, grafos de estados críticos). As principais vantagens das PN sobre as outras técnicas de análise de segurança são (NASA-GB-1740.13-96, 1996): 1) As PN podem ser usadas para derivar requisitos de tempo em sistemas de tempo real; 2) As PN permitem que os sistemas sejam descritos usando uma notação gráfica; 3) As PN podem ser construídas em qualquer fase do ciclo de vida de desenvolvimento do sistema (e do software); 4) Elas podem ser aplicadas para determinar a análise de pior caso e os riscos potenciais de falhas de tempo; 5) As PN permitem uma abordagem global do sistema já que podem modelar o hardware, o software e o comportamento humano do sistema usando a mesma linguagem; 6) As PN disponibilizam uma linguagem de modelagem que pode ser usada para análise e simulação formal; e 7) Adicionando tempo e probabilidades a cada umas das PN incorpora-se informações de tempo e probabilidade à análise.

Uma PN é composta de nós (lugares), transições e arcos, que definem a sua composição estrutural. Os nós são usados para descrever possíveis estados locais do sistema (chamados de condições ou situações). Transições são usadas para descrever eventos que modificam o estado do sistema. Os arcos especificam a relação entre os estados locais e os eventos de duas formas: 1) o estado local no qual o evento pode ocorrer, e 2) a transformação de estado local induzida pelo evento. Símbolos são marcadores que residem nos nós, e são usados para especificar o estado da PN. Se um nó descreve uma condição, pode assim conter um ou nenhum símbolo, a condição é verdadeira se um símbolo está presente e falsa no caso contrário. Se um nó define uma situação, o número de símbolos contido no nó é usado para especificar a situação. Um modelo PN, que é um modelo de um sistema real que usa o paradigma descritivo PN, é

representado graficamente por um grafo bipartido no qual os lugares (nós) são desenhados como círculos, e as transições são desenhadas na forma de barras ou caixas. Os símbolos são desenhados como pontos pretos dentro dos nós. A Figura 5.20 mostra um modelo simples de rede de Petri para um interruptor, no qual os lugares (nós) definem as condições (*on* e *off*) conectados por quatro arcos a duas transições (*switch-on* e *switch-off*).

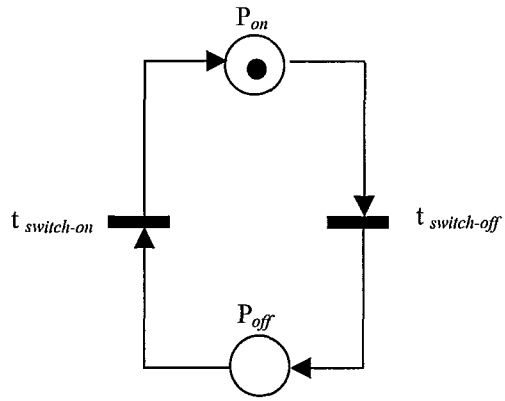

Figura 5.20– Modelo de rede de Petri de um interruptor (o nó P_{on} contém o símbolo, condição verdadeira).

Infelizmente, as PN requerem uma grande quantidade de análise detalhada para construir mesmo pequenos sistemas, tornando-as assim muito caras. Para diminuir este problema, foram propostas algumas alternativas de técnicas de modelagem PN, cada uma talhada para executar um tipo específico de análise de segurança (Leveson, 1987). Por exemplo:

- Redes de Petri Temporizadas (TPN), que levam em conta o fator dependência de tempo em sistemas de tempo real (Erau, 1995);
- Redes de Petri Inversas, especificamente necessárias para executar análise de segurança, onde não é necessário modelar todos os possíveis estados alcançáveis;

- Redes de Petri de Estado Crítico Inversas, que refinam a análise das PN modelando apenas os estados alcançáveis nos níveis de criticalidade predefinidos.

5.3.3.4 Notação de Transformação e Propagação de Falhas

Fenelon et al. (Fenelon, 1993) (Fenelon, 1994) desenvolveu a FPTN (*Failure Propagation and Transformation Notation*), uma notação integrada para melhorar algumas das limitações da FMEA e da FTA quando aplicadas a software. Trata-se de uma notação gráfica que permite trabalhar nos dois modelos: no modo *top-down* da FTA e no modo *bottom-up* da FMEA. A FPTN permite também modelar o comportamento de falhas de um sistema. Na FPTN um sistema é representado através de uma série de módulos, cada um deles correspondendo a um elemento funcional dentro do sistema. Os módulos tem atributos padronizados: 1) um nome; 2) um nível de criticalidade; e (onde aplicável) 3) indicação de qualquer mecanismo de recuperação que possa existir naquele módulo (Figura 5.21). Os módulos podem ser aninhados hierarquicamente, e podem ser tanto do tipo "caixa preta" (módulos que não podem ser decompostos no mais baixo nível de abstração considerado) ou "caixa branca" (decomponíveis com uma estrutura interna conhecida). Na FPTN, os módulos são conectados através de falhas que se propagam entre eles, e não através dos dados que fluem normalmente entre eles.

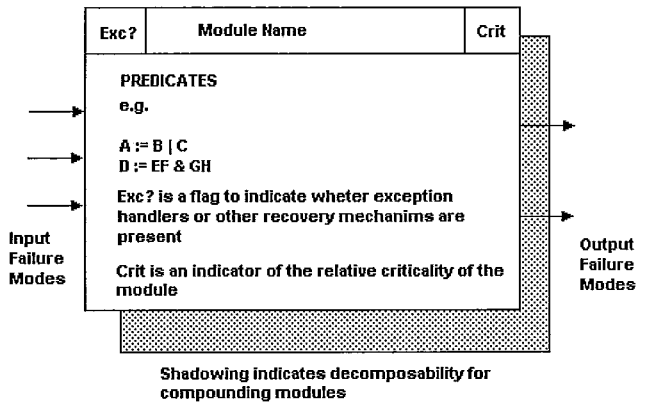

Figura 5.21– Alguns elementos da notação FPTN (Fenelon, 1993).

Dentro de cada um dos módulos da FPTN deve ser listado o seguinte (Fenelon, 1994):

- Os modos de falha que são gerados internamente pelo próprio módulo, i. e. aqueles que não dependem de qualquer condição externa.
- O modos de falha que são incondicionalmente impedidos de se propagar além do módulo (i. e. falhas identificadas por tratadores de exceção ou outros mecanismos de recuperação).
- Um conjunto de equações caracterizando o relacionamento entre a entrada e a saída dos modos de falha, onde os modos de falha de entrada são aqueles para os quais o componente está suscetível e os modos de falha de saída são aqueles para os quais eles passam para o ambiente.

No seu nível mais básico a FPTN pode funcionar como uma abstração e representação de árvores de falha complexas e tabelas de FMECA. Nos primeiros estágios de uma análise de segurança a FPTN pode ser usada também como uma

ferramenta de modelagem da arquitetura quando o sistema é decomposto numa série de módulos FPTN que refletem a estrutura operacional do software.

5.3.3.5 Análise Causa e Consequência

A Análise Causa e Consequência (*Cause Consequence Analysis - CCA*) combina as buscas para-frente e para-trás. A CCA começa com um evento crítico e determina as causas do evento (usando buscas *top-down* ou busca para-trás) e as consequências que podem resultar dele (busca para-frente). A CCA usa uma representação de sistema na forma de diagrama de blocos onde os blocos são conectados por arcos (as ligações causais). As Figuras 5.22 e 5.23 mostram os principais símbolos dos diagramas causa-consequência. O diagrama pode mostrar tanto as dependências de tempo como os relacionamentos entre eventos. Nos diagramas causa-consequência o sistema é descrito através de caixas de eventos que descrevem alterações nos estados do sistema e caixas de decisão que descrevem efeitos alternativos de um evento dependendo de uma caixa de condição ou árvore de condição (Nielsen, 1975). Árvores de condição são árvores de falhas, onde o evento de topo é uma condição que torna a caixa de decisão verdadeira, e uma caixa de condição é um nó simples de uma árvore de condição. A notação da técnica permite, por exemplo, a representação de relacionamentos inibidores entre eventos tais como mútua exclusão.

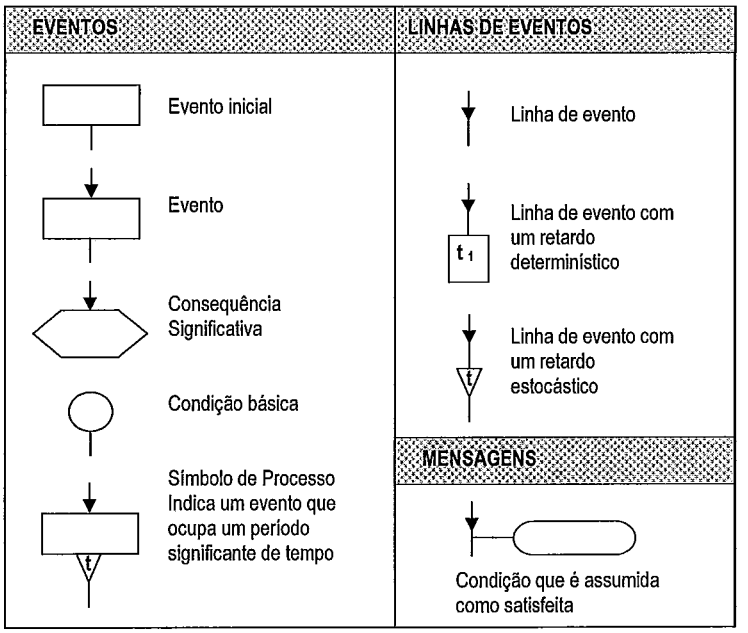

Figura 5.22– Símbolos de diagramas Causa-Consequência (Nielsen, 1975).

Comparado com a técnica de árvore de falhas quando usada sozinha, a CCA tem duas vantagens (Taylor, 1975): 1) Ela mostra explicitamente a sequência de eventos, que é útil no estudo de partida, desligamento, e outras sequências de problemas de controle, e 2) Os diagramas CCA podem representar intervalos (atrasos) de tempo, caminhos alternativos de consequências, e combinações de eventos.

A CCA tem algumas desvantagens, por exemplo, os diagramas podem tornar-se muito grandes e difíceis de serem usados, são necessários diagramas separados para cada evento iniciador, e os resultados só estão relacionados com a causa que está sendo analisada, apesar de que eles poderiam ser causados por outros eventos iniciadores. A Figura 5.24 mostra um diagrama de causa-consenquência simples.

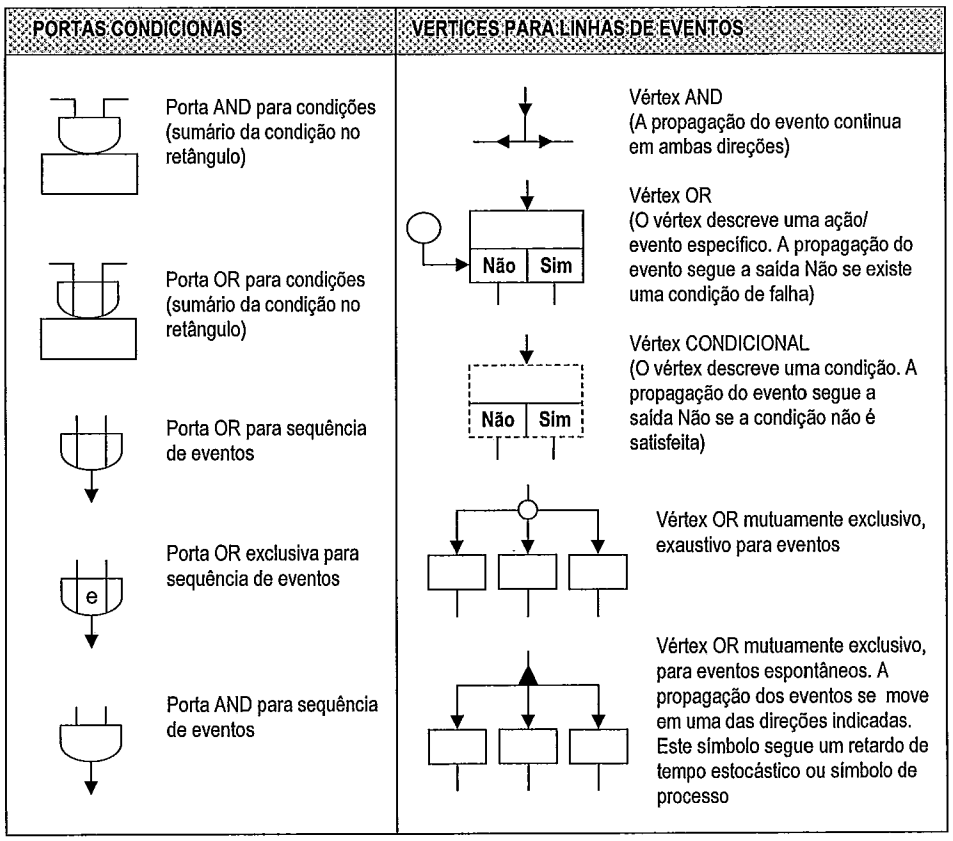

Figura 5.23– Símbolos usados nos diagramas Causa-Consequência (cont.) (Nielsen, 1975).

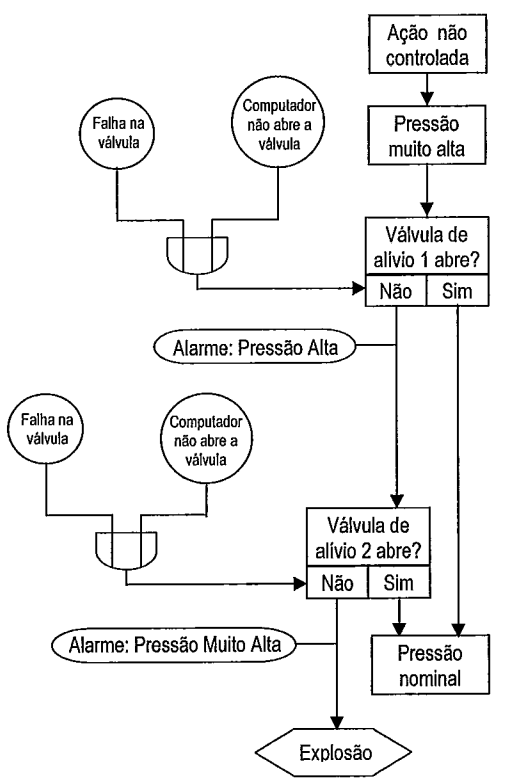

Figura 5.24 – Exemplo de um diagrama Causa-Consequência.

5.3.3.6 Software Sneak Analysis

A Análise *Sneak* para Software (*Software Sneak Analysis - SSA*) baseia-se na análise (*sneak circuit analysis - SCA*) desenvolvida para avaliar circuitos elétricos, e adaptada mais tarde para software (Ippolito, 1995). A SSA é usada em sistemas críticos para identificar caminhos latentes que podem causar a ocorrência de funções não desejadas ou inibir funções desejadas, assumindo que todos os componentes estão funcionando como devem. A SSA toma por base a análise da documentação de engenharia e manufatura. Trata-se de uma técnica de trabalho intensivo que requer treinamento especializado, e devido ao seu alto custo, deve ser conduzida somente em áreas onde o potencial de vulnerabilidade é alto (NASA-PD-AP-1314, 1995).

A SSA e a SCA podem ser usadas juntas para analisar um sistema inteiro, o que permite a identificação de vulnerabilidades causadas pela interação entre software e hardware, software e humano, e hardware e humano. Em (Juslin, 1995) a SSA é vista como uma combinação das abordagens FMEA e FTA, onde as saídas de software que levam a eventos vulneráveis são identificadas e suas causas estudadas.

Sneaks são condições, ou falhas, de projeto que foram incorporadas inadvertidamente ao desenho (projeto) de hardware, software e sistemas integrados. Elas não são causadas por falha de componentes. As três razões principais que podem causar *sneaks* são:

1) Complexidade do sistema;
2) Complexidade organizacional (organizações grandes e complexas frequentemente desenvolvem produtos com vários *sneaks*, principalmente devido a dificuldade de definir acuradamente a consistência das interfaces – problemas de integração de sistema); e

3) Rápidas mudanças de tecnologia (quando o tempo necessário para analisar e testar novos sistemas é reduzido).

A SSA deve ser usada para descobrir o software que causa uma ou mais das seguintes condições *sneak*:

1) Caminhos sneak – que desviam o fluxo da lógica para caminhos inesperados.
2) Temporização sneak – que pode causar ou impedir a ativação (ou inibir) de funções em um tempo inesperado.
3) Indicações sneak – que podem causar a apresentação de *displays* ambíguos, ou falsos, das condições de operação do sistema.
4) Rótulos sneak – que podem causar erros de operação devido a ativação de controles não apropriados.

A SSA foi usada primeiramente em código fonte de software, onde este é convertido numa rede topológica de árvores que são examinadas para identificar quais dos seis padrões topológicos (derivados de circuitos elétricos) aparecem nas árvores. A análise é executada usando listas de verificação de questões sobre o uso e o relacionamentos entre os componentes topológicos básicos. A SSA usa como base a documentação do sistema na forma de diagramas, grafos e desenhos. A fase preferida para começar a SSA é a fase de projeto, mas pode também ser executada durante qualquer fase do desenvolvimento dependendo da documentação disponível. Da mesma forma que as outras técnicas, se ela é executada durante as fases finais de desenvolvimento, tende a aumentar os custos devido aos custos potências de uma possível modificação de projeto. A SSA está baseada na identificação dos modos de operação projetados, e não nas falhas de equipamentos ou software.

A SSA depende da experiência e da capacidade dos engenheiros e analistas envolvidos, e os resultados dependem em grande parte da disponibilidade e da qualidade da documentação do sistema. É pouco provável que sejam encontrados muitos problemas sérios usando-se essa análise.

5.4 Conclusão

Foram estudadas várias técnicas e propostas para análise de vulnerabilidade. Alguns desses métodos são mais adequados do que outros, para serem aplicados em software, dependendo do tipo de aplicação do sistema.

Podemos observar que diferentes métodos de análise de vulnerabilidade cobrem diferentes aspectos da análise de segurança (Tabela 5.6). Alguns desses métodos e técnicas necessitam do desenvolvimento de modelos dinâmicos para a sua aplicação, enquanto outros são aplicáveis a partir da documentação existente do sistema. Alguns são aplicáveis em qualquer fase do ciclo de vida de desenvolvimento enquanto outros necessitam de informações precisas para a sua aplicação, como vimos ao longo desse estudo.

Um aspecto que deve ser observado acerca desses métodos estudados é que a maioria foi desenvolvida para estudar funções de processos contínuos e aspectos de hardware, e foram adaptados para software, dificultando assim a sua aplicação em sistemas computadorizados convencionais, i.e. de natureza descontínua. Em relação às RNA o aspecto de continuídade é relevante devido a sua natureza interpoladora.

Nós estamos interessados em técnicas de análise de vulnerabilidade exploratórias que possam ser aplicadas o mais cedo possível no processo de

desenvolvimento (ciclo de vida) especialmente na etapa de especificação de requisitos, onde se obtém o melhor custo/benefício em relação a possíveis modificações na modelagem do sistema em função da identificação de requisitos de segurança. Portanto, além da SFMEA outros métodos que revelam as falhas de software nas fases iniciais do desenvolvimento de sistema são a SFTA e a HAZOP.

	HAZOP	CCA	FTA	ETA	SFMEA	RP	SSA	SMHA	DFM	FPTN	Digrafos
Identificação de acidentes potenciais	S	S		O	S			S			
Identificação das causas de acidentes	S	S	S	O	S	S	S	S	S	S	O
Identificação das consequências das falhas	S	S		S	S			S		S	S
Falhas de hardware	O		S	S	S	S			S		S
Falhas de software			O		S	S		S	S	S	O
Aspectos qualitativos			S	S	O		O				O
Erros humanos	O	S	O	O		O	S				
Falhas sistemáticas	S	O	O	O	O	O		S	S	S	S
Falhas aleatórias	S	O		S	S	O		O	O	O	S

S: Cobre sistematicamente

O: Cobre ocasionalmente ou implicitamente

Tabela 5.6 – Diferentes métodos de análise de vulnerabilidade cobrem diferentes aspectos relacionados à segurança de sistemas computadorizados.

Como vimos a identificação das vulnerabilidade de software deve levar em consideração o estado do sistema e as condições reais do ambiente operacional do sistema, já que o software por si só não é inseguro. Portanto, é necessário identificar comportamentos do software que podem contribuir para colocar um sistema em situações vulneráveis, pois estas informações são o ponto de partida para identificação dos requisitos de segurança de software. Isto porque, como já vimos, os principais pontos críticos de software relacionados à segurança são:

1) requisitos de segurança não identificados;

2) integração de subsistemas (interfaces de subsistemas); e

3) gerência de redundância.

Assim, uma análise de vulnerabilidade de sistemas críticos que empregam software baseado em RNA vai certamente requerer a aplicação de mais de uma técnica, devido ao potencial característico de cada uma dessas técnicas. Escolher, os métodos e técnicas apropriados para um projeto é uma atividade que requer uma visão global do sistema como um todo, e depender somente de um enfoque não é prudente dado ao pouco conhecimento sobre a área em questão, i.e. SCS computadorizados que empregam RNA.

As abordagens indutiva e dedutiva são complementares, e esta complementaridade é essencial nos processos de identificação e análise de vulnerabilidade, permitindo uma busca em ambas direções, i. e. das causas para as consequências e vice-versa. A análise de vulnerabilidade e a análise de segurança são processos iterativos, tarefas de equipe de engenharia que devem considerar todos os aspectos de todas as tecnologias envolvidas no sistema no seu processo de desenvolvimento. Combinações de técnicas e métodos para domínios de aplicação específicos permitem um melhor fluxo de informações entre os especialistas das várias áreas envolvidas no processo de análise de segurança.

Para o caso das Redes Neurais Artificiais (RNA), devemos considerar as características particulares deste tipo de software não convencional. As RNA apesar de serem software possuem uma natureza contínua (i. e. geralmente uma pequena variação na entrada resulta em uma pequena variação correspondente na saída), e seu comportamento é altamente dependente das entradas. Portanto, é necessário fazer uma análise de vulnerabilidade dos subsistemas que podem ter alguma influência sobre as

entradas da RNA de forma a se identificar quais as possíveis falhas que podem gerar entradas erradas, i. e. fora do envelope de operação normal da RNA.

Em princípio todas as técnicas apresentadas são válidas para serem utilizadas na análise de segurança das entradas e de vulnerabilidade das saídas de uma RNA. No entanto, em uma análise mais detalhada de cada técnica acreditamos que para a identificação das possíveis falhas dos subsistemas que antecedem as entradas da RNA, e portanto ocasionando falhas nessas entradas a combinação das técnicas SFMEA e SFTA sugerida em (Lutz, 1996b) surge como a opção de maior potencialidade. A justificativa para esta escolha baseia-se principalmente nos seguintes aspectos:

1. A SFMEA identifica requisitos fracos ou ausentes, além de identificar defeitos latentes de software. Tal característica é de grande interesse para as RNA já que, para um ciclo de vida evolucionário, no início do seu desenvolvimento a RNA possui uma especificação de requisitos que na maioria das vezes precisa ser aprimorada ao longo do desenvolvimento.
2. A SFMEA focaliza atenuar as dependências ou interações implícitas que podem propagar dados errôneos para outros módulos de software, o que é altamente desejável para os módulos de software que antecedem a camada de entrada da RNA.
3. A SFMEA como parte do processo de análise de requisitos contribui para a avaliação dos requisitos de sistema.
4. A simplicidade do método torna fácil para os desenvolvedores revisar os resultados e incorporá-los nas iterações seguintes do projeto, permitindo assim a harmonização de técnicas com o conceito incremental do ciclo de vida evolucionário de segurança proposto (CIVES).

5. A SFMEA analisa a robustez do software na presença de anomalias, bem como sua correção funcional, o que é de particular interesse para as RNA.
6. A integração da análise indutiva (SFMEA), seguida de uma análise dedutiva (SFTA) permite a descoberta de modos de falha desconhecidos, anomalias múltiplas ou coincidentes e dependências implícitas entre processos de software, o que é perfeitamente aplicável ao caso de interesse onde as entradas da rede podem ser oriundas de múltiplos processos, não necessariamente dependentes.

Para a análise de vulnerabilidade a ser aplicada nas saídas de uma RNA de um SCS a técnica HAZOP surge como opção natural, não só pela sua ampla aceitação como técnica exploratória para a identificação de vulnerabilidades do sistema mas também pela sua característica inerente de analisar causas e efeitos relacionados a desvios do comportamento esperado, o que no caso das RNA torna-se bastante atraente pois os desvios de comportamento podem ser mapeados diretamente, e principalmente, somente nas entradas da rede. Deste modo a formulação de palavras guias e a definição, e análise, dos desvios fica restrita ao comportamento de cada neurônio de entrada, simplificando a aplicação da HAZOP e consequentemente aumentando as perspectivas de benefícios que a HAZOP propicia na identificação de estados vulneráveis.

6 Conclusões

O objetivo primeiro deste trabalho foi o de propor um conjunto de diretrizes que permitisse uma abordagem para a certificação de Redes Neurais Artificiais (RNA) para Sistemas Críticos quanto à Segurança (SCS). A certificação de RNA para SCS tende, principalmente nos dias de hoje, a se tornar um ponto crítico para este tipo de aplicação, pois a flexibilidade e a gama de opções oferecidas pelas RNA para a aplicação em SCS, principalmente em áreas sensíveis como a nuclear e a aviação, esbarra na natural necessidade destas áreas de garantir a confiabilidade e a segurança dessas aplicações. As características de confiabilidade e de segurança podem ser garantidas somente através de um processo de certificação, e neste ponto encontra-se um dos maiores obstáculos à utilização de RNA como ferramentas realmente operacionais.

Através das pesquisas realizadas neste trabalho tivemos a oportunidade de constatar a inexistência de qualquer proposta concreta para certificar RNA, principalmente em SCS. Também notamos o reduzido número de trabalhos, revelando um estágio ainda imaturo destas tecnologias, que abordam o problema em questão. É interessante notar no entanto que a maioria dos trabalhos encontrados possui estreita ligação com entidades privadas de alta tecnologia (Peterson, 1993, Nabney, 1997, Lisboa, 2001) ou com aplicações militares, e talvez por isso um aprofundamento do tema não esteja disponível. Observa-se também que além da inexistência de uma metodologia ou processo de certificação, as próprias bases do processo são ainda tratadas de modo informal, como a própria definição de um ciclo de vida de desenvolvimento de RNA. No caso de SCS as definições encontradas sobre este tema tendem a abordar somente aspectos muito específicos, e não apresentam nenhuma sugestão que possa servir de base para uma formalização do processo de certificação.

Na realidade é oportuno enfatizar no momento que não pretendemos afirmar que o trabalho aqui apresentado representa o fim deste processo, ao contrário, nossa intenção foi a de estabelecer um potencial início deste caminho que necessariamente deverá ser percorrido para se obter a formalização de um processo de certificação de RNA para SCS.

Neste sentido apresentamos um ciclo de vida para o desenvolvimento de RNA em SCS baseado em um modelo evolucionário e adequado aos elementos de segurança necessários. Tal ciclo apresentado no capítulo 4, que denominamos de CIVES, integra uma estratégia de desenvolvimento apropriada às características de aprendizado indutivo das RNA com etapas de análise de vulnerabilidade e segurança, aplicadas diretamente às camadas de entrada e saída da rede. Procuramos incorporar dentro do CIVES uma consolidação de vários estudos da área, extraindo destes trabalhos as características que consideramos relevantes para as diversas etapas do processo de desenvolvimento. Dentro deste trabalho procuramos não só consolidar os vários aspectos do ciclo de desenvolvimento das RNA mas também contribuímos com requisitos que consideramos relevantes como o conceito de auto-proteção de fronteiras dos dados de treinamento.

O aprendizado obtido durante as pesquisas realizadas, aliado à experiência adquirida no desenvolvimento de um caso teste (SMIC), permitiu também estabelecer os pontos de ligação entre a modelagem da rede e as necessidades de segurança inerentes aos SCS. Esta ligação, possibilitou integrar as tarefas de análise de vulnerabilidade e de segurança dentro do contexto de desenvolvimento de RNA. De modo a refletir esta integração em uma ferramenta realmente operacional aprofundamos a pesquisa em relação ao estado da arte das técnicas de análise de segurança procurando sempre enfocar as necessidades e características particulares das RNA. O resultado

deste estudo apresentado no capítulo 5, procurou em suas conclusões justificar uma base de soluções para as técnicas mais adequadas tanto para a análise de segurança das entradas, SFMEA combinada com SFTA, como para a análise de vulnerabilidade das saídas da RNA, através da técnica HAZOP.

Finalmente acreditamos que nossa busca de um processo de certificação para RNA em SCS possa ser vista como uma "busca em amplitude" e que para a formalização deste processo é necessário um trabalho futuro que implique em uma "busca em profundidade". Observamos que a escolha de um "busca em amplitude" para a pesquisa deste trabalho não pode ser definida claramente como uma opção, pois devido às próprias bases hoje existentes para o processo de certificação de RNA em SCS tal abordagem foi a que maior potencial oferecia no momento.

A natureza contínua das RNA em contraposição à característica discreta do software convencional, permite a exploração de uma série de índices não só de desempenho como também de avaliação da rede, possibilitando inclusive uma visualização de definições de probabilidades de falhas para as saídas da rede e por consequência a utilização desses índices em critérios de confiabilidade.

Apesar de não estar diretamente ligado com o tema em questão é interessante notar que as RNA, como ressaltado no trabalho de Bolt (1992), possuem uma natureza intrínseca de tolerância a falhas, fato este que poderá ser explorado em futuras pesquisas.

Referências Bibliográficas

ALMEIDA, J. C. S., 2001, Método de Identificação de Transientes com Abordagem Possibilística, Otimizado por Algoritmo Genético, Tese de Mestrado, COPPE/UFRJ, Rio de Janeiro, Brasil.

ALVARENGA, M. A. B., 1997, Diagnóstico do Desligamento de um Reator Nuclear Através de Técnicas Avançadas de Inteligência Artificial, Tese de Doutorado, COPPE/UFRJ, Rio de Janeiro, Brasil.

ALVES, A. C. D., 1993a, Um Sistema de Análise de "*TRIP*" em Reatores PWR Usando Redes Neurais, Tese de Mestrado, COPPE/UFRJ, Rio de Janeiro, Brasil.

ALVES, A. C. D., MARTINEZ, A. S., SCHIRRU, R.,1993b, Um Sistema de Análise de "*TRIP*" em Reatores PWR, Utilizando Redes Neuronais, In IX ENFIR, pp343-348, Caxambú, MG, Brasil, Outubro.

ANÔNIMO, 1998, "Station Software Risks", Aviation Week & Space Technology, Março.

ANTONIOU, A., 1979, Digital Filters: Analysis and Design, McGraw-Hill, Inc..

APLIN, J. D., 1997, "Primary Flight Computers for the Boeing 777", Microprocessors and Microsystems 20 473-478.

ATOM, 1999, Gamma Ray Spectra Catalog, Idaho National Engineering & Environmental Laboratory, www.atom.nw.ru/catalog/ .

AUSTIN, J. , 1991, "Neural Networks for Safe Knowledge Based Systems", Colloquium on Expert Systems and Safety , Feb, pp. 4.1 - 4.5, IEE.

BACHELIER, P., 1990, "L'Environnement Aerien du Futur FMS/ATC/Pilote", L'Aéronautique et l'Astronautique, N.145, 1990-6.

BARTAL Y., LIN, J., UHRIG, R. E. , 1995, "Nuclear Power Plant Transient Diagnostics Using Artificial Neural Networks that Allow "Don't Know" Classifications", Nuclear Technology, Vol. 110, Junho, pp. 346-349.

BEDFORD, D. F., MORGAN, G., AUSTIN, J. 1996, "Requirements for a Standard Certifying the use of Artificial Neural Networks in Safety Critical Applications", Int. Conf. of Artificial Neural Networks.

BELL, R., REINERT, D.,1993, "Risk and System Integrity Concepts for Safety-Related Control Systems", Microprocessors and Microsystems; V.17, N.1.

BHATTACHARYA, S., ONOMA, A., BASTANI, F., 1997,"High-Assurance Systems", Communications of the ACM, V.40 N.1, Janeiro 1997.

BISHOP, C. M., 1995a, Neural Networks: A Principled Perspective, relatório técnico NCRG/95/014, Dept. of Computer Science and Applied Mathematics, Aston University, Birmingham, Reino Unido.

BISHOP, C. M., 1995b, Neural Networks for Pattern Recognition, 1ª edição, Oxford University Press Inc., New York.

BLYTH, A., 1997, "Issues Arising from Medical System's Failure", Software Engineering Notes, ACM SIGSOFT, V.22, N.2, Março.

BOLT, G. R., 1992, Fault Tolerance in Artificial Neural Networks, D.Phil. Thesis, University of York, Advanced Computer Achitecture Group, Department of Computer Science, Novembro.

BRACEWELL, R., 1965, The Fourier Transform and its Applications, McGraw-Hill Book Co., N.Y.

BROOMFIELD, E. J., CHUNG, P. W., 1997, "Safety Assessment and the Software Requirements Specification", Reliability Engineering and System Safety, 55, 295-309.

CAMARGO, J. B., JUNIOR, S., MELNICOF, S. S., ALMEIDA, J. R., 1997, "O Uso de Fatores de Qualidade na Avaliação da Segurança de Software em Sistemas Críticos", Conferência Internacional de Tecnologia de Software, Curitiba, PR, Brasil, Abril.

CANEDO, J.C., SCHIRRU, R., COMERLATO, C. A. et al., 2002, Sistema de Identificação de Radionuclídeos a Partir de Emissões Gama Utilizando Redes Neurais Artificiais, XIII Encontro Nacional de Física de Reatores, aceito para publicação.

CARVALHO, L. A. V. de, 1989, Síntese de Redes Neuronais com Aplicações à Representação do Conhecimento e Otimização, Tese de Doutorado, COPPE/UFRJ, Rio de Janeiro, Brasil.

CARVALHO, L. A. V. de, BARBOSA, V., et al., 1991, "Redes Neuronais Artificiais, a Volta do Cérebro Eletrônico", Ciência Hoje, vol 12, No. 20, pp12-21.

CARVALHO, L. A. V. de, 1995, "Entrevista", Revista Decidir, No. 6, pp14-1.

COMERLATO, C.A., VIDAL DE CARVALHO, L.A., CAMINO, F.M., 1998, Hazard Analysis from a Software Safety Perspective, relatório técnico, LAAS Report 98572, Dezembro, 1998, Laboratoire d'Analyse et d'Architecture des Systèmes, Toulouse, França.

COMERLATO, C.A., SCHIRRU, R., 2000, Estudo de Viabilidade para Substituição do Computador PDP-11/44 do Sistema de Monitoração Isotópica da Chaminé de Angra 1, relatório técnico, projeto COPPETEC PEN-1413, Abril, 2000.

COMERLATO, C.A., VIDAL DE CARVALHO, L.A., SCHIRRU, R., et al., 2002, Proposta de um Processo de Desenvolvimento para Módulos de Redes Neurais Artificiais para Sistemas Críticos quanto à Segurança, XIII Encontro Nacional de Física de Reatores, aceito para publicação.

DORF, R. C., BISHOP, R. H, 1995, Modern Control Systems, 7ª edição, Addison Wesley, Inc..

DOWSON, M., 1997, "The ARIANE 5 Software Failure", Software Engineering Notes, ACM SIGSOFT, V.22, N.2, Março.

ERAU, J. F., DEMMOU, H., SALEMAN, M., 1995, Dynamic Fault Trees Based on Synchronized Petri Nets, relatório técnico, LAAS Report 95373, Setembro, Laboratoire d'Analyse et d'Architecture des Systèmes, Toulouse, França.

FENELON, P., MCDERMID, J. A., 1993ª, "An Integrated Toolset for Software Safety Analysis", Journal of Systems and Software, Julho.

FENELON, P., MCDERMID, J. A., NICHOLSON, M., PUMFREY, D. J., 1994, "Towards Integrated Safety Analysis and Design", ACM Computing Reviews,2(1) p. 21-32.

FENELON, P. , KELLY, T. P., MCDERMID, J. A., 1995, "Safety Case for Software Application Reuse", COMPASS'95, Springer.

FU, L., 1994, Neural Networks in Computer Intelligence, McGraw-Hill International Editions, Computer Science Series.

GARDINER, S., 1999 (Ed.), Testing Safety-Related Software: A Practical Handbook, Springer-Verlag, Reino Unido.

GARRET, C., GUARRO, S. B., APOSTOLAKIS, G., 1995, "The Dynamic Flowgraph Methodology for Assessing the Dependability of Embedded Software Systems", IEEE Transactions on Systems, Man, and Cybernetics, V.25, N.5, Maio.

GARRET, C., APOSTOLAKIS, G., 1998; "Context and Software Safety Assessment", HESSD.

GERHART, S., CRAIGEN, D., RALSTON, T., 1994, "Experience with Formal Methods in Critical Systems", IEEE Software, Janeiro.

GRADY, J. O., 1998, System Validation and Verification, CRC Press LLC.

HAGELAUER, P., CAMINO, F. M., 1998, "A Soft Dynamic Programming Approach for On-Line Aircraft 4D-Trajectory Optimization", European Journal of Operations Research, 107 (1) pág. 87.

HANSEN, K. M., RAVN, A. P., STAVRIDOU, V., 1998, "From Safety Analysis to Software Requirements", IEEE Transactions on Software Engineering, Julho.

HARAUZ, J., 1997, "International Trends in Software Engineering and Quality System Standards from the Perspective of Ontario Hydro", Conferência Internacional de Tecnologia de Software Curitiba, PR, Brasil, Abril.

HAYKIN, S., 1994, Neural Networks - A Comprehensive Foundation, Prentice-Hall, Inc.

HAYKIN, S., 1996, Adaptative Filter Theory, Third Edition, Prentice-Hall, Inc.

HEBBRON, B. D., FENELON, P., 1997, "The Application of Hazard Analysis and Operability Studies to Real Time Structured Requirements Models", Reliability Engineering and System Safety, 55, 311-325.

HECHT-NIELSEN, R., 1990, Neurocomputing, Addison-Wesley, Pub. Co..

HESSELINK, H. H.,1995; "A Comparison of Standards for Software Engineering based on DO-178B for Certification of Avionics Systems"; Microprocessors and Microsystems, V.19, N.10, Dezembro.

IAEA, 1999, Verification and Validation of Software Related to Nuclear Power Plant Instrumentation and Control, Technical Reports Series No. 384, International Atomic Energy Agency, Vienna, 1999.

IEC, 1990, International Standard IEC 1025; Fault Tree Analysis (FTA), Geneva: International Electrotechnical Commission (IEC).

IEC, 1995, Draft International Standard IEC 1508; Functional Safety: Safety-Related Systems; Geneva: International Electrotechnical Commission (IEC).

IEEE-1012, 1998, IEEE Standard for Software Verification and Validation, IEEE Software Engineering Standards, Vol. II, 1999 Edition.

IEEE/EIA-12207, 1997, IEEE Std 12207.2, Software Life Cycle Processes, IEEE Software Engineering Standards, Vol. I, 1999 Edition.

IPPOLITO, L. M., WALLACE, D. R., 1995, A Study on Hazard Analysis in High Integrity Software Standards and Guidelines, U.S. Department of Commerce – Technology Administration – National Institute of Standard and Technology.

ISAKSEN, U., BOWEN, J. P., NISSANKE, N., 1996, System and Software Safety Analysis; Technical Report RUCS/97/TR/062/A, Department of Computer Science, The University of Reading, UK (http://www. comlab.ox.ac.uk /archive/safety.html), Dezembro.

IVERSON, D. L., PATTERSON-HINE, F. A., 1995, "Advances in Digraph Model Processing Applied to Automated Monitoring and Diagnosis", Reliability Engineering and System Safety, 49, 325-334.

JÉZÉQUEL, J. M., MEYER, B., 1997, "Design by Contract: The Lessons of Ariane", IEEE Computer, 129-130, Janeiro.

JOCH, A., 1995, "How Software Doesn't Work", Byte magazine, Dezembro.

JUSLIN, K., 1995, "Status on the Finnish Activities regarding) Qualification of Programmable Autommation Systems", IAEA Advisory Group Meeting In: Advanced Control Systems to Improve Nuclear Power Plant Reliability and Efficiency, Vienna, Março.

KAHNE, S., FROLOW, I., 1996, "Air Traffic Management: Evolution With Technology", IEEE Control Systems, Agosto.

KANOUN, K., 1996, Dependability of Fault-Tolerant Systems – Explicit Modeling of the Interactions Between Hardware and Software Components, relatório técnico, LAAS Report 96046, Fevereiro, Laboratoire d'Analyse et d'Architecture des Systèmes, Toulouse, França.

KELLER, P.E., KOUZES, R.T., 1994, Gamma Spectral Analysis via Neural Networks, IEEE Nuclear Science Symposium (NSS'94), Nuclear Science Symposium & Medical Imaging Conference: 1994 IEEE Conference Record, (IEEE Press, Piscataway, NJ, USA, 1994), Vol. 1, pp. 341-345.

KELLY, J. P. J., MCVITTIE, T. I., YAMAMOTO, W. I., 1991, "Implementing Design Diversity to Achieve Fault Tolerance", IEEE Software, Julho.

KLETZ, T. A., 1997, "HAZOP – Past and Future", Reliability Engineering and System Safety, 55, 263-266.

KNIGHT, J., LITTLEWOOD, B., 1994, "Critical Task of Writing Dependable Software", IEEE Software, Janeiro.

KRISHNA, C. M., LEE, Y. H., 1994, Scanning the Issue: Real-Time Systems; Proceedings of the IEEE, Vol. 82, No. 1, Janeiro.

KUO, D. H., HSU, D. S., CHANG, C. T., 1997, "A Prototype for Integrating Fault Tree/Event Tree/ HAZOP Analysis", Computers Chemical Engineering, V. 21, Suppl.,pp. S923-S928.

LALA, J. H., HARPER, R. E., 1994, "Architectural Principles for Safety-Critical Real-Time Applications", Proceedings of the IEEE, Vol. 82, No. 1, Janeiro.

LAPP, S. A., POWERS, G. J., 1977, "Computer-aided Synthesis of Fault-trees", IEEE Transactions on Reliability, Abril.

LAPRIE, J. C., 1991, "Dependability: Basic Concepts and Terminology in English, French, German, Italian and Japanese", Vol. 5, Dependable Computing and Fault-Tolerant Systems; Springer-Verlag, Fevereiro.

LAPRIE J. C., 1996, Software-Based Critical Systems, relatório técnico, LAAS Report 96327, Agosto, Laboratoire d'Analyse et d'Architecture des Systèmes, Toulouse, França.

LAPRIE, J. C., 1997, Sûreté de Fonctionnement et Qualité du Logiciel; relatório técnico, LAAS Report 97444, Dezembro, Laboratoire d'Analyse et d'Architecture des Systèmes, Toulouse, França.

LEVESON, N. G., STOLZY, J. L., 1987, "Safety Analysis Using Petri Nets", IEEE Transactions on Software Engineering, Março.

LEVESON, N. G., 1995, Safeware: System Safety and Computers, Addison-Wesley Pub. Co..

LEVESON, N.G., 1997, Safety Analysis of Air Traffic Control Upgrades – Final Report; (http://www.cs.washington.edu/ homes/leveson), Setembro.

LEWIS, R. O., 1992, Independent Verification and Validation, Wiley - Interscience Publication Co..

LION, J. L., 1996, Ariane 5-Rapport de la Commission d'enquête Ariane 501-Echec du vol 501, (Communiqué de presse conjoint ESA-CNES, Paris, le 19 juillet 1996) http://www.esrin.esa.it/htdocs/tidc/Press/Press96/ ariane5rep.html, Julho.

LISBOA, P. J. G., 2001, Industrial Use of Safety-Related Artificial Neural Networks, Contract Research Report 327/2001, Liverpool John Moores University for the Health and Safety Executive, Reino Unido.

LITTLEWOOD, B., STRIGINI, L., 1992, "The Risks of Software", Scientific American, Novembro.

LUTZ, R. R., WOODHOUSE, R. M., 1996a, Experience Report: Contributions of SFMEA to Requirements Analysis; Second IEEE International Conference on Requirements Engineering, Abril.

LUTZ, R. R., AMPO, Y., WOODHOUSE, B., BRUHN, A., 1996b, Software Product Assurance Handbook: Software Failure Mode & Effects Analyses; NASA Jet Propulsiosn Laboratory, Junho.

MACHADO, L., 1996, Modelagem do Conhecimento para Sistemas Inteligentes de Monitoração de Segurança em Tempo-Real para Usinas Nucleares, Tese de Mestrado, COPPE/UFRJ.

MARTINEZ, A. , OLIVEIRA, L. F., SCHIRRU, R., et al., 1986, "A New Concept of Safety Parameter Display System", Annals of the Seminar on Nuclear

Engineering in Latin America, ANS - American Nuclear Society, México.

MARTINEZ, A. , SCHIRRU, R., THOMÉ, Z., 1988, "Improving SPDS for Normal Operation", Proceedings of an International Conference on Man-Machine Inference in the Nuclear Industry (Control and Instrumentation, Robotics, and Artificial Intelligence), IAEA, Japão.

MCCONNELL, S., 1996, Rapid Development, Microsoft Press, Washington, EUA.

MCDERMID, J. A., PUMFREY, D. J., 1994, "A Development of Hazard Analysis to Aid Software Design", COMPASS'94, IEEE Computer Society Press.

MIL-STD-1629A, 1980, Military Standard: Procedures for Performing a Failure Mode, Effects and Criticality Analysis, U.S. Department of Defense, Washington, D.C., Novembro.

MIL-STD-882D, 1997, Military Standard: System Safety Program Requirements (Draft Jan 97); U.S. Department of Defense, Washington, D.C., Janeiro (http://www.afmc. wpafb.af.mil/organizations/HQ-AFMC/SE/ssd.htm)

MoD, 1997, Draft Defence Standard 00-55 Issue 2; Requirements for Safety Related Software in Defence Equipment, Part 1:Requirements; Glasgow, Scotland: U.K. Ministry of Defence, Directorate of Standardization. (http://www.dstan.mod.uk/).

MoD, 1996a, Draft Defence Standard 00-56 Issue 2; Safety Management Requirements for Defence Systems; Glasgow, Scotland: U.K. Ministry of Defence, Directorate of Standardization, 1996 (http://www. dstan.mod.uk/).

MoD, 1996b, Interim Defence Standard 00-58 (Parts 1 and 2)/Issue 1; HAZOP Studies on Systems Containing Programmable Electronics; Glasgow, Scotland:

U.K. Ministry of Defence, Directorate of Standardization, 1996 (http://www. dstan.mod.uk/).

MOJDEHBAKHSH, R., TSAI, W., KIRANI, S., ELLIOTT, L., 1994, "Retrofitting Software Safety in an Implantable Medical Device", IEEE Software, Janeiro.

MORGAN, G., AUSTIN, J., BOLT, G., 1995, "Safety Critical Neural Networks", IEE Conference on Neural Networks.

MOURA, C. A. T., SANTELLANO, J., LSILVA, . M. N., CUNHA, A. M., 1995, "Análise de Segurança de Software de Uso em Aeronáutica", III Congresso e Exposição Internacionais de Tecnologia da Mobilidade, São Paulo: SAE Brasil, Novembro.

MOURA, C. A. T., OMAR, N., SANTELLANO, J., 1996, "Uma Estratégia de Análise de Segurança de Software para Aplicações Críticas", X Simpósio Brasileiro de Engenharia de Software (Workshop: Pesquisa de Teses em Engenharia de Software); UFSCar/USP São Carlos, SP, Brasil, Outubro.

MOURA, C. A. T., PAULA, E. G., BRAGA, R. O.,1997, "Aplicações Críticas quanto à Segurança: O Caso do Software Embarcado em Helicópteros" Conferência Internacional de Tecnologia de Software Curitiba, PR, Brasil, Abril.

NABNEY, I. T., PAVEN, M. J. S. , ELDRIDGE, R. C., LEE, C., 1997, Practical Assessment of Neural Network Applications, SafeComp 97, Proceedings of the 16th International Conferenceon Computer Safety, Reliability and Security, ed. P. Daniel, pp. 357-368, Springer Verlag.

NASA-PD-AP-1314, 1995, Sneak Circuit Analysis Guideline for Electro-Mechanical Systems; Nasa Marshall Space Flight Center, Outubro, 1995.

NASA-GB-1740.13-96, 1996, NASA Guidebook for Safety Critical Software – Analysis and Development, 1996.

NASA-GB-001-97, 1997, Formal Methods Specification and Analysis Guidebook for the Verification of Software and Computer Systems - Volume II: A Practitioner's Companion, Maio,1997.

NASCIMENTO Jr., C. L., YONEYAMA, T, 2000, Inteligência Artificial em Controle e Automação, Primeira Edição, Ed. Edgard Blücher Ltda/FAPESP, Brasil.

NETO, J.S., SILVA, M.C., PINHEIRO, R.F., SOARES, M., MARTINEZ,A.S., COMERLATO, C.A., OLIVEIRA, E.A., 1996, "Sistema de Monitoração de Inventário do Reator para Usina Nuclear Angra I", Anais do VI Congresso Geral de Energia Nuclear, Rio de Janeiro, RJ, Brasil, Outubro 1996.

NEUMANN, P. G., 1995, Computer-Related Risks; ACM Press, 1995.

NIELSEN, D., 1975, "Use of Cause-Consequence Charts in Practical Systems Analysis Reliability and Fault Tree Analysis", In: Theoretical and Applied Aspects of System Reliability and Safety Assessment, SIAM, Philadelphia, EUA, (849-880), 1975.

NUREG/CR-6316, 1995a, MILLER, L. A., HAYES, J. E., MIRSKY, S. M, Guidelines for the Verification and Validation of Expert System Software and Conventional Software - Project Summary; U.S. Nuclear Regulatory Commission (NUREG/CR-6316 SAIC-95/1028 Vol. 1), Março, 1995.

NUREG/CR-6316, 1995b, MILLER, L. A., GROUNDWATER, E. H., HAYES, J. E. MIRSKY, S. M., Guidelines for the Verification and Validation of Expert System Software and Conventional Software - Survey and Assessment of Conventional Software Verification and Validation

Methods; U.S. Nuclear Regulatory Commission (NUREG/CR-6316 SAIC-95/1028 Vol. 2), Março 1995.

NUREG/CR-6316, 1995e, MILLER, L. A., GROUNDWATER, E. H., HAYES, J. E. MIRSKY, S. M., Guidelines for the Verification and Validation of Expert System Software and Conventional Software - Evaluation of Knowledge Base Certification Methods; U.S. Nuclear Regulatory Commission (NUREG/CR-6316 SAIC-95/1028 Vol. 5), Março 1995.

PARNAS, D. L., SCHUWEN, A. J. van, KWAN, S. P., 1990, "Evaluation of Safety-Critical Software", Communications of the ACM, V.33 N.6, Junho 1990.

PARTRIDGE, D. , YATES, W. B., 1995, "Engineering Multiversion Neural-Net Systems", Neural Computation, 8, N. 4, pp. 869-893.

PARTRIDGE, D., 1996, "The Case for an Inductive Computing Science", IEEE Computer, Janeiro, pp. 36-41.

PATTERSON-HINE, F. A., KOEN, B. V.,1989, "Direct Evaluation of Fault Trees Using Object-Oriented Programming Techniques", IEEE Transactions on Reliability, Vol. 38, N. 2, Junho 1989.

PEREIRA, C. M. N. A., SCHIRRU, R. MARTINEZ, A. S., 1998, Learning an Optimized Classification System from a Data Base of Time Series Patterns Using Genetic Algorithms, In: Ebecken, N. S. S., *Data Mining*, 1ª. ed, South Hampton, Reino Unido, WIT Computational Mechanics Publication.

PETERSON, G. E., 1993, "A Foundation for Neural Network Verification and Validation", Proceedings SPIE - The International Society for Optical Engineering, Science of Artificial Neural Networks II, Vol. 1966, pp. 196-207, Orlando, Florida, EUA, Abril.

PETERSON, I., 1996, Fatal Defect: Chasing Killer Computer Bugs; Vintage Books, A Divison of Random House Inc., New York.

PFLEEGER, S. L., FWNTON, N., PAGE, S., 1994, "Evaluating Software Engineering Standards"; IEEE Computer, V. 27,N.9, Setembro.

PLACE, P. R. H., WOOD, W. G., 1990, Survey of Formal Specification Techniques for Reactive Systems, Technical Report CMU/SEI-90-TR-5 ESD-TR-90-206, Software Engineering Institute – Carnegie Mellon University, Maio 1990.

PLACE, P. R. H., KANG, K. C., 1993,Safety-Critical Software: Status Report and Annotated Bibliography; Technical Report CMU/SEI-92-TR-5 ESC-TR-93-182,Software Engineering Institute – Carnegie Mellon University, Junho 1993.

POTOCKI, J. P., 1993, "Computer Software in Civil Aircraft", Microprocessors and Microsystems,V. 17 N.1 (17-23) 1993.

PRESSMAN, R. S., 1997, Software Engineering – A Practitioner's Approach, 4ª edição, McGraw Hill Books, 1997.

RABÉJAC, C., 1993, Sûreté de Fonctionnement de Logiciels Critiques: Méthodes Formelles, relatório técnico, Rapport LAAS n.93386 MMS/DTI/LIS/93-4, Laboratoire d'Analyse et d'Architecture des Systèmes, Toulouse, França.

RAKITIN, S. R., 1997, Software Verification and Validation - A Practitioner's Guide, Artech House, Inc..

REDMILL, F., CHUDLEIGH, M., CATMUR, J., 1999, System Safety: HAZOP and Software HAZOP, John Wiley & Sons, 1999.

REESE, J. D., LEVESON, N. G., 1997, Software Deviation Analysis: A "Safeware" Technique, International Conference on Software Engineering, Maio, 1997.

ROBERTS, S., TARASSENKO, L., PARDEY, J., SIEGWART, D., 1994,"A Validation Index for Artificial Neural Networks", Proceedings of International Conference on Neural Networks & Expert Systems in Medicine & Healthcare, Plymouth, Agosto, pp. 23-30.

ROJAS, R., 1996, Neural Networks - A Systematic Introduction, Springer-Verlag, Berlin, 1996.

RTCA, 1992, Software Considerations in Airborne Systems and Equipment Certification; RTCA/DO-178B; EUROCAE/ED-12B. Washington: Radio Technical Commission for Aeronautics. Paris: European Organisation for Civil Aviation Electronics, 1992.

RUSHBY, J., 1993, Formal Methods and the Certification of Critical Systems, Technical report CSL-93-7, Computer Science Laboratory - SRI International, Menlo Park, CA - USA, Dezembro 1993 (Também publicado sob o título Formal Methods and Digital Systems Validation for Airborne Systems NASA CR 4551).

RUSHBY, J., 1995, Formal Methods and their Role in the Certification of Critical Systems; Technical report CSL-95-1, Computer Science Laboratory - SRI International, Menlo Park, CA, A, Março 1995.

RUSSELL, S., NORVIG, P., 1995, Artificial Intelligence - A Modern Approach, Prentice-Hall, Inc.

SCHMITT, D., 1990, "Advanced Technology Constant Challenge and Evolutionary Process", L'Aéronautique et l'Astronautique, N.145, 1990-6.

SHIN, K. G., RAMANATHAN, P., 1994, Real-Time Computing: A New Discipline of Computer Science and Engineering, Proceedings of the IEEE, Vol. 82, No.1, Janeiro 1994.

SILVA, L. M. N. , SANTELLANO, J. ,MOURA, C. A. T., 1995, Homologação de Software para Aplicações na Indústria Aeroespacial; I Encontro de Iniciação Científica e Pós-Graduação (ENCITA'95); São José dos Campos, SP: ITA, Outubro 1995.

STALHANE, T., WEDDE, K. J., 1998, "Modification of Safety Critical Systems: An Assessment of three Approaches", Microprocessors and Microsystems, (21), 611-619, 1998.

STOREY, N., 1996, Safety-Critical Computer Systems, Addison-Wesley, 1996.

STUPARU, A., 1996, "Analyse Comparative de Différents Systèmes de Sûreté Programmés Français", Recherche Transports Sécurité N. 53, Outubro-Dezembro, 1996.

SUOKAS, J., 1988, "The Role of Safety Analysis in Accident Prevention", Acid. Anal. & Prev., V. 20, N. 1, pp. 67-85, Pergamon Journals Ltd.,1988.

SWEET, W., 1995, "The Glass Copckit", IEEE Spectrum; Setembro 1995.

TAYLOR, J. R.,1975, "Sequential Effects in Failure Mode Analysis", In: Reliability and Fault Tree Analysis – Theoretical and Applied Aspects os System Reliability and Safety Assessment, SIAM, Philadelphia, EUA, (881-894) 1975.

TOOLA, A., 1993, "The Safety of Process Automation", Automatica, V. 29, N. 2, pp. 541-548, 1993.

TSOULFANIDIS, N., 1983, Measurement and Detection of Radiation, McGraw-Hill Book Company, N.Y..

VALETTE, R., 1994; Application and Theory of Petri Nets, Springer Verlag, 1994.

VERÍSSIMO, P., DE LEMOS, R., 1989, Confiança no Funcionamento: Proposta para uma Terminologia em Português, Publicação conjunta INESC e LCMI/UFSC, 1989.

VERÍSSIMO, P., 1996, Segurança e Confiabilidade: na Ordem do Dia dos Sistemas Distribuídos, Jornada do Colégio de Engenharia Electrotécnica, Ordem dos Engenheiros, Lisboa - IST, 1996.

VESELEY, W. E., GOLDBERG, F. F., ROBERTS, N. H., HAASL, D. F., 1981, Fault Tree Handbook, Washington, DC: US Nuclear Regulatory Commission, NEREG-0492, 1981.

WEN, W., CALLAHAN, J. M., 1996, "NeuroWare Engineering: Develope Verifiable ANN Systems", Proceedings of the IEEE Joint Symposia on Intelligence and Systems, Rockville, MD, EUA,1996.

WING, J. M., 1990, "A Specifier's Introduction to Formal Methods", IEEE Computer, Setembro 1990.

YEN, I. L., PAUL, R., MORI, K., 1998, "Toward Integrated Methods for High-Assurance Systems" IEEE Computer, Abril 1998.

Apêndice A

A.0 Redes Perceptron de Múltiplas Camadas (*MLP*)

Uma rede perceptron de múltiplas camadas (MLP) consiste tipicamente de um conjunto de nodos de entrada, uma ou mais camadas intermediarias (*hidden layers*) e uma camada de nodos de saída. Os sinais de entrada propagam-se através da rede na direção das entradas para as saídas (*feed forward*) camada a camada, como mostra a Figura A.0. As redes MLP são uma generalização da rede perceptron de uma camada (uma camada de entrada e uma camada de saída).

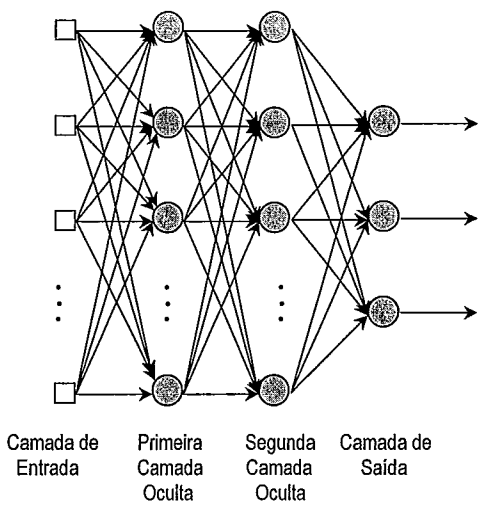

Figura A.0 - Grafo da arquitetura de uma rede MLP com duas camadas intermediárias.

As MLP tem sido aplicadas com sucesso para resolver uma diversidade problemas de difícil solução através de treinamento supervisionado com o algoritmo de retropropagação (*back-propagation*) de erro, baseado no princípio de correção do erro da regra de aprendizado (Fu, 1994).

O processo de retropropagação de erro consiste basicamente em dois passos através das diferentes camadas da rede: um *passo para a frente* e um *passo para trás*. No passo *para a frente* um vetor de entrada é aplicado às entradas da rede e o seu efeito propaga-se através da rede, camada a camada, até que um conjunto de saídas é gerado como saída da rede. Durante o passo para frente os pesos da rede são todos fixos. Durante o *passo para trás* os pesos são todos ajustados de acordo com a regra de correção do erro, ou seja, a resposta atual da rede é subtraída da resposta desejada para produzir um *sinal de erro*. Este sinal de erro é então propagado para trás ao longo da rede na direção contrária das conexões da rede (daí o nome retropropagação). Os pesos são ajustados de forma a aproximar a resposta da rede à resposta desejada. O processo de aprendizado (treinamento) executado desta forma é chamado de aprendizado por retropropagação.

A rede MLP tem três características distintas:

1. O modelo de cada nodo (neurônio) da rede contém uma *não linearidade* na saída final. Esta não linearidade é *suave* (i. e. diferenciável em qualquer ponto). Uma forma comum de não linearidade que satisfaz este requisito é a não *linearidade sigmoidal* definida por meio da função logística: $y_j = 1 / (1+ exp(-v_j))$, onde v_j é a atividade líquida interna do neurônio j, e y_j é a saída do neurônio. A presença de não linearidades é importante, porque caso contrário, a relação entrada-saída da rede poderia ser reduzida àquela do perceptron de uma camada, i. e. linear.
2. A rede contém uma ou mais camadas intermediarias (ocultas) que não fazem parte da entrada ou da saída da rede. Estes neurônios ocultos permitem à rede aprender tarefas complexas através da extração progressiva de características significativas a partir dos padrões de entrada (vetores).

3. A rede apresenta um alto grau de conectividade, determinada pelas sinapses da rede. Uma alteração na conectividade da rede exige uma mudança na população das conexões sinápticas ou dos seus pesos.

O poder da rede MLP reside na combinação dessas características, em conjunto com a sua habilidade para aprender (ajuste dos pesos) a partir da experiência através do treinamento. Mas estas mesmas características são responsáveis pelas deficiências do estado atual do conhecimento sobre o comportamento da MLP:

1. A presença de unidades não lineares distribuídas e a alta conectividade da rede tornam a análise teórica da MLP extremamente difícil de executar.
2. O uso de neurônios ocultos dificulta a visualização do aprendizado. Implicitamente, o processo de aprendizado deve decidir quais características do padrão de entrada devem ser representadas pelos neurônios ocultos. Portanto o processo de aprendizado torna-se muito mais difícil devido ao fato da busca ter que ser conduzida em um espaço muito maior de possíveis funções, e uma escolha tem que ser feita entre as representações alternativas do padrão de entrada (a definição do vetor de entrada).

A.1 O Método de Retropropagação de Erro

Para apresentar o método *backpropagation* (retropropagação) vamos usar o método do gradiente descendente da função erro, e o erro calculado segundo o erro médio quadrático. Outros métodos e cálculos de erro podem ser usados (Fu, 1994).

O algoritmo de retropropagação procura pelo mínimo da função de erro no espaço de pesos usando o método, por exemplo, do gradiente descendente. A configuração dos pesos que minimiza a função de erro é considerada como a solução do problema de aprendizado. Como este método necessita do cálculo do gradiente da função erro a cada iteração, é necessário garantir a continuidade e a diferenciabilidade da função erro. Portanto é necessário o uso de uma função contínua, outra além da função degrau, como a função logística (sigmóide), uma das funções de ativação mais populares para as redes do tipo *backpropagation* como vimos acima.

A função logística trata-se de uma função real $s_c : \Re \rightarrow (0,1)$ definida pela expressão

$$s_c(x) = \frac{1}{1+e^{-cx}} \qquad \text{(A.1.1)}$$

onde a constante c pode ser selecionada arbitrariamente. A forma da curva sigmóide muda de acordo com o valor de c, e no limite quando $c \rightarrow \infty$ a sigmóide converge para uma função degrau na origem.

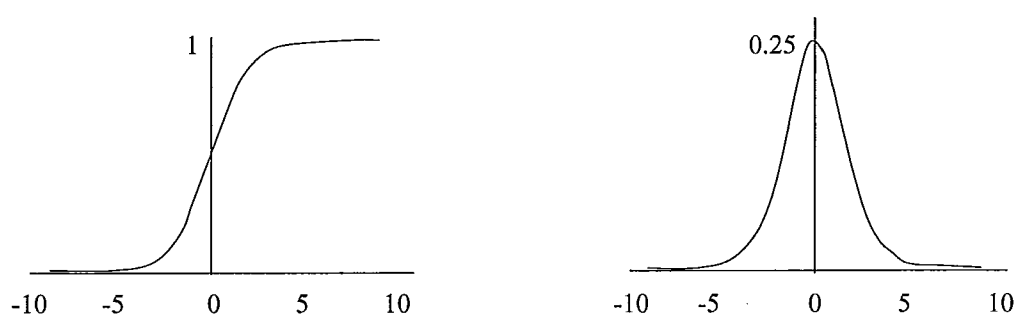

Figura A.1 - A função sigmóide (a direita) e a sua primeira derivada (a esquerda).

A derivada da função sigmóide em relação a x é

$$\frac{d}{dx}s(x) = \frac{e^{-x}}{(1+e^{-x})^2} = s(x)[1 - s(x)] \qquad \text{(A.1.2)}$$

Para o caso dos perceptron uma função de ativação simétrica apresenta algumas vantagens para o treinamento. Uma alternativa para a sigmóide é a sigmóide simétrica *S(x)* definida como

$$S(x) = 2s(x) - 1 = \frac{1 - e^{-x}}{1 + e^{-x}} \qquad \text{(A.1.3)}$$

Trata-se da função tangente hiperbólica para o argumento x/2 cuja forma é mostrada na Figura A.2.

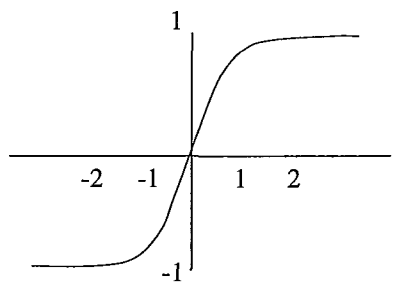

Figura A.2 - A função sigmóide simétrica.

Várias outra funções foram propostas (Rojas, 1996) e o algoritmo de retropropagação aplica-se a todas elas. Uma função de ativação diferenciável torna a função computada pela RNA diferenciável (assumindo-se que a função de integração em cada nodo é simplesmente uma soma das entradas) já que a própria rede computa somente composições de funções. Assim, a função de erro também é diferenciável.

Uma vez que se quer seguir a direção do gradiente da função erro para encontrar o mínimo global, é importante que não existam regiões nas quais a função de erro é

totalmente plana. Como a sigmóide tem sempre uma derivada positiva, a inclinação da função de erro fornece uma maior ou menor direção de descida que pode ser seguida. Segundo Rojas (1996) podemos pensar no algoritmo de busca como um processo físico no qual permite-se que uma pequena esfera role sobre a superfície da função de erro até alcançar o fundo desta.

Existe um preço a se pagar com o uso da função sigmóide como função de ativação. O maior problema é que, sob determinadas circunstâncias, aparecem mínimos locais na função de erro que não existiriam caso a função degrau tivesse sido usada (Rojas, 1996). Daí resulta a importância de se ter vários conjuntos de treinamento com diferentes valores iniciais para os pesos, pois o mínimo local mais próximo do ponto de início poderia ser encontrado.

A.2 O Problema do Aprendizado

Cada unidade da rede (nodo) é capaz de avaliar somente uma função primitiva da sua entrada. De fato a rede representa uma cadeia de composição de funções que transformam um vetor de entrada em um vetor de saída (chamado um padrão). A rede é uma implementação particular de uma função que mapeia o espaço de entrada no espaço de saída, que nós chamamos de função rede. O problema do aprendizado consiste em encontrar a configuração ótima de pesos de forma que a função rede φ aproxime uma dada função f o melhor possível (a função objetivo). Entretanto, nós não conhecemos a função f explicitamente, mas somente implicitamente através de alguns exemplos (pares ordenados de vetores de entrada e saída).

Consideremos uma rede *feedforward* com n entradas e m saídas, com um número qualquer de camadas intermediárias. Consideremos também um dado conjunto de treinamento $\{(\mathbf{x}_1, \mathbf{d}_1),...,(\mathbf{x}_p, \mathbf{d}_p)\}$ composto por p pares de vetores de dimensão n e m respectivamente, que são chamados de padrões de entrada e saída. Consideremos as funções primitivas em cada nodo da rede contínuas e diferenciáveis. Os pesos da rede são números reais escolhidos aleatoriamente. Quando um padrão de entrada $\mathbf{x}_i$ do conjunto de treinamento é apresentado para a rede, ela gera uma saída $\mathbf{o}_i$, geralmente diferente da saída desejada $\mathbf{d}_i$. O que se quer é tornar $\mathbf{o}_i$ e $\mathbf{d}_i$ idênticos para $i = 1,...,p$ através do algoritmo de aprendizado. Mais precisamente, nós queremos minimizar a função erro da rede definida por

$$E = \frac{1}{2}\sum_{i=1}^{p}\left\|o_i - d_i\right\|^2 \qquad \text{(A.2.1)}$$

Após minimizar esta função para o conjunto de treinamento, novos padrões desconhecidos são apresentados à rede e espera-se que ela seja capaz de interpolar. A rede deve reconhecer se um novo vetor de entrada é similar aos padrões aprendidos para gerar uma saída similar.

O algoritmo de retropropagação é usado para encontrar um mínimo local da função de erro. A rede é iniciada com pesos escolhidos aleatoriamente (existem heurísticas para isso, Fu, 1994). O gradiente da função erro é calculado e usado para corrigir os valores dos pesos iniciais. A tarefa consiste então em computar este gradiente repetidamente.

O primeiro passo para otimizar este processo consiste em estender a rede de forma que ela calcule a função de erro automaticamente. A Figura A.3 mostra como isto é feito.

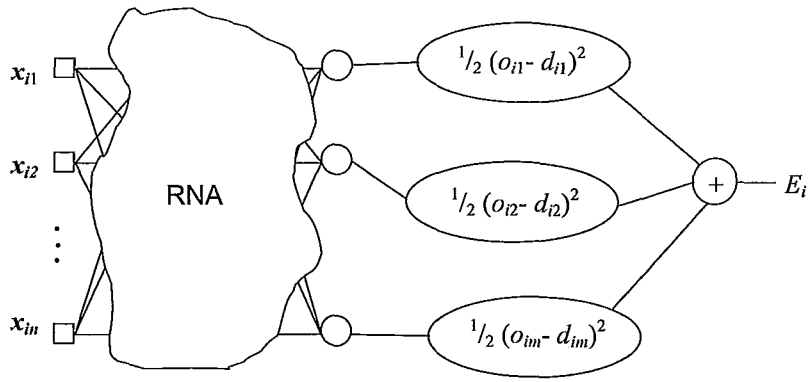

Figura A.3 - Extensão da rede para cálculo da função de erro (Rojas, 1996).

Cada uma das saídas das unidades j está conectada a um nodo que avalia a função $\mathbf{1/2}$ $(o_{ij} - d_{ij})^2$, onde o_{ij} e d_{ij} correspondem ao componente de ordem j do vetor de saída o_i e do vetor alvo (desejado) d_i. As saídas dos m nodos adicionais são coletadas em um nodo que as adiciona fornecendo a soma E_i como sua saída. O mesmo cálculo deve ser efetuado para cada um dos padrões d_i da rede. Uma unidade (nodo) coleta todos os erros quadráticos e calcula a sua soma E_1+...+E_p. A saída desta rede estendida é a função E.

Temos agora uma rede capaz de calcular o erro total para um dado conjunto de treinamento. Os pesos da rede são os únicos parâmetros que podem ser modificados para reduzir o erro quadrático médio E. Uma vez que E é calculado pela extensão da rede exclusivamente através da composição das funções dos nodos, ele é uma função contínua e diferenciável dos ℓ pesos $w_1, w_2,...,w_\ell$ da rede. Nós podemos assim minimizar E através do uso de um processo de gradiente descendente, para o qual nós precisamos calcular o gradiente da função E

$$\nabla E = (\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, ..., \frac{\partial E}{\partial w_\ell}) \qquad \text{(A.2.2)}$$

Cada um dos pesos é atualizado usando o incremento

$$\Delta w_i = -\gamma \frac{\partial E}{\partial w_i} \quad \text{para } i = 1, ..., \ell, \qquad \text{(A.2.3)}$$

Onde γ representa a constante de aprendizado, i. e., um parâmetro proporcional que define o comprimento do passo de cada iteração na direção negativa do gradiente.

Com esta extensão da rede original todo o problema de aprendizado reduz-se agora a uma questão de calcular o gradiente da função rede com respeito aos seus pesos. Uma vez que se tem um método para calcular este gradiente, podemos ajustar os pesos da rede iterativamente de forma a encontrar um mínimo da função de erro onde $\nabla E = 0$.

A.3 O Algoritmo *Backpropagation* (Retropropagação)

1. Todos os pesos devem ser inicializados aleatoriamente com valores baixos.

2. Cálculo de ativação . O nível de ativação de uma entrada é determinado pelo padrão (vetor) apresentado à rede.

3. O nível de ativação o_j de uma unidade oculta e de uma unidade de saída é determinado por

$$o_j = s\left(\sum w_{ji} o_i\right) \qquad \text{(A.3.1)}$$

onde w_{ji} é o peso de uma entrada o_i e s é uma função sigmóide.

4. Treinamento dos pesos. Iniciar pelas unidades de saída e proceder para trás para as camadas ocultas repetidamente. Ajustar os pesos através de

$$w_{ji}(t+1) = w_{ji}(t) + \Delta w_{ji} \qquad \text{(A.3.2)}$$

onde $w_{ji}(t)$ é o peso da unidade i para a unidade j no tempo t (ou na *t-ésima* iteração) e Δw_{ji} é o ajuste de peso.

5. A alteração do peso é calculada por

$$\Delta w_{ji} = \gamma \delta_j o_i \qquad \text{(A.3.3)}$$

onde γ é taxa de aprendizado independente ($0 < \gamma < 1$, e. g., 0.3) e δ_j é o gradiente do erro na unidade j. A convergência é as vezes mais rápida adicionando-se um termo de momento:

$$w_{ji}(t+1) = w_{ji}(t) + \gamma \delta_j o_i + \alpha \left[w_{ji}(t) - w_{ji}(t-1) \right] \qquad \text{(A.3.4)}$$

onde $0 < \alpha < 1$.

6. O gradiente do erro, para as unidades de saída, é dado por

$$\delta = o_j(1-o_j)(d_j - o_j) \qquad \text{(A.3.5)}$$

onde d_j é a saída desejada e o_j é a saída atual da unidade j.

7. O gradiente do erro, para as unidades ocultas, é dado por

$$\delta_j = o_j(1-o_j)\sum_k \delta_k w_{kj} \qquad \text{(A.3.6)}$$

onde δ_k é o gradiente do erro na unidade k para a qual aponta um conexão da unidade oculta j.

8. Repetir as iterações até a convergência nos termos do critério de erro selecionado. Uma iteração consiste na apresentação de um padrão, cálculo das ativações e modificação dos pesos.

Apêndice B

B.0 Estudo de Caso

O SMIC (Sistema de Monitoração Isotópica da Chaminé) é um sistema de monitoração contínua on-line em tempo real dos efluentes radioativos da chaminé da usina nuclear Angra 1. Os efluentes são monitorados através da análise isotópica desses efluentes liberados para o meio ambiente determinando a taxa de liberação precisa de cada isótopo. É um sistema importante para a segurança, pois permite monitorar as emissões no caso de um acidente, monitorando também as emissões normais do dia a dia. Trata-se de um sistema totalmente automatizado, composto por equipamentos de diferentes fabricantes integrados de forma a funcionarem perfeitamente em conjunto. Um minicomputador, em conjunto com o software de aplicação, é o responsável pelo funcionamento e controle de todo o SMIC. A Figura B.0 apresenta um diagrama de blocos simplificado do sistema.

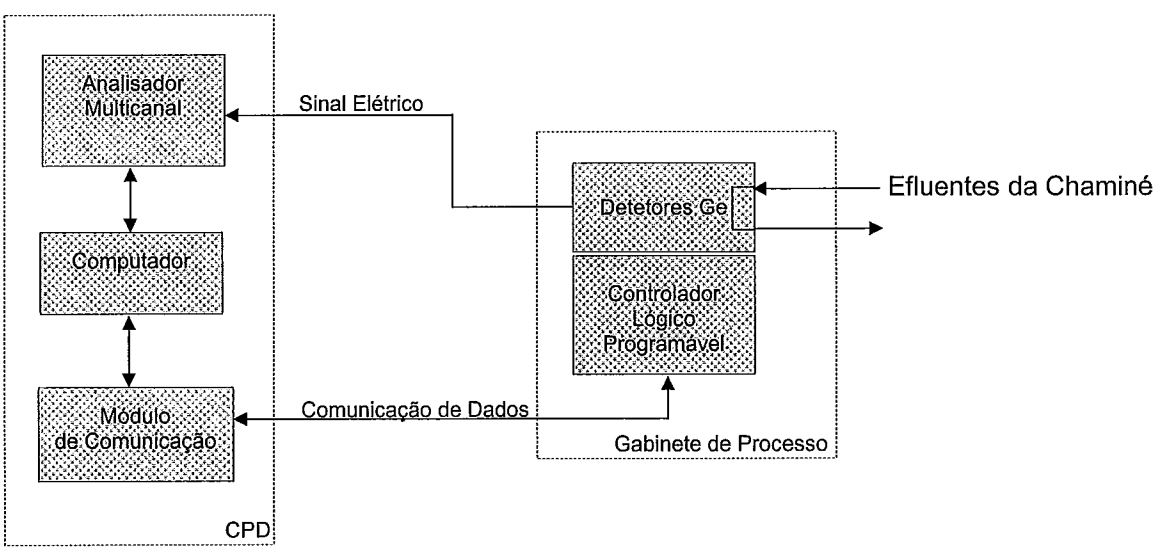

Figura B.0 – Diagrama simplificado do SMIC.

Além do minicomputador, o sistema tem dois outros subsistemas baseados em microprocessador digital: o Controlador Lógico Programável (CLP) e o Analisador Multicanal (AMC) e todo hardware eletrônico necessário para a aquisição de dados.

O CLP é responsável pelo controle físico (válvulas, bombas, etc.) de coleta das amostras no gabinete de processos próximo a chaminé. O AMC é responsável pela contagem de emissões radiativas dos efluentes coletados gerando os arquivos de dados necessários para análise dessas amostras, pelo minicomputador através do software aplicativo.

O software do SMIC é composto por várias tarefas de tempo real que controlam a aquisição dos efluentes radioativos (particulado, iodo e gás), executam análises isotópicas dos dados adquiridos e geram relatórios periódicos dos resultados das análises. O software é executado sob um sistema operacional de tempo real no minicomputador.

Podemos ver portanto que se trata de um sistema relacionado à segurança da planta, tanto para a sua operação normal como para as situações de emergência. A sua robustez, disponibilidade e confiabilidade devem ser as mais altas possíveis em função da sua importância para operação da usina.

B.1 O Processo de Identificação de Radionuclídeos

O processo de identificação de radionuclídeos a partir de emissões gama implica no reconhecimento do espectro de radiação gama do radionuclídeo a ser identificado. Em geral o espectro de radiação gama é obtido por meio de um processo de detecção e

contagem de emissões gama, através de sistemas contendo detetores, amplificadores e um multicanal.

A abordagem tradicional para a análise espectral de raios gama pode ser categorizada como uma busca por picos e ajuste de curvas. Esta abordagem envolve um processo iterativo de decomposição e reconstrução do espectro original até que um modelo matemático aproximado do espectro verdadeiro possa ser obtido. É um processo que requer um processamento considerável em termos de máquina, não tolerando pequenos desvios nos dados de entrada. A Figura B.1 apresenta o aspecto de um espectro simulado de Eu-152 em uma escala linear de contagens por canais (eixo abcissas). A Figura B.2 apresenta um espectro real de Eu-152 obtido do SMIC.

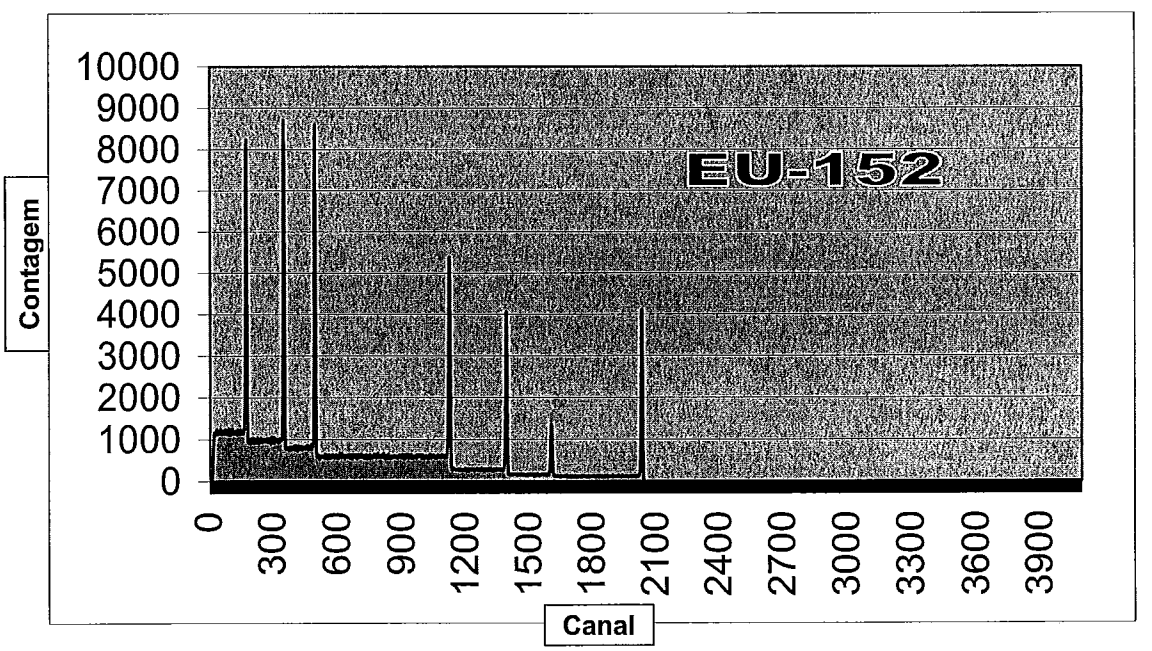

Figura B.1 –Espectro do elemento Európio 152 gerado através do programa *Cspectr* (Atom, 1999).

B.2 O Processo de Calibração

A maioria desses sistemas tradicionais de identificação de radionuclídeos só é capaz de identificar os espectros de radionuclídeos quando devidamente calibrados, necessitando assim de uma calibração periódica dos equipamentos.

O processo de calibração consiste no ajuste dos equipamentos de modo que o espectro de um radionuclídeo conhecido, usado como padrão, seja gerado corretamente com seus picos de energia devidamente posicionados nos canais de energia correspondentes ao elemento, segundo o banco de dados do sistema.

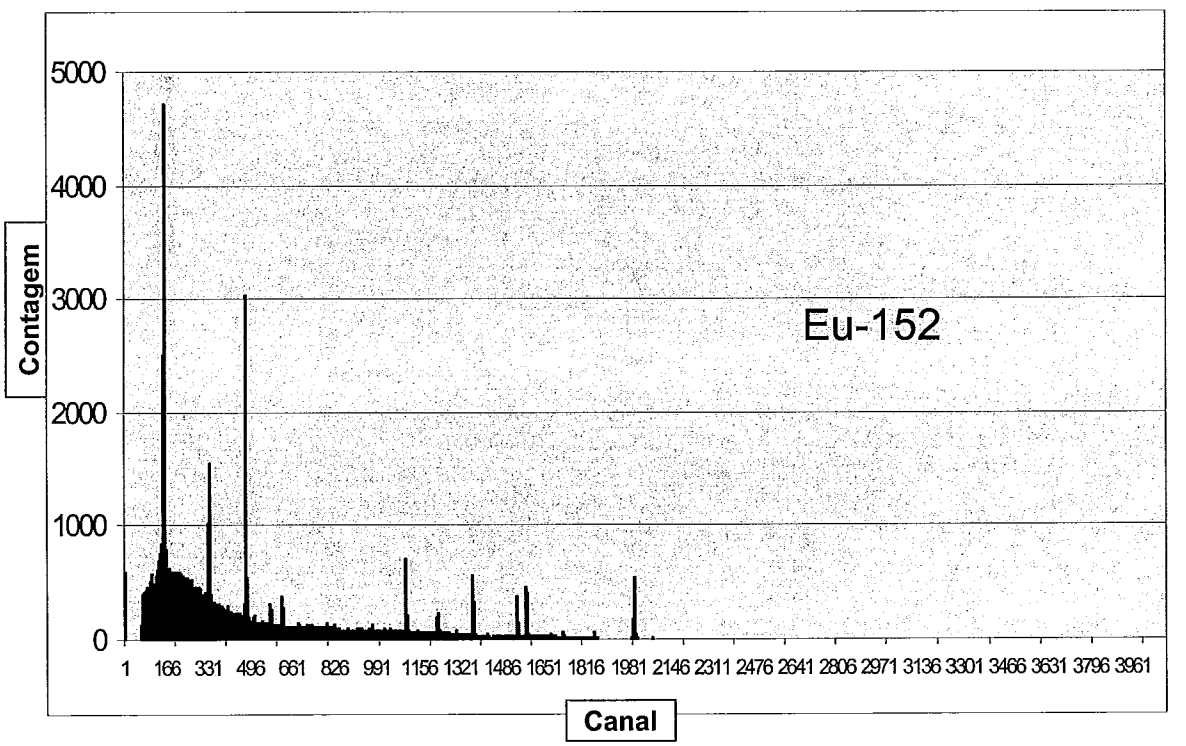

Figura B.2 –Espectro real do elemento Európio 152, obtido de um passo de calibração do SMIC.

B.3 O Problema

Em virtude da idade do equipamento, principalmente dos módulos eletrônicos do AMC, o SMIC vem apresentando problemas na sua calibração automática, sendo por isso incapaz de efetuar uma calibração automática a partir do radionuclídeo padrão. Desta forma o SMIC é incapaz de entrar em funcionamento, necessitando a intervenção de um operador humano para efetuar uma calibração manual.

No caso específico do SMIC, o ciclo inicial antes de cada período de coleta de amostras é composto por três passos:

1. No primeiro passo, o sistema expõe ao detetor de Germânio (Ge) uma amostra de Európio-152 (Eu-152) durante um tempo predeterminado (10 minutos) e conta as emissões gama do isótopo ao longo de 4096 canais do AMC.
2. Neste segundo passo o espectro de Eu-152 gerado no passo anterior é submetido ao software de análise onde são identificados os centróides e as áreas dos picos. É então aplicada aos centróides a equação de calibração com os valores da última calibração de energia. Os valores de energia obtidos a partir dessa equação são então comparados com os valores de energia dos picos de Eu-152, do banco de dados do SMIC para identificação do elemento, neste caso, o próprio Eu-152. Esta operação tem objetivo garantir que a relação CANAL-ENERGIA do equipamento AMC está correta.
3. No terceiro passo, o sistema conta a radiação de fundo, i. e., o detetor de Ge é exposto "em vazio" ao ambiente durante um tempo predeterminado (10 minutos). Esta operação tem por objetivo gerar o espectro de *background* do

sistema como um todo (ambiente, invólucro, blindagem, etc.) que será descontado do espectro da amostra coleta dos efluentes da chaminé.

O segundo passo é a calibração propriamente dita, onde surge o problema. Isto ocorre quando o sistema não é capaz de identificar os picos do Eu-152, o que acontece quando existe um pequeno desvio na relação CANAL-ENERGIA.

No caso de um sistema computadorizado automatizado como o SMIC, responsável pela monitoração de efluentes radiativos, e relacionado à segurança, a incapacidade de uma calibração automática compromete a sua disponibilidade e robustez, podendo implicar em consequências perigosas para o processo que tal sistema monitora.

B.4 Abordagem do Problema

A abordagem do problema de identificação de radionuclídeos através de RNA aplica o reconhecimento de padrão de todo o espectro. Este reconhecimento é feito através de uma única multiplicação vetor-matriz que resulta numa rápida (tempo real) identificação das amostras.

Para uma amostra composta por uma combinação de isótopos, o espectro da amostra $\underline{E}$, é (aproximadamente) uma superposição linear dos espectros individuais dos isótopos, $\underline{e_i}$. A equação (B.4.1) ilustra este fato, onde α_i é concentração relativa de cada isótopo na amostra.

$$\underline{E} = \sum_i \alpha_i \underline{e}_i \qquad \text{(B.4.1)}$$

Assim, o sistema deve ter uma resposta linear em relação às entradas. Isto implica em um desvio da maioria das RNA que implementam uma resposta não linear. Mesmo com uma resposta linear, a abordagem através de RNA apresenta vantagens em relação às abordagens tradicionais em termos de velocidade e simplicidade. Uma RNA modelada para responder de forma linear emprega funções de ativação lineares. Portanto, uma RNA *feedforward* que implementa funções de ativação lineares pode ser reduzida à uma RNA com uma camada de entrada e uma camada de saída.

Abordamos o problema de calibração SMIC de Angra 1, através da aplicação de Redes Neurais Artificiais (RNA) para a identificação de radionuclídeos. Optamos inicialmente, por ser a visão da aplicação do caso ideal (sem falhas), por experimentar a identificação de espectros através de RNA, partindo do trabalho de Keller (1994).

Em seu trabalho Keller (1994), faz um estudo comparativo entre dois tipos de RNA na identificação de radionuclídeos. São desenvolvidas duas RNA: uma rede perceptron linear (sem camadas ocultas) e uma rede OLAM (*Optimal Linear Associative Memory*) para dois detetores diferentes: NaI (Iodeto de Sódio) e Ge (Germânio).

Para a rede perceptron foi usada uma função de ativação modificada (linear para entradas positivas e zero para entradas negativas) conhecida como função perceptron. Foi usada para treinamento a regra delta em um processo iterativo. Com funções de ativação lineares o algoritmo de treinamento é matematicamente idêntico ao algoritmo de *backpropagation* já que a derivada dos termos na retropropagação será sempre igual a unidade neste caso.

A rede OLAM foi escolhida já que é útil em situações onde as entradas são constituídas por combinações lineares de padrões conhecidos (e. g. espectros de radiação gama). A OLAM é uma melhoria da abordagem inicial da matriz de memória associativa, no senso de que ela projeta um padrão de entrada em um conjunto de vetores ortogonais onde cada vetor ortogonal representa um único padrão (exemplar). Podemos ver que esta OLAM, é uma rede hetero-associativa. O processo de treinamento é direto através da ortogonalização da matriz, onde cada padrão do conjunto de treinamento é projetado em um único eixo ortogonal no espaço de saída.

No experimento de (Keller, 1994) foi usado um espectrômetro de raios gama ao qual foi conectada a RNA. Neste protótipo foi usado um detetor NaI, e são gerados dados através de 512 canais do espectrômetro. Todos os canais são usados pela RNA de tal forma que existe uma entrada para cada canal. Existe somente uma camada de processamento (camada de saída) na RNA cujo número de neurônios de saída é igual ao número de isótopos que se quer identificar, 8 neste caso. Um aspecto positivo desta abordagem para a análise espectral de raios gama é que todo o espectro é usado no processo de identificação ao invés de somente os picos. Por esta razão, ele é potencialmente mais útil para o processamento de dados de espectrômetros de raios gama de baixa resolução como os que empregam detetores NaI.

Os resultados iniciais obtidos por Keller (1994) demonstraram o potencial do paradigma das RNA para a análise espectral de raios gama. O estudo demonstrou a superioridade do desempenho da rede OLAM em relação à rede perceptron linear na classificação, acurácia e velocidade de treinamento. O desempenho em relação à classificação pode ser atribuído ao processo de ortogonalização usado pela OLAM durante o treinamento. Uma vez que esse processo não é iterativo, a rede OLAM

apresenta um tempo de treinamento significativamente menor do que a rede perceptron linear.

Uma desvantagem da rede OLAM, é que são necessários espectros quase perfeitos no processo de treinamento. Caso seja necessário, a rede OLAM pode ser treinada com espectro gerados através de simulações de Monte Carlo. A rede perceptron linear pode ser treinada com dados com ruído e com defeitos desde que esteja disponível um grande conjunto de treinamento.

A partir desses resultados obtidos por Keller (1994) partimos para a nossa abordagem do problema, apresentando antes os fundamentos teóricos da rede OLAM.

B.5 A Rede OLAM

A função de uma rede associativa é reconhecer vetores de entrada previamente aprendidos, mesmo que exista algum ruído. A vantagem da memória associativa em relação a outras abordagens é que somente as informações locais precisam ser consideradas. A resposta de cada unidade é determinada exclusivamente pela informação fluindo através dos seus próprios pesos. Podemos considerar três tipos de redes associativas:

1. Redes Heteroassociativas mapeiam m vetores de entrada $\mathbf{x}^1, \mathbf{x}^2,...,\mathbf{x}^m$ em um espaço n-dimensional em m vetores de saída $\mathbf{y}^1, \mathbf{y}^2,...,\mathbf{y}^m$ em um espaço k-dimensional, de forma que $\mathbf{x}^i \rightarrow \mathbf{y}^i$.
2. Redes Autoassociativas são um subconjunto especial das redes heteroassociativas, nas quais cada vetor é associado com ele mesmo, i. e., $\mathbf{y}^i = \mathbf{x}^i$ para $i = 1,...,m$. Estas redes são mais usadas para a correção de vetores de entrada com ruído.

3. Redes de reconhecimento de padrão são também um tipo especial de redes heteroassociativas. Cada vetor $\mathbf{x}^i$ é associado a um escalar i. O objetivo de tal rede é identificar o "nome" do padrão de entrada.

Estes tipos de rede podem ser implementados com uma única camada de unidades computacionais. A Figura B.3 mostra a estrutura de uma rede heteroassociativa.

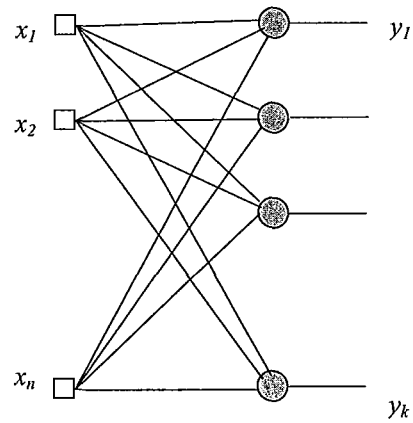

Figura B.3 – Rede heteroassociativa sem retroalimentação.

Seja w_{ij} o peso entre a unidade de entrada i e a unidade j. Seja **W** a matriz de pesos n x k. o vetor linha $\mathbf{x} = (x_1, x_2,...,x_n)$ produz o vetor de excitação **e** através da computação

$$\mathbf{e} = \mathbf{xW} \qquad \text{(B.5.1)}$$

A função de ativação é computada a seguir para cada uma das unidades. Se ela for a função identidade, as unidades são apenas associadores lineares e a saída **y** é exatamente **xW**. Em geral m diferentes vetores linha n-dimensionais $\mathbf{x}^1, \mathbf{x}^2,...,\mathbf{x}^m$ tem que estar associados a m vetores linha k-dimensionais $\mathbf{y}^1, \mathbf{y}^2,...,\mathbf{y}^m$. Seja **X** a matriz m x n cujas linhas são cada uma um dos vetores de entrada e seja **Y** a matriz m x k cujas linhas

são os vetores de saída. Estamos procurando a matriz de pesos **W** para a qual seja obtido

$$\mathbf{XW=Y.} \qquad \text{(B.5.2)}$$

No caso autoassociativo, associamos cada vetor com ele próprio e a equação (B.5.2) torna-se

$$\mathbf{XW=X} \qquad \text{(B.5.3)}$$

Se $m = n$, então **X** é uma matriz quadrada. Se ela for inversível, a solução é

$$\mathbf{W=X^{-1}Y,} \qquad \text{(B.5.4)}$$

o que significa que encontrar **W** equivale a resolver um sistema de equações lineares.

Estamos interessados em métodos de aprendizado alternativos capazes de minimizar a interferência cruzada (*crosstalk*) entre os padrões armazenados. Um dos métodos preferidos é usar a matriz pseudoinversa ao invés da matriz de correlação ($\mathbf{X^TX}$).

Como já comentamos, estamos procurando por uma matriz de pesos **W** tal que **XW = Y** (B.5.2). Uma vez que, em geral, $m \neq n$ e os vetores $\mathbf{x^1}$, $\mathbf{x^2}$,...,$\mathbf{x^m}$ não são necessariamente linearmente independentes, a matriz **X** não pode ser invertida. Mas, podemos procurar determinar a matriz **W** que minimiza a norma quadrática da matriz **XW - Y**, ou seja, a soma dos quadrados de todos os seus elementos. Um resultado bem conhecido da álgebra linear é que a expressão $\| \mathbf{XW - Y} \|^2$ é minimizada pela matriz

$$\mathbf{W = X^{+}Y,} \qquad \text{(B.5.5)}$$

onde $\mathbf{X}^+$ é a chamada pseudoinversa da matriz **X**.

Uma condição suficiente para uma associação perfeita seria

$$\mathbf{X}^+\mathbf{X} = \mathbf{I} \qquad \text{(B.5.6)}$$

onde **I** é a matriz identidade. Quando esta condição é satisfeita, a resposta atual produzida pela aplicação da matriz de vetores **X** à matriz de pesos **W** iguala o valor desejado **Y** da matriz armazenada, como é mostrado por

$$\begin{aligned} \mathbf{WX} &= \mathbf{YX}^+\mathbf{X} \\ &= \mathbf{Y} \end{aligned} \qquad \text{(B.5.7)}$$

onde foram usadas as equações (B.5.5) e (B.5.6) respectivamente. A condição da equação (B.5.6) pode ser realizada através do uso de um conjunto de vetores (i. e., colunas da matriz **X**) que são linearmente independentes. Isto por sua vez requer que m $\geq$ n, onde m é a dimensionalidade do espaço de entrada e n é o número de associações armazenadas. Sob estas condições, a matriz pseudoinversa $\mathbf{X}^+$ é definida por

$$\mathbf{X}^+ = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T \qquad \text{(B.5.8)}$$

onde $\mathbf{X}^T$ é a matriz transposta da matriz **X**. Onde se vê que a condição para a associação perfeita da equação (B.5.6) é satisfeita de imediato.

B.6 O Tratamento do Problema

Com base nos resultados obtidos por Keller (1994) partimos para a implementação de uma RNA do tipo matriz linear associativa, já que tal modelo de rede foi o que melhor desempenho apresentou (Keller, 1994).

Os espectros usados estão na forma de arquivos ASCII contendo 4096 pares de dados, onde o primeiro valor corresponde ao número do canal (1 a 4096) e o segundo ao número de contagens para aquele canal.

B.6.1 Rede OLAM Autoassociativa

Nosso primeiro passo foi construir uma RNA autoassociativa de forma a reproduzir o espectro de entrada na saída da rede. A rede OLAM com 4096 entradas, uma para cada canal, reproduziu o espectro conforme mostram as Figuras B.6.1 a B.6.4 abaixo. A entrada da rede é um espectro e a saída é a reprodução desse espectro.

Treinamento da Rede OLAM Autoassociativa

- Entrada X = Espectro Multicanal
- Saída Y = Espectro Multicanal
- Resultado W = Matriz de Transformação

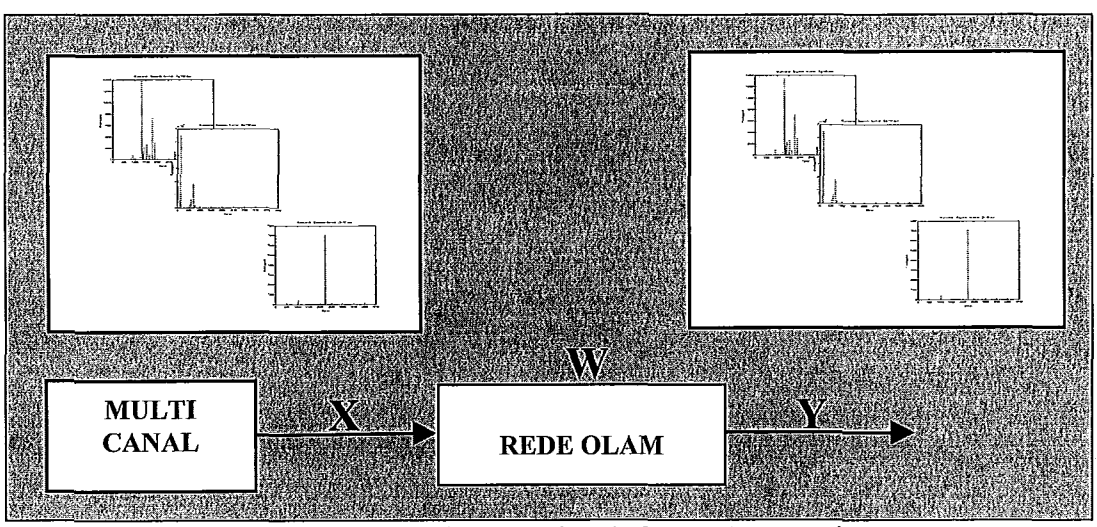

Figura B.6.1 – Treinamento da rede OLAM Autoassociativa.

Operação da Rede OLAM Autoassociativa

- Entrada x = Espectro Multicanal
- Saída y = Resposta Espectro (y = W * x)

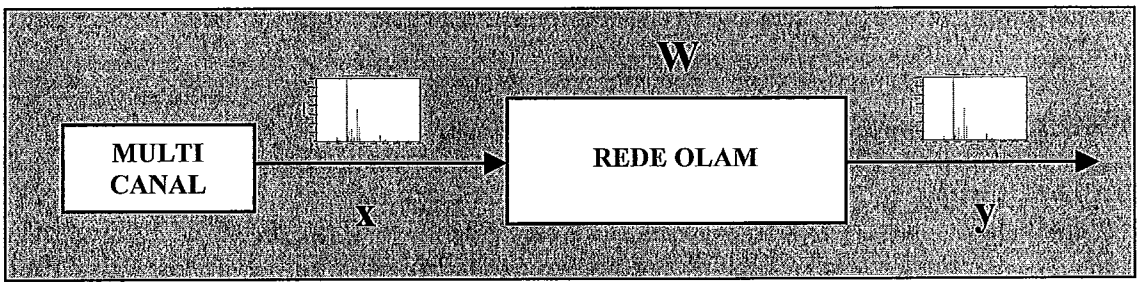

Figura B.6.2 –Operação da rede OLAM Autoassociativa.

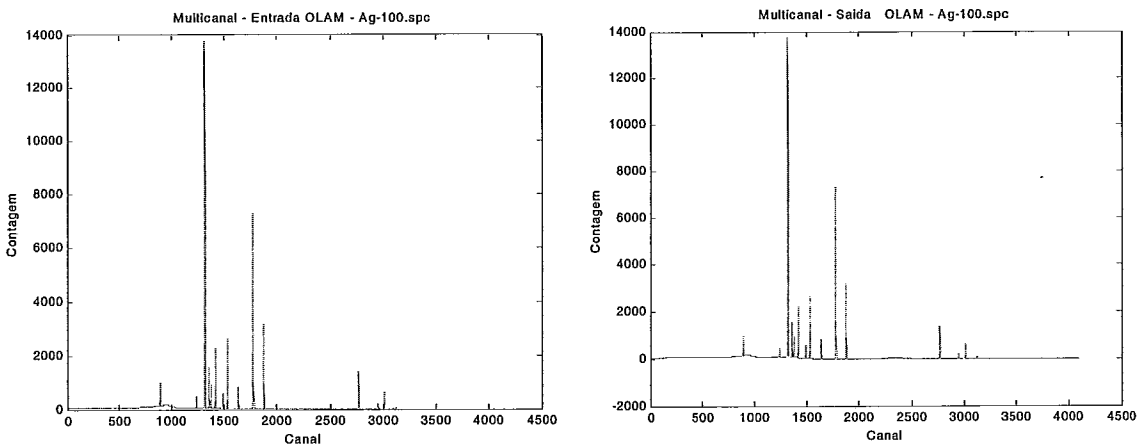

Figura B.6.3 –Espectro de entrada (Ag-100) e a resposta da rede OLAM.

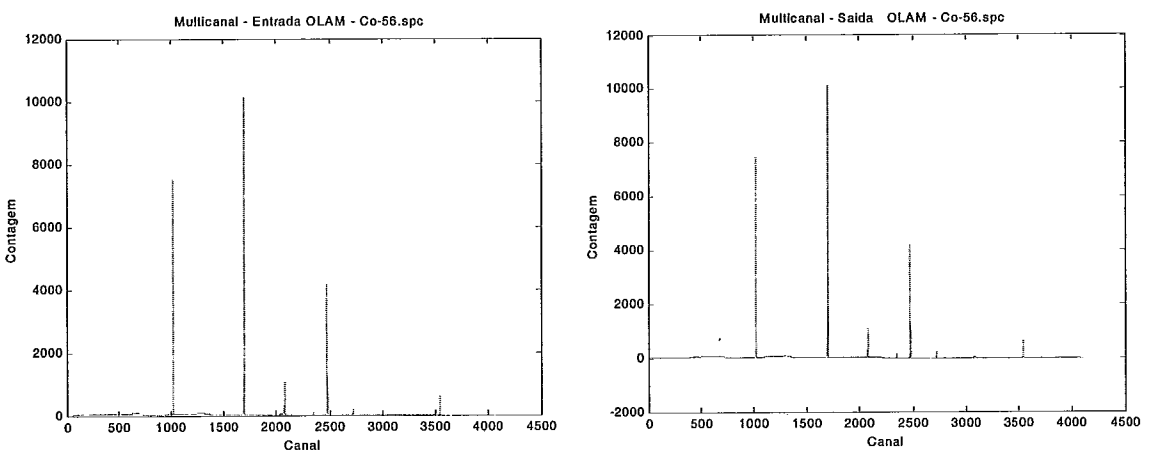

Figura B.6.4 –Espectro de entrada (Co-56) e a resposta da rede OLAM.

B.6.2 Rede OLAM Autoassociativa (Teste Espectro Atenuado)

Nesse teste apresentamos à rede um espectro com 50% de atenuação para observar a sua resposta. A rede respondeu corretamente.

- Entrada x = Espectro Atenuado
Eu-152 (Aten = 1/2)
x = 1/2 Eu-152

- Saída y = Resposta (Nuclídeo + %)
y = W*1/2 x
= 1/2 W*x

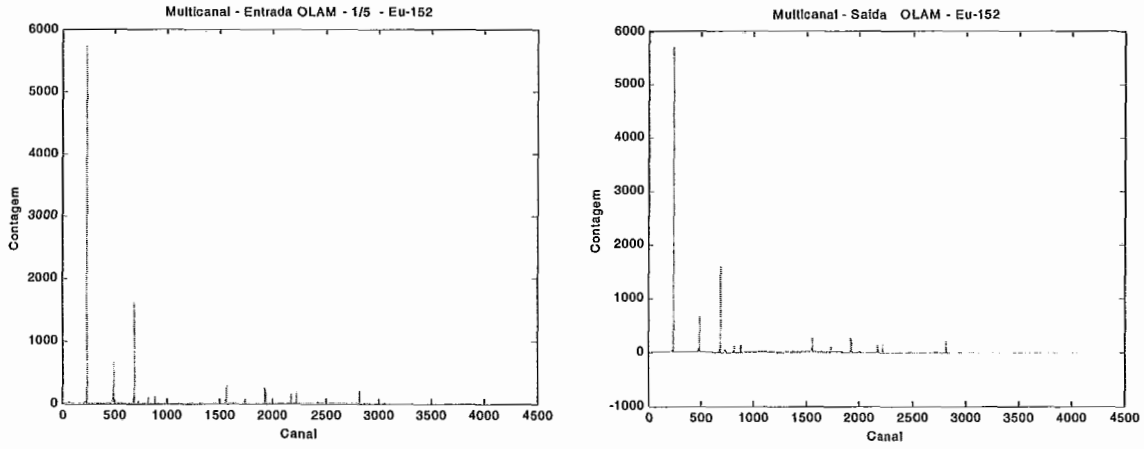

Figura B.6.5 –Espectro de entrada (1/2 Eu-152) e a resposta da rede OLAM.

B.6.3 Rede OLAM Autoassociativa (Teste Coquetel de Espectros)

Nesse teste apresentamos à rede um espectro composto de vários elementos combinados (coquetel) para observar a sua resposta. A rede respondeu corretamente.

- Entrada x = Coquetel

$$x = k1\ n1 + k2\ n2$$

- Saída y = **Resposta Espectro**

$$= W^*\ (k1\ n1 + k2\ n2)$$
$$= k1\ W^*\ n1 + k2\ W^*\ n2$$
$$= k1\ y1 + k2\ y2$$

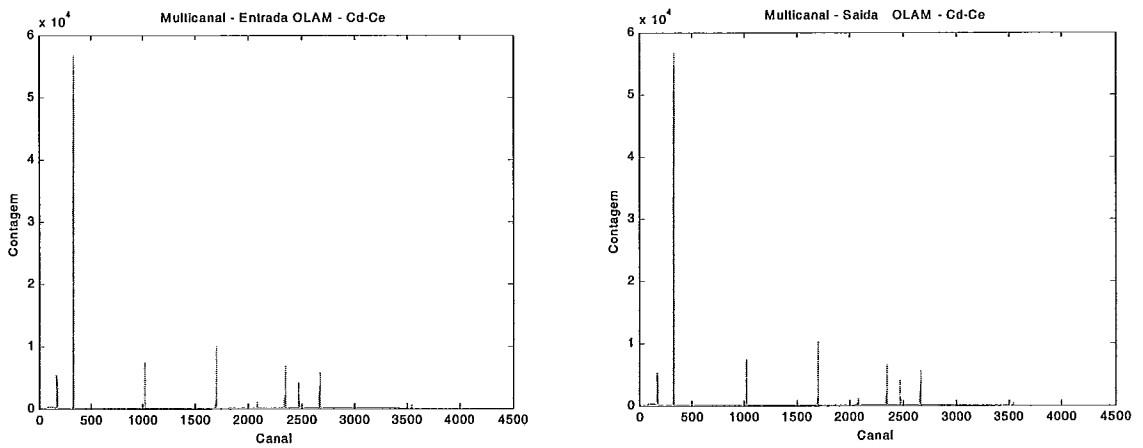

Figura B.6.6 –Espectro de entrada (Coquetel) e a resposta da rede OLAM.

B.6.2 Rede OLAM Heteroassociativa

Nessa etapa o objetivo é a identificação e quantificação dos radionuclídeos apresentados à rede. A rede reconhece o elemento segundo seu "nome" e a quantidade em relação a aprendida no treinamento.

1 - Treinamento

- Entrada X = Espectros Multicanal
- Saída Y = Nuclídeo + %
- Resultado W = Matriz de Transformação

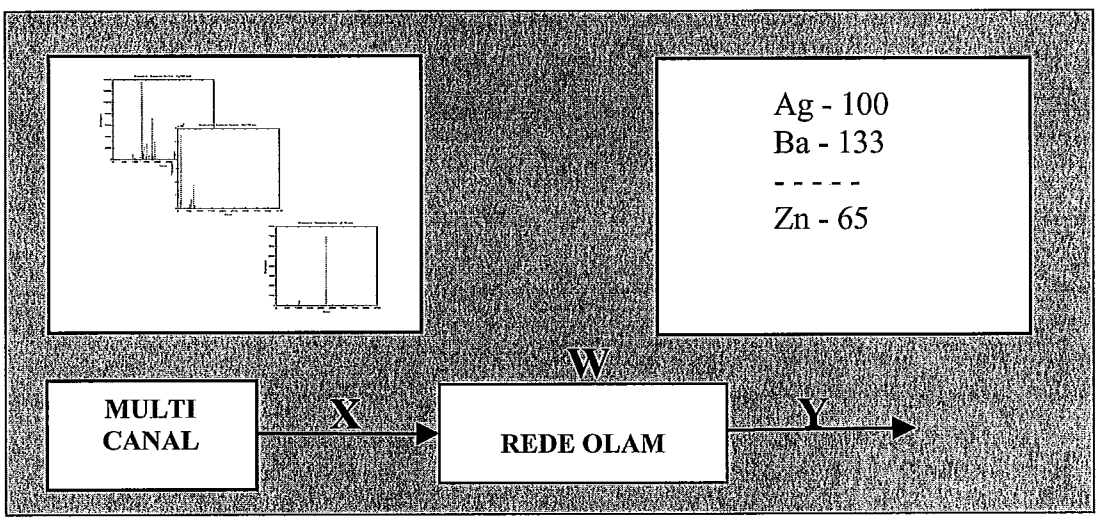

Figura B.6.7 –Treinamento da rede OLAM Heteroassociativa.

2 - Operação

- Entrada x = Espectro Multicanal
- Saída y = Resposta (Nuclídeo + %)
y = W * x

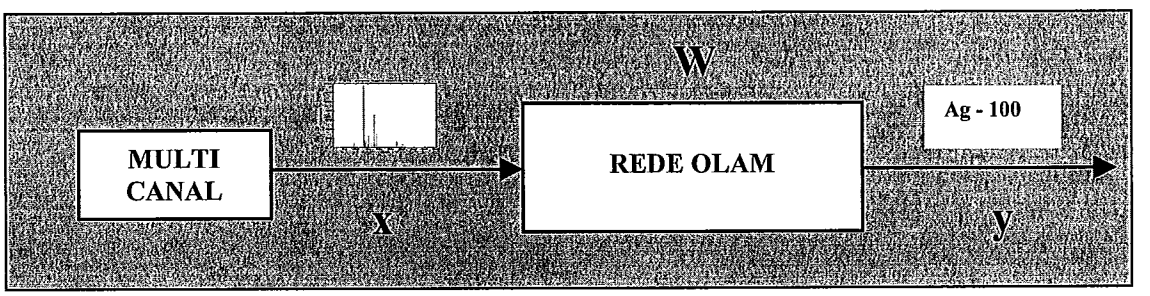

Figura B.6.8 –Operação da rede OLAM Heteroassociativa.

B.6.2.1 Rede OLAM Heteroassociativa (Teste de Coquetel)

A rede deve reconhecer os elementos que compõem a amostra (coquetel) segundo os seus respectivos "nomes" e as quantidades em relação as aprendidas no treinamento.

- Entrada x = Coquetel
x = k1 n1 + k2 n2

- Saída y = Resposta (Nuclídeo + %)
= W* (k1 n1 + k2 n2)
= k1 W* n1 + k2 W* n2
= k1 y1 + k2 y2

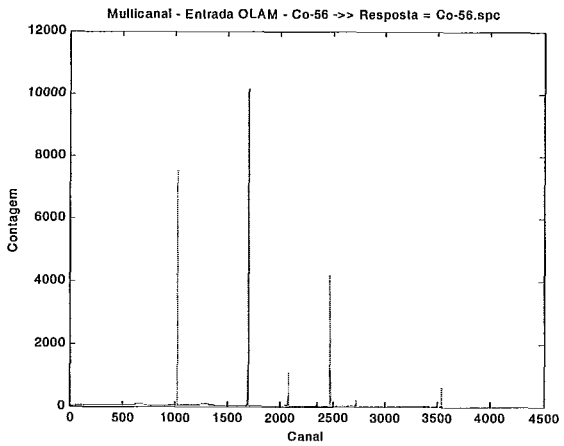

Figura B.6.9 – A rede identificou o Co-56.

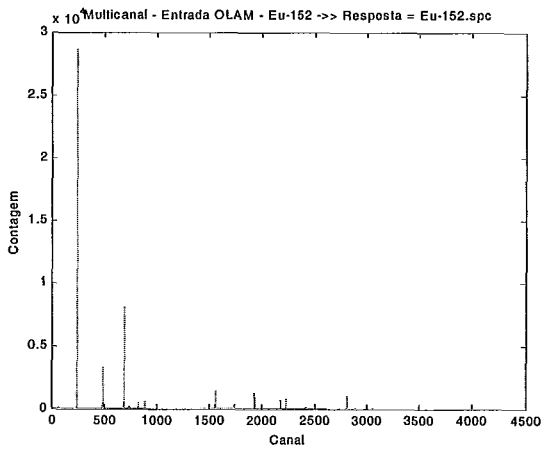

Figura B.6.10 – A rede identificou o Eu-152.

B.6.3 Rede OLAM (Teste de Translação)

Com o objetivo de observar o comportamento da rede OLAM, deslocamos (translação) um espectro em 100 canais e a rede não foi capaz de identificar conforme mostra a Figura B.6.11 abaixo.

- Entrada **x = Translação do Espectro**

Eu-152 (Translação = 100 canais)

- Saída **y = A rede não reproduz.**

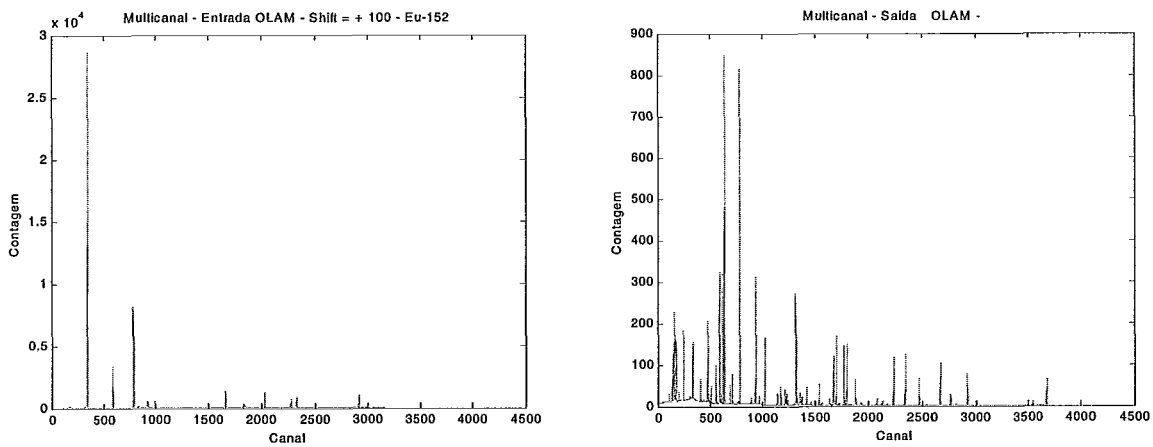

Figura B.6.11 – A rede não foi capaz de reconhecer o espectro (Eu-152) deslocado.

B.7 Reconhecimento de Padrão Invariante à Translação

De forma a tornar robusto o processo de memória associativa, no senso de ser capaz de reconhecer padrões independentemente da sua localização, empregamos a transformada discreta de Fourier (Antoniou, 1979, Bracewell, 1965). Assim, caso o espectro esteja deslocado (translação) em relação à origem ainda é possível identificá-lo.

B.7.1 A Transformada de Fourier

A Transformada Discreta de Fourier (*DFT*) de uma função discreta *x(n)* é definida por (Antoniou, 1979, Bracewell, 1965):

$$DFT\{x(n)\} = X(jk) = Ax(k) * exp\ [j\phi x(k)] \qquad \text{(B.7.1.1)}$$

$$Ax(k)=abs[X(jk)] \quad \text{e} \quad \phi x(k)=angle[X(jk)] \qquad \text{(B.7.1.2)}$$

A *DFT* representa cada ponto da função por um número complexo (amplitude e fase) e apresenta propriedades de linearidade, periodicidade e simetria.

Propriedade de Linearidade:

Se

$$DFT\{x(n)\} = X(jk) \; e \; DFT\{y(n)\} = Y(jk)$$

$$Ax(k)=abs[X(jk)] \quad \text{e} \quad \phi x(k)=angle[X(jk)]$$

$$Ay(k)=abs[Y(jk)] \quad \text{e} \quad \phi y(k)=angle[X(jk)] \qquad \text{(B.7.1.3)}$$

então

$$DFT\{k1\ x(n) + k2\ y(n)\} = k1\ Ax(k) * exp\ [j\phi x(k)] + k2\ Ay(k) * exp\ [j\phi y(k)]$$

$$DFT\{k1\ x(n) + k2\ y(n)\} = k1\ X(jk) + k2\ Y(jk) \qquad \text{(B.7.1.4)}$$

Observar no entanto que

$\phi[DFT\{\mathrm{k1}\ x(n)\}\} = angle[X(\mathrm{j}k)]$ e $\phi[DFT\{\mathrm{k2}\ y(n)\}\} = angle[Y(\mathrm{j}k)]$ e

$A\{DFT\{\mathrm{k1}\ x(n)\}\} = \mathrm{k1}Ax$ e $A\{DFT\ \{\mathrm{k2}\ [y(n)\}\} = \mathrm{k2}Ay$ e

$A\{DFT[\mathrm{k1}\ x(n) + \mathrm{k2}\ y(n)]\} <> \mathrm{k1}\ Ax + \mathrm{k2}\ Ay$ (B.7.1.5)

Periodicidade

$X[j(k+rN)] = X\ (j\mathrm{k})$ (B.7.1.6)

o que indica que $X\ (j\mathrm{k})$ é uma função periódica com período N.

Simetria

$X[j(N-k)] = X^*\ (jk)$ e

$\mathrm{Re}\ X[j(\mathrm{N}-k)] = \mathrm{Re}\ X(jk)$

$\mathrm{Im}\ X[j(N-k)] = -\mathrm{Im}\ X[jN-k)]$ (B7.1.7)

Considerando as propriedades de linearidade, periodicidade e simetria da transformada discreta de Fourier, podemos empregá-la como fase de préprocessamento dos dados de uma rede OLAM de forma que esta responda qualitativa e quantitativamente a respeito de uma amostra de radionuclídeos para os quais a rede foi treinada, independentemente da sua localização (translação) ao longo do eixo dos canais de energia. A implementação eficiente da *DFT* é denominada *Fast Fourier Transform* (*FFT*) (Antoniou, 1979), e passaremos a usar sua denominação no lugar de *DFT*. As

Figuras 7.0 e 7.1 apresentam o espectro de alguns radionuclídeos e as suas respectivas amplitude e fase após a aplicação da FFT.

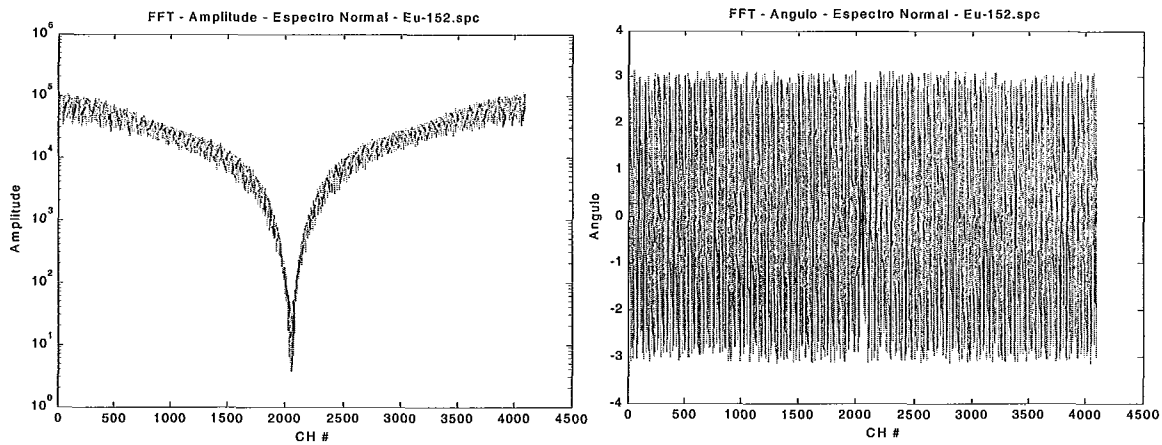

Figura B.7.0 – FFT (amplitude e fase) do espectro normal do Eu-152.

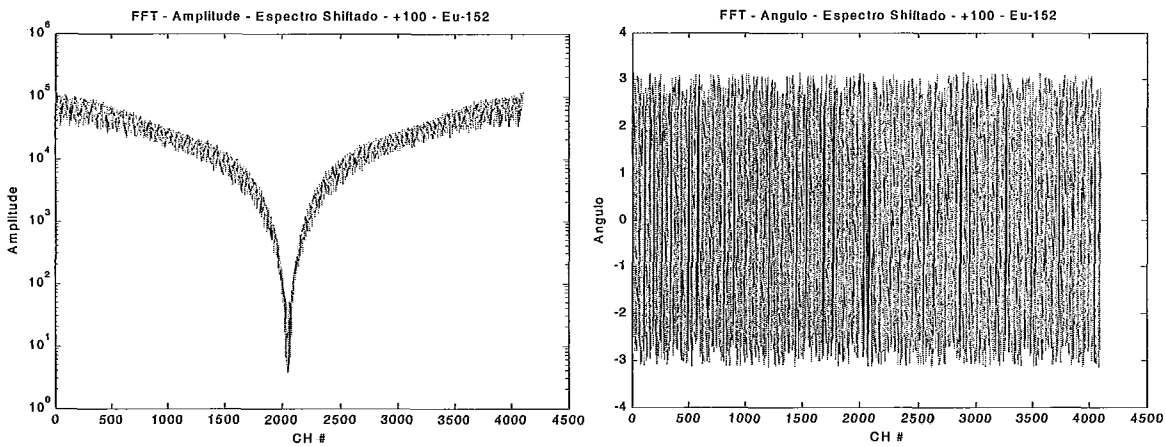

Figura B.7.1 – FFT (amplitude e fase) do espectro deslocado do Eu-152.

As figuras B.7.0 e B.7.1 mostram a FFT dos espectros normal e deslocado do Eu-152, respectivamente. A amplitude da FFT é igual nos dois casos mas, a fase não, ver eq. (B.7.1.5). Embora a amplitude seja a mesma para um espectro do nuclídeo deslocado, devemos usar a amplitude e a fase pois a amplitude de uma combinação de espectros não é igual à combinação das amplitudes da FFT, ver eq. (B.7.1.5). Devido à

propriedade de simetria da FFT, a rede pode utilizar apenas metade dos pontos da amplitude e metade dos pontos da fase da FFT. Assim a rede continua sendo uma rede de 4096 entradas como no caso do uso direto dos espectros. As Figuras B.7.2 e B.7.3 mostram o processo de treinamento da rede a partir do préprocessamento dos espectros provenientes do multicanal através da aplicação da FFT.

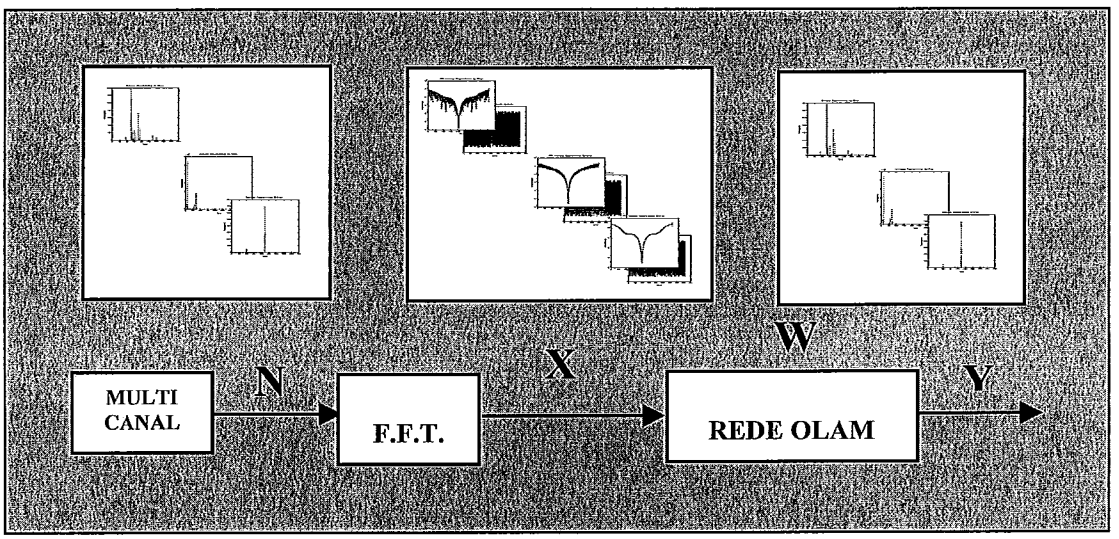

Figura B.7.2 – Treinamento da rede para reprodução dos espectros (autoassociação) com préprocessamento por FFT(amplitude e fase).

Na Figura B.7.2 a rede recebe como entrada os espectros FFT (amplitude e fase) e como saída desejada o espectro esperado. Na Figura B.7.3 a rede recebe como entrada os espectros FFT (amplitude e fase) e como saída desejada os rótulos e as quantidades correspondentes às amostras de entrada do treinamento.

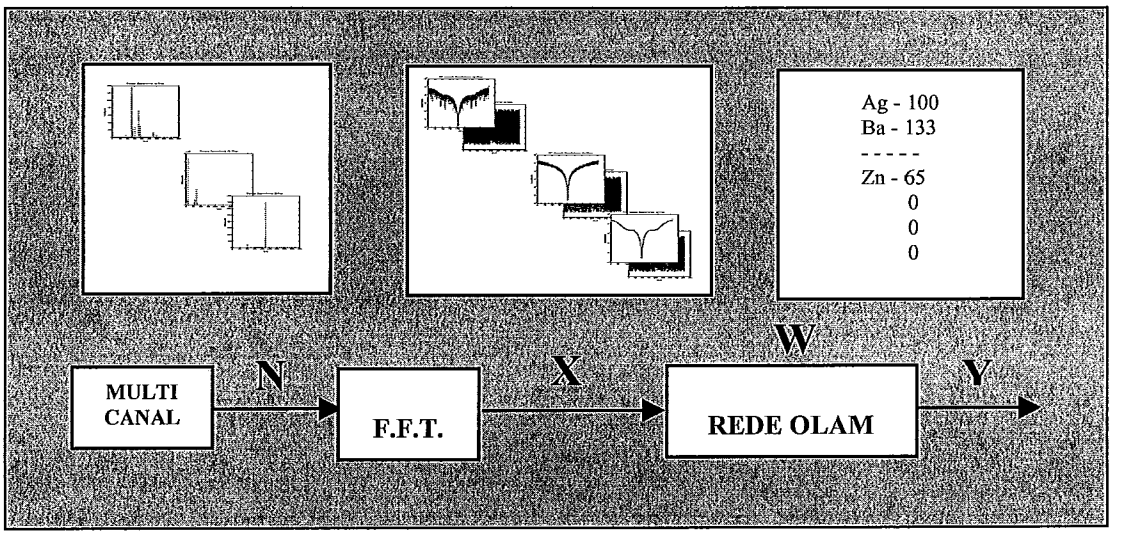

Figura B.7.3 – Treinamento da rede para classificação (heteroassociação) com préprocessamento por FFT(amplitude e fase).

A figura B.7.4 mostra a resposta da rede para um espectro de Eu-152 deslocado. A rede com préprocessamento por FFT dos espectros foi capaz de responder e quantificar espectros de amostras de elementos individuais e suas combinações. Isto se deve ao fato da FFT ser uma transformação linear e invariante à translação eqs. (B.7.1.3 a B.7.1.7) e ao fato da rede ser também linear.

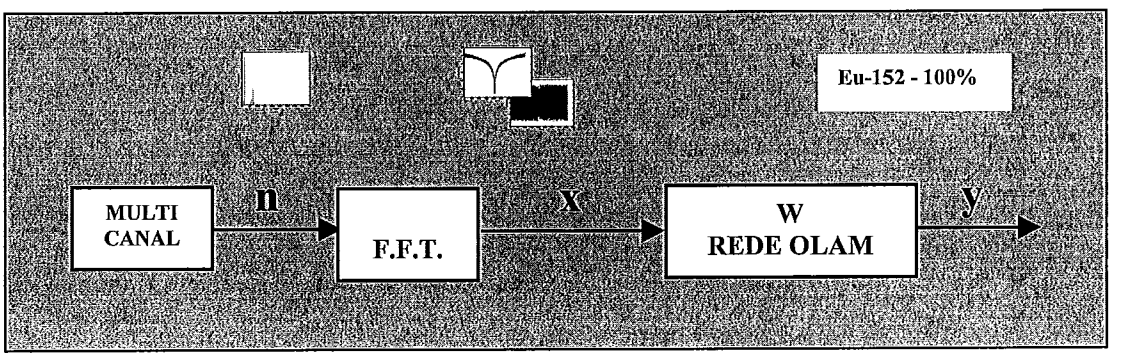

Figura B.7.4 – Resposta da rede ao espectro do Eu-152 deslocado

Os resultados obtidos mostram que as RNA do tipo memórias associativas podem ser utilizadas no reconhecimento de espectros gama, o que requer espectros sem deformações. O problema do deslocamento (translação) dos espectros pode ser resolvido se os dados de entrada forem préprocessados para extrair características invariantes do espectro de entrada. A Transformada Rápida de Fourier foi utilizada com sucesso para identificar espectros deformados por translação. A abordagem para a solução do problema genérico de deformação requer uma grande quantidade de dados de treinamento, quais sejam espectros com diversos graus de deformação, aumentando assim o número de casos de treinamento da rede exigindo um compromisso entre o número de casos de treinamento e o *crosstalk.*