UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Software Tolerante a Falhas para Aplicações Tempo Real

por

FERNANDA KRUEL DENARDIN

Dissertação submetida à avaliação,
como requisito parcial
para a obtenção do grau de
Mestre em Ciência da Computação

Profª. Drª. Ingrid Eleonora Schreiber Jansch-Pôrto
Orientadora

Porto Alegre, maio de 1997.

CIP - CATALOGAÇÃO NA PUBLICAÇÃO

Denardin, Fernanda Kruel

Software Tolerante a Falhas para Aplicações Tempo Real / por Fernanda Kruel Denardin. — Porto Alegre: CPGCC da UFRGS, 1997.

128f.: il.

Dissertação (mestrado) — Universidade Federal do Rio Grande do Sul. Curso de Pós-Graduação em Ciência da Computação, Porto Alegre, BR-RS, 1997. Orientadora: Jansch-Pôrto, Ingrid Eleonora S.

1. Tolerância a Falhas. 2. Tempo Real. 3. Software Tolerante a Falhas. 4. Sistemas Tempo Real Tolerantes a Falhas. I. Jansch-Pôrto, Ingrid Eleonora S. II. Título.

Aos meus pais,

Luiz Vilmar e Nádia

e ao meu noivo,

Loidemar.

Agradecimentos

Muitas pessoas me auxiliaram e encorajaram durante o desenvolvimento deste trabalho e eu não poderia tê-lo completado sem o seu auxílio. Esta dissertação é fruto de um trabalho desenvolvido durante dois anos, que envolveu direta ou indiretamente todos os que me cercam. Meu sincero agradecimento a todos vocês que me ajudaram, da sua maneira, do seu jeito... Mas de forma especial:

À professora Ingrid Jansch-Pôrto, pela sua disponibilidade, além da segura orientação e do auxílio despendidos sempre que necessário durante estes dois anos de trabalho. E por ter sido incansável na sua tarefa...

À professora Maria Lúcia Blanck Lisbôa, pela colaboração, interesse e materiais cedidos para o desenvolvimento desta dissertação.

Ao professor João César Netto, pelo interesse e incentivo, oferecidos em um dos momentos mais dificeis desta caminhada.

A todos colegas - alunos e professores - do grupo de tolerância a falhas, pelo interesse e pela confiável troca de idéias.

Aos demais colegas, professores e funcionários do CPGCC da UFRGS, pela sua dedicação constante.

À CAPES, pelo apoio concedido através de uma bolsa de estudos, que viabilizou a realização deste trabalho.

Enfim, aos meus pais, aos meus irmãos, a toda minha família, aos meus amigos - em especial, a Ana Carolina e a Rejane - e ao meu noivo, que estiveram sempre ao meu lado: pela sua tolerante compreensão, pela constante paciência e pelo apoio oferecido em todas as horas, nas mais diversas situações...

Muito obrigada!

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

ABNT	Associação Brasileira de Normas Técnicas.
ARINC	*Aeronautical Radio Inc.*
ATC	*Asynchronous Transfer of Controls.*
CAN	*Controller Area Network.*
CCITT	Comitê Consultivo Internacional de Telefonia e Telegrafia.
CLP	Controlador Lógico Programável.
COMTRAC	*Computer Aided Traffic Control.*
CRC	*Cyclic Redundancy Check.*
CSMA/CA	*Carrier Sense Multiple Access with Collision Avoidance.*
CSMA/CD	*Carrier Sense Multiple Access with Collision Detection.*
DEAL	*DEdos Application Language.*
DEDOS	*DEpendable Distributed Operating System.*
DREAM	*Distributed Real-time Ever Available Micro-computing.*
E/S	Entrada e Saída.
FBD	*Function Block Diagram.*
FIP	*Flux Information Processus ou Factory Instrumentation Protocol.*
FTCS	*Fault-Tolerant Computing Systems.*
HARTOS	*Hexagonal Architecture Real-Time Operating System.*
IL	*Instruction List.*
IPC	*InterProcess Communication.*
LAN	*Local Area Network.*
LD	*Ladder Diagram.*
LON	*Local Operating Network.*
MARS	*Maintanable Real-time System.*
MTBF	*Mean Time Between Failure.*
MTTF	*Mean Time To Repair.*
PC	*Personal Computer.*
PCT	Processo Crítico no Tempo.
PNV	Programação N-Versões.
Profibus	*Process Field Bus.*
RPC	*Remote Procedure Call.*

RTB	*Real-Time Based.*
RT-CDL	*Real-Time Common Design Language.*
RTO	*Real-Time Object.*
RTR	Reflexivo de Tempo Real.
RTT	*RealTimeTalk.*
SOTR	Sistema Operacional Tempo Real.
ST	*Structured Text.*
TDMA	*Time Division Multiple Access.*
THT	*Token Hold Time.*
TMR	*Triple Modular Redundancy.*
TRT	*Token Rotation Time.*
TTP	*Time-Triggered Protocol.*
UCP	Unidade Central de Processamento.
UTI	Unidade de Terapia Intensiva.

Resumo

Esta dissertação aborda um ramo da computação que se encontra em crescente desenvolvimento: a computação em tempo real. Os sistemas de computação tempo real surgiram a partir da necessidade de substituição do controle humano, que muitas vezes é falho, em situações complexas ou críticas, onde máxima confiabilidade e disponibilidade são exigidas para garantir a segurança do sistema. A área de aplicação diferencia-se de outras convencionais por possuir diferentes tipos de restrições de tempo e operar em ambientes não-determinísticos. Entretanto, atualmente tais sistemas estão tornando-se grandes, complexos, distribuídos, adaptativos e cada vez mais presentes nas aplicações do dia-a-dia, o que tende a exigir soluções mais simples e generalizadas.

Pelo fato de tais sistemas normalmente atuarem sobre aplicações críticas, é importante salientar que, em algumas situações, pequenos erros no sistema podem levar a grandes catástrofes. Mesmo atrasos mínimos no tempo de resposta são problemáticos, podendo ocasionar degradações ou ações erradas no mundo físico controlado pelo sistema tempo real. Como nestes casos máxima confiabilidade e disponibilidade são exigidas para garantir a sua segurança, tornou-se importante a construção de sistemas tempo real tolerantes a falhas. Dessa forma, é visivelmente crescente a necessidade de utilização de mecanismos capazes de abordar os requisitos de tempo real e tolerância a falhas de forma integrada durante o desenvolvimento do sistema. Assim, o processo de desenvolvimento de sistemas tempo real confiáveis torna-se mais simples e mais eficiente.

A necessidade de maior conhecimento do uso de tolerância a falhas para obter segurança no funcionamento de aplicações tempo real levou ao desenvolvimento deste trabalho, onde buscou-se um caminho de solução para a adequação das técnicas de tolerância a falhas a estas aplicações. Sabe-se que para produzir software confiável e, desta forma, de maior qualidade, além do emprego de boas técnicas de engenharia de software, é necessário compreender os principais conceitos e técnicas de tolerância a falhas. Por outro lado, é importante ter-se conhecimento dos mecanismos oferecidos pelas diversas camadas de software de um sistema - protocolo de comunicação, sistema operacional e linguagem de programação - para apoiar estas atividades de tolerância a falhas.

Este trabalho busca analisar os mecanismos e técnicas usados na implementação de software tolerante a falhas frente às situações mencionadas, uma vez que nem todas as técnicas conhecidas podem ser indistintamente aplicáveis a estas situações. Os resultados desta análise são organizados na forma de uma taxonomia, visando assim auxiliar projetistas de desenvolvimento de software a tomarem decisões importantes na construção de sistemas tempo real tolerantes a falhas. Os mecanismos são agrupados de acordo com o nível de implementação: sistemas operacionais, linguagens de programação e protocolos de comunicação, destacando suas características e aplicabilidade. Por fim, o uso da classificação é demonstrado com a análise de três casos-exemplo.

Palavras-chave: tempo real, tolerância a falhas, software tolerante a falhas, sistemas tempo real tolerantes a falhas.

TITTLE: "Fault-Tolerant Software to Real-Time Applications"

Abstract

This dissertation is about a computer science field which is in growing development, that is, real-time computation. Real-time computing systems have emerged from the necessity of substituting human control which is sometimes failed in complex or critical situations. In these ones maximum availability and reliability are requested in order to guarantee the system dependability. The application area differs from the conventional ones because it has particular time constraints and operates in non-deterministic environments. Nevertheless, nowadays such systems are becoming large, complex, distributed and adaptive but tend to demand simpler and generalized solutions as they are more present in daily applications.

Since such systems normally act on critical applications it is important to reinforce, that in some situations, subtle systems errors may generate big catastrophes. Even slight delays in response time are troublesome and they may cause degradation or wrong acts in physical world controlled by real-time systems. In these cases maximum reliability and availability are requested in order to guarantee system dependability. Thereby, the requirement of including mechanisms capable of achieving real-time and fault tolerance in an integrated way during the system design has been increased. Thus, the developing process of reliable real-time systems becomes simpler and more effective.

The necessity of improving designers knowledge on using fault tolerance in order to obtain dependability on real-time applications has motivated this study. Our main goal has been to find an adequate way of using fault tolerance techniques to these applications. It is known that the development of reliable software not only requires appropriate software engineering techniques but also demands understanding of main politics and mechanisms used to implement fault tolerance techniques in these situations. Otherwise, it is very important to know the related support that is offered by the different software levels of a system - communication protocol, operating system and programming language.

This study has as purpose analyzing the mechanisms and techniques used in implementation of fault-tolerant software applied to the previously mentioned situations. The basic supposition is that not all the known techniques may be applied indistinctly to these situations. The properties of the software are organized according to a taxonomy, where the mechanisms are bracketed in groups according to implementation level: operating systems, programming languages and communication protocols. In this presentation, the characteristics and applicability of the software tools are stood out in order to help developing-software designers to decide what is important to build fault-tolerant software. Finally, the use of the classification is demonstrated by analyzing three case-examples.

Keywords: real-time, fault tolerance, fault-tolerant software, fault-tolerant real-time systems.

1 Introdução

Desde o surgimento dos primeiros computadores, acompanhando a evolução tecnológica, eles foram se tornando gradativamente mais presentes em nossas vidas. O uso generalizado dos computadores deve-se, em grande parte, à redução de preços; assim a tendência é de um aumento deste mercado. Como principal conseqüência deste fato, criou-se uma forte relação de dependência entre o homem e as máquinas. Em vista disso, é esperado que estas cumpram adequadamente suas funções, não frustrando nem prejudicando seus usuários. A obtenção de qualidade também contribuiu significativamente para a generalização do uso dos computadores.

Por outro lado, em situações que envolvem maior necessidade de funcionamento correto, a possibilidade de substituir o controle humano é interessante, visto que este muitas vezes é falho. Esta substituição é justificada considerando-se que, mesmo no controle de aplicações simples, a capacidade humana apresenta uma baixa velocidade de reação a eventos externos e limitações no processamento de informações, além do fato de que o ser humano é suscetível à fadiga e apresenta um comportamento dificilmente modelável. Assim, pode-se concluir que não é confiável deixar o controle de processos mais complexos, e até mesmo críticos, sob sua exclusiva responsabilidade. Nesse sentido, foram desenvolvidos sistemas computacionais capazes de realizar de forma mais adequada este tipo de controle: os sistemas tempo real.

A área de aplicação dos sistemas tempo real inclui sistemas que controlam, por exemplo, comutadores telefônicos, robôs, usinas nucleares e pilotos automáticos de aviões. Estes não são sistemas comuns ou que executam tarefas triviais; pelo contrário, qualquer tipo de falha, inclusive as temporais, pode comprometer todo o sistema, gerando desde prejuízos financeiros, ambientais e até perdas de vidas humanas.

O uso intensivo de computadores controlando diversos aspectos da vida moderna é parcialmente justificado pelo melhor desempenho qualitativo e quantitativo, incluindo a utilização dos sistemas tempo real. Porém, em decorrência disto, tornou-se necessário que os sistemas de computação utilizados apresentem altos índices de confiabilidade, visando garantir a adequada realização das tarefas que lhes são atribuídas. Quando os sistemas de computação controlam situações críticas, seu funcionamento correto é imprescindível, podendo causar até mesmo grandes catástrofes caso isto não ocorra. Os problemas envolvidos na computação confiável estão presentes desde o surgimento dos primeiros computadores e, como tantos outros aspectos importantes da computação, têm Von Neumann à sua frente [VON 56]. Estudos mais aprofundados voltam a demonstrar o interesse científico na área, uma década após [CAR 68]. Desde então, existe o interesse em estudar e desenvolver técnicas especiais que buscam garantir o funcionamento adequado dos sistemas de computação, uma vez que estes não estão livres de falhas: as técnicas de tolerância a falhas. Estas técnicas consideram a possível existência de falhas residuais após a conclusão das fases de projeto de um sistema, bem como o aparecimento de falhas durante a vida útil do mesmo.

Tolerância a falhas não é uma área nova, o evento internacional mais importante - o *International Fault-Tolerant Computing Systems* (FTCS) - tem sido realizado anualmente desde 1970. No entanto, até o momento, o enfoque maior foi dado à

tolerância a falhas do hardware em implementações práticas. Esta preferência é justificável porque é razoável pensar-se primeiro em computadores que nunca param e que são confiáveis, para após pensar-se em programas com estas características [LIS 93]. Este certamente é um dos motivos pelos quais tolerância a falhas encontra-se atualmente mais avançada na implementação de técnicas de hardware. Mas também os componentes de hardware tornaram-se muito mais confiáveis enquanto que o software, mais complexo, passou a ser mais difícil de testar e de avaliar quanto à correção de funcionalidade. Além disso, o baixo custo atual dos componentes de hardware, facilitou ainda mais a implementação de mecanismos de tolerância a falhas nesse contexto, sem estimular significativamente a pesquisa em novas opções. Entretanto, há aplicações que não são adequadamente satisfeitas com estas soluções, como é o caso de todas as que contam com a estrutura de hardware pré-definida e precisam elaborar mecanismos a nível de software de suporte. Em vista disso, é relativamente novo o interesse de pesquisa na área de desenvolvimento de tecnologia para software tolerante a falhas.

Hoje em dia, uma questão tecnológica de interesse crescente neste espectro de aplicações é justamente a tolerância a falhas em sistemas tempo real. Este interesse deve-se, em primeiro lugar ao fato de que tolerância a falhas em tempo real é fundamental para acentuar a confiabilidade dos sistemas de aplicação baseados em computação crítica segura. Além disso, os investimentos em pesquisa na área de tolerância a falhas se justificam pois o mercado de computação tempo real tem crescido constantemente e a tendência é que este crescimento seja acelerado nos próximos tempos [KIM 95].

Em decorrência, tornou-se importante um estudo aprofundado das aplicações envolvidas pela computação tempo real, visando a devida adequação das técnicas de tolerância a falhas a estes sistemas. É nesse contexto que se enquadra o trabalho aqui apresentado, abordando especificamente software para sistemas tempo real tolerantes a falhas.

1.1 Justificativa e Motivação

Sistemas tempo real exigem altos índices de confiabilidade por serem essencialmente críticos; ou seja, o custo das conseqüências de um defeito pode ser significativamente alto, ocasionando vários danos ou produzindo conseqüências catastróficas, tais como perdas econômicas e até de vidas humanas. Desta forma, torna-se inerente ao projeto dos sistemas tempo real a aplicação de técnicas de tolerância a falhas.

A pesquisa aqui desenvolvida é necessária e se justifica por buscar esclarecer aspectos sobre tolerância a falhas específicos para situações de tempo real, uma vez que nem todas as técnicas conhecidas podem ser indistintamente empregadas nesta extensa gama de aplicações envolvida. Visa-se, com isso, auxiliar projetistas de software a desenvolverem produtos de mais alta qualidade e que atendam especificações estritas.

A principal motivação para o desenvolvimento deste trabalho é a possibilidade de reutilizar técnicas e mecanismos de tolerância a falhas implementados em software, para aplicações com especificações temporais, assim como tem sido utilizado o hardware. Desta forma serão definidas as propriedades e a aplicabilidade destes recursos disponíveis à área de aplicação considerada: sistemas tempo real.

1.2 Objetivos

O objetivo principal deste trabalho é oferecer diretrizes para o projeto, a análise e o desenvolvimento de software para sistemas tempo real tolerantes a falhas construídos com base em hardware existente, ou seja, soluções baseadas em hardware convencional. Dessa forma, supõe-se que projetistas de sistemas tempo real devem ser capazes de construir sistemas confiáveis usando apenas tecnologia existente, o que resulta em sistemas mais simples e baratos, e sua atividade principal passa a ser o desenvolvimento de software adequado a estas aplicações.

Este software pode ser desenvolvido com maior facilidade se as ferramentas escolhidas possuírem o suporte desejado à implementação dos mecanismos necessários. Também se deseja que este software não seja a origem de falhas no sistema. Assim, dentre os passos iniciais está a definição dos requisitos básicos para o desenvolvimento de software desta categoria, a fim de posteriormente poder-se avaliar a aplicabilidade das técnicas de tolerância a falhas e o reuso de ferramentas e recursos pré-existentes a estes sistemas. Modos específicos de resposta a falhas também correspondem a um dos requisitos dos sistemas de tempo real. Entretanto, agregar técnicas e mecanismos que assegurem dada resposta não pode ser um ato isolado - as demais especificações funcionais do sistema precisam ser consideradas.

O software aqui referido será diferenciado nos níveis de: protocolos de comunicação, sistemas operacionais e linguagens de programação. Considera-se que há ações que se realizarão na camada de hardware do sistema, mas cujos resultados das falhas se propagarão aos níveis superiores; outras falhas serão geradas nestas camadas superiores (software). Conseqüentemente, deverão existir mecanismos para realização do processo de tolerância a falhas: detecção de erros, avaliação e confinamento dos danos, recuperação de erros e tratamento de falhas.

Assim como tem sido empregado hardware pré-existente para a construção de sistemas tempo real, considera-se como objetivo de projeto a utilização de software de suporte existente para o desenvolvimento destes sistemas. Mas a integração dos diversos componentes de software de suporte exige o conhecimento das propriedades e características de cada um - neste trabalho, o objetivo é analisar estes componentes frente às suas propriedades comportamentais diante da ocorrência de possíveis falhas no sistema, considerando-se a existência de restrições no tempo de resposta. Dessa forma, serão avaliadas, a cada nível de software, as facilidades por ele oferecidas para o desenvolvimento de software para sistemas tempo real tolerantes a falhas.

O estudo dos componentes e ferramentas de software analisadas a ser relatado é baseado em informações constantes em manuais ou artigos publicados - diante das restrições de tempo para realização da presente dissertação, não foi viável fazer experimentações práticas com o software. A forma de organização escolhida constitui-se em uma classificação ou taxonomia para ser usada como material de consulta em tempo de projeto. Conseqüentemente, esta taxonomia corresponde basicamente a uma compilação analítica dos conhecimentos atuais sobre o assunto proposto, que orienta os projetistas de sistemas tempo real na escolha do software a ser usado no desenvolvimento de produtos mais confiáveis e de maior qualidade. O processo de realização deste trabalho é esquematizado na figura 1.1.

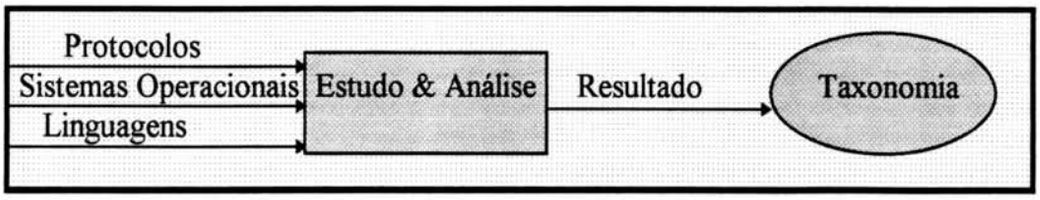

FIGURA 1.1 - ESQUEMA DOS OBJETIVOS DO TRABALHO

Convém esclarecer também quanto à área de aplicabilidade do presente trabalho. Os sistemas descritos de acordo com as características aqui apresentadas - com projeto de certa forma massificado - correspondem tipicamente a sistemas de controle de processos industriais, por exemplo, ou sistemas de transporte de superfície. Estas aplicações necessitam de segurança de funcionamento, traduzida principalmente por confiabilidade e disponibilidade, mas estes parâmetros não são considerados extremamente críticos. Por outro lado, sistemas tais como de controle de usinas nucleares, de controle espacial ou de transporte aéreo exigem todo um procedimento específico de projeto, inclusive de hardware, e não são alvo deste trabalho por esta razão.

1.3 Metodologia

As atividades de tolerância a falhas necessitam do apoio da linguagem de programação de implementação adotada no sistema de controle de aplicações, além de encontrar respaldo no seu sistema operacional e no hardware de suporte [LIS 93]. Assim, a metodologia de desenvolvimento deste trabalho baseia-se em uma organização dos componentes de software de um sistema em diferentes camadas: os protocolos de comunicação, o sistema operacional e, por fim, as linguagens de programação.

Para produzir software confiável em aplicações reais, além de empregar técnicas básicas de tolerância a falhas sobre determinados módulos do sistema, é indispensável determinar o suporte necessário nos demais níveis, para viabilizar a implementação das técnicas definidas. Desta forma, este trabalho apresenta um estudo diferenciado, visando ressaltar as características e recursos necessários em protocolos de comunicação, em sistemas operacionais e em linguagens de programação conhecidos, de intuito acadêmico ou disponíveis no mercado. Cada nível, é preliminarmente analisado para definir o que se espera desse para a sua aplicação em sistemas tempo real tolerantes a falhas. Após, segue-se a apresentação dos exemplos de implementação e uma análise comparativa de cada um dos exemplos apresentados, na forma de uma taxonomia.

Uma taxonomia é uma ferramenta de classificação que reconhece exemplos diferentes de um mesmo tipo genérico a ser descrito [MAR 91]. A taxonomia apresentada aqui será diferenciada para cada um dos níveis em questão, apresentando as características desejáveis para sua utilização no ambiente proposto e a forma como cada uma das alternativas estudadas implementa estas características.

Por fim, são estudados alguns casos de sistemas tempo real relativamente críticos. Visto que sistemas tempo real apresentam características diferenciadas e bem-definidas, de acordo com a aplicação, não seria possível obter-se um resultado ou

indicação única do suporte de comunicação, sistema operacional e linguagem de programação. Assim, optou-se por estudar alguns casos específicos, para cada um dos quais buscou-se identificar a melhor solução possível. Dessa forma, com base na definição, caracterização da aplicação e na taxonomia desenvolvida, são apresentadas sugestões ou opções de protocolos de comunicação, sistemas operacionais e linguagens de programação mais adequados para serem utilizados no contexto apresentado.

1.4 Trabalhos Relacionados

Tem-se conhecimento de *frameworks* e ambientes de desenvolvimento existentes, que apresentam propostas relacionadas aos objetivos deste trabalho. Exemplos apresentados na literatura [HEC 76, AND 83 e PUR 91] serão brevemente descritos nesta seção.

Hecht [HEC 76] fez uma revisão do que existia a respeito de tolerância a falhas em software para sistemas tempo real, até a época em que foi publicado o trabalho. Porém, ao contrário do que é proposto aqui, apenas foram analisadas as técnicas básicas, como os blocos de recuperação, para implementar tolerância a falhas em software para sistemas tempo real. Além disso, é apresentada uma técnica para análise de confiabilidade do software resultante desta implementação com tolerância a falhas. Diante da constatação de que, até o momento da publicação do trabalho, tinham sido encontradas apenas implementações fragmentadas de software tolerante a falhas em aplicações práticas tempo real, foi proposta uma abordagem estrutural de desenvolvimento, envolvendo todos os níveis do sistema, para a implementação de tolerância a falhas. Esta abordagem foi inclusive demonstrada e avaliada, como forma de provar que funcionava. Por outro lado, os fatores econômicos que envolvem a implementação de tolerância a falhas em software também foram analisados. Entretanto, como aquele artigo já tem duas décadas, ele não aborda todas as técnicas e ferramentas de software atuais.

Anderson e Knight [AND 83] propuseram desenvolver um *framework* que permitisse a construção dos sistemas da categoria aqui tratada, já prevendo que estes deveriam continuar a oferecer as respostas adequadas em tempo real, apesar de circunstâncias onde falhas no software normalmente causariam uma perda de serviço. Foi garantido por eles que esta abordagem poderia ser aplicada de forma direta a muitos sistemas tempo real, pois aproveitava a própria estrutura do sistema para simplificar a recuperação de erros e introduzia um esquema de classificação de erros. Desta forma, foram definidas respostas para cada tipo de erro previsto na classificação, permitindo que o serviço fosse mantido em funcionamento mesmo quando esta falha ou erro previsto estivesse presente no ambiente.

No mais recente trabalho, Purtilo e Jalote [PUR 91] apresentam um ambiente genérico para o desenvolvimento de software tolerante a falhas, sem preocupação com as características específicas das aplicações tempo real, que é o enfoque abordado no presente trabalho. O ambiente permite que seja implementada tolerância a falhas de forma diferenciada em módulos distintos do programa, utilizando-se programação diversitária: programação n-versões (PNV) ou blocos de recuperação nos módulos, sendo o usuário livre para empregar esta técnica em diferentes partes do sistema. Assim, da forma como o ambiente opera, é permitido que um sistema grande suporte tolerância

a falhas apenas nos módulos mais críticos. Nesse caso, apenas tais módulos precisam ser replicados e não todo o sistema, o que, às vezes, poderia ser inviável, tanto do ponto de vista econômico quanto da complexidade. O ambiente permite a distribuição do software de aplicação em arquiteturas heterogêneas, sendo a transformação de dados entre as diversas máquinas e várias linguagens de programação realizada de forma transparente à aplicação. Apesar do ambiente não oferecer nenhum recurso específico para tempo real, a princípio poder-se-ia considerar sua utilização no desenvolvimento de sistemas tempo real. Algumas de suas características permitem esta dedução: a possibilidade de implementar um programa em qualquer linguagem de programação e a possibilidade de dividir um programa mais extenso em subprogramas (módulos), diferenciando os críticos dos demais. Mas como este trabalho [PUR 91] é para ambientes genéricos, enquanto que a pesquisa aqui relatada é dirigida a sistemas tempo real, os resultados de ambos são diferentes.

Logo, os trabalhos aqui comentados têm em comum o fato de preverem o desenvolvimento de software tolerante a falhas, mas não atingem os objetivos propostos neste. Entretanto serviram como base para que se tenha conhecimento sobre o que existe de concreto para a implementação de sistemas desta categoria específica. Além disso, no presente trabalho há informações mais atualizadas tanto sobre técnicas de tolerância a falhas como dos conceitos dos sistemas tempo real, o que produz resultados novos sobre a questão.

1.5 Estrutura do Trabalho

O presente trabalho encontra-se estruturado em oito capítulos, cujo conteúdo é basicamente o seguinte:

- Capítulo 2 - Sistemas Tempo real: aqui é feita uma contextualização deste trabalho, onde sistemas tempo real são definidos, caracterizados, classificados e exemplificados.
- Capítulo 3 - Tolerância a Falhas em Software: neste capítulo é definida a necessidade de aplicação das técnicas de tolerância a falhas em software aos sistemas tempo real e são apresentadas e analisadas as principais técnicas conhecidas para isto.
- Capítulo 4 - Comunicação: este capítulo define alguns conceitos, pressupostos e requisitos básicos no que se refere ao sistema de comunicação das aplicações tempo real, abordando, analisando e comparando os protocolos de comunicação. É apresentada ainda uma adaptação de um mecanismo de comunicação do sistema operacional para ser utilizado em sistemas tempo real.
- Capítulo 5 - Sistemas Operacionais: este capítulo apresenta alguns conceitos básicos e estabelece os principais requisitos de um sistema operacional tolerante a falhas para aplicações tempo real. Tomando por base os preceitos anteriores, são apresentados e comparados alguns dos principais sistemas conhecidos.

- Capítulo 6 - Linguagens de Programação: este capítulo apresenta alguns conceitos relacionados ao tema e estabelece os principais requisitos e recursos oferecidos por uma linguagem de programação tolerante a falhas para aplicações tempo real. Após são apresentadas e comparadas algumas das principais linguagens de programação conhecidas.
- Capítulo 7 - Estudos de Caso: neste capítulo são definidos e apresentados alguns casos práticos de sistemas tempo real. Em seguida, cada um deles é analisado de acordo com as taxonomias estabelecidas, visando definir o ambiente de desenvolvimento que melhor se adequa às suas características.
- Capítulo 8 - Conclusão: por fim, este capítulo apresenta as conclusões gerais, uma análise do que foi obtido com a realização deste trabalho, a sugestão de trabalhos futuros possíveis e extensões a serem desenvolvidas nesta área.

As comparações mencionadas nos capítulos 4, 5 e 6 são dirigidas às características importantes sob o ponto de vista de tolerância a falhas e da execução tempo real do sistema.

2 Sistemas Tempo Real

A partir da necessidade de um controle mais rígido sobre certas aplicações, foram desenvolvidos sistemas computacionais capazes de realizar, de forma mais adequada, este tipo de controle: os sistemas tempo real. Este capítulo faz uma breve apresentação desta categoria de sistemas, definindo-os, caracterizando-os, classificando-os e exemplificando-os.

2.1 Definição

Um sistema tempo real pode ser definido como aquele que muda seu estado computacional em função do tempo (no caso, de acordo com o relógio real e não com o interno ao computador - *clock*) [KOP 93]. De outra forma, segundo Murata [MUR 87], sistemas tempo real podem ser definidos como sendo aqueles cujo bom funcionamento é avaliado, não somente pelo resultado lógico computado, mas também pelo tempo decorrido ou pelo instante em que tal resultado tenha sido produzido.

Estes sistemas são ditos de "tempo real" por que as tarefas das aplicações que eles controlam devem ocorrer em um determinado instante, de uma maneira independente do sistema controlador. Assim, o relógio que controla as ações do sistema tempo real não é o relógio interno do computador, mas sim o relógio que controla a dinâmica dos estados do processo no ambiente em que o sistema está inserido [SAN 91]. Desta forma, os sistemas tempo real são tipicamente constituídos por [STA 88]:

- Processo controlado: que pode ser, por exemplo, o chão[1] de uma fábrica automatizada, com os seus robôs, estações de montagem e demais itens necessários para o seu funcionamento.
- Processo controlador: que, para o exemplo citado, corresponde ao computador e à interface humana, que gerenciam e coordenam as atividades ocorridas no chão da fábrica.

O processo controlado pode ser considerado como o ambiente com o qual o computador (processo controlador) interage. Nestes sistemas, portanto, o computador trabalha em paralelo com o sistema ou processo em questão, controlando-o, monitorando-o e acionando-o. Com isto, pode-se concluir que uma das necessidades fundamentais dos sistemas tempo real é a de se manterem constantemente sincronizados com o processo controlado, ou seja, a atuação do sistema deve corresponder à dinâmica dos estados do processo.

A maioria dos sistemas tempo real possui um propósito específico e complexo, além de requerer um alto grau de dependabilidade, traduzida por confiabilidade, segurança, integridade e disponibilidade, exigindo assim tolerância a falhas. Por outro

---

[1] Este termo tem sido utilizado em informática industrial referindo-se aos equipamentos de controle industrial de uma fábrica.

lado, uma das maiores dificuldades encontradas no desenvolvimento de tais sistemas é que eles normalmente são caros, pois exigem um profundo conhecimento sobre a aplicação onde atuam. Além disso, são difíceis de testar, pois devem ser verificados diretamente ou através de simulação, que é uma tarefa bastante cara e complexa; e, ainda, pequenas mudanças no sistema tornam necessária a realização de uma nova rodada de testes [AMA 96].

2.2 Arquitetura

Uma arquitetura é uma forma de abstração que contém a estrutura e o comportamento de todos os componentes do sistema considerado [SEL 96]. Assim, a arquitetura de um sistema tempo real típico pode ser descrita através da figura 2.1, explicada no texto que segue.

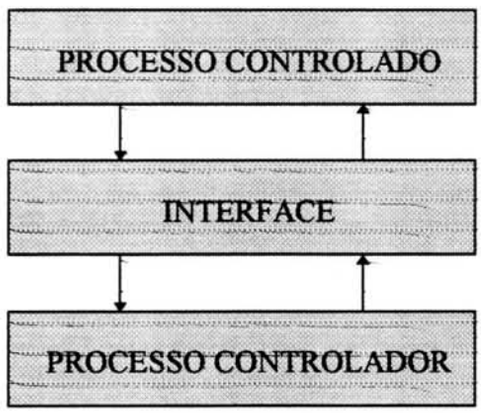

FIGURA 2.1 - ARQUITETURA DE UM SISTEMA TEMPO REAL

Visualizando-se a arquitetura de um sistema tempo real apresentada na figura 2.1, é possível caracterizar cada uma de suas partes componentes como segue:

- Processo Controlado: corresponde ao mundo real que está sendo controlado, como os seguintes exemplos: o chão de uma fábrica, os comandos e controles de um avião, os aparelhos de monitoração de um paciente na UTI (Unidade de Terapia Intensiva) em um hospital, entre outros.
- Interface: refere-se ao uso de dispositivos dependentes do processo, onde vários tipos de dispositivos, tais como sensores e atuadores, devem ser conectados ao sistema tempo real de acordo com a aplicação. São estes dispositivos e interfaces que tornam possível a atuação dos sistemas computacionais em tempo real, pois é através deles que o computador é conectado ao processo a ser controlado.
- Processo Controlador: corresponde ao sistema computacional que está controlando o mundo real, ou seja, o programa de controle. O projeto de um processo controlador pode ser dividido em níveis correspondentes às camadas de software que são utilizados no seu desenvolvimento: os protocolos de comunicação, o sistema operacional (e seus mecanismos de comunicação), as linguagens de programação utilizadas e, por fim, as aplicações. Estas camadas

trocam informações e interagem entre si. Dessa forma, o processo controlador corresponde a uma abstração da estrutura apresentada na figura 2.2.

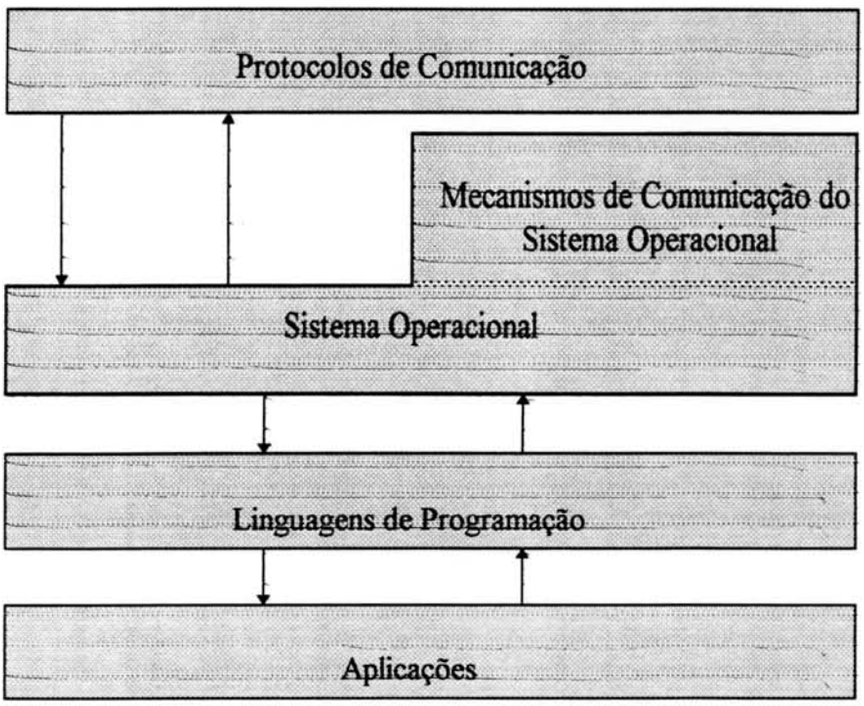

FIGURA 2.2 - ESTRUTURA DO PROCESSO CONTROLADOR

Assim, a arquitetura expandida de um sistema tempo real típico, onde são apresentadas as camadas componentes do software utilizado no projeto do processo controlador, é mostrada através da figura 2.3.

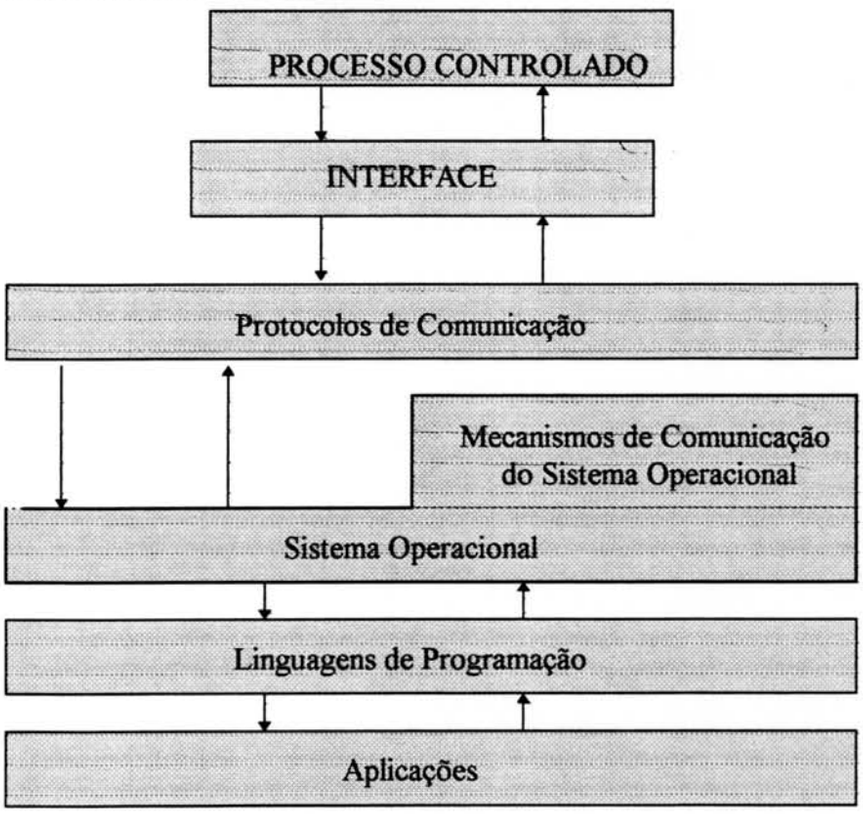

FIGURA 2.3 - ARQUITETURA EXPANDIDA DE UM SISTEMA TEMPO REAL

2.3 Paradigmas de Projeto

Quanto à forma de reação às modificações externas (entradas), existem dois paradigmas distintos utilizados no projeto de sistemas tempo real, cuja opção por usar um ou outro interfere também na definição do sistema de comunicação utilizado. Estes paradigmas são [KOP 94]:

- Arquitetura disparada por eventos: nesse caso, todas as atividades - ativação de tarefas, comunicação, entre outras - são iniciadas como uma conseqüência de eventos externos, ou seja, alguma mudança significativa do estado do sistema. Este paradigma também é conhecido como dirigido a interrupções. Um exemplo de sistema implementado com base nesse paradigma, chama-se DELTA-4, proposto por Chereque, conforme citado por Veríssimo [VER 93b].
- Arquitetura disparada por tempo: nesse caso, todas as atividades são dirigidas pela progressão do tempo global. Dessa forma, todas as tarefas ou ações de comunicação do sistema são periódicas e ocorrem em momentos pré-determinados. Este paradigma é menos flexível que o anterior, porém é mais fácil de analisar e testar. Um exemplo de sistema implementado com base nesse paradigma, chama-se MARS (*Maintanable Real-time System*), apresentado por Kopetz e citado por Veríssimo [VER 93b].

Normalmente, os sistemas cuja arquitetura é disparada por eventos não são previsíveis, ou seja, não é possível saber antecipadamente quando serão realizadas determinadas ações. Enquanto que os sistemas disparados por tempo permitem uma maior previsibilidade. Por outro lado nas arquiteturas disparadas por tempo, é necessário que haja um sincronismo entre os nodos (se o sistema for distribuído), de acordo com o relógio tempo real do sistema. Já nos sistemas disparados por eventos este sincronismo é desnecessário.

Para ilustrar a diferença entre os paradigmas existentes, Kopetz e Grünsteidl [KOP 94], apresentam um sistema de controle de elevador em um edifício que possui vários elevadores paralelos. Na implementação de uma arquitetura disparada por eventos, cada aperto no botão de chamada do elevador por um cliente sinaliza o sistema de controle do elevador através de uma interrupção. O sistema então imediatamente decide qual elevador deve atender à requisição. Por outro lado, a implementação de uma arquitetura disparada por tempo percorre todos os botões de chamada do elevador periodicamente, por exemplo, a cada segundo. Dessa forma, ela trata todos os botões apertados no último segundo de forma igual. Baseado no resultado desta pesquisa, o sistema de controle de elevador gera um novo escalonamento do elevador a cada segundo.

Assim, em uma situação de baixa carga, onde um botão apenas é pressionado a cada intervalo de poucos segundos, a implementação disparada por eventos atende mais adequadamente às solicitações dos clientes. Já quando se trata de uma situação de alta carga, onde vários botões são pressionados durante um segundo, a implementação disparada por tempo apresenta um *overhead* organizacional menor e um serviço melhor e mais previsível. Este *overhead* organizacional deve-se às chamadas a serviços de interrupção e ao escalonamento.

Na prática, a maioria sistemas tempo real utilizados apresentam uma arquitetura híbrida, onde algumas atividades são periódicas, ou seja, disparadas por tempo; enquanto que outras são não-periódicas, isto é, disparadas por eventos.

2.4 Classificação Quanto aos Requisitos de Tempo

Na literatura, os sistemas tempo real são classificados, com algumas variações, de acordo com o atendimento aos requisitos de tempo. Mas de uma forma geral, pode-se classificá-los, usando a nomenclatura apresentada por [SAN 91], como:

- Tempo real brando (*soft*): quando a aplicação pode tolerar pequenos atrasos com relação ao instante em que devem ser produzidos os resultados. Ou seja, as conseqüências de uma falha temporal não implicam em perdas humanas ou põem em risco a vida. Exemplos desta classe são sistemas de transações bancárias, reservas de passagens e teleprocessamento.
- Tempo real crítico (*hard*): quando o fato de não cumprirem com precisão as restrições temporais pode levar a conseqüências catastróficas no ambiente em que os sistemas estão inseridos. Nesta classe encontram-se aplicações onde o custo de uma falha temporal pode ser muito alto, como no caso de controle industrial, computador de bordo de aviões e monitoração de pacientes em UTI.

Na literatura mais recente [KOP 93], é apresentado ainda um subgrupo dos sistemas tempo real críticos, que apresentam um comportamento denominado de "melhor esforço". Estes sistemas são projetados especialmente para atuarem em situações onde é fundamental que o controle seja completo, irrestrito e confiável, visando garantir a disponibilidade do sistema, mesmo quando é necessária uma certa adaptabilidade às mudanças do ambiente (pois os parâmetros encontram-se fora do previsto), em situações como por exemplo no controle de tráfego aéreo. Neste caso, mesmo quando as taxas de uso de um determinado aeroporto são ultrapassadas, o sistema deve continuar sendo capaz de monitorar a situação e manter o controle. Normalmente, estes sistemas são específicos para cada aplicação e diferenciados de acordo com a sua área de atuação.

Existem ainda os sistemas tempo real denominados de longo período de duração, quando os sistemas controlados devem ter a capacidade de ficar sem manutenção por um tempo considerável, sem que isto gere qualquer tipo de prejuízo. São considerados exemplos desta classe os sistemas de controle empregados em viagens espaciais, exploração submarina e satélites.

2.5 Processos Tempo Real

Na maioria dos sistemas tempo real, atividades que devem ocorrer em um determinado tempo coexistem com aquelas que não possuem nenhuma restrição. Stankovic e Ramamritham [STA 88b] generalizam todas atividades sob a denominação de tarefas. Entretanto uma tarefa com restrições temporais é considerada uma tarefa

tempo real. Na literatura, tarefas tempo real são algumas vezes chamadas de tarefas críticas no tempo.

Segundo Fricks [FRI 89], em ambientes tempo real estas tarefas críticas à aplicação, cujo processamento deve ser executado atendendo a restrições de tempo rígidas, finitas e previamente especificadas, são consideradas como processos críticos no tempo (PCT). A importância deste conceito reside no fato de que uma coleção desses processos pode representar a totalidade de processamento de um sistema tempo real particular. Os parâmetros associados a um PCT, cuja arquitetura de projeto seja disparada por eventos, correspondem aos seguintes:

- Instante de Ativação (*arrival time* ou *start time*): corresponde ao instante da ocorrência do evento associado ao processo, que vai disparar a sua ativação. Nenhum processo pode ser ativado antes da ocorrência do seu evento associado.
- Prazo Limite: é o instante de tempo máximo para o atendimento ao evento ter sido concluído. Juntamente com o instante de ativação delimita um período de tempo fixo no qual o serviço executado pelo processo deve ter sido garantidamente concluído, denominado de Janela Tempo real (*frame size*), que corresponde à quantidade máxima de tempo permitida ao processo em execução.
- Tempo de Resposta (*response time*) ou de Latência (*latency*): refere-se ao período de tempo compreendido entre a ativação e o fim do atendimento efetivo ao evento. Em sistemas tempo real, principalmente os críticos, deve-se garantir que os tempos de resposta atendam às restrições de tempo.
- Tempo de Processamento (*run time* ou *quantum size*): corresponde ao tempo gasto pelo processador para executar o processo sem interrupções, ou seja, representa a quantidade de serviços da UCP (Unidade Central de Processamento) utilizados em segundos. É dependente do ambiente (por exemplo da velocidade da UCP).
- Período de Iteração (*frame time*): é o período de tempo compreendido entre as sucessivas ocorrências do evento associado a um processo (PCT). Na prática usualmente os PCTs são periódicos e seus prazos limites são iguais aos respectivos períodos de iteração, os quais são constantes. Já os PCTs esporádicos típicos apresentam um prazo limite menor do que o período de iteração, que é variável.
- Taxa de Requisição (*request rate*): refere-se ao inverso do período de iteração de um processo.

Normalmente, os valores destes parâmetros podem ser determinados pelos engenheiros do sistema, uma vez que estes valores correspondem a características dos dispositivos de controle ou aos requisitos do sistema tempo real.

Na figura 2.4, é esquematizado um processo crítico no tempo em uma arquitetura disparada por eventos.

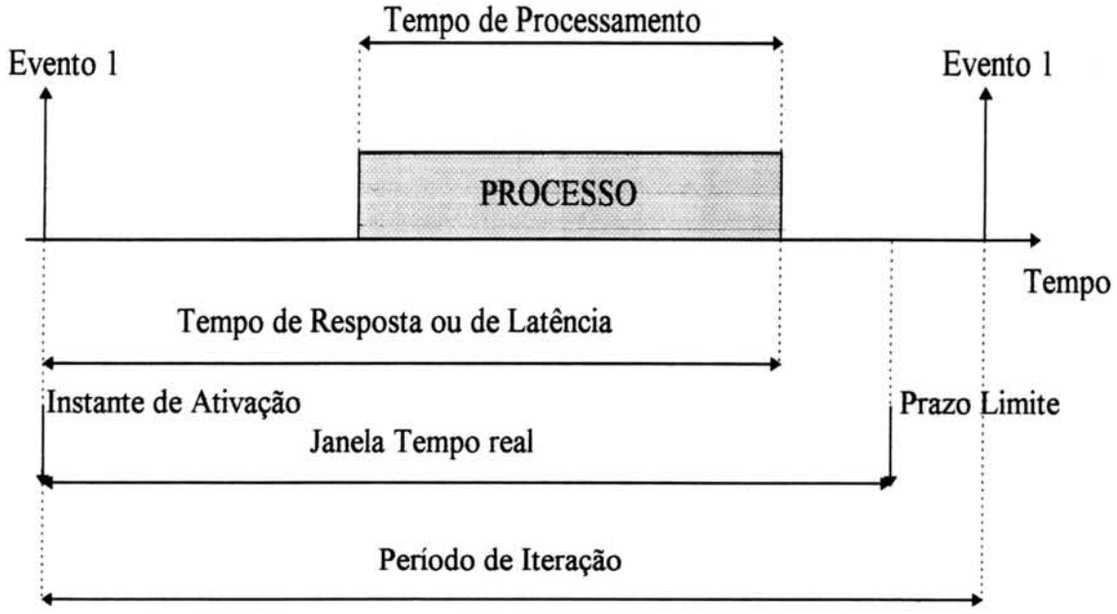

FIGURA 2.4 - PROCESSO CRÍTICO NO TEMPO

2.6 Tipos de Aplicações

Sistemas tempo real são normalmente considerados sistemas de controle ou comando, que englobam desde áreas de aplicações simples até áreas bastante complexas. Em um extremo deste amplo espectro de aplicações em que os sistemas tempo real podem ser inseridos, encontram-se dispositivos simples como, por exemplo, o controlador de uma máquina de lavar roupas. No outro extremo há sistemas bastante complexos e críticos como, por exemplo, o sistema de controle de uma usina nuclear.

Historicamente, a automação industrial foi o primeiro campo de atuação dos sistemas computacionais de tempo real [KOP 93]. Entretanto, este campo expandiu-se consideravelmente e hoje em dia outros exemplos conhecidos de aplicações de sistemas tempo real podem ser encontrados nas mais diversas áreas, tais como: transações bancárias, teleprocessamento, reservas de passagens aéreas ou rodoviárias, experimentos em laboratórios, testes de motores de automóveis, aviação (tráfego aéreo e controle de vôo), viagens espaciais, exploração submarina, robótica, defesa militar, aparatos médicos em casos de tratamentos intensivos (UTIs) e de urgência, apenas para citar algumas entre tantas existentes.

As aplicações de baixo risco como o controle de uma máquina de lavar roupas, não justificam gastos altos de projeto e implementação. Normalmente, há poucos investimentos em itens de segurança e elas comportam pouco ou nenhum recurso de tolerância a falhas.

Por outro lado, as aplicações de altíssimo risco, como usinas nucleares, aviação ou viagens espaciais, não comportam soluções genéricas exigindo um projeto cuidadoso e específico para elas, através do qual todas as situações de falha possíveis são previstas e eliminadas ou tratadas.

Os sistemas intermediários, entre os dois recém caracterizados exigem soluções que podem ser tratadas de forma menos personalizada, mas ainda segura. Para estes

pode-se usar as soluções existentes com as devidas correções ou modificações requeridas para cada caso.

Portanto, as aplicações de tempo real não podem ser caracterizadas e tratadas como um caso uniforme, mas precisam ser sub-agrupadas de acordo com suas necessidades. Além destes, ainda aparecerão outros fatores de funcionamento na caracterização das aplicações.

Dessa forma, fica claro que este trabalho aborda especificamente os sistemas de controle intermediários, não levando em consideração os sistemas extremamente simples, nem os considerados mais complexos e críticos.

2.7 Características Desejáveis

Há um relativo consenso dentre os autores consagrados, no sentido que sistemas tempo real diferem dos sistemas tradicionais pelo fato de terem restrições de tempo e tratarem, em muitos casos, com situações críticas. Isto implica que, qualquer tipo de falha, inclusive falhas temporais, pode causar conseqüências catastróficas. Então, ao contrário da maioria dos sistemas onde há uma separação entre correção e desempenho, uma das características dos sistemas tempo real é que correção e desempenho estão fortemente relacionados.

Nesse sentido, através de algumas importantes características computacionais pode-se verificar as principais diferenças entre os sistemas tempo real e os sistemas convencionais de processamento de dados. Diversos autores [MUR 87, SAN 91 e STA 88b] apresentam as seguintes características como sendo as desejáveis em um sistema tempo real:

- Garantia no cumprimento das especificações referentes ao tempo de resposta: um sistema tempo real deve detectar modificações no estado do processo controlado, para poder executar rapidamente uma ação de controle sobre ele, mesmo que estas mudanças sejam assíncronas. Isto deve ser feito para que não haja uma ação retardada sobre o processo ou que dados sejam perdidos. Desta forma, um tempo máximo de resposta deve ser estabelecido e obedecido.
- Velocidade de entrada e saída: as operações de E/S (Entrada e Saída) de um sistema tempo real devem ser rápidas o suficiente para que o sistema não perca dados e para que possa atuar dentro do tempo adequado sobre o processo controlado. Entretanto, existe uma variação significativa no tempo de resposta entre diferentes processos: em aplicações aeroespaciais, por exemplo, este tempo é da ordem de grandeza de nanossegundos, enquanto que, em algumas aplicações industriais, a ordem de grandeza deste tempo pode ser de horas ou dias. Do mesmo modo, em relação à velocidade de E/S de um sistema, pode haver uma variação desde uma informação por segundo até milhões de informações por segundo, dependendo da aplicação.
- Definição do atendimento às ações do processo: o software do sistema tempo real deve suportar meios para que o programador defina o comportamento do sistema em relação ao processo que será controlado ou monitorado por ele. Por exemplo, a sincronização necessária com o processo, que ocorre de

acordo com as mudanças de estado, pode ser atingida utilizando-se conceitos como interrupção e multitarefa. Entretanto, fica a cargo do programador inseri-los adequadamente no código do sistema, definindo assim o seu comportamento frente às mudanças de estado do processo.

- Suporte à execução em tempo real: ambientes de programação convencionais não são apropriados para executar aplicações de tempo real. Assim, no desenvolvimento de software para sistemas tempo real, devem ser utilizadas linguagens apropriadas (linguagem de programação tempo real), sistemas operacionais que suportem estas aplicações (sistema operacional tempo real) e protocolos de comunicação com requisitos específicos (protocolos de comunicação tempo real).

Um sistema tempo real é composto basicamente por tarefas que interagem entre si e por uma parte de código co-residente responsável por controlar estas tarefas: o sistema operacional [RIP 93]. Nesse sentido, além dessas características apresentadas, outras inerentes aos sistemas tempo real são, por exemplo a concorrência existente entre as tarefas do sistema, a comunicação necessária, a alocação dos recursos e a prioridade das tarefas, de acordo com o quão críticas elas são; estas características devem ser tratadas pelo sistema operacional, conforme será visto no capítulo 5 desta dissertação.

Finalmente, um sistema tempo real deve ser inevitavelmente rápido e previsível, ou seja, quando uma tarefa for ativada deve ser possível determinar com certeza o tempo necessário para o término de sua execução. Por outro lado, confiabilidade pode ser considerada como um pré-requisito para os sistemas tempo real, que não podem existir se seus componentes não forem suficientemente confiáveis para garantir a execução correta de cada tarefa no tempo esperado.

2.8 Falsas Idéias

Segundo alguns autores [KUR 94, STA 88 e STA 88b], os sistemas tempo real são alvo de diversas idéias falsas a seu respeito, que acabam por influenciar na sua compreensão e no desenvolvimento de pesquisas nesta área. Nesta seção tais idéias serão apresentadas, juntamente com a discussão que existe em torno delas, a fim de que a próxima geração não seja influenciada negativamente por elas. Algumas das principais falsas idéias a respeito de sistemas tempo real são as seguintes [STA 88]:

- Não há ciência no projeto de sistemas tempo real: é realmente verdade que o estado da arte em sistemas tempo real é predominantemente *ad-hoc*. No entanto, isto não impede que seja possível uma abordagem científica para estes sistemas. Prova disso é que boas soluções científicas, das mais diversas áreas, desenvolveram-se na tentativa de resolver problemas práticos.

- Os avanços em hardware de supercomputadores devem levar em consideração os requisitos de tempo real: o avanço no projeto de supercomputadores normalmente explora o uso de processadores paralelos para melhorar o *throughput* do sistema, mas isto não significa que as restrições de tempo sejam automaticamente satisfeitas. Assim, deve haver esta preocupação adicional para que estas máquinas possam abrigar sistemas tempo real. A história da computação mostra que a demanda por mais poder de computação tem

sempre ultrapassado o limite do que é oferecido. Neste sentido, é de se esperar que a disponibilidade de mais poder computacional aumentará a área de abrangência de sistemas tempo real, requerendo mais funcionalidade e agravando assim os problemas de tempo.

- Computação tempo real é equivalente à computação rápida: o objetivo da computação rápida é minimizar o tempo médio de resposta de um dado conjunto de tarefas, enquanto que o objetivo da computação tempo real é satisfazer a restrição individual de tempo de cada uma das tarefas. Ao invés de rapidez, a propriedade mais importante que um sistema tempo real deve ter é a previsibilidade, ou seja, seu comportamento funcional e temporal deve ser tão determinístico quanto for necessário para satisfazer as especificações do sistema. Computação rápida é útil na execução da especificação temporal estrita (tempo real crítico), mas ela sozinha não garante previsibilidade. Assim, previsibilidade, e não velocidade, é o principal objetivo no projeto de sistemas tempo real.

- Pesquisa em sistemas tempo real é engenharia de desempenho: um importante componente da pesquisa de sistemas tempo real é a investigação de efetivas estratégias para alocação de recursos de modo que satisfaçam estritamente os requisitos do comportamento temporal. Sob este prisma, os aspectos de pesquisa em sistemas tempo real podem realmente ser considerados como engenharia de desempenho. Entretanto, o problema mais grave na pesquisa tempo real talvez seja investigar se as regras definidas para o tempo podem ser traduzidas em um mecanismo de sincronização, o que não pode ser visto como engenharia de desempenho.

- Os problemas no projeto de sistemas tempo real têm sido resolvidos em outras áreas de ciências da computação ou pesquisa operacional: pesquisas em sistemas tempo real devem, certamente, tentar explorar as técnicas de solução já desenvolvidas em áreas de pesquisa estabilizadas. No entanto, há problemas únicos em sistemas tempo real e que não foram resolvidos em qualquer outra área. Por exemplo, engenharia de desempenho em ciência da computação tem estado mais interessada em analisar os valores médios dos parâmetros de desempenho, enquanto que uma consideração importante no projeto de sistemas tempo real é determinar se alguns prazos poderão ser cumpridos ou não.

- Não é significativo falar sobre garantia de desempenho tempo real, porque não se pode garantir que o hardware não falhará e que o software é livre de erros ou que as atuais condições de operação não vão violar os limites especificados no projeto: pode-se minimizar a probabilidade de falhas nos sistemas. No projeto de sistemas tempo real, deve-se tentar alocar criteriosamente recursos para garantir que qualquer restrição crítica de tempo possa ser satisfeita com os recursos disponíveis, assumindo que o hardware e o software operam corretamente e que o ambiente externo não exige do sistema nada além do que este foi projetado para manipular. Certamente, não há como garantir algo que esteja fora de controle, mas dentro do possível, deve-se garantir o correto funcionamento, utilizando-se das técnicas de tolerância a falhas, como será visto no capítulo 3.

- Sistemas tempo real funcionam em um ambiente estático: dependendo do modo de operação, um sistema tempo real pode ter que satisfazer diferentes conjuntos de restrições de tempo. Assim, um tópico importante na pesquisa de sistemas tempo real é o projeto de escalonadores hierárquicos para tomar decisões de alocação dos recursos para diferentes granularidades de tempo. Há alguns anos, os sistemas tempo real têm assumido um caráter reconfigurável e, por isso, não devem funcionar em um ambiente estático.

2.9 Próxima Geração de Sistemas Tempo Real

O principal requisito para o desenvolvimento da próxima geração de sistemas tempo real é que haja uma metodologia de desenvolvimento, na qual sejam observadas as características de tempo, confiabilidade e funcionalidade. Dessa forma, o seu comportamento funcional e confiável deve ser determinado em função da capacidade do sistema prover serviços ininterruptos e dentro do prazo previsto aos clientes, possivelmente utilizando-se de técnicas de tolerância a falhas [STA 88b].

Os sistemas tempo real, de acordo com Musliner [MUS 95], são impulsionados para a próxima geração pelos avanços emergentes na área de computação em geral, mas principalmente nos componentes de hardware; e pela necessidade de utilização de aspectos de inteligência artificial. No entanto, estas forças intensificam as dificuldades científicas e os problemas de engenharia na construção de sistemas tempo real, pois adicionam entidades complexas a serem integradas nas aplicações atuais e futuras. Por exemplo, a tecnologia do hardware (e do software) tornou a computação distribuída e multiprocessada uma realidade, e brevemente, os avanços nesta área levarão ao surgimento de redes de multiprocessadores. Mas para que isso seja possível, devem ser desenvolvidos em uma escala crescente trabalhos básicos ou científicos que considerem os requisitos de tempo em aplicações tempo real tanto no projeto quanto na verificação do sistema, visando o acompanhamento destas tendências.

Pode-se prever que a próxima geração atuará em áreas similares às atuais, provavelmente expandindo-as um pouco. No entanto, os sistemas deverão ser mais complexos, englobando características que seguem as tendências nos sistemas de computação em geral, como: distribuição, inteligência artificial e adaptabilidade. Mas principalmente, a próxima geração deve ser projetada para ser dinâmica e flexível. Exemplos de aplicações futuras que poderão ser controladas por estes sistemas de tempo real são sistemas de manufatura inteligentes e estações espaciais.

Quanto aos sistemas de inteligência artificial, visto que estes possuem uma grande adaptabilidade e complexidade, tornam impossível o pré-cálculo de todas as possíveis combinações das tarefas que devem ocorrer durante a execução do processo controlado. Este fato impede o uso das normas de escalonamento estático comuns nos sistemas tempo real atuais. Com este objetivo, são necessárias novas pesquisas para o escalonamento das tarefas nos sistemas tempo real, que levem em consideração o aspecto dinâmico obtido através do uso de inteligência artificial. Segundo Musliner [MUS 95], este estágio de desenvolvimento já está em andamento, com a inteligência artificial caminhando em direção a domínios mais realísticos de aplicação, que requerem respostas em tempo real, ao mesmo tempo que os sistemas tempo real estão caminhando em direção a aplicações mais complexas, que requerem comportamento inteligente. O

ponto de encontro das duas áreas deu origem a um novo sub-campo de pesquisa, denominado como "controle tempo real inteligente" ou "inteligência artificial tempo real", definido por um interesse comum entre pesquisadores de ambas as áreas.

Por outro lado, Kim [KIM 95] afirma que há uma idéia de evolução entre a comunidade científica de ciência da computação, a qual sugere que a próxima geração da computação tempo real possa seguir o estilo de projeto geral (não tempo real), adicionado ao estilo de projeto que envolve restrições de tempo. Isto significa que as tecnologias compatíveis com a próxima geração de computação tempo real devem ser aplicáveis a um grande espectro de sistemas de computação, cobrindo desde sistemas não-tempo real até sistemas tempo real críticos. Por isso, a próxima geração de sistemas tempo real requer um método poderoso de estruturação de sistemas que seja aplicável a todos os tipos de computação (tempo real ou não). Ainda segundo ele, tais métodos de estruturação devem ser preferencialmente orientados a objetos, considerando os benefícios da modularidade, generalidade e abstração naturais desta abordagem. Um exemplo de método de estruturação de sistemas orientados a objetos poderoso é o modelo RTO.k (*Real-Time Object*), apresentado em [KIM 94].

Finalmente, sistemas de computação têm sido largamente utilizados no controle de processos em tempo real dos mais variados tipos de aplicações. Observa-se que as características computacionais destes processos controladores são fortemente influenciadas pelas características do processo controlado. Assim, para o desenvolvimento de sistemas tempo real é preciso ter um profundo conhecimento sobre a aplicação em que este será inserido. Este fato pode vir a ser um fator que atrase o seu desenvolvimento e a sua expansão, mas cada vez mais, tais sistemas podem e devem ser aplicados a um grande número de situações. No entanto, apesar de haver uma grande tendência no crescimento do uso e da área de abrangência dos sistemas tempo real, pouco tem sido feito no sentido do seu desenvolvimento e aperfeiçoamento. Talvez este fato seja influenciado pelas falsas idéias anteriormente apresentadas. O que se espera, então, é que tais idéias sejam abolidas e que a pesquisa em sistemas tempo real seja incentivada e aumente rumo à próxima geração, onde pretende-se obter sistemas mais eficientes, mais flexíveis e soluções melhor integradas com o ambiente onde atuam.

3 Tolerância a Falhas

Diversos autores apresentam visões distintas sobre a postura que deve ser assumida frente a falhas de software, entre estas pode-se citar a seguinte:

> "...existem duas razões principais para preocupação a respeito de falhas de software: (a) as conseqüências operacionais no sistema de defeitos resultantes de falhas de software e (b) os custos de reparação dos defeitos." [ABB 90]

Este capítulo busca apresentar uma visão geral sobre tolerância a falhas em software, apresentando seus problemas, dificuldades, conseqüências e as principais técnicas utilizadas. Nesse sentido, é discutido o interesse em aspectos de tolerância a falhas em software, especialmente na área de desenvolvimento de tecnologia para software aplicado a sistemas tempo real, que está começando a ser pesquisada. Este interesse deve-se principalmente ao fato de que tolerância a falhas em tempo real é fundamental para assegurar a confiabilidade dos sistemas de aplicação baseados em computação crítica segura, e, desta forma, produzir software de maior qualidade [SIN 94].

Atualmente, o principal desafio no desenvolvimento de tecnologia em tolerância a falhas tempo real parece ser a integração. O estado da arte atingiu um ponto onde técnicas promissoras que são efetivas na realização de capacidades de tolerância a falhas específicas de subsistemas quando utilizadas em isolamento, não estão adaptadas à integração com outros subsistemas. Há muito mais pesquisa a ser feita para melhorar estas técnicas individualmente, porém uma questão mais urgente e importante é a sua integração [KIM 95].

3.1 Conceitos Básicos de Tolerância a Falhas

O objetivo desta seção não é introduzir, de forma aprofundada, conceitos de tolerância a falhas; para isso, sugere-se consultar obras que constituem em referências clássicas [AND 81, JAL 94, LAP 85, NEL 87, NEL 90, PAU 89 e PRA 96]. Aqui serão apresentadas apenas as idéias básicas relacionadas à nomenclatura usada no texto.

Tolerância a falhas refere-se ao emprego de técnicas e mecanismos em sistemas de computação que visam garantir a continuidade do fornecimento de seus serviços, de acordo com as suas especificações, apesar da ocorrência de falhas em seus próprios componentes ou em seu meio ambiente. Desta forma, tolerância a falhas ou computação tolerante a falhas significa a habilidade de produzir resultados coerentes mesmo na presença de falhas, erros e outras condições anômalas ou inesperadas no sistema.

A escolha das técnicas e mecanismos vai produzir diferentes tipos de respostas nas saídas - comportamento externo do sistema diante da ocorrência de falhas - que podem ir desde a simples sinalização de anormalidade até a aparência externa de que

tudo está perfeito (ou seja, o sistema consegue lidar completamente com as falhas, mascarando-as).

Tomando por base um sistema de supervisionamento, onde a continuidade de funcionamento adequado é uma exigência, as principais características esperadas em um sistema tolerante a falhas são [FRA 87]:

- capacidade de detectar falhas temporárias e permanentes em seus circuitos e programas;
- garantia de que nenhuma falha simples de hardware deve comprometer a integridade dos dados do sistema;
- preservação da execução correta de programas e de suas funções de E/S mesmo na presença de falhas, quaisquer que sejam as fontes de erro;
- capacidade de auto-reconfiguração, de modo a colocar os componentes com defeito fora de operação até que eles sejam reparados;
- execução correta de algoritmos específicos, mesmo na presença de falhas, de modo que as conseqüências dos defeitos de um sistema possam ser superadas pelo uso de redundância;
- não reinicialização do sistema na ocorrência de falhas a ponto de ser desligado, devendo utilizar técnicas automáticas de recuperação de erro;
- garantia de alta disponibilidade e de alta confiabilidade, além de modularidade e simplicidade.

Para garantir que estas características estejam presentes no sistema, pode-se aplicar tolerância a falhas tanto a nível de hardware, quanto a nível de software. Em ambos os casos, normalmente são utilizadas técnicas de redundância de componentes - chamada redundância espacial - ou de repetição de computações - chamada redundância temporal - que serão apresentadas na seção 3.4.

Segundo Lisbôa [LIS 93], as falhas de software são creditadas à complexidade do desenvolvimento do mesmo. Dessa forma, pode-se concluir que a estrutura do software é importante sob o ponto de vista da suscetibilidade a erros e que a especificação de requisitos é essencial para a produção de software confiável. Assim, dependendo do tipo de software que está sendo desenvolvido, pode-se utilizar atividades de prevenção, eliminação ou de tolerância a falhas. Porém, o importante é que pelo menos uma destas esteja presente durante todo o processo de desenvolvimento do software. As duas primeiras atividades são incluídas exclusivamente no projeto através do emprego de boas técnicas de engenharia de software. Já as atividades de tolerância a falhas são incluídas durante o projeto, mas agem durante o uso do sistema. Sua forma mais eficiente baseia-se na aplicação de técnicas de mascaramento das falhas para que estas não sejam percebidas pelo usuário.

As técnicas de tolerância a falhas não são usadas para qualquer tipo de ação de remoção de falhas e sim para garantir que, apesar da existência de falhas, o programa continuará executando de forma a atender a sua especificação. Em especial, os sistemas tempo real devem executar com garantias de um comportamento correto e contínuo do sistema computacional; para isto, eles devem apresentar características de dependabilidade como disponibilidade, confiabilidade, segurança e integridade. Nesse

sentido, é imprescindível a aplicação de técnicas de tolerância a falhas a estes sistemas, visando manter a sua operação normal na presença de eventuais falhas internas.

Segundo Anderson e Lee [AND 81], tolerância a falhas é uma atividade dinâmica, centrada na detecção e recuperação de erros em tempo de execução, mas que pode ser dividida em quatro fases consecutivas, que são:

- Detecção de erros: consiste em perceber que existe um desvio de comportamento do sistema, a partir da sua especificação.
- Confinamento e avaliação de danos: implica em avaliar a abrangência da falha e evitar sua propagação, visto que, desde a ocorrência até a detecção de uma falha, esta pode ter se espalhado entre os demais componentes do sistema.
- Recuperação de erros: significa voltar ao ponto do sistema onde ocorreu a falha, para recomeçar a execução livre de falhas.
- Tratamento da falha: quando é efetuada a manutenção ou a substituição das máquinas (hardware) ou dos processos (software) que causaram a falha.

Estas fases estão interligadas, pois cada fase depende da execução adequada das fases anteriores para que possa ser realizada com sucesso.

Na maioria das vezes, as atividades de tolerância a falhas dependem da exatidão dos mecanismos para detecção de erros e podem, por sua vez, iniciar uma série de procedimentos preventivos ou conseqüentes (enquadrados nas fases seguintes do processo), tais como:

- reconfigurar o sistema, usando técnicas de redundância;
- modificar o programa executivo e atuar em tomadas de decisões sobre o processo de recuperação;
- inserir no programa rotinas de prevenção e de situações de emergência que resultarão em um modo degradado de operação;
- executar tentativas repetidas de evitar a perda de dados;
- registrar cronologicamente valores assumidos e de erros, bem como avaliar os efeitos de falhas temporárias que podem eventualmente tornarem-se críticas no decorrer do tempo;
- gerar condições de alarme indicando a presença de erros, que podem requerer intervenções manuais.

Estas ações variam de acordo com o tipo de aplicação envolvido e dos requisitos exigidos e vão necessitar do emprego de técnicas adequadas e específicas. É importante ressaltar que a detecção e a localização da falha são essenciais para os passos seguintes da reconfiguração.

3.2 Graus de Tolerância a Falhas

Por melhor que um sistema tenha sido projetado, especificado e testado, não há como garantir que ele seja totalmente confiável, isto é, que esteja sempre ativo e comportando-se conforme suas especificações. Mesmo que se suponha o projeto e

implementação perfeitos (o que é difícil), durante a vida útil do sistema podem ocorrer defeitos em componentes de hardware ou situações não previstas no software. Assim, em diversas aplicações, independentemente do tipo de falha, de suas conseqüências e gravidade, é norma de qualidade que o sistema de computação disponha de mecanismos de tolerância a falhas para impedir que os efeitos destas prejudiquem o seu funcionamento.

Entretanto, segundo Kim [KIM 95], tolerância a falhas não é uma propriedade que um sistema apresente do tipo: tudo ou nada. Como os trabalhos de Kim têm por objetivo aplicações tempo real, esta classificação é interessante para o escopo desta dissertação por similaridade de enfoques. De acordo com o autor, existem graus de aplicação da propriedade. Dessa forma, são definidas cinco diferentes categorias de tolerância a falhas, de acordo com os benefícios oferecidos para as aplicações. Conforme apresentado por Lisbôa [LIS 95], estas categorias podem ser visualizadas através da tabela 3.1.

TABELA 3.1 - GRAUS DE TOLERÂNCIA A FALHAS

Grau	Danos Previstos	Capacidade de Recuperação
<4>	Sem perda de ações visíveis.	Tolerância a falhas a nível de ação (recuperação de uma ação visível interrompida).
<3>	Perda de uma ou mais ações visíveis.	Lenta recuperação de um serviço (sem perda de hardware).
<2>	Perda de um ou mais serviços.	Recuperação parcial de hardware (degradação de serviços).
<1>	Perda de todos, exceto um núcleo mínimo de serviços críticos.	Recuperação mínima de um núcleo de hardware (serviços críticos mínimos).
<0>	Perda de serviços críticos.	Nenhuma tolerância a falhas.

Assim, cada uma destas categorias apresenta resumidamente as seguintes características:

- Grau 4 (o topo): representa tolerância a falhas no nível da ação, ou seja, todas as ações visíveis são executadas com sucesso apesar de defeitos nos componentes. A máxima confiabilidade é garantida.
- Grau 3: ocorre quando a recuperação de uma falha é lenta e, muitas vezes, é necessário abandonar a execução de uma ação visível que foi interrompida devido a ocorrência de uma falha. Entretanto não há perda a nível de hardware.
- Grau 2: resulta em alguma degradação de serviço por que, nesse caso, alguma parte do serviço crítico pode ser sacrificada, resultando em uma recuperação apenas parcial de hardware.

- Grau 1: representa o pior caso do grau 2. Sua finalidade é manter um conjunto mínimo de funções de serviço críticas, com a recuperação de uma configuração de hardware mínima.
- Grau 0: neste caso, não há tolerância a falhas, pois um serviço crítico não pode ser mantido. Diz-se então que tolerância a falhas não é fornecida.

3.3 Tolerância a Falhas em Sistemas Tempo Real

Conforme exposto anteriormente, é fundamental que técnicas de tolerância a falhas estejam presentes em aplicações envolvendo risco de vida humana, podendo levar à situações catastróficas, como em centrais nucleares, sistemas de transporte, monitoração de pacientes em hospitais (UTIs), sistemas aeroespaciais, entre outros. Assim, de acordo com os graus apresentados na seção 3.2, é desejável que o grau de tolerância a falhas de um sistema tempo real seja o máximo, ou seja, 4, garantindo sua máxima confiabilidade.

Por outro lado, além de problemas de funcionamento de algum componente, ou incorreção de dados, nesse caso, deve ser levada em conta a pontualidade em que um serviço é oferecido, pois este é um fator fundamental em sistemas tempo real. Portanto, sistemas desta categoria incluem atrasos entre as falhas para as quais devem estar aptos a tolerar e continuar operando normalmente. Logo, a implementação de tolerância a falhas em sistemas tempo real deve levar em consideração alguns requisitos importantes, tais como:

- disponibilidade de pequeno espaço de tempo para detecção de erros e localização das falhas e implementação de soluções;
- impossibilidade de voltar atrás e refazer operações anteriores, na maioria das vezes;
- exigência de redundância em funcionamento permanente para garantir a continuidade do processamento;
- comportamento livre de falhas, o que significa que, em caso de ocorrência de uma falha não-mascarável, o sistema permanece em estado seguro ou vai imediatamente para um.

O comportamento livre de falhas de um sistema tempo real vai variar de acordo com a aplicação, para que seja garantido um estado seguro. Por exemplo, em um sistema controlador de vôo de um avião, cujo estado inicial é "no chão" e o estado atual, quando ocorre a falha, é "voando"; o estado seguro é manter o sistema no estado atual, para evitar que o avião se descontrole e caia. Por outro lado, em um sistema controlador de forno industrial, cujo estado inicial é "desligado" e o estado no instante da falha é "aquecendo", o estado seguro é retornar ao estado inicial, para evitar que o forno possa explodir.

Desta forma, a existência de restrições leva à escolha de técnicas adequadas, as quais serão tratadas ainda neste capítulo. Se há pouco tempo para a localização das falhas, significa que possivelmente o processo deve prever granularidade compatível, restringindo a módulos substituíveis. Quanto à recuperação, que deve ser imediata, deve-

se pensar em técnicas por avanço ou que não provoquem efeito dominó (retornos muito drásticos). Além disso, em muitos casos, é simplesmente impossível voltar atrás e refazer operações anteriormente executadas, uma vez que o processo controlador está atuando fisicamente e de forma direta sobre o processo controlado.

Quanto ao tipo de falhas, em sistemas tempo real, as falhas mais importantes a serem toleradas ou evitadas são [KEL 91]:

- Temporização: quando uma tarefa não é completada no espaço de tempo especificado.
- Detecção e recuperação de erro: quando o mecanismo de detecção ou recuperação não trata adequadamente um erro ou é invocado quando o erro não existe.
- Interação: quando o comportamento de módulos que interagem entre si não foi suficientemente especificado ou analisado.
- Controle de concorrência: envolvem problemas de comunicação e sincronização entre processos; além de proteção e coerência de dados e *deadlock*.

Um fato importante é que estas falhas podem ser introduzidas em todas as fases do ciclo de vida do sistema: especificação, projeto, implementação, testes e manutenção. Elas podem surgir de defeitos e omissões nos requisitos ou especificações iniciais, da metodologia de projeto, de interpretações impróprias das especificações e de manutenção incorreta. Nesse sentido é fundamental que sejam utilizadas boas técnicas de engenharia de software juntamente com as de tolerância a falhas em todas as etapas do desenvolvimento.

Quando a questão refere-se ao desenvolvimento de software de segurança crítica, o sistema deve ser projetado de modo que a parte crítica seja preferencialmente isolada dos demais componentes e reforçada quanto à robustez. Por outro lado, é necessário reforçar a atenção, pois em muitos casos uma falha de um componente não-crítico pode comprometer todo o sistema. Isto por que, como diz o ditado:

"Uma corrente é tão forte quanto o mais fraco dos seus elos."

Tecnicamente, esta questão pode ser abordada através dos aspectos matemáticos da confiabilidade. A corrente é o elemento figurado de um conjunto de componentes C_i ligados em série. Por hipótese, $C_i(t)$ representa o evento de C_i estar operando adequadamente no tempo t. A confiabilidade (R(t)) do conjunto será expressa, em termos matemáticos, pela seguinte expressão:

$$R_{série}(t) = P\{C_1(t) \cap C_2(t) \cap ... \cap C_n(t)\}$$

e, assumindo que os eventos $C_i(t)$ são independentes,

$$R_{série}(t) = R_1(t).\ R_2(t)..... R_n(t).$$

Assim, em uma configuração série, é necessário que cada elemento do sistema funcione adequadamente. Basta que um deles falhe para que o sistema perca confiabilidade. Portanto, a expressão acima também pode ser traduzida como o cálculo da probabilidade de que nenhum dos elementos do conjunto falhe [PRA 96].

Assim, para evitar que uma falha em um componente menos crítico gere uma conseqüência que possa comprometer o funcionamento global de todo o sistema, deve-se despender igual atenção a todas as partes do sistema.

3.4 Técnicas de Tolerância a Falhas

Técnicas de tolerância a falhas podem ser implementadas em software, para resolver e detectar falhas do hardware e do próprio software. Dessa forma, tais técnicas devem ser capazes de perceber qualquer manifestação presente no código executável de um programa que possa implicar em diminuição de confiabilidade. Para que seja possível implementar tolerância a falhas em software, deve-se incorporar componentes adicionais ao programa, como alternativas para funções consideradas críticas, a fim de permitir a continuidade do processo de execução do software frente às situações anormais.

Os métodos usuais para a tolerância e prevenção de falhas baseiam-se, tal como em hardware, em mascaramento através de redundância, ou seja, na existência e disponibilidade de alternativas e multiplicação de recursos. Esta é a forma tradicional de tolerar falhas de projeto e programação na fase operacional do software, oferecendo diversidade de projeto e de implementação.

Inclusive mecanismos de tratamento de exceções em software podem ser considerados como um tipo especial de redundância. Quando é detectada uma exceção durante a execução de um programa, o controle é transferido para um manipulador de exceções que termina a execução do programa (modelo de terminação) ou desvia o programa falho para algum outro programa que dê continuidade a sua execução (modelo do recomeço) [LIS 93].

As técnicas mais utilizadas na implementação de tolerância a falhas em software são as pertencentes à programação diversitária, cujo principal objetivo é minimizar a probabilidade de versões projetadas independentemente conterem erros similares. A base da programação diversitária é obter-se vários componentes de software com a mesma função, mas projetados de forma totalmente independente e não apenas simples cópias. Algumas destas técnicas baseiam-se em mascaramento, cujo objetivo é impedir que erros se propaguem até sua manifestação como defeitos no sistema, corrigindo-os de forma dinâmica. Estas técnicas baseiam-se no uso de redundância dos componentes. Exemplos de técnicas que se enquadram como sendo de programação diversitária são: blocos de recuperação [RAN 75], programação n-versões (PNV) [AVI 85], conversação [JAL 94] e programação auto-verificadora [LAP 90]. Já a técnica de re-expressão de dados [JAL 94] baseia-se na diversidade de dados. As seções seguintes explicam e analisam o funcionamento de cada uma destas técnicas.

3.4.1 Técnicas de Recuperação

As técnicas de recuperação visam transformar o estado atual incorreto do sistema em um estado livre de falhas, escolhido e definido durante a especificação, a partir do qual pode ser retomada a operação normal do sistema. Estas técnicas podem ser de dois tipos:

- Avanço: consiste na transformação do estado errôneo, conduzindo o sistema a um novo estado livre de falhas, que ainda não tenha ocorrido.
- Retorno (retrocesso): significa conduzir o sistema a um estado consistente pelo qual ele passou antes da ocorrência do erro, descartando toda a informação referente ao estado atual. Para isto, deve-se salvar o contexto do sistema durante a sua execução.

Uma possível implementação destas técnicas (de ambos os tipos) são os mecanismos de tratamento de exceções.

Mas a implementação mais conhecida é a técnica de recuperação por retorno denominada blocos de recuperação [RAN 75]. Trata-se de uma técnica de tolerância a falhas para software baseada em redundância espacial dinâmica, pois apesar da existência de diversos componentes redundantes, apenas um subconjunto encontra-se ativo em um dado momento. A base da técnica é a substituição de computações errôneas (que apresentaram falhas).

O funcionamento desta técnica corresponde ao seguinte: várias tentativas de blocos alternativas, todas implementando a mesma funcionalidade, são combinadas para garantir a execução de uma parte crítica do software. Na entrada do bloco de recuperação o estado do programa é salvo e a primeira alternativa é executada. Quando termina a sua execução, os resultados computados são avaliados através de um teste de aceitação. Se o teste falhar, a computação é retrocedida, de forma a assumir o estado da entrada no bloco e uma versão alternativa é executada. Este funcionamento é ilustrado na figura 3.1.

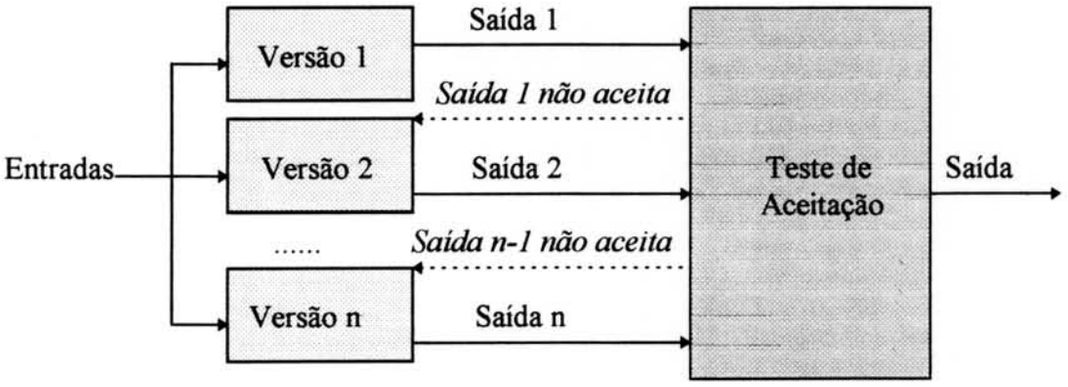

FIGURA 3.1 - FUNCIONAMENTO DO BLOCO DE RECUPERAÇÃO

De um modo geral, as principais vantagens das técnicas de recuperação são as seguintes:

- capacidade de fornecer recuperação diante de defeitos não previstos;
- independência da aplicação;
- facilidade de ser implementada como mecanismo de recuperação de erros.

Entretanto, conforme estudado em trabalho anterior [DEN 96], sistemas tempo real críticos não devem aplicar técnicas de recuperação por retrocesso. Isto deve-se ao fato de que em sistemas desta categoria é difícil e, muitas vezes, impossível voltar atrás em atitudes já tomadas como, por exemplo, durante o processo de fundição de metais. Nestes sistemas, pode ser difícil recomeçar uma ação, visto que ela já pode ter sido

executada fisicamente no processo controlado. Por outro lado, como sistemas deste tipo têm restrições severas quanto ao prazo para conclusão de determinadas tarefas, muitas vezes, devido ao fato de ter que se executar mais de uma vez a mesma computação, pode ocorrer um atraso indesejável na conclusão da tarefa. Dessa forma, dependendo da situação, é preferível repetir uma saída anterior do que voltar atrás e refazer uma computação (o que pode resultar em um atraso considerável). Esta atitude é aplicável em sistemas de controle de forno industrial, onde a saída corresponde à temperatura do forno, por exemplo.

Além disso, como nos blocos de recuperação são empregados retornos para recuperar erros, quando se trata de um sistema concorrente, que tenha comunicação entre processos, isto pode ocasionar um problema sério: o efeito dominó. Supondo-se que um processo deva retornar a um estado anterior (armazenado), devido a uma comunicação entre processos, este processo pode requerer que outros processos retrocedam ao seu último estado armazenado e assim sucessivamente até que um estado consistente seja obtido. O efeito dominó corresponde ao retorno descontrolado de processos ao estado anterior, que pode levar ao começo da computação. É caracterizado por repetidos retornos, impedindo o progresso da computação. Esta situação é extremamente indesejável devido às restrições temporais, para a conclusão de tarefas, impostas ao sistemas aqui considerados. A figura 3.2 ilustra a ocorrência do efeito dominó em um sistema, conforme apresentado por [KOO 87].

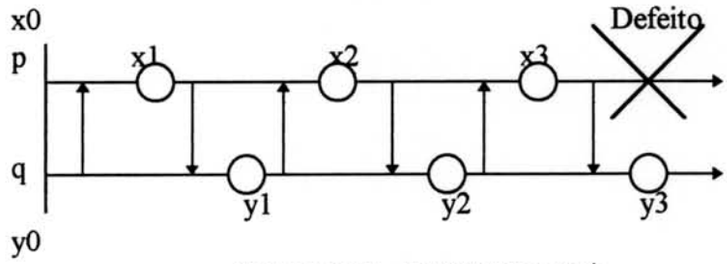

FIGURA 3.2 - EFEITO DOMINÓ

Neste exemplo, os processos p e q criaram uma seqüência de pontos de recuperação independente. A combinação da troca de mensagens e dos pontos de recuperação não permite nenhum estado consistente de pontos de recuperação para p e q exceto o estado inicial $x0, y0$. Dessa forma, depois que p falha, ambos devem retornar ao início de suas computações.

Em resumo, podem ser detectados os seguintes problemas a respeito da consistência da recuperação:

- Efeito dominó: problema que pode ocorrer devido ao estabelecimento desordenado de pontos de recuperação, que podem provocar o retorno do processo a um estado inconsistente.
- Necessidade de repetição da entrada: este problema pode ocorrer devido à reexecução de tarefas. Para evitá-lo devem ser observadas restrições na estrutura e na comunicação entre tarefas.
- Geração de saídas: o problema, neste caso, ocorre se o erro for detectado depois de uma saída ter sido liberada, devido à impossibilidade do seu cancelamento.

Há ainda o fato de que um sistema tempo real trata-se normalmente de um conjunto de processos concorrentes, o que dificulta até certo ponto que se salve o contexto global antes de começar a executar o módulo implementado por bloco de recuperação. Por fim, pode-se dizer que estes problemas contribuem para que se chegue à conclusão que a técnica de blocos de recuperação deve, de uma forma geral, ser considerada cautelosamente quando se trata da implementação de tolerância a falhas em sistemas tempo real, principalmente críticos. Atualmente, estão surgindo algumas variações de implementação para blocos de recuperação visando uma maior adequação destas técnicas a sistemas tempo real.

Entretanto, as técnicas de recuperação por avanço, que incluem o tratamento de exceções, podem ser perfeitamente aplicáveis nas situações aqui consideradas.

3.4.2 Programação N-Versões

Uma das técnicas de mascaramento de software mais utilizadas é a programação n-versões (PNV) [AVI 85]. Trata-se de uma técnica de tolerância a falhas de software baseada em redundância espacial estática, pois os componentes adicionais de software (versões ou variantes) são executados concorrentemente.

Na aplicação da programação n-versões, várias versões de um módulo de software com a mesma funcionalidade, porém projetadas e desenvolvidas independentemente, são executadas em paralelo e seus resultados computados são comparados por alguma unidade específica, o módulo de votação. O resultado computado pela maioria das versões é o aceito como saída do sistema. O funcionamento desta técnica é ilustrado na figura 3.3.

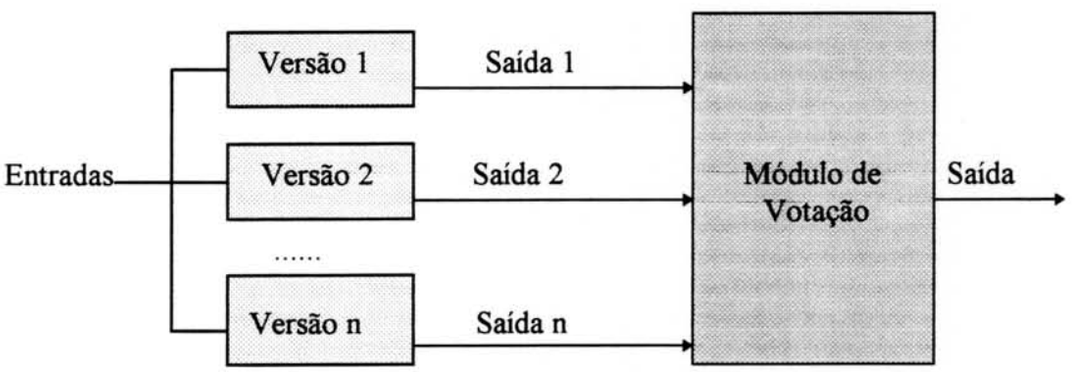

FIGURA 3.3 - FUNCIONAMENTO DA PNV

PNV pode ser utilizada tanto para detectar problemas causados por falhas de software quanto de hardware, uma vez que, quando é detectado um erro, não há como saber se ele é decorrente de uma falha na versão do programa ou no processador que está executando.

De um modo geral, esta técnica pode ser utilizada em sistemas tempo real, pois além de permitir a execução de processos concorrentes, não exige que o estado da computação seja retrocedido em nenhum instante. Entretanto, se esta técnica for utilizada em processos concorrentes alguns problemas podem surgir, visto que cada linguagem de programação utilizada pode assumir um estilo particular de comunicação (por exemplo: troca de mensagens, memória compartilhada ou ainda a abordagem

cliente-servidor) e integrá-los como versões alternativas é uma tarefa bastante complexa. Além disso, para um melhor desempenho da técnica é aconselhável que se tenha diversos processadores disponíveis e atuantes, o que nem sempre é viável. Porém quando o tempo de resposta não é um fator tão crítico no funcionamento do sistema (tempo real brando), é possível utilizar-se de recursos como o pseudo-paralelismo.

Por outro lado, a principal vantagem desta técnica é que ela proporciona níveis de disponibilidade e de integridade significativamente mais altos, de forma totalmente transparente aos usuários do sistema. Porém, PNV é utilizada somente em casos extremos, pois apresenta várias desvantagens, tais como:

- o custo de instalação de um sistema com PNV é duas ou três vezes maior que um sistema normal;
- dificuldade de sincronização de respostas (saídas) dos diversos módulos;
- precisão das saídas na comparação, visto que diferentes algoritmos podem encontrar respostas um pouco diferenciadas (por exemplo, 1,951 e 1,954) para o mesmo problema - uma solução para esta questão pode ser encontrada no estabelecimento de faixas de aceitação para as respostas de um mesmo problema.
- o software adicional implica em maiores custos de manutenção;
- exige hardware adicional, implicando também em custos de manutenção deste, além da disponibilidade de componentes e serviços para tal fim.

Desta forma, é importante avaliar o grau de redundância necessário para garantir a tolerância a falhas em cada aplicação específica, levando-se em consideração principalmente os seguintes aspectos:

- custo de eventuais paradas do processo;
- probabilidade relativa de ocorrência de parada para cada alternativa;
- níveis de risco causado por desativações não-intencionais do processo;
- tempo de suporte disponível a nível de componentes e serviços para cada alternativa do sistema.

Como o custo de PNV é relativamente alto, visto que é necessário despender diferentes cuidados e atenção para cada versão, o uso de três versões é a prática mais conhecida na aplicação desta técnica.

Esta técnica é muitas vezes referida como uma analogia à TMR (*Triple Modular Redundancy* ou Redundância Modular Tripla) utilizada quando se implementa tolerância a falhas em hardware. Entretanto TMR propõe, em regra, replicação de módulos de hardware idênticos, o que previne apenas a ocorrência de erros conseqüentes a falhas de funcionamento em componentes e não a falhas de projeto. No caso do software, as falhas que se procura controlar são justamente as provenientes do projeto.

3.4.3 Conversação

Segundo Jalote [JAL 94], a técnica denominada conversação foi proposta como uma extensão ao bloco de recuperação, para que seja possível a sua utilização quando se trabalha com processos concorrentes, visando principalmente prevenir o efeito dominó. Este objetivo a torna interessante, a princípio, para sua aplicação em sistemas tempo real.

O funcionamento da técnica corresponde ao seguinte: um ponto de recuperação é usado quando um processo entra em uma conversação. Este processo deve comunicar-se apenas com os processos que já entraram na conversação (e guardaram seus pontos de recuperação), para que não seja propagado um dado ainda não válido - o que é denominado "contrabando de informação". Quando todos os processos completam suas operações, é executado um teste de aceitação global. Em caso de falha, todos os processos retornam e executam sua próxima alternativa. Se o teste de aceitação é satisfeito, todos os processos podem descartar seus pontos de recuperação e prosseguir, acabando sincronamente a conversação [CLE 93]. A figura 3.4 ilustra o funcionamento de uma conversação em um sistema concorrente.

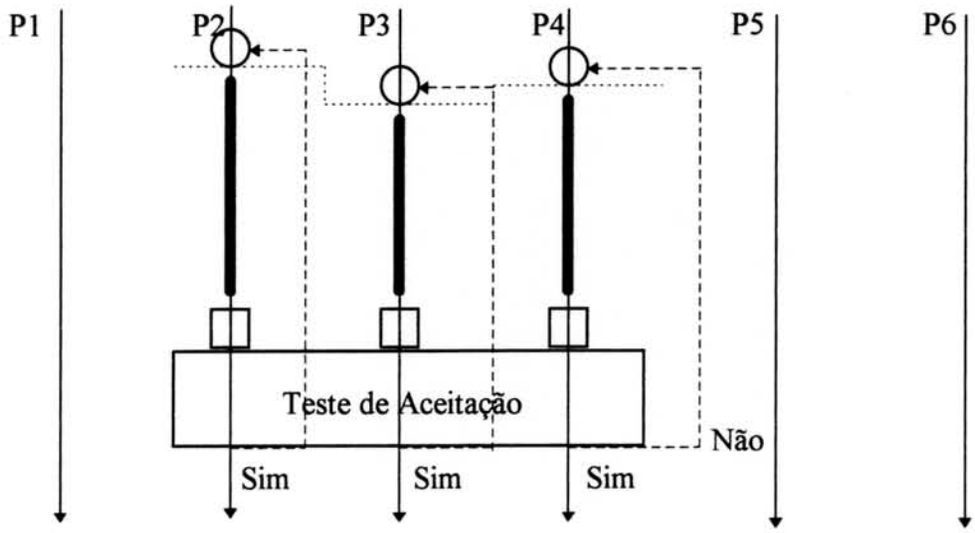

FIGURA 3.4 - REPRESENTAÇÃO DA CONVERSAÇÃO ENTRE PROCESSOS

Onde:
- → Ponto de recuperação
- → Linha de recuperação
- → Operação completa
- → Parte do processo onde há multiplicidade de alternativas para execução repetida.

Esta figura pode ser interpretada como uma seqüência dos seguintes passos, que representam o conversação:

1. Todos os processos que vão participar da conversação armazenam seus pontos de recuperação.
2. Os processos trocam informações entre si, enquanto seguem a execução de seus comandos.

3. A conversação é encerrada.
4. Os processos realizam simultaneamente o teste de aceitação.
5. Se todos processos passam pelo teste, eles descartam sincronamente os pontos de recuperação e continuam sua execução normal.
6. Se um dos processos falha no teste de aceitação, todos retornam aos seus pontos de recuperação e executam uma versão alternativa.

A restrição na comunicação limita a propagação de erros e elimina a possibilidade de ocorrência do efeito dominó. Além disso, fica garantido que nenhum erro subseqüente force os processos a retornar além do último teste de aceitação realizado com sucesso. Nesse sentido, sua utilização em sistemas tempo real parece ser aceitável em alguns casos, desde que as ações realizadas durante a conversação sejam vistas como ações atômicas, ou seja, do tipo "tudo ou nada". Dessa forma, as ações serão executadas fisicamente no processo controlado somente após a passagem pelo teste de aceitação, evitando que seja preciso voltar atrás e refazer ações já tomadas. Se esta restrição não implicar em atrasos no cumprimento dos prazos especificados, então a técnica pode ser aplicada em sistemas tempo real. Uma outra possibilidade (plausível para alguns casos) é a previsão de valores para uso "emergencial" quando o teste de aceitação não obtém resposta adequada.

3.4.4 Programação Auto-Verificadora

Segundo Laprie [LAP 90], esta técnica baseia-se em redundância ativa dinâmica. Um programa auto-verificador resulta da adição de redundância a um programa, permitindo que ele possa verificar seu próprio comportamento dinâmico durante a sua execução. Um componente de software auto-verificador consiste de uma versão e um teste de aceitação ou duas versões e um algoritmo de comparação. Um componente de software é dito ser de auto-verificação se ele possuir a capacidade de julgamento sensato de seus próprios resultados de computação [KIM 95].

Dessa forma, tolerância a falhas é garantida pela execução paralela de, no mínimo, dois componentes auto-verificadores. A cada execução do sistema, um componente é o ativo, enquanto que o(s) outro(s) permanece(m) como reserva(s) a quente. Os resultados são produzidos simultaneamente por ambos, mas somente o ativo divulga-os à aplicação. Quando o componente ativo falha, um componente reserva assume o seu lugar e continua a computação. Se um reserva falha, o componente ativo continua a executar seu serviço normalmente. A figura 3.5 ilustra o funcionamento desta técnica.

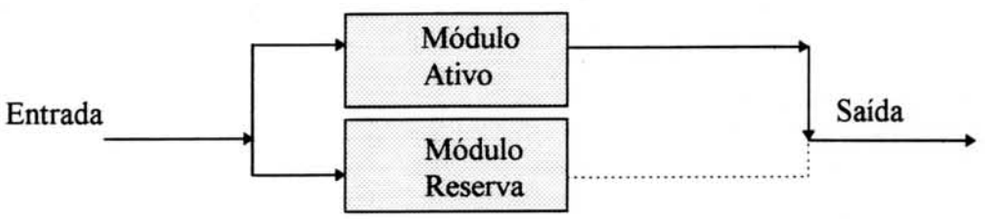

FIGURA 3.5 - PROGRAMAÇÃO AUTO-VERIFICADORA

Assim, a técnica de programação auto-verificadora, conforme descrita por Laprie e apresentada por Lisbôa [LIS 93], caracteriza-se pela existência simultânea dos seguintes elementos:

- programa com redundância, que possibilite a verificação de seu próprio comportamento dinâmico durante o processo de execução;
- componente de software para auto-verificação, que consiste de uma variante e de um teste de aceitação ou de duas variantes e de um algoritmo de comparação.

Esta técnica combina as de programação n-versões e de blocos de recuperação, através da adaptação de uma outra técnica semelhante, conhecida na tolerância a falhas em hardware como *hot-standby*, citada por Denardin [DEN 96]. Desta forma, por ser uma técnica híbrida, carrega consigo as vantagens, mas também algumas desvantagens das anteriores. O principal problema abordado por Lisbôa [LIS 93] é que para garantir o chaveamento das funções entre os componentes auto-verificadores, é necessário que se disponha de um mecanismo para assegurar a consistência de entrada.

Conforme apresentado e discutido por Kim [KIM 95], pode-se concluir que esta técnica pode, de um modo geral, ser utilizada em sistemas tempo real, desde que não implique em um atraso indesejável durante o chaveamento das funções entre os componentes primário e sombra.

3.4.5 Re-expressão de Dados

Esta técnica, diferentemente das demais, não implementa diversidade a nível de componentes de software, mas sim a nível de dados, conforme apresentado por Jalote [JAL 94]. Assim, o mesmo componente (ou módulo) de software é executado para conjuntos distintos de dados. Dessa forma, um algoritmo de decisão determina a saída final do sistema, dependendo da saída obtida a partir da execução do software com estes diferentes conjuntos de dados.

Na técnica de diversidade de dados é fundamental que re-expressão de dados seja possível. Pode-se definir re-expressão de dados como a geração de conjuntos de dados logicamente equivalentes. Dessa forma, um algoritmo de re-expressão de dados, altera os valores das variáveis, de acordo com as especificações do sistema.

Apesar do que parece a princípio, re-expressão de dados não é uma técnica limitada a alterar valores de números reais. Também pode-se, com ela, alterar valores obtidos a partir de sensores e mapear qualquer tipo de dados do programa. Por exemplo, as coordenadas de um avião podem ser facilmente alteradas, mudando o sistema de coordenadas para uma nova origem.

Em geral, a implementação de re-expressão de dados depende da aplicação e requer uma análise cautelosa para determinar as transformações válidas. Assim, é óbvio, que isto não é possível em todas as situações. Mas há muitas aplicações, inclusive de sistemas tempo real, como sistemas de controle que utilizam sensores e computações de estatística, onde esta técnica é claramente viável de ser aplicada.

A principal vantagem da abordagem de diversidade de dados é o custo, pois apenas uma versão do programa necessita ser escrita e mantida. Entretanto, o seu uso

requer um entendimento cuidadoso de todas as dependências dos dados que serão alterados. Mas, em princípio, atendidas as restrições temporais, a técnica não apresenta outras desvantagens para uso em sistemas tempo real.

Duas estruturas foram propostas por Ammann, conforme citado por [JAL 94], como formas de implementação de diversidade de dados:

- Bloco tentativo: é modelado depois do bloco de recuperação e possui a mesma semântica, porém usa apenas um algoritmo e não múltiplas alternativas. Assim, se o teste de aceitação falha, depois de retornar ao ponto de recuperação, o algoritmo é reexecutado apenas depois da re-expressão de dados. Este processo pode ser repetido até a expiração do prazo máximo determinado para a execução do algoritmo. A figura 3.6 ilustra esta estrutura.

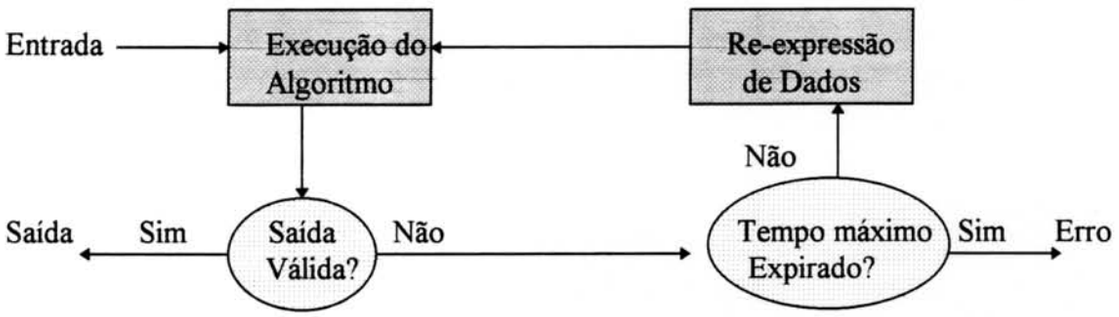

FIGURA 3.6 - ESTRUTURA DE BLOCO TENTATIVO

- N-Cópias: é modelada após a abordagem de programação n-versões e emprega diversidade de dados. Nesta estrutura, n cópias diferentes do programa são executadas em paralelo. Entretanto, diferentemente da programação n-versões, cada cópia do programa recebe diferentes dados de entrada, cada um representando uma re-expressão das entradas originais. A votação é feita com as saídas das diferentes cópias, porém ela é um pouco mais complicada, por que as saídas podem ser diferentes. A figura 3.7 ilustra esta abordagem de implementação para diversidade de dados.

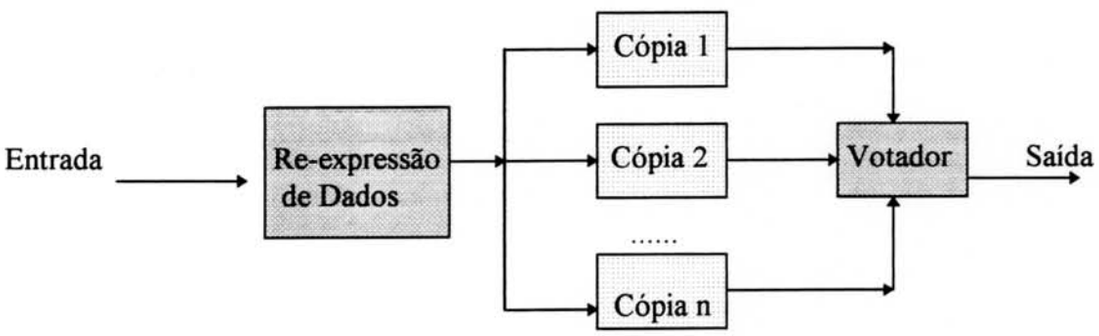

FIGURA 3.7 - ESTRUTURA DE N-CÓPIAS

4 Comunicação Tempo Real

A maioria dos sistemas tempo real apresenta configurações que consistem de um conjunto de nodos interconectados por um sistema de comunicação tempo real, caracterizando um sistema distribuído [KOP 93]. Pode-se dizer então, que um sistema tempo real é composto de vários subsistemas autônomos interligados, cada um contribuindo com a sua parcela (realizando as suas tarefas) para a execução das atividades impostas ao sistema. Dessa forma, um sistema tempo real precisa compartilhar as informações críticas e importantes entre os subsistemas autônomos de uma maneira confiável e dentro do prazo máximo estipulado. Isto é possível através do sistema de comunicação, que utiliza alguns mecanismos do sistema operacional e protocolos de comunicação para realizar esta tarefa de compartilhamento de informações.

Normalmente o canal é o único meio de comunicação entre os nodos de uma rede e nele é garantido o êxito somente na transmissão de uma mensagem por vez. Se duas ou mais mensagens são transmitidas de forma simultânea, há interferência por colisão e provavelmente nenhuma delas será recebida corretamente pelos nodos de destino. Assim, fica evidente a necessidade de regras que assegurem a coordenação de acesso dos nodos distintos ao mesmo canal de comunicação. Estas regras são denominadas de protocolos de comunicação [SAN 91].

Este capítulo visa apresentar os principais aspectos relacionados à questão do uso de protocolos de comunicação em sistemas tempo real. Em um primeiro momento, serão apresentados os requisitos importantes em um sistema de comunicação tempo real, após serão abordados aspectos de tolerância a falhas, seguidos pela descrição de alguns protocolos específicos para aplicações tempo real, que serão posteriormente comparados na forma de uma taxonomia. Além de protocolos de comunicação, foi estudado, e será apresentado, um dos mecanismos de comunicação do sistema operacional que pode ser adaptado para uso em sistemas tempo real.

4.1 Características da Comunicação Tempo Real

Para obter-se comunicação em sistemas tempo real, dois elementos principais são necessários: uma rede e um protocolo, ambos atuando em tempo real. É o protocolo que controla o acesso ao meio, definindo o tráfego da rede e todas as interações diretas ou indiretas entre nodos independentes que formam a rede. Ambos são requisitos muito simples de se obter, desde que apresentem um comportamento confiável, estejam disponíveis ao ambiente e cumpram os prazos de tempo previamente estabelecidos. Dessa forma, algumas das características desejáveis num sistema de comunicação tempo real são [VER 93]:

- Entrega de mensagens: o prazo de entrega conhecido e limitado deve ser respeitado.
- Comportamento determinístico e previsível: estas características permitem saber com precisão quando inicia e quando deve terminar a transmissão de

uma mensagem por um determinado nodo. É importante que o sistema continue comportando-se dessa forma mesmo na presença de fatores anormais (por exemplo, sobrecarga ou defeitos).

- Prioridades: deve haver o reconhecimento de urgência no tráfego das mensagens (prioridades), permitindo que algumas mensagens, mais urgentes, passem na frente de outras.
- Conectividade: a conectividade do sistema deve ser permanente, uma vez que o mundo real não espera (ou não pára) durante quedas (blecautes) do sistema de comunicação.

Pode-se dizer que as duas primeiras características listadas correspondem a qualidades elementares de um sistema de comunicação tempo real, expressas através da seguinte definição:

> "Uma rede tempo real confiável apresenta um tempo de entrega de mensagens conhecido e limitado, mesmo na presença de fatores anormais ao sistema, tais como sobrecarga e defeitos." [VER 93]

Isto implica que a restrição de tempo deve ser respeitada no sentido que uma mensagem gerada em um nodo fonte deve chegar ao nodo destino garantidamente dentro de um certo intervalo de tempo previamente estabelecido. Esta idéia pode ser estendida pela visão expressa por Santos [SAN 91], segundo a qual: em um ambiente de tempo real crítico, é óbvio que os nodos devem ter um tempo garantido de acesso ao meio; por outro lado, uma vez que possui o acesso ao mesmo, nenhum nodo pode retê-lo por mais do que um certo tempo também definido, a fim de permitir que todos os demais nodos possam acessá-lo dentro de seus prazos e, por outro lado, evitando que um único nodo monopolize o canal.

Quando se trata de comunicação, é importante salientar que é bastante difícil encontrar um único protocolo ou mecanismo de comunicação correspondente a uma solução ótima, pois depende da situação na qual ele será aplicado [KOP 94a]. Isto ocorre por que os protocolos são o meio de obtenção de uma comunicação confiável e contínua entre processos distintos de um sistema tempo real. Como cada aplicação possui características específicas e nem sempre uniformes para a mesma área de uso, o protocolo deve adequar-se a elas, atendendo aos seus requisitos. Cabe ao projetista do sistema, que tem a responsabilidade de, entre outras coisas, definir qual protocolo será utilizado para garantir a comunicação entre os nodos componentes do sistema, identificar esta adequação.

Para que seja possível realizar uma análise comparativa entre os diversos protocolos de comunicação estudados, serão consideradas como importantes, neste trabalho, as seguintes características:

- Detecção de erros: esta característica indica se o protocolo é capaz de detectar erros, que ocorram durante a comunicação, antes que uma saída errada seja enviada para o ambiente onde ele atua. Exemplos destes erros são [KOP 94a]:
  - corrupção de mensagens;
  - perda de mensagens;

- perda de nodo.

- Entrega: esta característica indica se o tempo máximo de entrega de uma mensagem é conhecido e limitado, ou seja, se está dentro do prazo determinado.

- Flexibilidade: esta característica identifica se o protocolo permite flexibilidade quanto às aplicações que podem utilizá-lo ou se ele é específico para algum tipo de aplicação.

- Previsibilidade: esta característica identifica se o comportamento do protocolo é previsível e determinístico, ou seja, se é possível saber de antemão quando uma mensagem será transmitida e qual será o tempo necessário para isto.

- Sincronismo: esta característica analisa se o protocolo oferece recursos de sincronização dos relógios tempo real entre os nodos participantes de uma comunicação. Uma sincronização entre os relógios locais dos nodos é realizada sempre que for percebida uma diferença entre o relógio do nodo que envia e do nodo que recebe a mensagem. Isto é possível através da comparação entre o tempo real de chegada do pacote e o tempo planejado a *priori* para a chegada do mesmo.

- Tolerância a Falhas: esta característica indica se o protocolo é capaz de continuar a operação normal do sistema, embora com alguma degradação, mesmo após a ocorrência de uma falha. Assim, visa-se avaliar, por exemplo, se há garantias de que a comunicação entre os nodos não-falhos não será interrompida. Tolerância a falhas, conforme é aqui considerado, refere-se às atividades que devem ser executadas após a detecção de erros (que é uma característica a parte), que correspondem a: avaliação e confinamento dos danos, recuperação de erros e tratamento de falhas.

Em sistemas tempo real distribuídos é importante que a característica de tolerância a falhas envolva o devido tratamento das seguintes situações, que afetam o funcionamento adequado do sistema de comunicação, tais como [VER 93 a]:

- Falhas temporais (atrasos): ocorrem devido a sobrecarga do sistema.

- Falhas de omissão (perdas): ocorrem devido a erros durante a transmissão.

- Partições da rede: ocorrem devido a um defeito no meio físico.

Em relação às características anteriormente apresentadas, constata-se que algumas delas podem ser consideravelmente conflitantes entre si. Por isso, é possível que, muitas vezes, o fato de um protocolo apresentar fortemente alguma destas característica implique em outra característica ser restrita ou nula no protocolo. Dentre as características conflitantes, a flexibilidade *versus* a detecção de erros e/ou a tolerância a falhas é bastante visível. Isto ocorre por que para se garantir a flexibilidade de aplicações o comportamento de um nodo (ou de um sistema) não deve ser limitado, ou seja, não deve ser específico para uma única aplicação. Ao contrário disso entretanto, detecção de erro e tolerância a falhas somente são possíveis se o comportamento atual de um nodo (ou do sistema) pode ser comparado com algum conhecimento prévio sobre o comportamento esperado do nodo (ou do sistema), um modelo que limita a gama de aplicações que podem utilizá-lo.

Por outro lado, a escolha das técnicas que implementem atividades de detecção de erros e de tolerância a falhas deve ser cautelosa, visto que somente é possível e desejável que estas sejam utilizadas em um sistema tempo real se não implicarem em um atraso prejudicial a sua execução. Assim, por exemplo, muitas vezes, fazer com que a entrega de uma próxima mensagem ocorra dentro do seu prazo limite é preferível do que realizar retransmissão de uma mensagem anterior, que pode atrasar a execução do sistema como um todo e cujo conteúdo pode ter sido invalidado pela passagem do tempo.

4.2 Tolerância a Falhas na Comunicação Tempo Real

Por simplicidade, supõe-se aqui que os protocolos de comunicação atuantes no sistema tempo real foram adequadamente implementados e funcionam de acordo com a sua especificação na ausência de falhas. Porém, quando ocorrem falhas, é que podem surgir os problemas, como: perda de mensagens, atrasos na entrega, corrupção de conteúdos, partição da rede, entre outros. Uma solução para superar alguns destes problemas pode ser encontrada na utilização de hardware replicado, ou seja, disponibilizar mais de uma cópia das partes físicas do sistema responsáveis por manter a conexão, tais como modems, transformadores e cabos. Do ponto de vista da segurança de funcionamento (dependabilidade) do sistema, esta redundância espacial é essencial em sistemas críticos. Entretanto, devido ao custo envolvido na sua implementação e à complexidade do sistema, uma solução alternativa para os casos onde a disponibilidade não é exigida permanentemente, talvez seja a redundância temporal, que, nesse caso, corresponde apenas a uma repetição no envio de mensagens, por exemplo.

É importante ressaltar que um modelo de comunicação de um sistema tempo real deve enfocar as questões temporais referentes à transferência de informação ocorrida entre a observação de um sensor no ambiente, o processamento desta informação pelo computador e a saída de um valor para um dispositivo atuador no ambiente.

Por outro lado, a presença de certas condições básicas no desenvolvimento do projeto de um sistema de comunicação tempo real garante a obtenção de um sistema confiável e disponível, de acordo com a definição anteriormente apresentada. Para controlar as falhas mais comuns, estas condições referem-se, entre outras, à [VER 93]:

- garantia que o tempo de entrega, que envolve desde a requisição até a transmissão de uma mensagem, seja limitado e previsível;
- garantia de que seja concluída a entrega de cada mensagem, apesar da ocorrência de omissão, através da implementação de redundância temporal (repetição de mensagens);
- manutenção da conectividade do sistema em tempo integral.

É importante que o protocolo seja capaz de detectar erros de corrupção de mensagens, perda de mensagens e perda de nodos. Além disso as técnicas de tolerância a falhas devem ser capazes de suprir a perda de nodos e a perda de mensagens, garantindo ainda o determinismo das réplicas, ou seja, sua consistência, e controlando a redundância física do sistema [KOP 94a].

Para garantir uma maior confiabilidade da comunicação pode-se utilizar diversas abordagens de tratamento de erros ocorridos durante a execução do protocolo, como por exemplo:

- Mascaramento de erros: garante que a ocorrência de qualquer erro (enquadrado em um conjunto de hipóteses) ficará transparente à aplicação. Este modelo é mais utilizado em sistemas tempo real que exigem correção, mas suportam eventuais atrasos ou perda de mensagens [VER 93].
- Detecção de erros e recuperação por avanço: caso ocorra uma falha detectável, o protocolo desviará sua execução para um conjunto de ações específicas para o tratamento desta anomalia, que permitirá o prosseguimento das atividades.
- Detecção de erros e recuperação por retorno (retrocesso): caso ocorra uma falha detectável, o protocolo desviará sua execução para algum estado previamente conhecido, ou seja, que já foi anteriormente executado. Para tanto, devem ser desfeitas as ações posteriores que poderiam estar inadequadas devido à ocorrência da falha. Entretanto, esta abordagem pode ser inviável em muitos casos de sistemas tempo real.

4.3 Protocolos de Comunicação Tempo Real

Vários autores mencionam a existência de mais de cem protocolos de comunicação desenvolvidos nos últimos vinte anos. Esta riqueza de opções é resultado das limitações de uso em função das características diferenciadas das aplicações e da busca de determinados parâmetros nos diversos casos. Assim, dependendo dos objetivos do usuário, mesmo em aplicações de uma mesma classe, pode ser perseguido privilegiar parâmetros como desempenho, confiabilidade e facilidade de interação em relação a outros. As combinações entre estas prioridades e a definição das classes de aplicações resultam em amplas e diferentes implementações da forma de comunicação adequada a cada situação.

Neste trabalho será abordada, diante desses dados, apenas uma pequena parcela desse conjunto. Os protocolos aqui apresentados foram desenvolvidos especificamente para uso em sistemas tempo real. Atualmente, as redes locais (LAN - *Local Area Network*) formam a base da maioria dos sistemas tempo real distribuídos. Dessa forma, os protocolos que serão apresentados a seguir foram desenvolvidos para LANs e pertencem, de acordo como é definido o controle de acesso ao meio (canal), às seguintes categorias: CSMA/CA (*Carrier Sense Multiple Access with Collision Avoidance*), CSMA/CD (*Carrier Sense Multiple Access with Collision Detection*), controle de *token*, *minislotting*, controle central e TDMA (*Time Division Multiple Access*). Quanto ao paradigma utilizado no projeto da sua arquitetura podem ser: protocolos disparados por tempo e protocolos disparados por eventos, conforme descrito no capítulo 2 deste trabalho.

Considera-se aqui que o sistema distribuído tempo real, onde estes protocolos atuarão, consiste de:

- um canal de comunicação do tipo LAN replicado;

- nodos replicados, para tolerar falhas nos nodos;
- nodos do tipo falha-silencia[2] (*fail-silent*), ou seja, que sempre enviam um resultado correto ou não enviam nada.

4.3.1 Protocolo ARINC 629

O ARINC (*Aeronautical Radio Inc.*) 629, segundo sua descrição técnica citada por Kopetz [KOP 94a], é um protocolo do tipo *minislotting*. Este tipo de protocolo utiliza uma estratégia de acesso ao meio controlado por tempo. Dessa forma, o tempo é particionado em uma seqüência de mini-frações de tempo diferentes, cada uma destas possuindo o tamanho correspondente ao tempo de propagação de mensagens no canal de comunicação. Cada nodo deve, então, esperar por um diferente espaço de tempo, definido por algumas mini-frações (enquanto os demais ocupam o canal) antes de poder realizar a sua transmissão, ou seja, de lhe ser designada uma fração de tempo e, logo após a transmissão da mensagem, deve liberar o canal e esperar pela próxima fração para poder utilizar novamente o canal. Esta estratégia evita que um nodo detenha o canal indefinidamente.

O protocolo ARINC 629 foi desenvolvido especificamente pela indústria de aviões para controlar a comunicação em tempo real realizada em aplicações aeroespaciais. E, com base nele, já foram desenvolvidas algumas variações, de acordo com a situação em que ele é utilizado - aviões de pequeno ou grande porte, de carga ou de passageiros, entre outros.

Este protocolo é baseado no paradigma de projeto de sistemas tempo real que utiliza uma arquitetura disparada por tempo. Dessa forma, neste protocolo, o acesso ao meio é controlado por três parâmetros distintos de controle do tempo:

- Período de sincronização: é o intervalo esperado por todos os processos antes de começar uma transmissão. É idêntico para todos os nodos.
- Período do terminal: é o período individual de cada nodo, assim, o processo que for realizar uma transmissão deve esperar pelo período relativo ao nodo (terminal) a que ele pertence.
- Intervalo de transmissão: é espaço de tempo compreendido desde a alocação do canal por um processo até a sua liberação, ou seja, corresponde à mini-fração de tempo do processo.

O funcionamento básico deste protocolo corresponde, de maneira resumida, à seguinte descrição:

> "Por hipótese, dois processos, denominados P1 e P2, esperam para transmitir uma mensagem. Eles primeiro devem esperar pelo período de sincronização.

---

[2] Embora a tradução literal devesse ser "falha-silencioso", ela remete a uma idéia falsa, pois, na verdade, o nodo falha e "fica quieto", ou seja, não envia dados ou sinais errados. Assim, preferiu-se optar por "falha-silencia", visando transmitir o sentido de causa-conseqüência associado ao funcionamento.

> Após, P1 espera pelo período do terminal 1 (que é tempo individual do terminal que ele pertence) e inicia a transmissão, desde que o canal esteja livre. No início da transmissão de P1, seu mecanismo de controle de tempo deve ser ajustado de acordo com o seu intervalo de transmissão, para bloquear qualquer atividade de envio que possa ser realizada a mais pelo processo P1. Este mecanismo torna impossível que uma única estação monopolize a rede. Após P1 ter iniciado a transmissão, P2 deve esperar até que P1 finalize-a. Então, ele deve esperar pelo período do terminal 2 (a que ele pertence) e iniciar a transmissão de sua mensagem. Assim, todos os nodos devem enviar mensagens durante o período de tempo correspondente à mini-fração que lhe foi designada, completando sua atividade de envio antes de qualquer outro nodo iniciar uma nova transmissão (em um novo intervalo)." [KOP 94a]

Este protocolo apresenta uma flexibilidade razoável quanto às aplicações que podem utilizá-lo, visto que foi desenvolvido especificamente para ser utilizado na comunicação aeronáutica. Outra característica é que ele oferece bons mecanismos de detecção de erros, tais como CRC (*Cyclic Redundancy Check*) e *checksum*, e outros de tolerância a falhas (que, embora não sejam denominados na referência, são oferecidos), porém não realiza a retransmissão automática de mensagens [ARI 97].

Conforme a descrição aqui apresentada, pode-se considerar que, de acordo com as características anteriormente citadas como relevantes, o protocolo ARINC 629 apresenta o seguinte perfil:

- Detecção de erros: oferece bons mecanismos de detecção de erros (CRC e *checksum*).
- Entrega: o tempo é conhecido e limitado devido ao uso de mini-frações (*slots*) de tempo.
- Flexibilidade: apresenta uma flexibilidade razoável quanto às aplicações que podem utilizá-lo, sendo mais utilizado em aplicações aeroespaciais.
- Previsibilidade: por ser controlado por fatias de tempo (e disparado por tempo) deve ser previsível.
- Sincronismo: apresenta sincronismo entre os nodos implementado através do denominado período de sincronização.
- Tolerância a Falhas: igualmente oferece bons mecanismos de tolerância a falhas. Entretanto, não realiza a retransmissão automática de mensagens perdidas.

4.3.2 Protocolo CAN

O CAN (*Controller Area Network*), de acordo com sua descrição técnica citada por Kopetz [KOP 94a], trata-se de um protocolo do tipo CSMA/CA. Protocolos deste tipo controlam o acesso distribuído ao meio de comunicação, evitando a ocorrência de colisões entre mensagens, através de um bit arbitrário, por exemplo. Especificamente o protocolo aqui considerado segue o paradigma de projeto de sistemas tempo real que utiliza uma arquitetura disparada por eventos.

Este protocolo foi desenvolvido para o controle de aplicações tempo real automotivas, porém hoje em dia é amplamente utilizado na automação industrial e no controle de processos. De acordo com a sua definição, o protocolo CAN foi projetado para ser extremamente flexível quanto às aplicações que podem utilizá-lo, tanto que atingiu outros campos de atuação além dos quais para que foi inicialmente projetado.

A sua abordagem de implementação considera uma certa arbitrariedade lógica, que assume a existência de um estado recessivo e de um estado dominante no canal de comunicação. Dessa forma, um estado dominante tem prioridade sobre o estado recessivo, passando a sua frente. O funcionamento do protocolo pode corresponder, dependendo da arbitrariedade considerada para determinar os estados, ao seguinte:

> "Por hipótese, um bit '0' é codificado como um estado dominante e um bit '1' é codificado como um estado recessivo. Quando um nodo tem a intenção de enviar uma mensagem, ele deve colocar o primeiro bit do identificador da mensagem a ser enviada no canal. Se houver conflito (colisão) de mensagens, então o nodo com um '0' em seu primeiro bit do identificador ganha o canal e o que tiver '1' deve ficar esperando. Este processo é repetido sucessivamente para todos os bits do identificador da mensagem, enquanto os bits de ambos os nodos forem iguais. Assim, um nodo com todos os bits em '0' sempre será o ganhador do canal - este é o modelo de uma mensagem com a mais alta prioridade. Pode-se dizer então, que no protocolo CAN a prioridade de uma mensagem é sempre determinada pelo identificador desta mensagem, dependendo de qual foi definido como sendo o estado dominante e qual como o recessivo." [KOP 94a]

Quanto à capacidade de detecção de erros, no CAN é bastante elevada, visto que ele implementa mecanismos de detecção tais como CRC, *frame check* e monitoramento de bits, entre outros. No que se refere à tolerância a falhas, o protocolo igualmente apresenta bons mecanismos, principalmente de confinamento de erros. Além disso, é capaz de retransmitir mensagens perdidas. Estas características permitem denominá-lo como sendo um protocolo que permite uma comunicação de dados com integridade garantida [SCH 97].

Conforme a descrição aqui apresentada, pode-se considerar que, de acordo com as características anteriormente citadas como relevantes, o protocolo CAN apresenta o seguinte perfil:

- Detecção de erros: é bastante elevada, visto que ele implementa mecanismos de detecção tais como CRC, *frame check* e monitoramento de bits, entre outros.
- Entrega: por ter sido projetado especificamente para aplicações tempo real, supõe-se que seu tempo de entrega seja conhecido e limitado, para que possa permitir a operação normal do sistema, sem atrasos significativos. Entretanto, não foram obtidas informações sobre os mecanismos e facilidades que controlam este aspecto.
- Flexibilidade: é considerado extremamente flexível quanto às aplicações que podem utilizá-lo.
- Previsibilidade: os protocolos disparados por eventos não são suficientemente previsíveis.

- Sincronismo: o fato de ser disparado por eventos torna esta característica desnecessária no protocolo.
- Tolerância a Falhas: igualmente apresenta bons mecanismos, principalmente de confinamento de erros e, além disso, é capaz de retransmitir mensagens perdidas.

4.3.3 Protocolo FIP

O FIP (*Flux Information Processus* ou *Factory Instrumentation Protocol*), de acordo com seu relatório técnico citado por [KOP 94a], é um protocolo do tipo controle central. Durante a execução deste protocolo é designado um mestre, responsável pelo controle do acesso ao meio de comunicação. Quando se trata de um sistema capaz de manter multi-mestres, onde um é o central e os demais são os reservas; se o mestre central falhar, algum outro nodo (mestre reserva) deve assumir a sua posição para controlar o acesso ao meio de comunicação.

Quando um sistema controlado por um protocolo FIP é configurado, uma lista estática de nomes de variáveis, incluindo sua periodicidade, é gerada para o mestre central. Então, o mestre periodicamente difunde o nome de uma variável desta lista no meio (canal). O nodo que produziu esta variável responde com uma difusão do seu conteúdo. Todos os outros nodos escutam esta difusão no canal e alteram o conteúdo da referida variável, caso seja necessário. A devida operação de todas as estações ligadas ao canal de comunicação é monitorada por temporizadores, visando garantir com isso sua correta execução em tempo real.

O protocolo FIP oferece ainda um mecanismo para transferência de dados esporádicos no canal, além dos periódicos. Quando um escravo difunde sua variável periódica, depois de ter sido solicitado que fizesse isto pelo mestre, ele pode informar ao mestre (através de um bit no cabeçalho da sua mensagem) que também possui alguns dados esporádicos para transmitir. Se houver algum tempo disponível após o tráfego normal (de mensagens periódicas) no canal, o mestre elegerá a requisição de um dos escravos solicitantes para que este realize a transmissão de sua mensagem esporádica aos demais nodos.

Este protocolo possui uma elevada capacidade de detectar erros, sendo que algumas versões deste protocolo implementam CRC. É alta também a capacidade de tolerar eventuais falhas ocorridas, inclusive quando estas envolverem o mestre, que pode ser dinamicamente substituído pelos reservas, não impedindo a continuidade de execução do sistema. Além disso, a operação de cada nodo participante do sistema é monitorada através de temporizadores, para que seja possível detectar rapidamente se algum deles não estiver operando conforme o esperado. Entretanto, não implementa a retransmissão automática de mensagens perdidas durante a sua execução [FIE 96].

Por outro lado, de acordo com a análise realizada por Kopetz [KOP 94a], o protocolo FIP não apresenta praticamente nenhuma flexibilidade quanto às aplicações que podem utilizá-lo.

Conforme a descrição aqui apresentada, pode-se considerar que, de acordo com as características anteriormente citadas como relevantes, o protocolo FIP apresenta o seguinte perfil:

- Detecção de erros: possui uma elevada capacidade de detectar erros, algumas versões deste protocolo implementam CRC, e os nodos são constantemente monitorados.
- Entrega: o uso de temporizadores implica que os tempos de entrega de mensagens sejam conhecidos e bem limitados.
- Flexibilidade: não apresenta praticamente nenhuma flexibilidade de aplicações que possam utilizá-lo.
- Previsibilidade: quando se trata de mensagens periódicas, é previsível, visto que sua periodicidade é estabelecida na configuração do sistema; entretanto, as mensagens esporádicas não são previsíveis.
- Sincronismo: o mestre realiza o sincronismo entre os nodos, através de seu controle central.
- Tolerância a Falhas: é capaz de tolerar eventuais falhas (inclusive do mestre), porém não retransmite mensagens.

4.3.4 Protocolo LON

O LON (*Local Operating Network*), segundo uma visão geral do protocolo citada por Kopetz [KOP 94a], trata-se de um protocolo da classe CSMA/CD, que apresenta uma grande flexibilidade quanto às aplicações que podem utilizá-lo. Protocolos desta categoria controlam o acesso distribuído ao meio de acesso, sem requerer que qualquer tipo de controle central seja utilizado para isto. O protocolo aqui considerado, em particular, segue o paradigma de projeto de sistemas tempo real que utiliza a arquitetura disparada por eventos.

Este protocolo foi desenvolvido para controlar a comunicação em sistemas tempo real utilizados na automação de construções, mas hoje em dia seu uso já está bastante difundido em outras áreas de aplicação. Sendo assim, ele é extremamente flexível no que se refere ao tipo de aplicações que podem utilizá-lo. O protocolo LON é amplamente utilizado em sistemas dinâmicos, onde a estratégia de "melhor esforço" é utilizada.

O seu funcionamento depende de um gerador de números aleatórios. Assim, a idéia básica deste protocolo é iniciar a transmissão das mensagens no tempo determinado por um gerador de números randômicos. Com isso, busca-se reduzir a probabilidade de ocorrerem colisões no início da transmissão da mensagem e recuperar-se rapidamente destas colisões, caso elas venham a ocorrer.

Desta forma, um nodo que deseja transmitir alguma mensagem sempre acessa o canal em um tempo, determinado aleatoriamente, depois da transmissão da mensagem anterior ter sido concluída. O tamanho desta janela de tempo é dado em função da carga do canal de comunicação, visando minimizar a probabilidade de ocorrer uma colisão quando houver uma carga alta. Assim, este mecanismo oferece um controle estocástico do fluxo de mensagens.

Quanto à capacidade de detectar erros, tem-se conhecimento que algumas versões do protocolo utilizam técnicas comuns de detecção de erros como, por exemplo,

o CRC. Entretanto, esta não é uma prática utilizada na versão original do protocolo LON. Por outro lado, no que se refere à tolerância a falhas, é conhecida a sua capacidade de retransmitir mensagens perdidas. Além disso o tempo de espera gerado aleatoriamente é utilizado visando evitar colisões de mensagens e permitir que, caso elas venham a ocorrer, seja possível recuperar-se rapidamente [LON 96].

Conforme a descrição aqui apresentada, pode-se considerar que, de acordo com as características anteriormente citadas como relevantes, o protocolo LON apresenta o seguinte perfil:

- Detecção de erros: tem-se conhecimento que algumas versões do protocolo utilizam técnicas comuns de detecção de erros como, por exemplo, o CRC. Entretanto, esta não é uma prática utilizada na versão original do protocolo.
- Entrega: por ter sido projetado especificamente para aplicações tempo real, supõe-se que seu tempo de entrega seja conhecido e limitado, para que possa permitir a operação normal do sistema, sem atrasos significativos. Entretanto, não foram obtidas informações sobre os mecanismos e facilidades que controlam este aspecto.
- Flexibilidade: é extremamente flexível, o que permite que possa ser utilizado em uma grande gama de aplicações tempo real. O protocolo LON é amplamente utilizado em sistemas dinâmicos, onde a estratégia de "melhor esforço" é utilizada.
- Previsibilidade: os protocolos disparados por eventos já não são suficientemente previsíveis. Este, em especial, por trabalhar com um gerador de números aleatórios, não pode ser considerado como sendo previsível e sim, estocástico.
- Sincronismo: o fato de ser disparado por eventos torna esta característica desnecessária no protocolo.
- Tolerância a Falhas: é conhecida a sua capacidade de retransmitir mensagens perdidas. Além disso, o tempo de espera gerado aleatoriamente é utilizado visando evitar colisões e permitir que, caso elas venham a ocorrer, seja possível recuperar-se rapidamente. Entretanto, nada foi encontrado na referência a respeito de implementação de técnicas específicas de tolerância a falhas pelo protocolo LON.

4.3.5 Protocolo Profibus

O Profibus (*Process Field Bus*), segundo sua descrição citada por Kopetz [KOP 94a], trata-se de um protocolo do tipo controle de *token*. Em sistemas onde a comunicação é controlada por *token* - denominados *token bus* - o direito de transmissão é definido por uma mensagem especial de controle, a mensagem *token*. O nodo que possuir esta mensagem *token*, em um dado momento, é o que poderá utilizar o canal para transmitir.

Esta estratégia induz à formação de um anel lógico no sistema distribuído, pois a posse do *token* deve inicialmente pertencer a um nodo, ser passado aos demais e voltar ao inicial, para uma nova rodada.

Dois parâmetros de controle de tempo determinam a resposta de um sistema *token bus*, que são monitorados por um mecanismo de controle de tempo (*timeout*) são eles:

- Tempo de Ocupação do *Token* (THT - *Token Hold Time*): determina o período de tempo mais longo durante o qual um nodo pode ocupar ("segurar") o *token*.
- Tempo de Rotação do *Token* (TRT - *Token Rotation Time*): determina o período de tempo mais longo designado para que o *token* dê uma volta completa, passando por todos os nodos.

Um problema sério a ser considerado em qualquer sistema do tipo *token bus* refere-se à perda da mensagem de *token*, que pode ocorrer, por exemplo, se a estação (nodo) que possui o *token* falhar. Nesta situação, o tráfego de mensagens na rede ficará temporariamente perturbado até que algum nodo detecte o silêncio do canal através da monitoração por um mecanismo de controle de tempo. Entretanto, o processo de recuperação do *token* é, na maioria das vezes, bastante complexo e consome muito tempo, o que pode ser prejudicial em sistemas tempo real.

Conforme a descrição aqui apresentada, pode-se considerar que, de acordo com as características anteriormente citadas como relevantes, o protocolo Profibus apresenta o seguinte perfil:

- Detecção de erros: é implementada apenas através da monitoração do canal por um mecanismo de controle de tempo.
- Tempo de Entrega: é conhecido e limitado pelo THT, que vai liberar o canal para outro nodo.
- Flexibilidade: devido ao uso de *token* como forma de controle do canal, a flexibilidade quanto às aplicações que possam utilizar este protocolo é relativamente baixa.
- Previsibilidade: pela existência do THT e do TRT para a entrega de mensagens, o protocolo é previsível.
- Sincronismo: há um sincronismo entre os nodos, determinado pelo processo de passagem do *token*.
- Tolerância a Falhas: caso o nodo que possua o *token* falhe a execução do sistema é interrompida, até que esta situação seja detectada e recuperada. Entretanto, o protocolo continua operando normalmente quando ocorrem falhas em outros nodos, apenas isolando o nodo falho.

4.3.6 Protocolo TTP

O TTP (*Time-Triggered Protocol*) [KOP 94] é um protocolo de comunicação projetado especificamente para ser utilizado em arquiteturas disparadas por tempo.

Entretanto, ele também usa comunicação disparada por eventos para mensagens esporádicas quando ele muda rapidamente para um modo de emergência, por exemplo.

Este protocolo oferece os serviços requeridos para a implementação de um sistema tempo real tolerante a falhas, tais como: transmissão previsível de mensagens, mensagem de notificação de recebimento (*acknowledgment*) na comunicação em grupo, sincronização de mensagens, *membership* e gerenciamento de redundância. Para tanto, não necessita de mensagens extra e gera apenas um pequeno aumento no tamanho da mensagem.

Trata-se de um protocolo do tipo TDMA, os protocolos desta categoria implementam uma estratégia de acesso distribuído estático ao meio de comunicação. O direito de usar o canal para transmissão é controlado pela passagem do tempo real. Assim, é requerido que uma base de tempo global esteja disponível para todos os nodos do sistema, ou seja, um relógio tempo real. Na estratégia TDMA o tempo é fracionado e as frações são sincronizadas com o relógio, assim cada nodo possui a sua fatia de tempo para transmitir, com um período fixo de duração, o que caracteriza uma comunicação determinística e previsível.

Dessa forma, no TTP a capacidade total do canal é subdividida estaticamente em um número de frações. Uma fração de envio é atribuída a cada nodo do sistema. A seqüência de frações de envio em um conjunto de nodos é denominada uma rodada TDMA. Cada nodo pode enviar um pacote contendo uma ou mais mensagens em cada rodada TDMA. Se não houver nenhum dado para enviar, então um pacote vazio deve ser transmitido. Um pacote periódico contém informação de notificação de recebimento de mensagens (*acknowledgment*), que também é interpretado como uma sinalização de que o nodo está operante. Eventualmente, quando um nodo possui muitas mensagens ou mensagens urgentes para transmitir, pode lhe ser designada uma fração de tempo maior ou pode ser permitido que ele transmita mais de um pacote em uma rodada TDMA, recebendo mais de uma fração de tempo.

Uma sincronização tolerante a falhas entre os relógios locais dos nodos é realizada no TTP sempre que for percebida uma diferença entre o relógio do nodo que envia e do nodo que recebe a mensagem. Isto é possível através da comparação entre o tempo real de chegada do pacote e o tempo planejado a *priori* para a chegada do mesmo.

O protocolo TTP foi desenvolvido para aplicações automotivas de segurança crítica, mas hoje seu horizonte de aplicações é bem mais extenso. O objetivo do seu projeto era garantir o devido cumprimento dos prazos e tolerar falhas, integrando todos os serviços necessários neste tipo de aplicações. Assim, por exemplo, o TTP oferece as seguintes garantias:

- Transporte de mensagens: é previsível, ou seja, apresenta poucas variações, mesmo na presença de condições de sobrecarga ou de falhas.
- Tolerância a falhas: tolera as falhas permanentes e transitórias, previstas no seu modelo, que ocorram no canal ou nos nodos, sem violar os prazos especificados. Apresenta uma latência mínima para detecção de erros, implementando CRC.

- Manipulação de blecautes temporários: detecta e manipula rapidamente qualquer interferência na operação do sistema, proveniente de algum distúrbio externo, que pode causar uma queda.
- Sincronização de relógios: estabelece um tempo global do sistema, entretanto, quando os relógios locais dos nodos não estiverem de acordo é executada uma estratégia de sincronização.
- Serviço de *membership*: é a base para implementar *multicast* atômico e gerenciamento de redundância, além de permitir a abstração dos nodos do tipo falha-silencia.
- Gerenciamento de redundância distribuída: remove os nodos falhos e (re)integra os nodos reservas ou reparados.
- Suporte para mudança de modo de operação rápida: permite uma mudança rápida e consistente de um modo de operação para outro, que correspondem a: inicialização, operação normal e emergência.
- Mínimo *overhead*: oferece os serviços especificados com um mínimo *overhead* no tamanho e no número das mensagens.
- Extrema flexibilidade com previsibilidade: oferece alta flexibilidade de aplicações que podem utilizá-lo, enquanto mantém o determinismo (previsibilidade na transmissão de mensagens).

Em vista destas características apresentadas, a análise daquelas consideradas importantes na seção 4.1 deste trabalho, já pode ser considerada como realizada, mas será apresentado um breve resumo do seu perfil:

- Detecção de erros: tempo de latência de detecção mínimo e implementação de mecanismos como CRC.
- Entrega: tempo conhecido e limitado, devido a filosofia TDMA utilizada.
- Flexibilidade: extrema flexibilidade quanto à área de aplicação.
- Previsibilidade: pouca variação mesmo na presença condições de sobrecarga ou falhas.
- Sincronismo: realiza temporariamente sincronização entre os relógios locais dos nodos.
- Tolerância a Falhas: tolera falhas permanentes ou temporárias no canal ou nos nodos.

Este protocolo é considerado bastante regular e, desta forma, oferece boas estratégias de detecção de erros e de tolerância a falhas. Como as aplicações tempo real, de uma forma geral, igualmente são muito regulares e requerem um mínimo tempo de latência na sua execução, os requisitos destas aplicações estão normalmente em concordância com as características dos serviços que são oferecidos pelo protocolo TTP, o que permite a sua utilização. Para maiores detalhes a respeito da implementação do TTP, sugere-se a leitura de artigo de autoria do responsável pelo projeto deste protocolo [KOP 94].

4.4 Mecanismos de Comunicação do Sistema Operacional Tempo Real

Além dos protocolos de comunicação de um sistema tempo real, propriamente ditos, foi estudado um dos mecanismos de comunicação oferecidos pelo sistema operacional. Uma abordagem desenvolvida An e Chu [AN 86] foi estudada e constatou-se que ela garante o uso de protocolos de compromisso ou de aceitação (*commit*) em sistemas tempo real. Esta abordagem será brevemente apresentada na presente seção. Trata-se de uma adaptação aos protocolos de compromisso, cujo resultado é um protocolo resiliente, de baixo custo, não-bloqueante, realizado em uma única fase, tolerante a múltiplas falhas de nodos e cuja utilização resulta em um mínimo *overhead* à aplicação.

Em Singhal e Shivaratri [SIN 94], é feita uma análise dos mecanismos de comunicação do sistema operacional, onde são discutidas algumas classes de protocolos, tais como os de aceitação ou compromisso (*commit*) e os de votação, usados no projeto de sistemas tolerantes a falhas. Em um primeiro instante, devido a forma como tais protocolos são executados, poder-se-ia pensar que eles não seriam adequados a aplicações tempo real. Entretanto, com algumas adaptações, eles são perfeitamente utilizados, como é o caso do protocolo de compromisso (*commit*) proposto por An e Chu [AN 86], que será aqui apresentado.

Parte-se do princípio que a sobrevivência de um sistema tempo real distribuído depende das múltiplas cópias dos arquivos, dispersas entre vários nodos. Quando é realizada uma atualização em uma das cópias, todas as demais devem receber esta atualização. Assim, todas as cópias devem ser idênticas para garantir a integridade dos dados do sistema, ou seja, a consistência múltipla. Entretanto podem surgir problemas se o tempo for diferente para a atualização de cópias localizadas em nodos diferentes ou se um dos nodos que participa do processo de atualização, por conter uma cópia do arquivo que está sendo atualizado, falhar. Para realizar esta atualização múltipla, onde existe um consenso entre os nodos participantes, são utilizados os mecanismos de comunicação do sistema operacional.

Quanto aos protocolos de votação pouco tem a ser dito, visto que não foi encontrada nenhuma referência à sua utilização em sistemas tempo real. A princípio, este protocolo é mais robusto do que o de compromisso. Entretanto, cada ação de leitura e escrita a ser realizada na cópia de um arquivo deve ser autorizada por um número expressivo de nodos do sistema. Isto implica que se for um sistema grande ou onde o tempo de execução é fundamental (tempo real), ele não pode ser utilizado, pois seu *overhead* é bastante significativo.

Os protocolos de votação usualmente mascaram falhas que venham a ocorrer no sistema e continuam operando, sem que esta falha seja percebida pelo restante do sistema. Já os protocolos de compromisso normalmente implementam um comportamento bem definido diante de falhas, devendo executar uma função pré-especificada, que pode ser uma alternativa àquela executada durante a sua operação normal.

A adaptação aqui apresentada é proposta com o intuito de permitir que este protocolo seja utilizado para garantir a consistência de um sistema tempo real distribuído. A abordagem tradicionalmente empregada, utiliza versões em duas fases e bloqueante (diante da ocorrência de falhas) na operação de consistência mútua entre as

cópias de um arquivo. O problema desta versão é que se o nodo que está coordenando a atualização falhar, perde-se um tempo relativamente alto, que pode ser crucial em sistemas tempo real, até que o coordenador se recupere e complete o protocolo. Enquanto isso nada pode ser feito no sistema, visto que é bloqueado o uso da cópia do arquivo que está sendo atualizado e os demais nodos nada podem fazer.

O conceito básico desta adaptação do protocolo de compromisso é que se uma atualização é proposta por um nodo operacional, todos os demais nodos operacionais que mantêm uma cópia do arquivo devem receber a atualização, mesmo na presença de múltiplas falhas. Assim, a consistência mútua entre as cópias do arquivo é preservada por todas atualizações efetuadas.

Para detectar a ocorrência de nodos falhos, todos os nodos operacionais devem enviar periodicamente uma mensagem do tipo: "Eu estou vivo" aos demais nodos. Este mecanismo torna desnecessária a implementação de outros mecanismos de controle do tempo ou mensagens de notificação do recebimento (*acknowledgment*) para detecção de erros no sistema.

No início da execução deste protocolo os nodos são numerados linearmente e as atualizações devem ser difundidas de acordo com esta seqüência. Se apenas alguns nodos recebem a atualização antes de ocorrer uma falha no coordenador, a consistência é garantida, sem perda de tempo, da seguinte forma: o nodo cujo número de seqüência for menor retransmite a última atualização recebida do nodo falho. Esta retransmissão deve ser feita igualmente, de acordo com a seqüência, para garantir a resistência a múltiplas falhas. Assim, se este nodo que estiver retransmitindo a atualização falhar, o nodo operacional com o menor número de seqüência assume a sua posição e retransmite a última atualização do nodo falho e assim é feito sucessivamente, até completar a atualização de todas as cópias do arquivo, garantindo a consistência mútua.

Dessa forma, este protocolo é capaz de recuperar-se rapidamente e continuar o seu serviço, sem prejudicar a execução tempo real do sistema, desde que seus mecanismos de detecção de erros sejam suficientemente rápidos, ou seja, que as mensagens do tipo "Eu estou vivo" sejam geradas com uma alta freqüência. Um de seus maiores problemas é a repetição de mensagens que um nodo pode receber (e deve descartar) e o fluxo de mensagens que é gerado por isso.

Entretanto, quando ocorrerem situações de atualização concorrente de arquivos, geradas a partir de nodos diferentes, é necessário que sejam utilizados protocolos próprios (como o de escrita exclusiva e de bloqueio do nodo) juntamente com o protocolo aqui apresentado.

A maior vantagem na utilização deste protocolo é que ele requer apenas um pequeno *overhead* adicional somente quando ocorrem falhas, sendo desta forma muito atraente para ser utilizado em aplicações tempo real. Além disso, ele é bastante simples de ser implementado.

4.5 Análise da Comunicação Tempo Real

Os protocolos de comunicação anteriormente apresentados e analisados, são comparados nesta seção, segundo algumas características importantes, que é desejável

que eles apresentem. As principais características consideradas a respeito dos protocolos de comunicação envolvem as questões levantadas na seção 4.1 deste trabalho. O mecanismo de comunicação estudado não entra nesta comparação, pois apresenta fins diferentes dos protocolos de comunicação e deveria ser comparado com outros mecanismos do sistema operacional.

A análise realizada, conforme apresentada na Tabela 4.1, atribui graus a cada um dos protocolos estudados, dessa forma:

- Grau 2: significa que o protocolo apresenta esta característica sempre.
- Grau 1: significa que esta característica está parcialmente presente no protocolo (com restrições).
- Grau 0: significa que esta característica não está presente no protocolo.

TABELA 4.1 - ANÁLISE COMPARATIVA DOS PROTOCOLOS TEMPO REAL

Características	ARINC 629	CAN	FIP	LON	Profibus	TTP
Detecção de erros	<2>	<2>	<2>	<1>	<1>	<2>
Entrega	<2>	<1>	<2>	<1>	<2>	<2>
Flexibilidade	<1>	<2>	<0>	<2>	<1>	<2>
Previsibilidade	<2>	<0>	<1>	<0>	<2>	<2>
Sincronismo	<2>	<0>	<2>	<0>	<2>	<2>
Tolerância a Falhas	<2>	<2>	<2>	<1>	<1>	<2>

5 Sistemas Operacionais Tempo Real

Um sistema operacional, é um conjunto de programas, cujo propósito pode ser visto sob os seguintes aspectos, também denominados "princípios" por Finkel citado por Fricks [FRI 89]:

- alocar recursos (como espaço em disco, tempo de processamento, dispositivos de E/S) e processos;
- mascarar detalhes do hardware ao usuário final;
- prover um ambiente de execução mais amistoso aos processos e aos seus respectivos usuários.

Porém, algumas necessidades adicionais devem ser suportadas pelo sistema operacional para que seja possível atender aos processos críticos no tempo (PCTs) pertencentes a um sistema tempo real. Nesse sentido, um sistema operacional tempo real deve principalmente ser capaz de satisfazer os requisitos referentes ao tempo de atendimento a um evento ocorrido no ambiente com o qual ele se relaciona [LOB 89].

As aplicações tempo real dependem totalmente do sistema operacional para manipular eventos múltiplos com limites de tempo fixos. Assim, todas atividades do sistema operacional, principalmente de gerenciamento de recursos - tais como: escalonamento de processos no processador, escrita de arquivos em disco, envio de dados através da rede, entre outros - devem funcionar de forma mais rigorosa e transparente nos sistemas tempo real do que nos outros tipos de sistemas.

Nesse capítulo será apresentada uma visão geral a respeito de sistemas operacionais tempo real (SOTRs), caracterizando-os, inclusive com utilização de mecanismos de tolerância a falhas. Serão apresentados ainda alguns exemplos dos sistemas operacionais tempo real mais conhecidos e utilizados, estes serão analisados e, após, será realizada uma comparação entre eles, que resultará em uma taxonomia.

5.1 Características de Sistemas Operacionais Tempo Real

De uma forma geral, os sistemas tempo real devem ter a capacidade de fornecer as respostas aos estímulos recebidos do ambiente externo em tempo hábil e de realizar os tratamentos convenientes das diversas variáveis dos processos com os quais interagem. Em vista disso, o sistema operacional deve estar apto a oferecer o suporte necessário a algumas características específicas dos sistemas tempo real como, por exemplo [FRI 89]:

- Prazos limites inflexíveis: as requisições para todos os processos devem ser atendidas dentro de prazos limites inflexíveis (*hard deadlines*), caso contrário, tem-se como conseqüência um dano irreparável no sistema.
- Ocorrência de eventos: em alguns casos, a execução dos processos está associada à ocorrência de eventos (nas arquiteturas disparadas por eventos, por exemplo), que podem gerar ativações externas (via interrupção externa,

disparada por eventos no ambiente físico), explícitas (via ativação por outro processo) ou iterativas (ativação em períodos regulares constantes) dos mesmos.

- Conclusão da execução: a execução de um processo crítico no tempo deve ser concluída sempre antes da ocorrência de novas requisições a este mesmo processo.
- Processos independentes: os processos, inclusive os críticos no tempo, não são inter-relacionados por relações de precedência, ou seja, o conjunto de processos é não-ordenado.
- Compartilhamento de recursos do sistema: cada processo pode necessitar de outro recurso além do tempo de processador durante sua execução, o que representa o compartilhamento de vários recursos disponíveis no sistema tempo real, por exemplo, periféricos, variáveis compartilhadas e seções críticas.

Estas características normalmente são fornecidas à aplicação através do sistema operacional, que provê o gerenciamento de tais exigências inerentes ao funcionamento correto dos sistemas tempo real.

Dessa forma, pode-se dizer que um sistema tempo real compõe-se de um conjunto de programas separados - denominados de tarefas ou processos - que podem ser executados de forma concorrente e são interrompidos pela chegada de uma interrupção assíncrona proveniente do ambiente externo [RIP 93]. Entretanto, a execução descontínua de uma tarefa é transparente para a própria tarefa, já que um sistema tempo real opera sob o controle do sistema operacional, que representa uma parte de código co-residente e é o responsável por este gerenciamento. Assim, o sistema operacional é o programa mestre: é ele quem decide que tarefa deve executar no processador em um dado momento e realiza as trocas de contexto requeridas, quando necessário. Além disso, é o sistema operacional que manuseia as interrupções do equipamento, que indica a disponibilidade de novas entradas nos dispositivos periféricos e que determina quando uma tarefa de resposta deve ser ativada. Em resumo, o sistema operacional é o responsável por organizar todo o trabalho do processador.

Pode-se reforçar a idéia de que um sistema tempo real consiste de um conjunto de tarefas - programas que competem pelo acesso à UCP - e do sistema operacional - um programa mestre, co-residente, que organiza este acesso à UCP. Devido a esta troca de tarefas utilizando a UCP, um dos serviços freqüentemente realizados pelo sistema operacional é o de trocas de contexto que, apesar de ser rápida, representa um sobrecusto ao sistema, pois diminui o tempo disponível para o processamento da tarefa. Apesar disso, essa pequena perda é facilmente restituída, pois o sistema operacional deve assegurar que a UCP seja mantida sempre ocupada, enquanto existir qualquer tarefa que tenha trabalho a fazer.

Outra tarefa importante, que cabe ao sistema operacional realizar, é o escalonamento das tarefas em tempo real. Este escalonamento deve ser dinâmico, na maioria das situações, pois as tarefas, muitas vezes, são ativadas por eventos externos que não podem ser previstos em tempo de compilação. É o escalonamento tempo real que determina quando as entradas são esperadas e quando as saídas devem ser geradas. Em vista disto, é fundamental que técnicas de tolerância a falhas sejam aplicadas a ele,

evitando que sejam geradas saídas incorretas ou em tempos superiores ao prazo limite especificado.

Ainda, devido a este papel central sobre todas as tarefas e interrupções tempo real, o sistema operacional encontra-se na melhor posição para prover serviços centralizados e para controlar o acesso às facilidades do hardware e do software que são compartilhadas pelas diversas tarefas do sistema. Portanto, controle do tempo, entrada e saída periférica e alocações de memória são domínios do sistema operacional. Entretanto, além desses serviços, é fundamental que as seguintes propriedades estejam presentes em sistemas operacionais onde são executadas aplicações tempo real, principalmente quando envolvem controle de processos [TOZ 83]:

- Atividades concorrentes: o sistema operacional deve permitir a existência e o gerenciamento de tais tarefas, que são possíveis através da utilização de recursos de multiprogramação.
- Política de alocação de recursos: os critérios de uso do processador, bem como dos outros recursos compartilhados, devem ser definidos e implementados através de mecanismos como por exemplo, a adoção de prioridades para as tarefas, atribuídas conforme sua função e tempo de processamento. Dessa forma, tarefas "mais urgentes" devem obter atendimento mais rápido do que as demais, quando houver conflito no uso do processador.
- Relógio de tempo real: este dispositivo - capaz de gerar interrupções periódicas regulares - permite realizar e controlar as operações dependentes do tempo. Por outro lado, o horário e a data mantidos no sistema são utilizados para o registro do histórico relativo às informações coletadas.
- Mecanismo de sincronização: a existência de diversas tarefas que compartilham os recursos do sistema, gera a necessidade de sincronização entre elas; desta forma os recursos compartilhados poderão ser utilizados sem conflitos e as tarefas poderão ser ativadas no momento conveniente, garantindo o funcionamento adequado do sistema tempo real.
- Mecanismo de comunicação: normalmente as tarefas existentes comunicam-se mutuamente para coordenar suas atividades. As filosofias básicas mais utilizadas para comunicação implementam mecanismos de: memória compartilhada e troca de mensagens.
- Interface com instrumentos para aquisição de dados e para atuação no ambiente: o sistema operacional deve ter recursos para incorporar, sem muitos transtornos, a manipulação com tais instrumentos (como sensores e atuadores), cujas operações de E/S podem não ter sido previstas quando ele foi fabricado ou instalado.
- Suporte ao armazenamento e à manipulação de dados: muitas vezes os dados coletados constituem um volume grande a ser tratado; por isso, o sistema operacional deve disponibilizar recursos de sistemas de arquivos ou banco de dados.
- Ambiente para desenvolvimento de programas: além de controlar processos, o sistema deve ter recursos para preparar novos programas tempo real utilizados para controle. Desta forma, o ambiente fornecido deve colocar à disposição

do programador ferramentas para construir, depurar e testar adequadamente estes novos programas.

Por outro lado, existem algumas necessidades funcionais básicas, que vêm a complementar estas propriedades e que devem ser atendidas por um sistema operacional tempo real; são elas [FRI 89]:

- Responder a eventos assíncronos: um sistema tempo real possui a habilidade de processar múltiplos eventos que podem ocorrer, no ambiente físico, em tempos e seqüências incapazes de serem previstas estaticamente. O sistema operacional, na maioria das vezes, não pode prever quando ocorrerão tais eventos externos, mas deve estar preparado para executar o processamento necessário quando ele vier a ocorrer.

- Permitir a coordenação do processamento de eventos: alguns eventos, devido à sua natureza, podem exigir maior urgência de atendimento; esta urgência é relativa a outros processamentos e é determinada pelos requisitos da aplicação. O sistema operacional deve permitir a atribuição de prioridades a eventos e ao seu processamento. Além disso, deve ser capaz de sincronizar o processamento de eventos distintos, visando a sua coordenação.

- Interagir com o relógio tempo real: muitas vezes é de extrema importância que o operador do sistema saiba o que está acontecendo e em que instante determinado evento ocorreu. Assim, há a necessidade de se ter um relógio tempo real disponível, bem como as condições necessárias para sua interação com o sistema operacional.

- Facilitar a adição de dispositivos periféricos: o sistema operacional não deve exigir software de interface com dispositivos de E/S dependentes do sistema, tornando mais fácil adicionar novos equipamentos de E/S. Isso ocorre porque sistemas tempo real são caracterizados pela elevada diversidade de dispositivos a ele conectados.

- Apoiar a implementação de ferramentas de depuração *on-line*: o procedimento de depuração nem sempre é imediato em ambientes tempo real. Porém seria de grande utilidade a implementação de métodos de visualização do comportamento dinâmico do sistema e de manipulação interativa do ambiente tempo real, a serem utilizados como ferramentas de depuração. Entretanto, para isso, deve-se obter uma efetiva assistência do sistema operacional, no sentido de facilitar a implementação para que possa ser realizada esta depuração.

Além disso, em sistemas tempo real é importante ter segurança de que o sistema operacional comporte-se conforme especificado, em todas as condições operacionais possíveis, ou seja, que ele seja previsível. Então, uma das mais importantes necessidades de um sistema operacional tempo real é contribuir na obtenção de graus elevados de confiabilidade e de disponibilidade do sistema como um todo, o que garante a sua segurança.

Para que seja possível realizar uma análise comparativa entre os diversos sistemas operacionais estudados serão consideradas como importantes, neste trabalho, as seguintes características:

- Contexto: esta característica refere-se à capacidade do sistema operacional de trocar rapidamente de contexto (alterar o estado do sistema), salvando o anterior, quando for necessário.
- Portabilidade: esta característica indica se o sistema operacional é portátil para qualquer tipo de ambiente tempo real.
- Previsibilidade: esta característica indica se o comportamento do sistema operacional como um todo é previsível, principalmente em relação aos prazos de tempo.
- Reconfigurabilidade: esta característica indica se o sistema operacional é dinamicamente reconfigurável após a detecção de uma falha.
- Redundância: esta característica indica se o sistema operacional apresenta gerenciamento de redundância de componentes.
- Tolerância a Falhas: esta característica refere-se à capacidade do sistema operacional tolerar falhas e envolve todas as etapas de tolerância a falhas (detecção de erros, avaliação e confinamento de danos, recuperação de erros e tratamento de falhas).

5.2 Tolerância a Falhas nos Sistemas Operacionais Tempo Real

Devido ao fato do sistema operacional possuir a responsabilidade de controlar o funcionamento geral do sistema, ele deve ser suficientemente confiável para garantir a execução correta de todas as atividades impostas ao sistema tempo real. Como conseqüência disso, é fundamental que técnicas de tolerância a falhas sejam aplicadas a ele.

A necessidade de um grau de segurança de funcionamento elevado é importante particularmente em aplicações ligadas ao controle de processos críticos no tempo, onde um defeito computacional, e até um simples atraso no tempo de resposta, pode resultar em prejuízos econômicos, deterioração da produção, perda de controle de qualidade ou danos ao ambiente. Além disso, em casos extremos, vidas humanas podem ser prejudicadas. Para oferecer a garantia de segurança em um sistema, deve-se considerar seus parâmetros de confiabilidade e disponibilidade, que são as medidas de segurança (dependabilidade) de um sistema.

Conforme exposto anteriormente, em um sistema tempo real, as respostas devem ater-se às especificações temporais e funcionais. Entretanto, o envio de um comando errado ao processo físico é tão falho quanto um atraso no seu envio. Além disso, caso seja detectada uma resposta errônea, mesmo antes da mesma atuar no ambiente controlado, possivelmente não haverá tempo disponível para voltar atrás e refazer toda a computação. Dessa forma, como o sistema operacional é o responsável pelo controle geral do sistema, inclusive no que se refere à interação com o ambiente físico que está sendo controlado, ele deve ser suprido de técnicas de tolerância a falhas para garantir seu funcionamento correto e contínuo dentro do tempo especificado.

Por outro lado, caso venha a ocorrer algum tipo de falha não-mascarável, o sistema operacional deve ser capaz de buscar uma solução alternativa tal como conduzir

a execução imediatamente para um estado computacional seguro. Entretanto, a decisão de qual estado pode ser considerado seguro varia dependendo da aplicação; assim, por exemplo:

- Caso 1: Em um controle de forno industrial, que tem como estado inicial "desligado" e como estado atual "aquecendo", na ocorrência de uma falha, para garantir segurança, não é interessante, manter-se o estado atual após detectada a falha. Isto por que, desta forma, o forno irá continuar aquecendo, sem nenhum controle, até que possa vir a explodir. Então, o mais seguro, nesse caso específico, é retornar o sistema ao seu estado inicial, ou seja, o de "desligado".
- Caso 2: Em um controle de avião, onde o estado inicial é "no chão" e o estado atual é "voando", na ocorrência de uma falha, ao contrário do anterior, para garantir segurança, deve-se manter o estado atual, do sistema, ou seja "voando", após a falha ter sido detectada. Dessa forma, evita-se uma queda abrupta do avião devido a algum descontrole que possa ocorrer na mudança para o estado "no chão".

5.3 Exemplos de Sistemas Operacionais Tempo Real

No decorrer da realização deste trabalho, foram estudados alguns sistemas operacionais conhecidos e disponíveis que implementam recursos de tempo real, os quais serão apresentados nesta seção. Os sistemas operacionais aqui abordados são os reunidos durante a fase de coleta bibliográfica, a qual foi dirigida a sistemas candidatos - pelas características anunciadas - ao uso em sistemas tempo real. Certamente não se esgota aqui a lista de possibilidades ou o universo de sistemas de suporte disponíveis, mas acredita-se que a abordagem apresentada e a análise subseqüente podem auxiliar o leitor a estender os procedimentos a outros sistemas.

5.3.1 Sistema Operacional DEDOS

O sistema operacional DEDOS (*DEpendable Distributed Operating System*) apresentado por Roosmalen [ROS 94], surgiu a partir de um projeto desenvolvido pelo Departamento de Ciência da Computação da Universidade de Tecnologia de Eindhoven. O objetivo deste projeto era definir e implementar aspectos de uma plataforma de execução para sistemas tempo real distribuídos, inclusive os de controle e supervisionamento. A idéia desta plataforma é permitir um alto grau de abstração e oferecer a possibilidade de construir sistemas desta categoria com a maior produtividade, flexibilidade e confiabilidade possíveis.

No ambiente DEDOS, é feita uma distinção entre as tarefas tempo real existentes em um sistema: as críticas (*hard*), que correspondem aos processos críticos no tempo (PCTs) e são disparadas por tempo; e as brandas (*soft*), que são disparadas por eventos. Estas classes distintas de tarefas recebem um tratamento diferenciado, de acordo com as restrições temporais do sistema. Assim, por exemplo: as tarefas consideradas críticas são

escalonadas estaticamente (*off-line*), enquanto que as brandas são escalonadas dinamicamente (*on-line*).

A plataforma DEDOS tem o propósito de ser utilizada em aplicações de controle de processos, em particular em sistemas embutidos. Tais sistemas são predominantemente construídos de forma isolada, mas a *posteriori* devem ser capazes de executar em conjunto com o restante do sistema, sem que para isto sejam necessários muitos ajustes na configuração do hardware. Dessa forma, a reutilização e a extensibilidade do código e do projeto desenvolvidos para esta plataforma são características consideradas de extrema importância.

Em sistemas tempo real, de uma forma geral, é importante que o desenvolvimento do projeto seja totalmente independente da plataforma onde ele será utilizado, pois são realizadas mudanças freqüentes no hardware de suporte. Assim, o desenvolvimento da aplicação não deve levar em conta aspectos como a velocidade de execução dos processadores, seu número e suas estruturas de interconexão. Nesse sentido, o ambiente DEDOS permite, através de sua linguagem de programação, que as aplicações sejam desenvolvidas de forma independente, o que as torna significativamente portáteis para outros equipamentos.

Além disso, o DEDOS oferece primitivas de multiprogramação através de modelo de processos, onde os processos são considerados como unidades de (re)alocação. Objetos contidos em diferentes processos comunicam-se através do mecanismo de chamada de procedimentos remotos (RPC - *Remote Procedure Call*). É o ambiente DEDOS que controla toda a distribuição do sistema, liberando o programador (ou projetista) desta tarefa, através de suas primitivas relacionadas à distribuição, que são oferecidas pelo próprio sistema operacional, e não pela sua linguagem de programação.

O DEDOS possui várias ferramentas específicas para o controle de tempo da aplicação, que podem ser utilizadas em conjunto com a sua linguagem de programação. Estas ferramentas correspondem, por exemplo, a um analisador temporal e a um escalonador tempo real.

Conforme a descrição aqui apresentada, pode-se considerar que, de acordo com as características anteriormente citadas como relevantes, o sistema DEDOS apresenta o seguinte perfil:

- Contexto: nada foi encontrado na referência base sobre este assunto. Entretanto, como o ambiente a ser utilizado pelo DEDOS possui vários processadores (é distribuído), esta não é uma característica que possa ser considerada importante.
- Portabilidade: por ser um sistema extremamente flexível e o seu projeto ser independente de plataforma de hardware, é considerado significativamente portátil.
- Previsibilidade: em sistemas operacionais desenvolvidos especificamente para utilização em sistemas tempo real, como o DEDOS, esta é uma característica indispensável. Entretanto, esta propriedade não é analisada na bibliografia consultada e o sistema não foi testado de forma a permitir a obtenção de conclusões próprias.

- Reconfigurabilidade: nada foi encontrado na referência sobre esta característica.
- Redundância: como esta é uma característica inerente aos sistemas distribuídos e o DEDOS foi projetado especificamente para estes, supõe-se que ele seja capaz de gerenciar a redundância de elementos no sistema em que está sendo utilizado.
- Tolerância a Falhas: a referência base não comenta nada sobre esta característica, apesar de garantir que o DEDOS permita o desenvolvimento de aplicações tempo real confiáveis.

5.3.2 Sistema Operacional HARTOS

O HARTOS (*Hexagonal Architecture Real-Time Operating System*), conforme descrito por Shin [SHI 92], trata-se de um sistema operacional projetado especificamente para esta arquitetura hexagonal de sistemas tempo real, que consiste de um sistema distribuído formado por nodos conectados por uma rede ponto-a-ponto.

Este sistema operacional possui duas versões. A primeira apresenta características como: dispor de um servidor de nomes distribuído e de serviços para suprir a comunicação entre os processos. A segunda versão possui adicionalmente: comunicação tempo real tolerante a falhas, com difusão generalizada (*broadcasting*) confiável, sincronização de relógios e comunicação de grupos, que não se encontra na versão 1 do HARTOS. Nesta seção será apresentada a segunda versão deste sistema operacional.

Quanto ao subsistema de comunicação esta versão implementa serviços como: uma base de tempo global e um canal de comunicação tempo real para garantir que a comunicação entre os processos seja realizada dentro dos limites de tempo estipulados no projeto do sistema. A base de tempo global, mantida por um algoritmo de sincronização de relógios, é que habilita o serviço de canal tempo real. Estas características permitem que o sistema suporte o tráfego de mensagens normais e de tempo real (que possuem tempo de entrega limitado).

A versão 2 do HARTOS inclui serviços de tolerância a falhas tais como difusão confiável e comunicação de grupo. A difusão confiável é essencial para implementar algoritmos de sincronização de relógios, diagnóstico de falhas distribuído e consenso distribuído quando houver falhas. Supostamente, um nodo não-falho entrega corretamente seu valor privado para todos outros nodos não-falhos; já os nodos falhos descartam, corrompem e possivelmente alteram a informação passada para eles. Então, eles devem ser descartados. Esquemas de diagnóstico distribuído podem ajudar a identificar falhas, mas não são capazes de cobrir todas as falhas possíveis.

O algoritmo de difusão confiável implementado pelo HARTOS funciona basicamente da seguinte forma: múltiplas cópias da mensagem são difundidas de forma generalizada, por caminhos distintos, para todos os nodos do sistema; os nodos que as recebem fazem a distinção entre a mensagem original e as múltiplas cópias, usando um esquema apropriado para o modelo de falhas, semelhante ao esquema de votação

majoritária. Uma espécie de cifragem é utilizada para evitar que ocorra a corrupção de mensagens, além disso o algoritmo suporta a perda de mensagens.

O HARTOS implementa ainda redundância temporal no envio das mensagens, ou seja, uma mensagem pode ser enviada múltiplas vezes para um mesmo nodo. Por outro lado, o seu mecanismo de RPC também pode, algumas vezes, realizar esta chamada de forma redundante. Para garantir a tolerância a falhas, várias cópias do módulo gerenciador de rede são mantidas, de forma redundante, em múltiplos nodos componentes do sistema e a consistência entre estas cópias é mantida através de difusão atômica do seu estado atual.

Quanto à previsibilidade na entrega de mensagens, o canal tempo real controla e garante o tempo decorrido até a entrega das mensagens. Esta característica garante o funcionamento adequado de qualquer sistema tempo real distribuído que venha a ser executado sob o HARTOS. Entretanto, o funcionamento global do sistema não é totalmente previsível. Também é importante ressaltar a sua flexibilidade e portabilidade decorrentes de todas estas facilidades oferecidas para a execução de sistemas desta categoria, desde que se verifique a arquitetura hexagonal distribuída para a qual ele foi projetado.

A reconfiguração da rede, em caso de falha no canal, também é uma tarefa fácil no HARTOS. Ela se baseia no algoritmo utilizado para realizar o balanceamento de carga do sistema, que permite o melhor aproveitamento dos recursos da rede por que é possível selecionar a melhor rota para transmitir uma mensagem. Porém, se ocorrer uma falha em um dos nodos componentes do sistema distribuído, pode haver perda total dos seus serviços, visto que, neste caso, o sistema não é dinamicamente reconfigurável.

Conforme a descrição aqui apresentada, pode-se considerar que, de acordo com as características anteriormente citadas como relevantes, o sistema HARTOS apresenta o seguinte perfil:

- Contexto: nada foi encontrado na bibliografia consultada sobre este assunto. Entretanto, como o ambiente a ser utilizado pelo HARTOS possui vários processadores (é distribuído), esta não é uma característica que possa ser considerada importante.
- Portabilidade: em decorrência das demais características, o HARTOS é considerado flexível e portátil, desde que se verifique a arquitetura para a qual ele foi projetado.
- Previsibilidade: é previsível apenas no tempo de entrega das mensagens, mas não totalmente no funcionamento global do sistema.
- Reconfigurabilidade: a capacidade de reconfiguração dinâmica do sistema é parcial, em caso de falha no canal, porém em caso de falha nos nodos do sistema, pode haver perda total de serviços.
- Redundância: implementa a redundância temporal de mensagens, a redundância de chamadas a procedimentos remotos (RPC) e a redundância do gerenciador de rede.
- Tolerância a Falhas: inclui serviços de tolerância a falhas tais como: difusão generalizada confiável e comunicação de grupo. Entretanto, os esquemas de

diagnóstico distribuído implementados podem ajudar a identificar falhas, mas não são capazes de cobrir todas as falhas possíveis.

5.3.3 Sistema Operacional MINIX

O sistema operacional MINIX, descrito por Tanembaum citado por Fricks [FRI 89] não se trata de um sistema operacional específico para aplicações tempo real. Entretanto, Fricks [FRI 89] propôs algumas modificações que permitem a sua utilização nestas situações. Por isso, a descrição breve contida neste trabalho será baseada nas alterações sugeridas na dissertação de Fricks, que o tornam um SOTR (Sistema Operacional Tempo real). Maiores detalhes podem ser obtidos diretamente na referência original.

O MINIX foi projetado para ser compatível com a versão 7 do UNIX, a qual foi escolhida como modelo por sua simplicidade e elegância, e para ser utilizado em simples plataformas de PCs (*Personal Computer*). O próprio nome MINIX significa mini-UNIX, porque se trata de uma versão deste sistema operacional suficientemente pequena para ser legível e compreendida por uma única pessoa que se proponha a estudá-lo.

No projeto do sistema operacional MINIX foi adotado o modelo cliente-servidor para sua estrutura interna. A idéia de utilização deste modelo é remover tanto código quanto possível do sistema operacional, tornando-o mais enxuto. Este código é levado aos níveis superiores do sistema, deixando um executivo (núcleo) de tamanho bastante reduzido. Dessa forma, muitas da funções do sistema operacional são implementadas a nível de aplicação.

Dentro do modelo cliente-servidor foi realizada uma implementação utilizando processos. Dessa forma, cada recurso possui um gerente distinto - o servidor - e os processos que necessitam utilizar o recurso - os clientes - devem enviar mensagens de solicitação do mesmo ao gerente (servidor) correspondente.

Um processo no SOTR proposto pode estar em um dos cinco estados distintos seguintes:

- Em execução (*running*): quando está utilizando o processador, ou seja, detém a sua atenção.
- Pronto (*ready*): quando está pronto para executar, disputando o processador com os demais processos prontos.
- Bloqueado (*blocked*): quando estiver executando e necessitar de um recurso que está indisponível, por exemplo, a impressora.
- Suspenso (*suspended*): quando for ordenado, por outro processo ou por si mesmo, que suspenda a sua execução, não podendo mais competir pelo acesso ao processador.
- Bloqueado-suspenso (*blocked-suspended*): quando estiver bloqueado e sua execução for suspensa.

Normalmente, um processo que está utilizando ou esperando pelo processador ou outro recurso do sistema é suspenso por outro processo quando este possuir uma prioridade (ou urgência) maior de execução. Pode-se dizer que, na verdade, existem

apenas três estados distintos: o processo está executando, está esperando por um recurso (que pode ser a UCP), ou foi suspenso por algum outro processo.

De acordo com os estudos realizados por Fricks, o tempo de chaveamento de contexto para o executivo do MINIX é bastante razoável, de forma a permitir a utilização do mesmo em ambientes tempo real[3].

Entretanto, para executar (e acomodar) uma aplicação efetiva tempo real é necessário ampliar o número de processos e tarefas para que o MINIX seja capaz de permitir a execução utilizando seus recursos de multiprogramação. Este novo valor deve ser determinado de acordo com as características particulares da aplicação e limitado pela sobrecarga de espaço ocupado em memória.

No projeto de Fricks são propostos apenas mecanismos de natureza defensiva, do ponto de vista do sistema operacional, e não de tolerância a falhas propriamente ditos. O problema principal é que o tratamento de falhas baseia-se fortemente em hardware adicional, o que não se encontra disponível em PCs comuns. Dessa forma, o mecanismo defensivo mais implementado na construção de sistemas operacionais com capacidade de tolerar falhas, e que por isso é proposto ao MINIX incorporar, é o confinamento de falhas. Os aspectos relativos ao confinamento de falhas analisados visando implementação no MINIX conduziram as conclusões expostas a seguir:

- os mecanismos usuais de proteção de memória e de proteção de processos foram descartados em função da arquitetura do processador e do hardware dos computadores considerados[4] no seu projeto;
- a proteção de arquivos, entretanto, foi considerada como viável e proposta para o projeto do MINIX com base no armazenamento estável, que visa manter a consistência interna dos dados nos dispositivos de armazenamento em disco mesmo na presença de falhas de hardware.

Apesar de não terem sido adotadas medidas significativas na proteção dos domínios de execução dos processos, providências de menor porte podem ser utilizadas no projeto, com o propósito de dotar a aplicação tempo real de maior confiabilidade e disponibilidade. Entre estas providências destacam-se a implementação de rotinas de atendimento à divisão por zero e ao transbordo em operação matemática (*overflow*). Adicionalmente, a nível de aplicação, podem ser utilizados temporizadores (*watchdog timers*) para verificar seqüências de eventos que devem ocorrer de forma ordenada em certas rotinas.

Conforme a descrição aqui apresentada, pode-se considerar que, de acordo com as características anteriormente citadas como relevantes, o sistema MINIX apresenta o seguinte perfil:

- Contexto: o tempo perdido com a troca de contextos é adequado para a utilização do MINIX em ambientes de tempo real.

---

[3] Entretanto, o trabalho de Fricks foi realizado tomando por base um processador da linha 286, enquanto que os parâmetros de desempenho dos processadores atuais são ainda melhores, o que, com certeza, melhora este tempo.

[4] Este hardware corresponde ao da linha de computadores 286, entretanto não foi realizada uma análise para verificar se este aspecto foi alterado nos computadores atuais.

- Portabilidade: quanto a esta característica nada foi encontrado na bibliografia. Porém, pelo fato de ter sido projetado para PCs, é suposta uma grande portabilidade do sistema.
- Previsibilidade: em sistemas operacionais desenvolvidos especificamente para utilização em sistemas tempo real, esta é uma característica indispensável. Entretanto, este não é o caso do MINIX, então supõe-se que não tenha sido sugerida nenhuma alteração no projeto original do MINIX e ele continue não apresentando esta capacidade.
- Reconfigurabilidade: como esta propriedade não é analisada nem proposta na bibliografia consultada e o sistema não foi testado de forma a permitir a obtenção de conclusões próprias, nada pode-se afirmar.
- Redundância: esta característica é implementada e controlada principalmente pelo mecanismo de armazenamento estável, ou seja, o sistema controla apenas a redundância de dados.
- Tolerância a Falhas: o mecanismo defensivo implementado para o MINIX ter a capacidade de tolerar falhas é o confinamento de falhas, relacionado apenas à proteção de arquivos e algumas outras providências referentes à proteção de processos.

5.3.4 Sistema Operacional MTOS-UX

O sistema MTOS-UX desenvolvido por Ripps [RIP 93] por volta de 1984, é um produto da Industrial Programming Inc. Trata-se de um SOTR genérico, que engloba características e facilidades consideradas importantes para sua área de aplicação, tais como:

- organização/estruturação em multitarefas;
- alto grau de independência entre as diversas tarefas componentes de um sistema;
- necessidade de ter resposta muito rápida a qualquer interrupção.

Talvez por isso, ele possa ser considerado um padrão entre os SOTRs existentes e tem sido utilizado com sucesso, nos últimos anos, em várias centenas de aplicações tempo real diferentes, segundo seu autor.

O MTOS-UX é portátil para vários equipamentos, encontrando-se disponível para famílias distintas de processadores tais como Motorola 680xx, Intel 80x86 e National Semiconductor 32x32. E pode ser executado em computadores mono ou multiprocessados.

No MTOS-UX, as aplicações são vistas como um conjunto de tarefas, que são definidas através da distribuição das especificações funcionais (tal como contida no modelo de especificações do sistema) entre programas executando concorrentemente (as tarefas). Estas tarefas podem ser: periódicas, quando disparadas por tempo; ou não-periódicas, quando disparadas por eventos. Assim, ele possui primitivas de controle, permitindo que o modo de coordenação básico de tarefas seja acrescido de um tempo máximo de intervalo. Este tempo é utilizado para controlar se a requisição de um serviço

a ser realizado por um recurso é finalizada dentro do prazo, visando não comprometer a operação normal do sistema. Se a requisição não pode ser realizada naquele tempo, ela é ignorada após determinado tempo e a tarefa continua, mas o sistema operacional informa a falha para a tarefa. Este intervalo com valor zero corresponde à uma "falha devido à não disponibilidade imediata".

Segundo seu autor, um SOTR pode ser visto como um programa baseado em objetos primitivos executáveis, estes objetos e respectivas funções de serviço suportados pelo MTOS-UX correspondem aos seguintes:

- Serviços de controle de tarefas: são utilizados, por exemplo, para criar, iniciar, transferir a coordenação, conectar a uma interrupção, alterar a prioridade, encerrar e apagar tarefas, entre outros.
- *Flags* de evento: representam informações significativas para a aplicação. A transmissão de informações através de *flags* de evento é empregada na coordenação intertarefa. Alguns serviços relacionados são, por exemplo, criar, ligar, desligar, esperar um tempo e apagar um grupo de *flags* de evento.
- Semáforos e variáveis compartilhadas controladas: como as tarefas compartilham grupos de dados alteráveis, devem ser realizados controles constantes, visando evitar que estes dados (por exemplo variáveis) sejam alterados simultaneamente por mais de uma tarefa. Os semáforos correspondem a uma forma tradicional de proteger dados compartilhados alteráveis; enquanto que variáveis compartilhadas controladas tratam-se de uma extensão ao conceito simples de semáforos, evitando a criação de tarefas ineficientes de captação exclusiva das variáveis.
- Trocas de mensagens: facilitam a comunicação entre tarefas. São implementadas duas formas diferentes de comunicação no MTOS-UX: a caixa de correio (sem limite de tamanho de mensagens) e o *buffer* de mensagens (de tamanho limitado).
- Entrada/saída: estas funções fornecem um mecanismo de comunicação entre as tarefas e o mundo exterior (sistema controlado). Podem ser realizadas em nível físico (através dos dispositivos periféricos) ou em nível lógico (através de arquivos do sistema).
- Sinais: são utilizados como uma forma de coordenação e comunicação entre tarefas. Sua forma de implementação é através de uma interrupção por software que pode ser manuseada a nível de tarefa. O MTOS-UX dedica alguns de seus sinais à recuperação de erros do sistema.
- Tempo e tempo do dia: além de controlar para que uma tarefa aguarde para executar durante um intervalo de tempo, o MTOS-UX trata referências ao relógio ou calendário real, que são mantidos e atualizados pelo sistema operacional.
- Gerenciamento de memória compartilhada: no MTOS-UX isto é realizado através de dois tipos de *pool*, um de blocos fixos e outro comum de memória, que pode ter vários blocos contíguos.

O MTOS-UX oferece ainda serviços que permitem a depuração e a recuperação de exceções, como forma de prover tolerância a falhas aos sistemas desenvolvidos em

seu ambiente. Para certas falhas ele simplesmente envia uma mensagem de erro à tarefa e continua a execução na próxima instrução. Se ocorrerem falhas mais graves ele pressupõe que seu código ou dados foram prejudicados fora do alcance do reparo e emite uma mensagem de erro fatal, parando completamente a execução. Dessa forma, pode-se dizer que a capacidade de recuperação do sistema operacional depende do tipo de falha detectada.

O trabalho de Ripps [RIP 93] é bastante técnico e preocupa-se em mostrar detalhes de implementação, o que não é relevante no desenvolvimento deste trabalho. Entretanto, quanto às características gerais do sistema, muito pouco foi encontrado. Mas conforme a descrição aqui apresentada, pode-se considerar que, de acordo com as características anteriormente citadas como relevantes, o sistema MTOS-UX apresenta o seguinte perfil:

- Contexto: nada foi encontrado na bibliografia consultada sobre esta característica. Entretanto, por tratar-se de um sistema multitarefa, esta propriedade deve estar embutida nele, ou seja, supõe-se que apresenta tempos de trocas de contexto bastante razoáveis, para que possa ser utilizado em sistemas tempo real.
- Portabilidade: é bastante portátil, uma vez que encontra-se disponível para várias famílias de processadores tais como Motorola 680xx, Intel 80x86 e National Semiconductor 32x32
- Previsibilidade: pode ser garantida apenas para as tarefas disparadas por tempo, denominadas periódicas.
- Reconfigurabilidade: não há itens na referência que esclareçam este ponto e como o sistema não foi testado, não há como saber.
- Redundância: nada foi encontrado na bibliografia consultada sobre esta característica.
- Tolerância a Falhas: possui primitivas de controle que informam a ocorrência de falhas de temporização para as tarefas, além de serviços de depuração e recuperação de exceções. Mas se ocorrerem falhas mais graves ele pressupõe que seu código ou dados foram prejudicados fora do alcance do reparo e emite uma mensagem de erro fatal, parando completamente a execução.

5.3.5 Sistema Operacional Portos-TF

O sistema operacional Portos-TF, descrito por Lobianco [LOB 89], é considerado um sistema operacional portátil de tempo real, que implementa primitivas de tolerância a falhas, em particular as de software. Para atender a estes requisitos de tolerância a falhas, o sistema operacional implementa primitivas que adotam o mecanismo de n-versões de forma transparente, o que permite que ele seja usado em aplicações que exigem alta confiabilidade. Quanto à portabilidade, o Portos-TF propõe ser necessário um mínimo de esforço no seu transporte para outros tipos de equipamentos. Além disso, um de seus propósitos é oferecer um ambiente de execução tempo real que suporte multitarefa, permitindo a execução concorrente de vários processos em uma mesma máquina.

Seu ambiente de uso é um sistema distribuído, composto de várias estações, não necessariamente iguais, interligadas por um subsistema de comunicação, com o mesmo sistema operacional (Portos-TF) sendo utilizado na maioria das estações, para garantir a portabilidade do software aplicativo tempo real. A primeira implementação do Portos-TF foi realizada em equipamentos do tipo PC, mas nada impede que ele seja utilizado em outra categoria de equipamentos, visto que é extremamente portátil.

O núcleo (*kernel*) desse sistema operacional é o responsável por gerenciar o uso do processador, suprir mecanismos de comunicação e sincronização entre processos (os portos) e oferecer primitivas que implementem mecanismos de tolerância a falhas. Além disso, o sistema operacional oferece um mecanismo de interrupção de relógio tempo real. Em complemento ainda, o núcleo do Portos-TF apresenta as seguintes funções de controle e gerenciamento [LOB 89]:

- criação e destruição de processos;
- obtenção do identificador do processo corrente;
- bloqueio e liberação de processos;
- obtenção e alteração da prioridade de processos;
- obtenção e alteração da fatia de tempo atribuída aos processos;
- congelamento e descongelamento da fatia de tempo atribuída aos processos;
- habilitação e desabilitação de geração de exceções;
- habilitação e desabilitação de escalonamento;
- escalonamento do processo corrente;
- obtenção do estado do sistema operacional.

Já que o Portos-TF é um sistema operacional que implementa primitivas de tolerância a falhas, ele deve permitir que a arquitetura seja dinamicamente reconfigurável como um todo. Isto é possível uma vez que o seu núcleo possui a habilidade de modificar e ampliar o sistema quando ocorrem falhas, em tempo hábil, sem comprometer os prazos de resposta do mesmo. Por outro lado, a implementação de alguns mecanismos de tolerância a falhas, resulta em uma queda de desempenho no funcionamento geral do sistema, se comparado com outro sistema equivalente sem tais mecanismos. Isso ocorre principalmente nos sistemas baseados em redundância (temporal ou espacial) durante a execução, devido ao processamento adicional que lhes é imposto. Entretanto, esta queda de desempenho (*overhead*) é suportável e preferível, desde que não inviabilize sua execução, uma vez que deve ser garantida a segurança e a correção do funcionamento global de um sistema tempo real.

Conforme a descrição aqui apresentada, pode-se considerar que, de acordo com as características anteriormente citadas como relevantes, o sistema Portos-TF apresenta o seguinte perfil:

- Contexto: nada foi encontrado na bibliografia consultada sobre esta característica. Entretanto, por tratar-se de um sistema multitarefa, esta propriedade deve estar embutida nele, ou seja, supõe-se que apresenta tempos de trocas de contexto bastante razoáveis, para que possa ser utilizado em sistemas tempo real.

- Portabilidade: uma de suas características principais refere-se ao fato de ele ser portátil.
- Previsibilidade: em sistemas operacionais desenvolvidos especificamente para utilização em sistemas tempo real, como é o caso do Portos-TF, esta é uma característica indispensável. Entretanto, esta propriedade não é analisada na bibliografia consultada e o sistema não foi testado de forma a permitir a obtenção de conclusões próprias.
- Reconfigurabilidade: é dinamicamente reconfigurável quando ocorrem falhas no sistema.
- Redundância: é capaz de gerenciar a redundância existente no sistema que o utiliza.
- Tolerância a Falhas: foi desenvolvido justamente para ser tolerante a falhas e, para isso, implementa vários mecanismos de tolerância a falhas, sem comprometer os prazos do sistema.

5.3.6 Sistema Operacional QNX

O QNX, conforme descrito no manual do próprio sistema [QNX 96], ao contrário de alguns sistemas operacionais estudados, trata-se de uma solução comercial de SOTR para ser utilizado em ambientes distribuídos compostos de simples PCs. Este sistema operacional disponibiliza aos seus usuários recursos importantes como o gerenciamento multitarefa, escalonamento dirigido a prioridades e rápida troca de contexto - todos considerados ingredientes essenciais para o funcionamento adequado de um sistema tempo real.

O QNX oferece um alto grau de eficiência, modularidade e simplicidade devido a dois princípios fundamentais:

- Arquitetura *microkernel*: o QNX consiste de um pequeno núcleo (*kernel*), que fornece apenas os serviços essenciais de um sistema operacional. Assim, por exemplo, ele é o responsável direto pela comunicação entre os processos, pela comunicação de baixo nível na rede, pelo escalonamento de processos e pela manipulação de interrupções de primeiro nível. As demais tarefas são desenvolvidas por vários componentes, responsáveis por serviços específicos, que serão descritos a seguir.
- Comunicação entre processos (IPC - *InterProcess Communication*): o QNX controla a comunicação entre processos, que pode ser basicamente de três tipos - por mensagens (a forma fundamental), por procurações (uma forma especial de mensagens) e por sinais (a forma tradicional).

São os componentes separados, responsáveis pela realização de vários serviços do sistema operacional, que garantem uma alta modularidade ao QNX. Tais componentes possuem uma interface bem definida e comunicam-se entre si através de mensagens, são eles:

- Gerenciador de Processos: é o responsável por criar novos processos no sistema e gerenciar os recursos considerados fundamentais associados ao

processo criado. Um processo possui basicamente um ciclo de vida composto de: criação (*creation*) - carga (*load*) - execução (*execution*) - término (*termination*). Além disso, um processo está sempre em um dos seguintes estados: pronto (*ready*); bloqueado (*blocked*); detido (*held*); espera-bloqueado (*wait-blocked*) ou morto (*dead*).

- Gerenciador do Sistema de Arquivos: oferece uma forma padronizada de armazenamento e acesso aos dados nos subsistemas de disco. É o responsável por manipular todas requisições para abrir (*open*), fechar (*close*), ler (*read*) e escrever (*write*) arquivos.
- Gerenciador dos Dispositivos: é a interface entre os processos e os dispositivos terminais, que são os dispositivos de E/S. É o responsável por controlar o fluxo de dados entre uma aplicação e os dispositivos.
- Gerenciador da Rede: disponibiliza aos usuários do QNX uma extensão das poderosas capacidades de manipulação de mensagens do sistema operacional. Possui as seguintes responsabilidades: *throughput* (por balanceamento de carga), tolerância a falhas (por conectividade redundante) e ligação entre as diversas redes QNX.

Como um sistema operacional tempo real, o QNX está habilitado a oferecer formas e meios de garantir a confiabilidade e o *throughput* das aplicações desenvolvidas com base nele. Quando nodos são conectados em duas ou mais redes, há mais de um único caminho de comunicação com o mesmo que pode ser utilizado. Nesse sentido, o QNX permite que se tenha uma rede com ligações redundantes dupla ou triplamente (na verdade, este número pode ser n). Porém, não são apenas múltiplos adaptadores de rede os responsáveis por suportar o balanceamento de carga e a tolerância a falhas; o QNX permite que se misture e combine todas as topologias de rede que forem necessárias, conforme os requisitos da aplicação que está sendo desenvolvida.

Se uma rede falhar, de forma a impossibilitar a comunicação com ela, o gerenciador de rede faz automaticamente o re-roteamento de todos os dados que deveriam passar por ali, através de outras redes. Isto ocorre automaticamente, sem qualquer envolvimento do software de aplicação, e resulta na capacidade de reconfiguração automática do sistema, suportada pela tolerância a falhas transparente da rede.

Pelo fato do QNX ter sido desenvolvido para atuar em ambientes tempo real, sua rede pode ser utilizada em conjunto com vários tipos de hardware e vários protocolos padrões. Por exemplo, em um ambiente de controle de processos, como um CLP (Controlador Lógico Programável), podem ser requeridos mais recursos do que em outros ambientes menos críticos, mas igualmente tempo real. Entretanto, sua rede e o QNX, de uma forma geral, deve suportar ambos tipos de aplicações da mesma forma. Esta característica permite enquadrar o QNX como um SOTR altamente portátil.

No QNX, o gerenciamento de tempo é baseado em um sistema de temporizador mantido pelo próprio sistema operacional. O temporizador contém a atual coordenada universal de tempo relativa a 0 (zero) horas, 0 (zero) minutos e 0 (zero) segundos do dia 01 (primeiro) de janeiro de 1970.

Conforme a descrição aqui apresentada, pode-se considerar que, de acordo com as características anteriormente citadas como relevantes, o sistema QNX apresenta o seguinte perfil:

- Contexto: disponibiliza uma rápida troca de contexto, por ser multitarefa.
- Portabilidade: é uma de suas características básicas, visto que foi projetado para ser executado em qualquer ambiente tempo real.
- Previsibilidade: em sistemas operacionais desenvolvidos especificamente para utilização em sistemas tempo real, como é o caso do QNX, esta é uma característica indispensável. Entretanto, esta propriedade não é analisada na bibliografia consultada e o sistema não foi testado de forma a permitir a obtenção de conclusões próprias.
- Reconfigurabilidade: o sistema é dinâmica e automaticamente reconfigurável, em caso de falhas na rede.
- Redundância: é capaz de controlar redundância de seus equipamentos, de conectores e de adaptadores de rede.
- Tolerância a Falhas: suporta tolerância a falhas, principalmente na rede de comunicação, permitindo com isso o desenvolvimento de aplicações mais confiáveis.

5.3.7 Sistema Operacional Thoth

O Thoth é descrito por Cheriton [CHE 79] como um sistema operacional tempo real projetado para ser portátil entre um grande conjunto de máquinas, sem preocupar-se com questões relacionadas com a confiabilidade do sistema. Normalmente, ele é executado em dois microcomputadores com arquiteturas diferentes. Os sistemas e programas de aplicação que são executados nele devem ser escritos em linguagens de alto nível. Uma vez que o sistema é implementado pelo mesmo software executando em diferentes tipos de hardware, o Thoth utiliza a mesma interface. Assim, um programa de aplicação que é usado no Thoth é altamente portátil.

Este sistema operacional suporta ainda múltiplos processos concorrentes, alocação dinâmica de memória, dispositivos de E/S independentes, um sistema de arquivos e vários terminais, ou seja, a maioria das características consideradas importantes para SOTRs. Além disso, apresenta uma rápida velocidade para troca de contexto durante a execução de seus múltiplos processos.

A alta portabilidade é o principal objetivo alcançado com o projeto do Thoth. Porém, além deste, outros objetivos foram atingidos, tais como:

- oferecer um sistema no qual os programas podem ser estruturados usando vários pequenos processos concorrentes;
- disponibilizar uma comunicação entre os processos bastante eficiente;
- permitir que o sistema acompanhe a demanda da aplicação tempo real através de limitações de tempo;

- desenvolver um sistema adaptável a uma grande variedade de aplicações tempo real.

No Thoth, uma função qualquer é invocada como uma subrotina ou como um processo separado. Quando é invocada como uma subrotina, a função é alocada no próprio processo requisitante. Quando é invocada como um novo processo, é alocada separadamente.

Os processos criados pelo sistema operacional podem encontrar-se, em um dado momento, em um dos seguintes estados: pronto para executar (*ready*); bloqueado (*blocked*) ou ativo (*active*).

São oferecidas, pelo Thoth, quatro primitivas de troca de mensagens entre os processos, como forma de comunicação: envio (*send*); recebimento (*receive*); resposta (*reply*) e divulgação (*forward*).

Ao que parece, pelo menos de acordo com a descrição apresentada por Cheriton, o Thoth não implementa primitivas de tolerância a falhas, não realiza a manipulação de redundância de componentes, e tampouco suporta a reconfigurabilidade dinâmica do sistema. É verdade que o material estudado preocupa-se mais em ressaltar a portabilidade como a característica fundamental do sistema, mas igualmente nada foi encontrado a respeito de previsibilidade.

Conforme a descrição aqui apresentada, pode-se considerar que, de acordo com as características anteriormente citadas como relevantes, o sistema Thoth apresenta o seguinte perfil:

- Contexto: oferece uma capacidade de trocar rapidamente de contexto quando alterna a execução (utilização do processador) entre os processos que estão prontos.
- Portabilidade: este é o objetivo principal do projeto do Thoth, por isso ele é altamente portátil.
- Previsibilidade: em sistemas operacionais desenvolvidos especificamente para utilização em sistemas tempo real, como é o caso do Thoth, esta é uma característica indispensável. Entretanto, esta propriedade não é analisada na bibliografia consultada e o sistema não foi testado de forma a permitir a obtenção de conclusões próprias.
- Reconfigurabilidade: não suporta a reconfigurabilidade dinâmica do sistema.
- Redundância: não realiza a manipulação de redundância de componentes do sistema.
- Tolerância a Falhas: não implementa nenhuma primitiva de tolerância a falhas.

5.3.8 Sistema Operacional CMX-RTX

Além dos sistemas operacionais anteriormente descritos, tem-se conhecimento de um outro sistema operacional tempo real bastante utilizado, denominado CMX-RTX. Entretanto, não foi possível realizar um estudo mais completo sobre este SOTR, no mesmo nível dos demais, uma vez que não foram encontradas informações suficientes

referentes às suas características gerais. Em vista desta dificuldade, não será realizada uma análise e comparação posterior do CMX-RTX com os demais sistemas operacionais estudados.

O material encontrado, através de buscas pela rede Internet [CMX 96, RTX 96 e RTX 97], detém-se mais nas aplicações tempo real que utilizam o sistema operacional CMX-RTX do que em uma descrição básica do seu funcionamento geral. Porém algumas informações obtidas a respeito de suas características, tais como: sua capacidade de realizar rapidamente trocas de contexto, ser altamente portátil e apresentar tempos de latência extremamente baixos, permitem enquadrá-lo como um SOTR de bom nível.

Embora esta menção ao sistema não atinja os objetivos do presente trabalho, a referência ao sistema foi aqui incluída como indicação de um software de possível interesse para pesquisa.

5.4 Análise dos Sistemas Operacionais Tempo Real

Os sistemas operacionais apresentados (seção 5.3) serão comparados nesta seção segundo aquelas características consideradas importantes ou desejáveis que eles apresentem. Estas características são as que envolvem as questões apresentadas anteriormente, na seção 5.1 deste trabalho. O sistema CMX-RTX, apresentado na seção 5.3.8, não será comparado com os demais por não ter sido suficientemente estudado.

A análise realizada, conforme será apresentada na Tabela 5.1, atribui graus, quantificando a análise qualitativa realizada para cada um dos sistemas operacionais estudados. Estes graus possuem os significados a seguir definidos:

- Grau 2: significa que o sistema operacional apresenta esta característica sempre.
- Grau 1: significa que esta característica está parcialmente presente no sistema operacional (com restrições).
- Grau 0: significa que esta característica não está presente no sistema operacional.

TABELA 5.1 - ANÁLISE COMPARATIVA DOS SOTRS

Características	DEDOS	HARTOS	MINIX	MTOS-UX	Portos-TF	QNX	Thoth
Contexto	<0>	<0>	<2>	<1>	<1>	<2>	<2>
Portabilidade	<2>	<1>	<2>	<2>	<2>	<2>	<2>
Previsibilidade	<1>	<1>	<0>	<1>	<1>	<1>	<1>
Reconfigurabilidade	---	<1>	---	---	<2>	<2>	<0>
Redundância	<1>	<2>	<1>	---	<2>	<2>	<0>
Tolerância a Falhas	<1>	<2>	<1>	<1>	<2>	<2>	<0>

6 Linguagens de Programação Tempo Real

A satisfação dos principais requisitos de um sistema tempo real, tais como previsibilidade, correção temporal e segurança de funcionamento; torna necessário o uso de técnicas, ferramentas e linguagens de programação específicas para o desenvolvimento, análise e validação de tais sistemas [FUR 96]. Neste capítulo serão definidas algumas características e recursos necessários nas linguagens de programação tempo real, incluindo aspectos de tolerância a falhas. Posteriormente, serão apresentados brevemente alguns exemplos de linguagens conhecidas, que serão analisadas e comparadas entre si, resultando em uma taxonomia.

6.1 Características das Linguagens de Programação Tempo Real

O desenvolvimento dos sistemas de aplicação é possível através da infra-estrutura fornecida pelas linguagens de programação de alto nível, que garantem a sua execução coerente. Tais linguagens devem possuir comandos ou chamadas do sistema operacional que permitam o acesso aos recursos do sistema e a comunicação.

Sob o ponto de vista de linguagens de programação, o aspecto temporal é abordado por mecanismos que permitem acessar o relógio do sistema, controlar a expiração de tempo e ativar processos com base em prioridades. Dessa forma, uma linguagem de programação tempo real deve refletir um modelo de programação que inclui a integração coerente entre os aspectos funcionais e temporais da aplicação a ser desenvolvida [LIS 93]. De acordo com as características de um sistema tempo real, pode-se concluir que seus programas de aplicação são constituídos de módulos independentes, que se comunicam mutuamente em resposta aos eventos externos ou ao término de intervalos de tempo pré-definidos. Nesse sentido, a linguagem de programação deve fornecer recursos para a existência de tais módulos, como por exemplo [TOZ 83]:

- recurso para a definição dos módulos de programas independentes;
- mecanismo para a troca de informação entre módulos distintos;
- mecanismo de sincronização para o uso de recursos comuns;
- comandos para ativar e suspender a execução de outros módulos;
- comando para auto-escalonamento;
- mecanismo de rotinas reentrantes e de co-rotinas, para o compartilhamento de rotinas;
- comando para determinar a própria execução;
- recurso de relógio de tempo real;
- recursos para comunicação do programa com o meio externo.

Além disso, os compiladores das linguagens de tempo real devem ter a capacidade de otimizar os códigos gerados para que a execução do programa seja eficiente. O problema é que eles nem sempre são muito rápidos, pois os passos adicionais de otimização são normalmente constituídos de algoritmos complexos. Este *overhead*, no entanto, é justificado pois o seu objetivo é gerar programas com bom desempenho na execução, não sendo o tempo de compilação tido como crítico na maioria das situações.

De forma geral, uma linguagem de programação tempo real deve incorporar, a nível de linguagem e/ou suporte de execução, características que permitam que a implementação de sistemas tempo real seja realizada de forma adequada [FUR 96]. Assim, tais linguagens devem garantir que sejam satisfeitos, além dos requisitos comuns a qualquer linguagem de propósitos gerais, os requisitos específicos de tempo real.

Para que seja possível realizar uma análise comparativa entre as diversas linguagens de programação estudadas neste trabalho, as seguintes características serão consideradas importantes:

- Flexibilidade: esta característica indica se a linguagem de programação é flexível, no sentido de poder ser utilizada em uma gama variada de aplicações.
- Gerenciamento de Tempo: esta característica identifica se a linguagem apresenta mecanismos específicos de gerenciamento de tempo.
- Priorização: esta característica refere-se à capacidade da linguagem de programação de atribuir prioridades distintas às tarefas.
- Reutilização: esta característica refere-se a capacidade da linguagem de permitir a reutilização de módulos de software.
- Tolerância a Falhas: esta característica indica se a linguagem oferece recursos de tolerância a falhas.
- Tratamento de Exceções: esta característica identifica se a linguagem apresenta mecanismos específicos para o tratamento de exceções.

6.2 Tolerância a Falhas nas Linguagens de Programação Tempo Real

Um bom suporte a nível de linguagem de programação pode simplificar consideravelmente a tarefa de escrever software distribuído, e principalmente de tempo real, tolerante a falhas. Algumas técnicas que podem ser incorporados a tais linguagens são, por exemplo: replicação, recuperação e notificação de falhas [SCH 95].

A manifestação observável de um comportamento inadequado, em termos de resultados ou desempenho, constitui-se em defeito do programa. Nesse ponto, já não é adequada a utilização de mecanismos de tolerância a falhas de programas, visto que eles não são usados para qualquer tipo de ação de remoção de falhas. O que pode ser garantido com o uso de tais mecanismos é que, apesar da existência de falhas, o sistema continuará comportando-se adequadamente [LIS 93]. Nesse sentido, os recursos oferecidos pelas linguagens de programação (ou ambiente de desenvolvimento), devem suportar, em tempo de execução, mecanismos específicos para as seguintes atividades:

- Mecanismos de detecção de erros: uma das formas de detecção de erros é através da monitoração da execução. Os mecanismos de tratamento de exceções oferecidos pelas linguagens de programação permitem a detecção de erros pela monitoração de operações que apresentem risco durante o processo de execução, por exemplo, as atividades de tempo real. Em caso de atividades tempo real formas alternativas de detecção de erros seriam a implementação de *watchdogs* ou de mecanismos de controle do tempo.

- Mecanismos de confinamento e avaliação de danos: algumas formas de minimizar a propagação de erros a outros componentes do programa envolvem mecanismos de tratamento de exceções, técnicas de operações atômicas, pontos de verificação e proteção de dados e processos.

- Mecanismos de recuperação de erros: técnicas de retorno ou avanço do estado computacional representam uma forma de restabelecer um estado normal de execução de um programa, permitindo a sua continuidade.

- Mecanismos de tratamento de falhas e continuidade do serviço: o tratamento de falhas também é responsabilidade dos sistemas de tratamento de exceções, através dos respectivos tratadores, que são os responsáveis pela continuidade da execução de um programa no evento de uma situação anormal. A continuidade do serviço pretendido pode ser obtida através da replicação dos módulos de software considerados críticos ou pelo mascaramento de um estado errôneo existente.

Estes mecanismos englobam desde técnicas de programação de uso geral, até aspectos específicos (sintáticos e semânticos) de algumas linguagens de programação e podem ser executados seqüencial ou concorrentemente.

As técnicas de programação tolerante a falhas, apresentadas na seção 3.4 deste trabalho, podem ser diretamente implementadas na aplicação, desde que a linguagem de programação utilizada ofereça facilidades e recursos adequados, tais como: mecanismos de tratamento de exceções, mecanismos de execução concorrente, mecanismos de conversação, mecanismos de recuperação de erros, entre outros. Além disso, para a implementação destas técnicas pode-se necessitar de ambientes específicos para o desenvolvimento e suporte de programas tolerantes a falhas, que inclusive já se encontram disponíveis para esta atividade.

Entretanto, o desenvolvimento de software tolerante a falhas requer, além do apoio da linguagem de programação usada na sua implementação, o apoio nas demais camadas de software, que é o enfoque deste trabalho. Por outro lado, é fundamental a compreensão dos conceitos de tolerância a falhas e sua aplicabilidade em todas as etapas do desenvolvimento do software.

6.3 Exemplos de Linguagens de Programação Tempo Real

No decorrer da realização deste trabalho, foram estudadas algumas das linguagens de programação existentes com propósitos específicos para o desenvolvimento de sistemas tempo real, que serão descritas nesta seção. Entretanto, não foi possível esgotar a lista de linguagens existentes e conhecidas e certamente nem

todas serão abordadas neste trabalho, mas acredita-se que a contribuição aqui apresentada, apesar de certas limitações, será de grande utilidade.

6.3.1 Linguagem Ada

A linguagem Ada surgiu quando o Departamento de Defesa dos Estados Unidos resolveu padronizar uma linguagem de programação para desenvolvimento de grandes sistemas, especialmente os sistemas embutidos de tempo real, na década de 70 [SIL 88]. Todavia, ela foi homologada para as aplicações tempo real do Departamento e sua prática pode se expandir para outros segmentos governamentais e industriais daquele e de outros países [RIP 93]. Dessa forma, Ada praticamente tornou-se um padrão na programação tempo real. Desde o seu surgimento, foram criadas várias versões da linguagem, sendo a última denominada Ada 95. Ada, de uma forma geral, foi projetada tendo em vista os seguintes aspectos:

- promoção da confiabilidade;
- simplificação da manutenção;
- eficiência.

Um dos pontos fortes de Ada para contribuir com estes aspectos é o suporte à elaboração de programas modulares, o que permite que grupos independentes de programadores desenvolvam partes do mesmo programa. Esta prática permite também a reutilização de software.

Ada apresenta algumas características que auxiliam a escrita de programas com correção, tais como:

- Verificação: é feita a verificação de compatibilidade de tipos e usos adequados das operações, durante a compilação, minimizando a maioria dos erros grosseiros, por ser fortemente tipada.
- Minimização de efeitos colaterais: os efeitos colaterais gerados devido ao uso de variáveis globais, são reduzidos pelo controle das mesmas permitido com o uso de pacotes.
- Declaração dos parâmetros: é declarado o modo como os parâmetros são utilizados nas especificações dos subprogramas.

Estas características contribuem para o desenvolvimento de sistemas mais corretos e menos propensos a falhas de software decorrentes de alguma especificação ou codificação errônea no que se refere aos itens anteriormente citados. Por outro lado, as facilidades para o encapsulamento de tipos abstratos de dados e suas operações em regiões bem delimitadas no corpo do programa, contribuem para o confinamento de erros.

Além disso, a linguagem coloca à disposição do usuário um extenso conjunto de mecanismos para tratamento de exceções e a possibilidade de trabalhar com processos concorrentes, utilizando sincronização e exclusão mútua. Por fim, estas facilidades permitem o desenvolvimento de software tempo real mais confiável.

Ada procura definir e controlar o comportamento tempo real do programa através de suas unidades de programação - uma das quais é denominada tarefa. Nesse sentido, uma tarefa não necessita ter uma prioridade de execução e quando ela tem não há serviço capaz de alterá-la[5] [RIP 93].

Entretanto, as características de tempo real embutidas em Ada não são limitadas a estas facilidades. A linguagem também oferece:

- Pausas: através do comando *delay*.
- Recuperação de erros: através do tratador de exceções.
- Suporte a data e hora: através do pacote CALENDAR.
- E/S físico: através do pacote TEXT_IO.
- Comunicação e coordenação: através do *rendezvous*.

Um programa Ada deve ter um núcleo básico ou seu equivalente para suportar o modelo de linguagens de tarefa e outras características de tempo real. Assim, sugere-se que o compilador Ada, quando utilizado para tempo real, tenha como suporte um sistema operacional também tempo real (SOTR). Esta combinação linguagem-sistema operacional tempo real tem sido utilizada por Ripps da seguinte forma: linguagem Ada e sistema operacional MTOS-UX e tem apresentado bons resultados.

Especificamente a linguagem Ada 95 possui as seguintes características relevantes para a implementação de sistemas tempo real [BUR 96]:

- Transferência de controle assíncrona (ATC - *Asynchronous Transfer of Controls*): esta característica oferece um mecanismo pelo qual uma tarefa pode obter de forma assíncrona a atenção de outra. Assim, é permitido que uma tarefa execute uma seção de código enquanto está esperando que uma chamada de entrada (ou o esgotamento de um prazo de tempo) ocorra. Se o código terminar antes da chamada de entrada ser aceita (ou do prazo de tempo expirar), então a chamada (ou o mecanismo de controle do tempo) é cancelada. Se a chamada é aceita (ou o prazo de tempo expirar) antes da seção de código terminar, então a execução do código é abortada.
- Atributos da tarefa: este recurso permite que sejam designados atributos arbitrários a serem adicionados às tarefas.
- Prioridades dinâmicas: juntamente com os demais recursos de tempo real, é oferecido um conjunto de facilidades para determinar dinamicamente as prioridades de cada tarefa. Para facilitar a programação de prioridades dinâmicas, a linguagem define este conjunto como um pacote da sua biblioteca.

Conforme a descrição aqui apresentada, pode-se considerar que, de acordo com as características anteriormente citadas como relevantes, a linguagem Ada apresenta o seguinte perfil:

---

[5] A prioridade de uma tarefa pode ser alterada quando ela é servidora dentro de um *rendezvous*, mas a prioridade imediatamente reverte ao valor original quando o *rendezvous* é encerrado.

- Flexibilidade: é uma linguagem flexível, no sentido que foi criada tendo-se em vista o desenvolvimento de grandes sistemas, inclusive de tempo real.
- Gerenciamento de Tempo: ao que parece, possibilita apenas a definição de mecanismos para controle do prazo máximo de tempo.
- Priorização: permite que sejam atribuídas prioridades dinamicamente às tarefas.
- Reutilização: a linguagem permite a reutilização de software, por ser estruturada em módulos.
- Tolerância a Falhas: apesar de não possuir recursos específicos de tolerância a falhas, apresenta algumas características que contribuem para a escrita correta de programas.
- Tratamento de Exceções: oferece explicitamente bons mecanismos de tratamento de exceções.

6.3.2 Linguagem Chill

A linguagem Chill, apresentada por Silva [SIL 88], foi criada visando permitir o desenvolvimento de sistemas de controle de comutação em telecomunicações. Tais sistemas normalmente apresentam as seguintes propriedades:

- são grandes e complexos;
- controlam aplicações em tempo real;
- devem operar com alto grau de confiabilidade e disponibilidade;
- devem permitir extensões sem perturbar o tráfego existente.

Chill surgiu em 1980 com o propósito de suprir a necessidade verificada pelo CCITT (Comitê Consultivo Internacional de Telefonia e Telegrafia) de se usar uma linguagem padronizada de alto nível para o desenvolvimento de software para centrais telefônicas. Dessa forma, as companhias operadoras tornaram-se mais independentes dos fabricantes de equipamentos. Assim, trata-se de uma linguagem moderna, de propósitos gerais, para desenvolvimento de grandes sistemas inclusive distribuídos, e suas facilidades são comparáveis às da linguagem Ada.

As principais características apontadas pelos projetistas para qualificar a linguagem Chill são [SIL 88]:

- permite o desenvolvimento de sistemas complexos e de tempo real;
- aumenta a confiabilidade do software, através de uma verificação extensiva em tempo de compilação;
- permite a geração de um código objeto altamente eficiente;
- é flexível e poderosa, a fim de atender uma ampla gama de aplicações na área de telecomunicações;
- explora os vários tipos de hardware disponíveis;

- encoraja o desenvolvimento modular e estruturado de programas;
- é fácil de se aprender e usar.

Além disso, a linguagem Chill suporta a elaboração de grandes programas por grupos independentes de programadores, através da utilização de módulos. A linguagem também oferece facilidades de programação concorrente, exclusão mútua, sincronização e comunicação entre processos. Além disso, é oferecido um excelente recurso de tratamento de exceções: esse mecanismo, além de exceções pré-definidas na linguagem, possibilita que o usuário defina exceções próprias, defina tratadores de exceções e sinalize a ocorrência de exceções através de cláusulas específicas.

Como uma facilidade para programação de sistemas tempo real, em Chill há a possibilidade de se definir prioridades diferentes para mensagens durante a sincronização e a comunicação entre processos. Entretanto, a linguagem não possui mecanismos para temporizações (ou mecanismos de controle do tempo).

A linguagem Chill não oferece a implementação explícita de mecanismos de tolerância a falhas, mas possui algumas facilidades que permitem que os programas sejam desenvolvidos com correção, tais como:

- Verificação estática de tipos de dados: através da verificação da compatibilidade entre eles e da aplicabilidade de determinadas operações, por ser fortemente tipada.
- Controle da forma de passagem de parâmetros entre subprogramas: através de atributos especificados nos cabeçalhos dos mesmos.
- Declaração explícita da recursividade de procedimentos (*procedures*): que deve ser feita já na definição das mesmas.
- Controle explícito dos procedimentos (*procedures*): ou seja, se estes serão manipulados dinamicamente ou passados como parâmetros, através da especificação dos seus atributos.
- Minimização dos efeitos colaterais: os efeitos colaterais gerados devido ao uso de variáveis globais, são reduzidos pelo controle das mesmas permitido com o uso de pacotes.

Estas facilidades contribuem para o desenvolvimento de sistemas mais corretos e menos propensos a falhas de software decorrentes de alguma especificação ou codificação errônea no que se refere aos itens anteriormente citados.

Conforme a descrição aqui apresentada, pode-se considerar que, de acordo com as características anteriormente citadas como relevantes, a linguagem Chill apresenta o seguinte perfil:

- Flexibilidade: tem o propósito de ser flexível a fim de cobrir uma ampla gama de aplicações na área de telecomunicações, entretanto é voltada para o desenvolvimento de software em centrais telefônicas.
- Gerenciamento de Tempo: não possui mecanismos para controle de temporizações.
- Priorização: há a possibilidade de se definir prioridades diferentes apenas para mensagens durante a sincronização e a comunicação entre processos.

- Reutilização: suporta a reutilização de software através da utilização de módulos na definição dos seus programas.
- Tolerância a Falhas: não oferece a implementação explícita de mecanismos de tolerância a falhas, mas possui algumas facilidades que permitem o desenvolvimento de programas com correção.
- Tratamento de Exceções: apresenta excelentes recursos para o tratamento de exceções.

6.2.3 Linguagem DEAL

A linguagem DEAL (*DEdos Application Language*), apresentada por Roosmalen [ROS 94], trata-se de uma linguagem orientada a objetos, baseada na linguagem C++. Entretanto, para que DEAL pudesse ser utilizada no desenvolvimento de sistemas tempo real, inclusive os críticos, foram acrescentadas algumas extensões e feitas algumas restrições à linguagem base.

A plataforma de uso desta linguagem é o sistema operacional DEDOS, caracterizado na seção 5.3.1 deste trabalho. Como já foi visto, o objetivo deste sistema operacional e, por extensão, da sua linguagem, é disponibilizar um ambiente para o desenvolvimento de sistemas tempo real de maior produtividade, mais flexíveis e mais confiáveis.

Sistemas tempo real normalmente são embutidos em sistemas maiores, sendo assim desenvolvidos isoladamente, mas devendo posteriormente funcionar em conjunto com todo o sistema. Neste sentido, uma característica considerada importante na linguagem é a capacidade de reutilizar e estender um projeto ou um código desenvolvido. Esta facilidade é resultado da escolha do paradigma da orientação a objetos para desenvolver a linguagem.

Além disso, a linguagem suporta a construção de sistemas disparados (ou dirigidos) por eventos e por tempo, através de conceitos e técnicas que permitem a composição do sistema hierarquicamente em módulos. Outra importante abstração que deve ser permitida por linguagens de alto nível é a independência da plataforma de execução do sistema, tornando este mais flexível e portátil, visto que em sistemas tempo real mudanças na plataforma são normalmente a nível de hardware; DEAL apresenta esta abstração.

Por outro lado, as características capazes de satisfazer os requisitos de temporização do sistema são especificadas no programa através de primitivas exclusivas, que possuem um significado independente da velocidade do processador, justamente para garantir esta abstração.

A linguagem DEAL não apresenta nenhuma primitiva relacionada à distribuição. Os módulos são implementados e o sistema operacional DEDOS é encarregado de organizar sua distribuição entre os processadores disponíveis. Assim, a distribuição do sistema torna-se transparente a nível de programação.

DEAL é baseada em C++ e, dessa forma, mantém os aspectos de orientação a objetos, como classes genéricas, e oferece liberdade no uso de objetos dinâmicos e apontadores. Entretanto, as tarefas tempo real críticas são exceções, pois o

comportamento do programa deve ser analisado estaticamente, ou seja, em tempo de compilação.

As primitivas que são adicionadas a C++ referem-se aos seguintes aspectos do sistema:

- Concorrência: os objetos da linguagem podem exibir várias atividades concorrentes; dessa forma, atividades são introduzidas como propriedades do objeto. Assim, uma classe também define atividades para suas instâncias, que são inicializadas na construção do objeto.
- Controle de concorrência: uma importante questão a ser manipulada pelo programador é a manutenção da consistência dos objetos através do uso devido das primitivas de controle da concorrência imposta. Isto somente é possível se as funções, classes e objetos forem declaradas como atômicas. Além disso, as múltiplas atividades dos objetos requerem um controle de fluxo e das dependências dos objetos na aplicação.
- Tempo real: em DEAL, o programador especifica quando deve ocorrer uma mudança interna de estado em termos dos momentos previamente determinados de transições internas e externas de estados. As relações de tempo são consideradas causais e apenas comandos executados pela mesma atividade podem ser relacionados no tempo, ou seja, sincronizados através de eventos ou guardas.

Entretanto as semânticas de tempo não são realizadas apenas a nível de linguagem, mas envolvem várias ferramentas no ambiente DEDOS, tais como: analisadores de tempo e escalonadores de tarefas.

Por ser uma extensão da linguagem C++, utilizada no desenvolvimento de sistemas tempo real, pode-se inferir que DEAL mantenha o sistema de tratamento de exceções da linguagem da qual é originária, que já é o mesmo de sua linguagem hospedeira (C).

Um aspecto importante a respeito do desenvolvimento do projeto do qual resultou esta linguagem é que as extensões de C++ não são provenientes de considerações formais realizadas, mas são guiadas pela experiência prática de utilização da linguagem no tipo de aplicação considerado.

Conforme a descrição aqui apresentada, pode-se considerar que, de acordo com as características anteriormente citadas (seção 6.1) como relevantes, DEAL apresenta o seguinte perfil:

- Flexibilidade: apresenta independência da plataforma de execução do sistema, tornando-se assim flexível e portátil, porém exige o sistema operacional DEDOS.
- Gerenciamento de Tempo: os requisitos de temporização do sistema são especificadas no programa através de primitivas exclusivas, que possuem um significado independente da velocidade do processador.
- Priorização: nada foi encontrado a este respeito no material consultado e a linguagem não foi utilizada na prática para ser possível afirmar algo em relação a esta característica.

- Reutilização: possui a capacidade de reutilizar e estender um projeto ou um código anteriormente desenvolvido, por ser estruturada em módulos independentes.
- Tolerância a Falhas: apesar de um de seus objetivos ser disponibilizar um ambiente para o desenvolvimento de sistemas tempo real mais confiáveis, nada foi encontrado a este respeito no material consultado. Entretanto, pode-se contar com a tolerância a falhas inerente ao paradigma da orientação a objetos.
- Tratamento de Exceções: pode-se inferir que mantém o sistema de tratamento de exceções de C++.

6.3.4 Linguagem Java-RTR

Java-RTR [FUR 96] é uma linguagem orientada a objetos, reflexiva e de tempo real, que estende a linguagem Java e implementa o modelo de programação reflexivo de tempo real (modelo RTR - Reflexivo de Tempo Real). Desta forma, caracteriza-se por sua estrutura reflexiva e por sua capacidade de representação de aspectos temporais, além da flexibilidade de Java. Esta linguagem tem um conjunto de potencialidades inerentes à reflexão computacional e permite uma representação explícita e flexível dos aspectos temporais encontrados nas aplicações tempo real.

Java é uma linguagem orientada a objetos, desenvolvida pela Sun Microsystems Inc., que possui uma série de características importantes tais como: robustez, segurança, simplicidade, independência de ambiente operacional, concorrência, distribuição, portabilidade e alto desempenho. Entretanto, Java não apresenta suporte para a representação explícita de restrições temporais, nem possibilita a construção de programas com tempo de execução previsível, o que dificulta seu uso em muitos casos de programação de aplicações tempo real. Dessa forma, a extensão à linguagem proposta por Furtado e Farines [FUR 96], permite expressar e implementar restrições temporais de sistemas tempo real que seguem uma abordagem dita de "melhor esforço".

Devido à utilização do paradigma da orientação a objetos no projeto, é permitida a estruturação, a manutenção, a extensibilidade e a reutilização de programas. Estas são características inerentes ao paradigma e facilitam o gerenciamento da complexidade do sistema.

Além disso, a abordagem da reflexão computacional permite que em uma situação de tempo real aspectos relativos a restrições de tempo, sincronização e exceções sejam implementadas a nível de meta-objetos, que são apresentados em [LIS 95]. Assim, o desenvolvimento e a implementação das aplicações tempo real são simplificados e adquirem uma maior flexibilidade, além de aumentar a legibilidade e facilitar o uso, a manutenção e a reusabilidade do sistema.

A linguagem Java-RTR, por ser reflexiva, permite ainda a separação explícita entre a funcionalidade normal e o controle do comportamento temporal do sistema, o que possibilita que novas restrições temporais e novos algoritmos de escalonamento sejam definidos sem que isto implique em mudanças na base do sistema.

A confiabilidade da linguagem provém de algumas características apresentadas por Java, como: ênfase na verificação estática, coletor de lixo automático (simplificando a programação e eliminando erros de gerenciamento de memória) e mecanismo de tratamento de exceções. Além disso, Java implementa uma série de mecanismos de segurança, que fazem dela uma linguagem segura mesmo quando utilizada em ambientes baseados em redes.

Algumas das facilidades temporais, que permitem a sua utilização em sistemas tempo real, oferecidas por Java-RTR são:

- Métodos com restrições temporais: uma das extensões de Java-RTR é a implementação de classes RTB (*Real-Time Based*), que diferenciam-se das classes convencionais de Java por suportarem a associação de restrições e de tratadores de exceções relacionados ao tempo de execução e à declaração de seus métodos. Este esquema introduz a flexibilidade necessária para expressar restrições temporais, desde as mais comuns até restrições como tempo de início, intervalo de ativação e polimorfismo temporal.
- Ativação de métodos com restrição temporal: a sintaxe de ativação de um método é estendida para que os valores dos atributos de restrição temporal associada ao método sejam fornecidos na forma de um conjunto de parâmetros atuais. A não-correspondência em número e/ou tipo entre atributos temporais de uma restrição temporal com os valores atuais destes atributos, conduzirá a um erro de compilação.
- Cláusula referente ao mecanismo de controle do prazo máximo de tempo: esta cláusula tem como objetivo limitar o tempo de espera pelo resultado da execução de um método ativado, devendo ser utilizada em conjunto com um comando de ativação de método. Dessa forma, tratadores de exceções poderão ser ativados nas seguintes situações: esgotamento do prazo de tempo antes do retorno do método ativado, rejeição da ativação do método e interrupção da execução deste.

Em adição ao mecanismo de controle de tempo, outras cláusulas e restrições semânticas relativas ao uso de *loops*, recursividade, estrutura dinâmica e coletor de lixo automático são necessárias para que o tempo de execução dos programas escritos em Java-RTR possa ser calculado e assim a análise de escalonabilidade, bem como de previsibilidade, destes programas possa ser realizada.

Quanto à comunicação entre objetos em Java-RTR, esta pode ser realizada através de mensagens síncronas e assíncronas, sendo que estas últimas facilitam a implementação de restrições temporais, pela adaptação do modelo RTR.

Conforme a descrição aqui apresentada, pode-se considerar que, de acordo com as características anteriormente citadas como relevantes, Java-RTR apresenta o seguinte perfil:

- Flexibilidade: por ser uma extensão de Java, apresenta uma grande flexibilidade, incluindo aplicações tempo real.
- Gerenciamento de Tempo: o modelo RTR confere à linguagem Java-RTR uma série de características de manipulação de tempo, tornando-a ideal para ser utilizada em sistemas tempo real.

- Priorização: nada foi encontrado a este respeito no material consultado e a linguagem Java-RTR não foi testada para poder-se aferir algo a respeito desta característica.
- Reutilização: devido à utilização do paradigma da orientação a objetos no projeto, é permitida a reutilização de seus programas.
- Tolerância a Falhas: a confiabilidade provém de recursos implementados por Java, que incluem recursos de segurança.
- Tratamento de Exceções: além dos mecanismos de tratamento de exceções de Java, suporta tratadores de exceções relacionadas com os prazos de tempo.

6.3.5 Linguagem Lustre

O desenvolvimento de sistemas críticos induz à utilização de linguagens de programação funcionais, que disponibilizam métodos formais de especificação e verificação de sistemas. Um exemplo deste tipo de linguagem é Lustre, que trata-se de uma linguagem síncrona de fluxo de dados, utilizada para a especificação e verificação formal de sistemas de tempo real críticos [LIS 93].

Ainda de acordo com esta referência, as linguagens orientadas a fluxo de dados possuem um alto grau de paralelismo, por serem estruturadas em função das dependências de dados em um programa. Em arquiteturas orientadas a fluxo de dados, a disponibilidade de operandos em uma expressão dispara a execução das operações referentes que devem ser executadas sobre estes operandos; assim uma instrução estará apta a executar quando todos os seus argumentos são conhecidos.

Lustre apresenta características de programação que permitem, entre outras coisas, facilmente:

- expressar implicações lógicas;
- expressar eventos temporais;
- expressar propriedades de segurança;
- expressar asserções para o processo de verificação.

Um programa Lustre trata-se de um conjunto de equações, especificando relações entre variáveis e E/S. Cada programa apresenta um comportamento cíclico, dessa forma, a cada ciclo, todas as variáveis e expressões do programa tomam o n-ésimo valor de suas seqüências simultaneamente. Os programas são estruturados em nodos, compostos por um conjunto não-ordenado de equações e asserções. Dessa forma, cada nodo pode ser utilizado como um operador básico em qualquer expressão. Tendo em vista a sua filosofia de estruturação pode-se afirmar que é difícil reaproveitar programas Lustre, pois cada um é desenvolvido de acordo com as suas expressões, que vão originar o seu conjunto de equações e asserções e a seqüência de suas variáveis e expressões.

De uma forma geral, as linguagens funcionais, como Lustre, encontram maior aplicabilidade na prevenção de falhas do que no desenvolvimento de sistemas tolerantes a falhas propriamente ditos. Nesse sentido, especificamente as linguagens funcionais orientadas para fluxo de dados são usadas para projetar aplicações críticas, especificar

suas propriedades e verificá-las. Porém, os mecanismos de tratamento de exceções não costumam ser implementados neste tipo de linguagem, que é o que ocorre com Lustre.

Associada à linguagem Lustre, pode-se utilizar a ferramenta de verificação denominada Lesar, que possibilita a aplicação de técnicas de verificação por enumeração exaustiva dos estados de abstração do programa ou pela construção simbólica do conjunto de estados que satisfazem uma propriedade global do sistema.

Quanto à sua aplicabilidade, de acordo com Pilaud [PIL 88], Lustre é uma das linguagens de programação mais propícias para implementar o software de um sistema de controle de trens, principalmente quando é exigido que se utilize técnicas de prevenção de falhas durante o seu projeto.

Conforme a descrição aqui apresentada, pode-se considerar que, de acordo com as características anteriormente citadas como relevantes, a linguagem Lustre apresenta o seguinte perfil:

- Flexibilidade: não pode ser considerada muito flexível, pois é utilizada para a especificação e verificação formal de sistemas de tempo real críticos. Uma das aplicações em que é mais utilizada é em sistemas de controle de trens
- Gerenciamento de Tempo: permite que sejam facilmente expressos os eventos temporais.
- Priorização: nada foi encontrado a este respeito no material consultado, mas acredita-se que a linguagem não ofereça recursos para isto, visto que é mais utilizada na especificação e na verificação de sistemas tempo real.
- Reutilização: é difícil reaproveitar programas, pois cada um representa um conjunto de equações e asserções válidas para uma determinada aplicação.
- Tolerância a Falhas: de uma forma geral, as linguagens funcionais, como Lustre, encontram maior aplicabilidade na prevenção de falhas do que no desenvolvimento de sistemas tolerantes a falhas propriamente ditos. Lustre permite ainda que sejam facilmente expressas algumas propriedades de segurança.
- Tratamento de Exceções: as linguagens funcionais, de uma forma geral, não implementam mecanismos de tratamento de exceções.

6.3.6 Linguagem RT-CDL

A linguagem RT-CDL (*Real-Time Common Design Language*), apresentada por Liu e Shyamasundar [LIU 90], foi projetada especificamente para o desenvolvimento de sistemas reativos de tempo real distribuídos confiáveis. O objetivo principal do seu projeto é oferecer ferramentas capazes de diminuir a lacuna existente entre a especificação e a implementação de programas tempo real. A linguagem é baseada no modelo de evento-ação, é rigorosamente definida por uma semântica operacional e possui uma grande capacidade de expressão de restrições temporais, tais como [LIS 93]:

- permitir a difusão temporizada de mensagens;
- responder a exceções de tempo real dentro do prazo determinado;

- expressar restrições temporais de forma natural;
- atribuir prioridades distintas aos processos, de acordo com seus limites de tempo;
- controlar a expiração de tempo para realização de uma tarefa.

Um programa RT-CDL é composto por um conjunto de processos distribuídos que executam em paralelo, possuem memória distribuída, comunicam-se via mensagens *broadcast*[6] e cooperam entre si para atingir um objetivo comum. Cada componente em RT-CDL é um módulo que possui três partes distintas: a especificação, o projeto e a implementação do módulo. Dessa forma, pode-se concluir que uma vez especificados, projetados e implementados os módulos podem ser reaproveitados em diversos programas.

Como RT-CDL é uma linguagem baseada em eventos, seus mecanismos são capazes de manipular e detectar eventos externos e internos e, dessa forma, é mais utilizada no desenvolvimento de programas que seguem o paradigma das arquiteturas disparadas por eventos.

A linguagem apresenta um mecanismo de tratamento de exceções específico para tempo real. Exceções em RT-CDL são consideradas eventos e, da mesma forma que outras construções da linguagem, devem ser tratadas o mais rápido possível - de preferência, logo que ocorram - podendo ser detectadas por:

- sistema de suporte;
- dispositivos dedicados;
- programas do usuário.

As linguagens tempo real, além das exceções normais a qualquer tipo de linguagem, devem tratar as denominadas de exceções de tempo real, que correspondem às seguintes, previstas por RT-CDL:

- Exceções por violação de tempo: estas podem ser detectadas por mecanismos de vigilância ou de expiração de tempo e admitem as seguintes variações: violação de tempo mínimo e de tempo máximo, que referem-se ao tempo de execução de comandos; e de duração, que diz respeito ao período de espera para a difusão e recebimento de mensagens na rede.
- Exceções críticas: são detectadas por monitores de operação e exigem um tratamento imediato; um exemplo desta pode ser a detecção de uma temperatura anormal em um reator. Estas exceções ocorrem durante o progresso de operações críticas, podem ser comunicadas de forma síncrona ou assíncrona e são detectadas com base na descrição do comportamento de cada dispositivo dedicado.
- Exceções de indeterminação: este tipo ocorre quando não é possível escolher uma ação dentre um conjunto de ações possíveis; são capturadas e tratadas quando os eventos possuem cláusulas que definem um conjunto válido de transições e de precedência de eventos.

---

[6] Difundidas de forma generalizada.

As principais causas destas exceções de tempo real são erros de software (por exemplo, no programa), defeitos de hardware (por exemplo, defeito de um processador ou canal de comunicação) e contenção (por exemplo, um tráfego pesado no canal de comunicação).

Conforme a descrição aqui apresentada, pode-se considerar que, de acordo com as características anteriormente citadas como relevantes, RT-CDL apresenta o seguinte perfil:

- Flexibilidade: nada foi encontrado a este respeito no material consultado; entretanto a linguagem é considerada flexível, desde que utilizada para o desenvolvimento de sistemas reativos de tempo real distribuídos em geral.
- Gerenciamento de Tempo: possui uma grande capacidade de expressão de restrições temporais, conforme citado anteriormente.
- Priorização: permite atribuir prioridades distintas aos processos, de acordo com seus limites de tempo.
- Reutilização: uma vez especificados, projetados e implementados os módulos podem ser reaproveitados em diversos programas.
- Tolerância a Falhas: apesar de não ser apresentada nenhuma consideração a este respeito na bibliografia consultada, permite, através do tratamento de exceções, um certo grau de tolerância a falhas e uma maior confiabilidade.
- Tratamento de Exceções: além das exceções normais a qualquer tipo de linguagem, trata as denominadas de exceções de tempo real. Porém seus recursos de tratamento de exceções são obtidos sem que apresente um mecanismo exclusivo para isso.

6.3.7 Linguagem RTT

A linguagem RTT (*RealTimeTalk*) descrita por Gustafsson [GUS 94], é uma linguagem orientada a objetos utilizada para a construção de sistemas tempo real críticos. A idéia da linguagem é utilizar os conceitos do paradigma da orientação a objetos, que permitem o desenvolvimento de aplicações complexas tais como as de tempo real, de forma coordenada com as garantias que o sistema deve ter quanto ao seu comportamento temporal. O principal objetivo desta linguagem é simplificar a modelagem e o projeto de sistemas tempo real previsíveis.

A linguagem é baseada em Smalltalk, mantendo a mesma sintaxe para seus comandos. A diferença é que tanto métodos quanto comandos separados podem ser escritos na linguagem C. Assim, é possível a reutilização de códigos já existentes. Entretanto algumas mudanças foram necessárias para a manipulação dos requisitos de tempo real, tais como:

- impedir a criação dinâmica de (novas) classes (em tempo de execução);
- impedir recursividade, *loops* ilimitados e *loops* sobre estruturas de dados ilimitadas;

- permitir apenas a invocação explícita de blocos, nunca via parâmetros ou variáveis.

A base de uma aplicação RTT são os modos operacionais e as transições entre estes modos. Os serviços que cada modo inclui correspondem a todas as computações existentes entre um estímulo do ambiente e uma resposta devolvida ao mesmo, e são definidos por:

- suas características temporais, calculadas a partir dos requisitos temporais;
- um grafo de precedência, determinando a ordem dos objetos;
- um grafo de relação entre objetos, definindo suas respectivas associações.

Todos os serviços e atividades construídos pela linguagem são periódicos, ou seja, são disparados por tempo.

Por RTT ser baseada em Smalltalk, pode-se assumir que ela tenha herdado o seu sistema de tratamento de exceções. Dessa forma, ela incorpora o tratamento à exceção básica de Smalltalk, que refere-se ao seguinte evento: uma mensagem é enviada a um objeto, mas esta mensagem não é especificada em qualquer classe da hierarquia. Além desta exceção, os métodos da aplicação podem capturar mensagens referentes a outras ocorrências excepcionais detectadas pelo sistema de suporte, e providenciar tratadores específicos para estas exceções. Mensagens de erro são suportadas pelo protocolo da classe. Objetos e subclasses podem ser usados para capturar as mensagens de erro e tomar providências para a correção da situação errônea [LIS 93].

Conforme a descrição aqui apresentada, pode-se considerar que, de acordo com as características anteriormente citadas como relevantes, a linguagem RTT apresenta o seguinte perfil:

- Flexibilidade: é adequada para implementar sistemas tempo real críticos, previsíveis e disparados por tempo.
- Gerenciamento de Tempo: todos os serviços e atividades construídos pela linguagem são periódicos, ou seja, são disparados por tempo e podem ser controlados.
- Priorização: nada foi encontrado a este respeito no material consultado e a linguagem RTT não foi utilizada na prática para poder-se analisar este ponto.
- Reutilização: por permitir que partes do código também sejam escritas em C, possibilita o reaproveitamento de códigos já existentes.
- Tolerância a Falhas: nada foi encontrado a este respeito no material consultado, entretanto pode-se contar com a tolerância a falhas inerente ao paradigma da orientação a objetos.
- Tratamento de Exceções: pode-se assumir que ela tenha herdado o sistema de tratamento de exceções de Smalltalk.

6.3.8 Modelo RTO.k

Além das linguagens de programação, foi estudado um modelo tempo real orientado a objetos denominado RTO.k (*Real-Time Object*). Este modelo foi desenvolvido por Kim [KIM 94] e é descrito como um esquema de estruturação de objetos tempo real. A abordagem apresentada pelo modelo propõe a realização de computação tempo real na forma de uma generalização de computação não-tempo real e ainda permite aos engenheiros do sistema definirem sistemas tempo real certamente confiáveis para aplicações de segurança crítica. Algumas das características principais do modelo são:

- Separação entre os dois tipos de métodos: o modelo contém dois tipos de métodos, os disparados por tempo (ou espontâneos) - SpM's - e os métodos convencionais de serviço - SvM's.
- Restrição básica de concorrência: trata-se de uma regra que é utilizada para prevenir conflitos que possam ocorrer entre os SpM's e os SvM's. É basicamente a seguinte: "a ativação de um SvM disparado por uma mensagem de um cliente externo somente é permitida quando não há conflito de execução com nenhum SpM".
- Prazo: é imposto um prazo para a execução de cada método no modelo RTO.k.
- Duração de validade máxima: após um certo intervalo de tempo, os dados tempo real de um objeto RTO.k tornam-se inválidos.

Especificamente para suportar as garantias de cumprimento de prazos que devem ser obedecidas pelos serviços RTO.k foi desenvolvido o modelo de um *kernel* de sistema operacional, denominado DREAM (*Distributed Real-time Ever Available Micro-computing*). Para maiores detalhes sobre a implementação deste modelo, sugere-se que sejam consultadas as referências de apoio [KIM 94, KIM 95 e KIM 96].

6.4 Análise das Linguagens de Programação Tempo Real

As linguagens de programação anteriormente apresentadas serão comparadas nesta seção, através das mesmas características exploradas na análise individual. Com exceção do modelo RTO.k, apresentado na seção 6.3.8, que não foi estudado suficientemente e cujas características não correspondem às mesmas das linguagens para que possa ser comparado. As principais características que serão consideradas a respeito das linguagens de programação envolvem as questões relacionadas na seção 6.1 deste trabalho.

A fim de facilitar a comparação, os aspectos apresentados de forma qualitativa na seção anterior são agora traduzidos em números. A análise, apresentada na Tabela 6.1, atribui graus a cada uma das linguagens de programação estudadas com a seguinte correspondência:

- Grau 2: significa que a linguagem de programação apresenta esta característica sempre.
- Grau 1: significa que esta característica está parcialmente presente na linguagem de programação (com restrições).
- Grau 0: significa que esta característica não está presente na linguagem de programação.

TABELA 6.1 - ANÁLISE COMPARATIVA DAS LINGUAGENS TEMPO REAL

Características	Ada	Chill	DEAL	Java-RTR	Lustre	RT-CDL	RTT
Flexibilidade	<2>	<1>	<1>	<2>	<1>	<1>	<1>
Gerenciamento de Tempo	<2>	<0>	<2>	<2>	<2>	<2>	<2>
Priorização	<2>	<1>	---	---	<0>	<2>	---
Reutilização	<2>	<2>	<2>	<2>	<0>	<2>	<2>
Tolerância a Falhas	<1>	<1>	<1>	<2>	<1>	<1>	<1>
Tratamento de Exceções	<2>	<2>	<1>	<2>	<0>	<2>	<1>

7 Estudos de Casos

O objetivo deste capítulo é demonstrar praticamente como as taxonomias desenvolvidas no decorrer deste trabalho podem auxiliar projetistas a tomarem decisões importantes durante o desenvolvimento de software para sistemas tempo real tolerantes a falhas. Conforme o escopo aqui tratado, estas decisões estão relacionadas ao software de apoio a ser utilizado: protocolo de comunicação, sistema operacional e linguagem de programação. Dessa forma, utilizou-se as taxonomias na definição de algumas sugestões para alternativas de projeto de três casos-exemplo de sistemas tempo real, como forma de validação. Estes estudos de caso envolvem situações distintas, com soluções aplicadas a cada caso, visando garantir a máxima confiabilidade e disponibilidade do sistema, obtendo segurança no seu funcionamento, a dependabilidade.

A idéia dessa demonstração é expor como devem funcionar e quais são os pré-requisitos de alguns sistemas tempo real responsáveis pelo controle de trens e de ônibus espaciais. E, a partir dessas descrições, indicar como podem ser tomadas algumas decisões a respeito de que protocolo de comunicação, sistema operacional e linguagem de programação usar durante o seu desenvolvimento, com base nas análises e comparações desenvolvidas nos capítulos 4, 5 e 6 desta dissertação.

Entretanto, a descrição destes casos é baseada nos artigos de David e Guidal [DAV 93], Hachiga [HAC 93] e Hennebert e Guiho [HEN 93], que apresentam apenas uma visão geral das aplicações consideradas. Dessa forma, eles não oferecem descrições detalhadas, mas apenas colocam em perspectiva algumas características das referidas aplicações. Porém, sabe-se que para realmente realizar a análise e sugestões para o projeto, as características do sistema devem ser bastante realistas e tão detalhadas quanto possível. Assim, dentro do que se obteve de informações, realizou-se um estudo preliminar de requisitos seguido de algumas sugestões possíveis para o projeto destes sistemas.

7.1 Sistema para o Ônibus Espacial Hermes

A descrição deste caso trata-se de uma adaptação daquela realizada por David e Guidal [DAV 93]. Dessa forma, para obter-se maiores detalhes sobre o sistema sugere-se a leitura do artigo referido.

O ônibus espacial Hermes foi projetado para realizar 30 missões de 10 dias cada, a serviço da Estação Espacial Européia, durante um período de 15 anos. O sistema que controla este ônibus deve operar com requisitos estritos de segurança de funcionamento, medida através da sua disponibilidade e confiabilidade (probabilidade menor que 10^{-6} de ocorrer algum evento catastrófico induzido por um defeito do sistema).

Este sistema é composto por quatro computadores (na verdade são três fisicamente presentes e um simulado) executando simultaneamente (em paralelo) e firmemente sincronizados. Os computadores estão interligados por uma rede e possuem comunicação com o mundo externo através de dispositivos sensores e atuadores. Cada

um deve divulgar seus dados de E/S através da rede aos demais, implementando o conceito de mascaramento de falhas baseado na votação bit-a-bit. A ocupação do canal é feita através de frações atribuídas periodicamente aos computadores.

É exigido, por ser uma aplicação extremamente crítica, que este sistema de computação seja tolerante a falhas. Ele deve suportar programação diversitária, onde as diferentes versões do software executam em diferentes computadores, implementando o conceito de programação n-versões. Com isso, o sistema deve garantir que:

- nenhuma falha simples levará a conseqüências graves ou catastróficas;
- nenhuma segunda falha, independente da primeira, levará a um efeito catastrófico.

Assim, a função de controle geral do sistema deve continuar operacional mesmo depois da ocorrência de duas falhas independentes. Para isso, é necessário que o sistema apresente a capacidade de reconfigurar-se dinâmica e rapidamente após uma falha. Por outro lado, deve-se salientar que este sistema não apresenta um estado seguro após a ocorrência de uma falha[7] muito simples de ser atingido: quando ocorre uma falha não controlável ou prevista, não há como parar o ônibus espacial. Isto implica em uma necessidade maior de empregar tolerância a falhas.

Além disso, quanto mais crítica for a conseqüência de uma falha, menor deve ser a probabilidade dela ocorrer. Nesse sentido, devem ser utilizados dois conhecidos conceitos de tolerância a falhas na implementação do sistema: mascaramento e programação auto-verificadora.

O sincronismo entre os computadores é fundamental para permitir a tolerância a defeitos nos sensores (que são redundantes) e aos erros na transmissão, aquisição e processamento dos dados; visto que todos os computadores devem receber o mesmo conjunto de dados de entrada ao mesmo tempo, para habilitarem-se a gerar um conjunto de comandos idênticos aos atuadores depois de uma votação bit-a-bit de suas saídas.

Dessa forma, no que se refere a tolerância a falhas, devem ser implementados os seguintes mecanismos:

- detecção de erros baseada na comparação de dados e na verificação de paridade;
- recuperação de erros com o propósito de desfazer os efeitos incorretos provocados pela falha, substituindo um dado errado por um correto, o que permite a execução do serviço sem interrupção;
- indicação se a ocorrência de um erro é proveniente de uma falha permanente ou transitória;
- verificação do sincronismo entre os computadores;
- verificação da comunicação na rede através do CRC associado a cada mensagem;

---

[7] O termo original é *fail-safe*, que significa a existência de um estado seguro para o qual o sistema deve ir em caso de falha.

- diagnóstico de falhas, que consiste em verificar erros detectados para evitar reconfigurações desnecessárias;
- isolamento do computador falho, para realizar a reconfiguração e evitar que os efeitos de uma falha se propaguem através da rede.

7.1.1 Sugestões de Projeto

Com base na descrição do funcionamento do sistema e de posse das taxonomias (e da análise das ferramentas) desenvolvidas neste trabalho, pode-se sugerir que sejam utilizados, por exemplo, os seguintes componentes de software de apoio:

- Protocolo de Comunicação: os computadores comunicam-se através da rede por frações de tempo que lhes são atribuídas; por isso, é necessário que seja utilizado um protocolo que faça uso desta filosofia. Além disso, o sincronismo entre os computadores é fundamental. Assim, o TTP apresenta-se como uma boa alternativa de projeto, pois além de apresentar estas características, implementa detecção de erros através de CRC e outros mecanismos de tolerância a falhas. Por tratar-se de uma aplicação aeroespacial, talvez uma alternativa de projeto também seja utilizar o ARINC 629, que foi projetado especificamente para estas aplicações, apesar de oferecer menos recursos que o TTP.
- Sistema Operacional: o sistema deve ser rápida e dinamicamente reconfigurável; além disso, ele é distribuído e exige alta confiabilidade e disponibilidade. Diante disso, o QNX parece ser o sistema ideal, pois permite reconfiguração dinâmica, é distribuído, implementa tolerância a falhas e pode ser executado em PCs comuns (uma vez que nada foi dito sobre o tipo de computador utilizado, pode-se supor que sejam estes). Outra alternativa seria o Portos-TF, que apresenta igualmente a capacidade de reconfiguração dinâmica, é distribuído, implementa tolerância a falhas e também é portátil. Porém, a implementação real deste projeto foi realizada utilizando o *kernel* tempo real ASTRES 1750, que não foi considerado neste trabalho.
- Linguagem de Programação: a linguagem Ada apresenta-se como uma opção interessante, visto que oferece todos os recursos necessários para implementação de um sistema tempo real crítico e tolerante a falhas e tornou-se um padrão no desenvolvimento de software para esta categoria de sistemas. Cabe salientar que, conforme descrito no artigo de David, o sistema do ônibus espacial Hermes foi implementado em módulos diferentes, que foram realmente escritos utilizando-se a linguagem Ada.

7.2 Sistema para Controle de Velocidade de Trem

A descrição deste caso trata-se de uma adaptação daquela realizada por Hennebert e Guiho [HEN 93]. Dessa forma, para obter-se maiores detalhes sobre o sistema sugere-se a leitura do artigo referido.

Face ao crescimento contínuo na demanda da linha A do RER (o metrô expresso regional de Paris), que transporta diariamente um milhão de passageiros, os responsáveis decidiram aumentar a sua capacidade em 25% (vinte e cinco por cento) nas horas de maior movimento. Entretanto, para isso é preciso:

- equipar os trens com um sistema automático de controle da sua velocidade contínua, para garantir a segurança do tráfego;
- oferecer uma ajuda *on-line*, que utiliza dados a respeito do avanço e da velocidade permitida para o trem, fornecida por uma sinalização na cabina do maquinista.

Em vista disso, foi proposta a construção de um sistema de computação responsável por controlar a velocidade e os movimentos do trem RER-A, tornando informatizadas inclusive informações de segurança crítica. Dessa forma, aspectos de dependabilidade do sistema deveriam ser preservados, através de técnicas de reforço da segurança, como a detecção *on-line* de erros e a validação do software.

O transporte em estradas de ferro possui o benefício de apresentar um estado estável em relação à segurança: parar o trem. Assim, em um sistema automatizado que apresenta um estado seguro após a falha, qualquer detecção de um erro que contrarie a segurança deve resultar em uma parada total do sistema com alta taxa de confiabilidade, gerada por uma freada do trem.

A partir da decisão de adoção do sistema informatizado, foram fixados os seguintes objetivos:

- usar exaustivamente técnicas de prevenção para evitar a ocorrência de erros que possam levar à parada total do sistema;
- transmitir continuamente informações descritivas sobre o estado da estrada de ferro que se encontra à frente da posição atual do trem, para onde ele está se deslocando;
- obter alta disponibilidade proveniente da tolerância a falhas e da capacidade de auto-verificação do sistema - a freqüência das paradas inadvertidas do sistema deve ser menor do que 2.10^{-3} por trem e por hora;
- garantir que a probabilidade de ocorrência de um defeito catastrófico ao sistema seja inferior a 10^{-9} por trem e por hora de utilização;
- tirar proveito de uma melhor adaptabilidade face às mudanças inevitáveis nas condições de uso durante a vida do sistema

O equipamento utilizado para este controle é composto por um equipamento fixo na estrada de ferro e um agregado ao trem (mas que nem todos os trens possuem) e periféricos de E/S. Cada equipamento fixo na beira da estrada é responsável por controlar uma área geográfica equivalente a uma interestação. Sua função é coletar informações, originárias da detecção da presença de um trem na área, sobre suas rotas e sinalizar aos demais trens. Então, o equipamento de bordo dos outros trens recebe estas informações e deve realizar as seguintes ações:

- identificar a existência de algum obstáculo à sua frente que possa impedir sua passagem;
- determinar a distância entre a posição do trem e a abcissa do obstáculo;

- comparar esta distância com a distância de desaceleração necessária para compensar a energia cinética e a energia potencial;
- gerar uma ordem que, de acordo com a diferença entre as distâncias, pode corresponder a uma freada automática e irreversível ou uma sinalização, na unidade *display* da cabina do maquinista, com uma informação de desaceleração indicando a velocidade alvo ou a velocidade permitida.

Esta sinalização deve ser mantida, mesmo com alguma degradação, no caso de uma falha no sistema, para permitir o tráfego de trens não equipados com este dispositivo.

Para garantir a segurança do sistema, mesmo na presença de falhas de hardware, é implementada uma abordagem que utiliza redundância dos equipamentos e uma unidade comparadora dos resultados gerados. Esta configuração está em acordo com as condições de funcionamento dos protocolos propostos. Já em relação ao software é feita essencialmente a prevenção de falhas durante a fase de projeto do mesmo. Além disso, a cada energização do sistema, ou durante uma reativação depois de um período em que ele esteve passivo, é necessário que um software de auto-verificação teste se todos componentes, inclusive as interfaces, estão operando adequadamente. Em caso negativo, o sistema não é inicializado. Esta providência evita que uma falha latente crie uma parada de emergência. Durante a execução do programa, qualquer erro de software detectado, não tratável dinamicamente, resulta na parada de emergência do sistema.

7.2.1 Sugestões de Projeto

Com base na descrição do funcionamento do sistema e de posse das taxonomias (e da análise das ferramentas) desenvolvidas neste trabalho, pode-se sugerir que sejam utilizados, por exemplo, os seguintes componentes de software de apoio:

- Protocolo de Comunicação: esta aplicação deve ser capaz de reagir rapidamente a eventos assíncronos, assim, a solução mais natural é enquadrada no paradigma de sistemas disparados por eventos. Com isso uma boa alternativa de projeto seria a utilização do protocolo CAN, que implementa detecção *on-line* de erros (CRC), uma das imposições do sistema. Outra boa opção seria uma versão do protocolo LON, que é amplamente utilizado em sistemas dinâmicos, onde a estratégia de "melhor esforço" é utilizada, como o caso aqui proposto, e suas versões também implementam detecção de erro por CRC.
- Sistema Operacional: uma das sugestões de projeto para o sistema é o MTOS-UX, que tem se apresentado como uma boa alternativa para implementação de sistemas tempo real. Além deste, destacam-se o Portos-TF e o QNX, que apresentam fortemente aquelas características desejáveis para o desenvolvimento de um software confiável (como gerenciamento de redundância, reconfigurabilidade e tolerância a falhas, por exemplo).
- Linguagem de Programação: é importante salientar que deve-se prevenir falhas no software do sistema. Diante disso, vale lembrar que Lustre é uma das linguagens de programação mais propícias para implementar o software de um

sistema de controle de trens e implementa prevenção de falhas, por ser uma linguagem funcional. Por outro lado, sendo este um sistema considerado reativo (disparado por eventos), RT-CDL também pode ser utilizada na sua implementação, pois oferece recursos de tratamento de exceções e alguns de tolerância a falhas, além de todas as facilidades para tempo real. RT-CDL possui justamente o objetivo de tentar minimizar a lacuna existente entre a especificação e a implementação de sistemas tempo real, aproximando estas duas fases de projeto, o que é conveniente para esta aplicação. É importante ressaltar que, de acordo com Hennebert, o software deste sistema foi escrito utilizando-se a linguagem Modula 2, que não foi abordada neste trabalho.

7.3 Sistema para o Controle do Trem Shinkansen

A descrição deste caso trata-se de uma adaptação daquela realizada por Hachiga [HAC 93]. Dessa forma, para obter-se maiores detalhes sobre o sistema sugere-se a leitura do artigo referido.

O Shinkansen é conhecido por ser um sistema de transporte público seguro, confiável, popular e rápido no Japão. Todos os dias este trem carrega mais de 700 mil passageiros entre localidades a centenas de quilômetros de distância. Uma de suas características é que o atraso máximo é de um minuto por trem, em média.

Para controlar o Shinkansen deve ser projetado um sistema computacional tolerante a falhas, a fim de permitir que ele continue operante mesmo na presença de condições anômalas, e que utilize uma arquitetura disparada por tempo [KOP 94].

Geralmente, há três tipos básicos de situações que devem ser evitadas para garantir a segurança dos trens:

- colisões com automóveis no cruzamento de estradas;
- colisões com outros trens;
- descarrilhamento do trem.

No caso deste trem, não há como ocorrer uma colisão com automóveis, porque seu trilho e a estrada localizam-se em níveis diferentes e não se cruzam. Já as colisões com outros trens e o descarrilhamento devem ser evitados através do controle da sua velocidade e da sua rota, o que deve ser realizado pelo sistema computacional.

Um sistema é considerado do tipo seguro a falhas se os efeitos de uma falha aparecerem como uma operação segura. O estado seguro após uma falha num sistema de trens corresponde a parar imediatamente o trem, quando for encontrado qualquer tipo de risco. Os conceitos empregados para garantir este estado seguro em trens são:

- definir explicitamente um estado seguro e um estado de risco;
- alterar a saída para um estado seguro no caso de defeito;
- implementar redundância através de hardware, software ou ambos;
- usar módulos independentes para redundância;
- detectar imediatamente erros em dados;

- identificar rapidamente falhas latentes.

Um sistema do tipo seguro a falhas pode ser composto de um conjunto de computadores de propósitos gerais e um mecanismo de detecção de erros, que pode ser implementado usando diversidade de software ou redundância de hardware. Como o hardware tem se tornado cada vez mais barato, a tendência é de se usar redundância de hardware. Mas este é um sistema crítico, assim parte-se da hipótese de que adicionalmente é interessante também utilizar redundância de software, o que vem ao encontro dos objetivos deste trabalho.

Dessa forma, os requisitos para implementar tolerância a falhas em software no Shinkansen são os seguintes:

- o trem deve funcionar 18 horas por dia (das seis horas à meia-noite);
- o sistema pode ficar inoperante, em decorrência de defeitos, por uma hora por ano, no máximo;
- MTTR (*Mean Time To Repair*) é preferível a MTBF (*Mean Time Between Failure*) porque os passageiros não estão dispostos a aceitar interrupções longas nos serviços do trem, mesmo que estas sejam escassas ou raras;
- o sistema utilizado para o controle do trem (sua rota e velocidade) deve ter uma disponibilidade maior do que o utilizado para processamento de seus dados;
- a expansão do software deve poder ser realizada concorrentemente com a operação do trem, ou seja, o sistema não deve ser interrompido quando for realizada a sua expansão.

Ao contrário do que foi feito nos casos anteriores, para este caso será apresentado um sistema de controle já existente e após será feita a análise de algumas decisões que poderiam ter sido tomadas durante o seu projeto. O sistema a ser descrito foi projetado especificamente para o controle de um trem de alta velocidade como o Shinkansen e chama-se COMTRAC (*Computer Aided Traffic Control*).

Hoje em dia, seria praticamente impossível o controle do funcionamento do Shinkansen sem o COMTRAC. Mais do que um simples controlador da rota do trem, o sistema teve sua função estendida e passou a controlar a operação de todas as tarefas do trem. São utilizados três sistemas de computação para controlar as funções do COMTRAC: o sistema de processamento de dados, o sistema de controle das rotas e o sistema de informação para o passageiro. Nesse sentido, o sistema foi dividido em três módulos básicos que possuem as funções descritas a seguir.

- Geração de escalonamento: o escalonamento do trem envolve necessariamente as tarefas de cada trem e da tripulação. O escalonamento básico é renovado várias vezes, baseado na demanda do transporte. O sistema de processamento de dados é o responsável por compilar o escalonamento sazonal ou diário do trem para o escalonamento básico. O sistema de processamento de dados também prepara as informações para os passageiros em cada estação e transmite o escalonamento diário do trem para o sistema de controle das rotas e para o sistema de informação para o passageiro.
- Regulagem do tráfego: ambos, o sistema de processamento de dados e o sistema de controle das rotas, são responsáveis por regular o tráfego. Uma vez

a cada dez minutos o sistema de processamento de dados prediz o estado do tráfego para as próximas três horas, enquanto tenta detectar qualquer conflito nas tarefas dos trens e das respectivas tripulações. Todos os trens em operação são rastreados independentemente pelo sistema de controle das rotas. A localização de cada trem é mostrada no painel. Se o sistema de controle das rotas detecta qualquer distúrbio no tráfego e decide mudar o escalonamento do trem, ele faz uma recomendação ao responsável pelo controle, que decide aceitar ou rejeitar a mudança proposta. Uma vez que a proposta tenha sido aceita, o arquivo de escalonamento é alterado e o sistema de processamento de dados é informado. Estes dois sistemas são responsáveis por regular o tráfego, entretanto qualquer decisão final é tomada por uma pessoa responsável pelo controle geral.

- Controle de informações aos passageiros sobre a rota do trem: o sistema de informação para o passageiro também rastreia cada trem. Quando um trem aproxima-se ou parte da estação, o sistema indica as suas rotas de origem e destino. Ele controla ainda o anúncio e as placas de chegada e partida em cada estação, informando os passageiros sobre o estado do tráfego. Uma vez por dia ele recebe informações do sistema de processamento de dados e se ocorre alguma mudança no escalonamento do trem, ele também deve ser informado para poder modificar as informações dadas aos passageiros sobre o tráfego.

O COMTRAC foi desenvolvido para operar sobre um sistema de controle de tráfego centralizado já existente. Para satisfazer os requisitos de tolerância a falhas, os sistemas de processamento de dados, de controle das rotas e de informação aos passageiros operam de forma redundante em dois, três e dois computadores respectivamente, enquanto que o sistema de controle de tráfego centralizado opera em três computadores, que implementam TMR. A razão para isto é que este controle deve ter a maior disponibilidade e continuar operando para que seja possível controlar o Shinkansen manualmente se necessário.

O sistema de processamento de dados e o sistema de informação aos passageiros executam em dois computadores; um está sempre ativo e o outro é o reserva. As extensões de software são desenvolvidas e testadas no computador reserva. Quando o computador ativo falha, o reserva assume o seu lugar; entretanto o tempo de chaveamento é perfeitamente aceitável pelo sistema. Já o sistema de controle de rotas usa três computadores; dois deles são ativos, utilizando o conceito "mesmo software em diferentes computadores" (reserva-quente), enquanto o terceiro é o reserva. O propósito de ter dois computadores ativos neste sistema é reduzir a probabilidade de um erro no computador causar um controle incorreto das rotas. Durante o processamento do sistema de controle de rotas são definidos pontos de recuperação, onde é feita a verificação se os dados dos dois computadores ativos estão em concordância. Para isso, as tarefas de ambos são sincronizadas. Quando os dois computadores ativos falham simultaneamente, deve-se esperar algum tempo para o reserva assumir o comando. Entretanto, neste meio tempo, o trem continua operando e há uma possibilidade de algumas ordens serem trocadas, alterando o seu arquivo. Conseqüentemente, este arquivo deve ser recuperado tão breve quanto possível, o que implica em também haver um sistema de arquivos redundante.

Em ordem de importância, do ponto de vista de tolerância a falhas, as maiores funções do COMTRAC são: controle das rotas dos trens/ informação aos passageiros;

regulagem do tráfego e geração do escalonamento. A função que deve sobreviver apesar da ocorrência de qualquer tipo de falha é o controle das rotas dos trens, por isso este sistema trabalha com três computadores e pode continuar operando enquanto pelo menos um deles estiver funcionando. Se os três computadores falharem, deve ser possível continuar controlando as rotas dos trens manualmente através do sistema de controle de tráfego centralizado.

Entretanto, para garantir o perfeito funcionamento do Shinkansen, além do COMTRAC, devem ser feitas manutenções periódicas nas estradas de ferros, trilhos e outros equipamentos indispensáveis para a segurança no funcionamento do trem. Existe disponível ainda um sistema de alarme que avisa caso esteja prevista a ocorrência de fenômenos de risco (como ventanias, tempestades, terremotos, entre outros).

7.3.1 Sugestões para o Projeto do COMTRAC

Com base na descrição do funcionamento do sistema e de posse das taxonomias (e da análise das ferramentas) desenvolvidas neste trabalho, pode-se sugerir que sejam utilizados para implementar o COMTRAC, por exemplo, os seguintes componentes de software de apoio:

- Protocolo de Comunicação: o sistema deve ser disparado por tempo, então igualmente o protocolo deve utilizar este paradigma. Assim, o TTP apresenta-se como uma boa solução, pois implementa ainda detecção de erros, tolerância a falhas e sincronização de nodos, além de outros recursos. Diante destas características, outra alternativa talvez fosse o protocolo ARINC 629, porém este é mais voltado para aplicações de aeronáutica.
- Sistema Operacional: por este ser um sistema distribuído, o sistema operacional deve ser específico para isto. Dentre os sistemas operacionais distribuídos apresentados, o QNX parece ser o mais adequado, visto que sobre DEDOS não se tem informações quanto aos seus recursos de tolerância a falhas e o HARTOS foi desenvolvido visando uma arquitetura específica (hexagonal). Porém o Portos-TF também apresenta-se como uma alternativa viável e, além deste, o MTOS-UX pode ser uma boa solução. Todos os sistemas citados como possíveis têm em comum o fato de utilizarem tolerância a falhas e outros recursos que facilitam a implementação de sistemas tempo real.
- Linguagem de Programação: Lustre é uma das linguagens de programação mais propícias para implementar o software de um sistema de controle de trens e implementa prevenção de falhas, por ser uma linguagem funcional. Mas outra alternativa seria a linguagem Ada, que encontra-se entre as linguagens mais utilizadas na implementação de sistemas tempo real confiáveis e já é considerada um padrão nesta área.

8 Conclusão

Em aplicações onde a máxima segurança é exigida, manifestando-se através de parâmetros como confiabilidade e disponibilidade, não é interessante e, nem mesmo possível, deixar o controle da situação exclusivamente sob o comando humano, visto que este muitas vezes é falho. No sentido de suprir esta deficiência, surgiram os sistemas de computação tempo real. Este tipo de sistema interage com o ambiente, reagindo a estímulos provenientes de eventos externos e produz resultados, dentro de prazos de tempo limitados e previamente especificados.

Entretanto, como qualquer outro sistema baseado em computador - composto de hardware e de software - nem mesmo estes estão livres de falhas. E, caso uma falha indesejável ocorra, pode-se pôr em risco o processo controlado, ou ter como conseqüências catástrofes no ambiente da aplicação em que estão inseridos, além de prejuízos econômicos e perdas de vidas humanas, uma vez que estes sistemas normalmente são aplicados no controle de situações mais críticas. Para garantir a continuidade do seu funcionamento correto, tornou-se indispensável agregar técnicas de tolerância a falhas a esta categoria de sistemas. Com isso, objetiva-se inclusive um aumento de qualidade no sistema resultante, visto que altos índices de confiabilidade e disponibilidade são inerentes a este tipo de aplicações. No entanto as técnicas de tolerância a falhas devem ser agregadas de forma bastante cautelosa ao projeto dos sistemas, considerando que esta área de aplicação tem nos tempos de resposta um requisito crítico e atrasos são também vistos como "falhas" muitas vezes inaceitáveis para estes sistemas.

Hoje em dia, o hardware pode ser considerado seguro, apresentando altos índices de confiabilidade e disponibilidade. Além disso, ele tornou-se significativamente barato, o que permite que as técnicas de tolerância a falhas sejam facilmente implementadas a este nível. Em vista disso, o projeto e a produção de software se tornará mais importante no futuro. Segundo Hachiga [HAC 93], as experiências têm mostrado que as estruturas de software mais simples são as mais confiáveis. E esta talvez seja uma referência a ser seguida quando se pensar no desenvolvimento de software para aplicações tempo real em um futuro não muito distante, onde pode-se vislumbrar a próxima geração de sistemas tempo real.

As aplicações de tempo real tolerantes a falhas constituem-se em uma tendência crescente no mercado atual, principalmente quando são considerados sistemas críticos no tempo ou na atuação. A constatação de um crescimento acelerado na utilização de sistemas tempo real no nosso dia-a-dia, faz pensar na próxima geração de sistemas tempo real, onde a tolerância a falhas deve ser uma prática inerente às suas propriedades. Além disso, certamente em pouco tempo, o comportamento funcional e confiável de um sistema tempo real será determinado em função da capacidade do sistema prover serviços ininterruptos e dentro do prazo previsto aos clientes, possivelmente utilizando-se de técnicas de tolerância a falhas. Outras características desejáveis nesta próxima geração e que estão pouco presentes ou desenvolvidas nos sistemas utilizados hoje em dia são, por exemplo, a flexibilidade e principalmente a previsibilidade, conforme foi possível constatar através das análises efetuadas neste trabalho.

Pode-se prever que a próxima geração atuará em áreas similares às atuais, provavelmente expandindo-as um pouco. No entanto, os sistemas deverão ser mais complexos, englobando características que seguem as tendências nos sistemas de computação em geral, como: distribuição, inteligência artificial e adaptabilidade. Mas principalmente, a próxima geração deve ser projetada para ser dinâmica e flexível. Exemplos de aplicações futuras que poderão ser controladas por estes sistemas de tempo real são: sistemas de manufatura inteligente e estações espaciais.

Entretanto, pode-se constatar, diante da bibliografia consultada e dos estudos efetuados, que ainda é necessário realizar muita pesquisa envolvendo tolerância a falhas nesta categoria de sistemas. Será preciso muito desenvolvimento ainda até a aplicação prática das técnicas de tolerância a falhas poder atingir o patamar desejado, visto que tais técnicas não podem ser indistintamente aplicáveis em qualquer situação. Acredita-se, no entanto, que o presente trabalho tenha sido um passo inicial, motivador desse estudo dirigido, pelo menos dentro do ambiente acadêmico (o Instituto de Informática da Universidade Federal do Rio Grande do Sul), onde a exploração prospectiva das questões relacionadas aos sistemas tempo real está recém começando a ser descoberta pelo grupo de tolerância a falhas.

Por outro lado, pode-se observar que sistemas de computação têm sido largamente utilizados no controle de processos em tempo real dos mais variados tipos de aplicações. E um dos problemas que se depara no desenvolvimento desse tipo de sistema é que as características computacionais destes processos controladores são fortemente influenciadas pelas características do processo controlado. Assim, para o desenvolvimento de sistemas tempo real, é preciso ter um profundo conhecimento sobre a aplicação na qual ele será inserido. Esta é uma dificuldade real no projeto de sistemas tempo real, pois é necessária uma integração adequada da equipe de informática e daquela que conhece e define os pontos funcionais da aplicação. Uma das atividades onde fica clara a necessidade desta integração é na definição da questão de falhas: avaliar suas conseqüências, procedimentos devidos e caracterização de estados seguros, são questões resolvíveis apenas no conjunto. E mesmo antes do uso, durante a verificação da correta transformação das especificações em um projeto, apenas uma equipe com significativo conhecimento da aplicação pode realizar uma avaliação qualificada.

Se as questões expostas no parágrafo anterior funcionam como um elemento de contenção no desenvolvimento e dificuldade na expansão numérica destes sistemas, reconhece-se neste assunto um campo bastante propício à pesquisa aplicada e principalmente aquela que integra os pesquisadores à área industrial. É evidente que ainda existe uma distância razoável entre o que é acadêmico e o que é economicamente viável ou o que o "consumidor" final conhece e sabe que pode exigir. A informação está se difundindo aos poucos. Com este objetivo, o presente trabalho abordou aspectos técnicos relacionados a preconceitos dos projetistas - nomeados como "falsas idéias" - que espera-se ajudar a abolir, além de toda uma justificativa na área a fim de que a pesquisa em sistemas tempo real seja incentivada e aumente rumo à próxima geração. Onde pretende-se obter sistemas mais eficientes, mais flexíveis, mais confiáveis e soluções mais bem integradas com o ambiente onde atuam. Muitos estudos já foram desenvolvidos nesse sentido, no entanto ainda há muito para ser feito...

No que diz respeito às técnicas de tolerância a falhas mais utilizadas no desenvolvimento de software, foi possível constatar que a busca por soluções aplicáveis a casos práticos e tempo real, fez com que as técnicas de recuperação por retorno, em

especial, fossem consideradas com cautela[8]. Desta forma, foi ressaltada a importância das técnicas de mascaramento baseadas em redundância para aplicações em sistemas tempo real. Estudos e projetos têm comprovado esta afirmação encontrada na bibliografia básica consultada. Por outro lado, uma técnica que merece destaque, pela facilidade de utilização em sistemas tempo real, é a que faz uso de diversidade de dados, a re-expressão de dados.

A realização do presente trabalho teve como objetivo principal contribuir na investigação de melhores condições para o desenvolvimento de software para sistemas tempo real tolerantes a falhas, visando com isso atingir uma maior qualidade a nível de análise, projeto e implementação do sistema. Nesse sentido, esta pesquisa buscou organizar as características de tolerância a falhas presentes no software de apoio - protocolos de comunicação, sistemas operacionais e linguagens de programação - que podem ser aplicados a situações de tempo real, uma vez que as características da aplicação não autorizam o uso indiscriminado dos princípios de tolerância a falhas. Estas características foram analisadas individualmente dentro do contexto de cada uma das camadas do software de apoio citadas.

Ao início da dissertação, há um ano, o ponto de partida da autora era o conhecimento geral dos fundamentos de tolerância a falhas, das características dos sistemas tempo real e das restrições existentes na aplicação de um ao outro. No decorrer deste tempo foi necessário aprofundar o conhecimento dos mecanismos empregados para implementação em software de tolerância a falhas. Então, foi definido que seriam usados três níveis de implementação, ou de tipos de software de suporte, que seriam os protocolos de comunicação (interna e externa ao sistema), o sistema operacional e as linguagens de programação. A partir da pesquisa bibliográfica identificou-se as ferramentas disponíveis nestes diferentes níveis, sempre mantendo como condição algum tipo de referência ou vínculo à questão "tempo real". É evidente que, em função da quantidade de material, do tempo disponível e da natureza do trabalho proposto, optou-se por manter a diversificação e pecar pelo nível mais superficial da análise. A diversificação referida também foi limitada pelas condições de contorno do trabalho e disponibilidade de referências.

Assim, mesmo alguns dos pontos propostos foram prejudicados e não foi possível investigá-los até a exaustão. Então, por exemplo a respeito de protocolos de comunicação, foram abordados apenas alguns de conhecida aplicação em sistemas tempo real, não sendo pesquisada a utilização de outros protocolos nestas situações. Dessa forma, não foi abordado nada a respeito do conhecido protocolo TCP/IP, para saber se ele prevê algum recurso de tempo real ou de tolerância a falhas e que tipo de primitivas seriam estas. Fica aqui mais um ponto a ser descoberto por outras pessoas que trilharem o caminho deste trabalho. Por outro lado, sobre os mecanismos de comunicação do sistema operacional, faltou uma investigação, por exemplo, a respeito da utilização dos protocolos de votação em sistemas tempo real, tal como foi feita com os protocolos de compromisso ou aceitação (*commit*).

Quanto aos sistemas operacionais tempo real, além de não ter sido possível abordar todos os sistemas existentes e disponíveis no mercado, mas apenas uma pequena

---

[8] As técnicas de recuperação por retorno somente serão aplicadas em situações onde, em caso de uma falha, o tempo de resposta possa ser relaxado. Dessa forma, pode-se considerá-las apenas em aplicações tempo real brandas.

parcela deles, acredita-se que a maior deficiência deste trabalho talvez tenha sido a falta de detalhamento com relação ao sistema operacional CMX-RTX. Entretanto, as referências encontradas através da rede são bastante promissoras; o que indica que parece ser interessante uma investigação mais aprofundada sobre este sistema.

Em tratando-se de linguagens de programação, cabe salientar que este trabalho limitou-se a abordar as linguagens consideradas de alto nível. Porém, quando se trabalha com CLPs (Controladores Lógicos Programáveis), principalmente no controle de processos industriais, conforme apresentado em monografia anterior [DEN 96], é utilizado outro tipo de linguagens que envolvem, por exemplo: linguagem por lista de instrução (IL - *Instruction List*); linguagem de texto estruturado (ST - *Structured Text*); linguagem de diagrama *Ladder* (LD - *Ladder Diagram*); linguagem de diagrama de bloco de função (FBD - *Function Block Diagram*) - todas descritas pela Norma Internacional para linguagens de programação de CLPs [IEC 93] e adotadas como padrões pela ABNT (Associação Brasileira de Normas Técnicas). Estas linguagens não foram estudadas durante o desenvolvimento deste trabalho, mas sabe-se que sua utilização é bastante difundida para este tipo de aplicação. Por isso, fica como sugestão de trabalho que seja desenvolvida uma pesquisa mais aprofundada sobre seu uso, seu funcionamento e suas características.

Além destes, embora não tenham sido explorados por este trabalho, alguns pontos mantêm-se relacionados como assuntos de interesse; entre os quais merece destaque a utilização prática dos protocolos de comunicação, dos sistemas operacionais e das linguagens de programação, visto que o estudo realizado foi totalmente teórico. Para realizar a análise das ferramentas e utilitários existentes, que resultou nas taxonomias apresentadas, foram tomados como base apenas artigos e manuais que descrevem os mesmos. Dessa forma, não há como garantir que os mesmos funcionem conforme especificado. Entretanto, muitos destes não se encontravam disponíveis no Instituto de Informática e não se dispunha de tempo suficiente para aprender e testar cada um deles. Desta forma, não foi possível qualificar completamente cada uma das ferramentas de acordo com as características propostas.

Da forma final, a taxonomia resultante pode ser vista como três módulos independentes referentes às áreas propostas, ou seja, obteve-se três taxonomias distintas, cada uma referindo-se a uma camada do sistema estudada. As tabelas resultantes destas taxonomias são novamente apresentadas nas Tabelas 8.1, 8.2 e 8.3, cada uma referindo-se respectivamente aos protocolos de comunicação, aos sistemas operacionais e às linguagens de programação tempo real.

Conforme salientado anteriormente, visando facilitar a comparação, os aspectos apresentados de forma qualitativa foram traduzidos em números. Dessa forma, as análises, apresentada nas Tabela 8.1, 8.2 e 8.3, atribui graus a cada uma das ferramentas (protocolo de comunicação, sistema operacional ou linguagem de programação) estudadas com a seguinte correspondência:

- Grau 2: significa que a ferramenta considerada apresenta esta característica sempre.
- Grau 1: significa que esta característica está parcialmente presente na ferramenta analisada (com restrições).
- Grau 0: significa que esta característica não está presente na ferramenta em questão.

Dessa forma, conforme apresentado na seção 4.5, obteve-se uma classificação para os protocolos de comunicação tempo real estudados, que originou a taxonomia apresentada na tabela 8.1.

TABELA 8.1 - TAXONOMIA PARA PROTOCOLOS TEMPO REAL

Características	ARINC 629	CAN	FIP	LON	Profibus	TTP
Detecção de erros	<2>	<2>	<2>	<1>	<1>	<2>
Entrega	<2>	<1>	<2>	<1>	<2>	<2>
Flexibilidade	<1>	<2>	<0>	<2>	<1>	<2>
Previsibilidade	<2>	<0>	<1>	<0>	<2>	<2>
Sincronismo	<2>	<0>	<2>	<0>	<2>	<2>
Tolerância a Falhas	<2>	<2>	<2>	<1>	<1>	<2>

Conforme apresentado na seção 5.4, obteve-se uma classificação para os sistemas tempo real estudados, que originou a taxonomia apresentada na tabela 8.2.

TABELA 8.2 - TAXONOMIA PARA SOTRS

Características	DEDOS	HARTOS	MINIX	MTOS-UX	Portos-TF	QNX	Thoth
Contexto	<0>	<0>	<2>	<1>	<1>	<2>	<2>
Portabilidade	<2>	<1>	<2>	<2>	<2>	<2>	<2>
Previsibilidade	<1>	<1>	<0>	<1>	<1>	<1>	<1>
Reconfigurabilidade	---	<1>	---	---	<2>	<2>	<0>
Redundância	<1>	<2>	<1>	---	<2>	<2>	<0>
Tolerância a Falhas	<1>	<2>	<1>	<1>	<2>	<2>	<0>

Conforme apresentado na seção 6.4, obteve-se uma classificação para as linguagens de programação tempo real estudadas, que originou a taxonomia apresentada na tabela 8.3.

TABELA 8.3 - TAXONOMIA PARA LINGUAGENS TEMPO REAL

Características	Ada	Chill	DEAL	JAVA-RTR	Lustre	RT-CDL	RTT
Flexibilidade	<2>	<1>	<1>	<2>	<1>	<1>	<1>
Gerenciamento de Tempo	<2>	<0>	<2>	<2>	<2>	<2>	<2>
Priorização	<2>	<1>	---	---	<0>	<2>	---
Reutilização	<2>	<2>	<2>	<2>	<0>	<2>	<2>
Tolerância a Falhas	<1>	<1>	<1>	<2>	<1>	<1>	<1>
Tratamento de Exceções	<2>	<2>	<1>	<2>	<0>	<2>	<1>

Como pode-se observar, o nível de elementos obtidos permite principalmente descartar ferramentas, quando estas não apresentam características imprescindíveis à aplicação, ou apresentam dificuldades à incorporação de certos mecanismos. A indicação prévia de características desejáveis ou exigidas pode sugerir ferramentas candidatas ao uso em determinadas aplicações. Descartar possibilidades e reforçar outras pode ser uma forma de objetivação na condução de um projeto, facilitando a convergência de uma solução - quando a gama de opções fica mais restrita, torna-se mais fácil e viável aprofundar-se nas informações.

A validação dos estudos realizados e das taxonomias resultantes foi proposta através dos estudos de casos. Desejava-se abordar um caso real, mas a maior dificuldade está relacionada à questão anteriormente abordada, que é a da especificação exata da aplicação, para a qual seria necessária estrita relação com os projetistas a nível de uma implementação prática. Assim, a idéia foi redirecionada para o aproveitamento de especificações definidas previamente e relatadas na bibliografia [DAV 93, HAC 93 e HEN 93]. Através deles foi possível constatar que as taxonomias e os estudos desenvolvidos podem ajudar os projetistas a tomarem decisões importantes durante o processo de desenvolvimento de software para sistemas tempo real, no que se refere à escolha do software de apoio - protocolo de comunicação, sistema operacional e linguagem de programação - a ser utilizado.

Fica ainda como sugestão, para o desenvolvimento de trabalhos futuros, que este venha a ser incrementado pelo aprofundamento dos temas e técnicas abordadas ou das ferramentas apresentadas; e até mesmo implementado, como um *framework* com suporte de auxílio a tomada de decisões, preferencialmente inteligente. Esta seria certamente uma solução mais fácil e confortável para auxiliar na definição e projeto de software para sistemas tempo real tolerantes a falhas.

Bibliografia

A lista de publicações relacionada a seguir, além de conter as referências bibliográficas citadas no decorrer deste trabalho, apresenta outras que incluem idéias estudadas durante a realização deste, servindo de embasamento teórico das informações aqui contidas.

[ABB 90] ABBOUT, Russel J. Resourceful Systems for Fault Tolerance, Reliability and Safety. **ACM Computing Surveys**, New York, v. 22, n.1, p. 35-68, Mar. 1990.

[ADA 95] ADARTS - **Ada-based Design Approach for Real-Time Systems.** Disponível por WWW em : http://www.software.org/pub/JASTdoc/ada_jast.htm. (dez. 1996).

[AMA 96] AMARO, C. et al. Objective Function Synthesis for Complex Real-Time Systems. In: IFAC/IFIP WORKSHOP ON REAL-TIME PROGRAMMING, 21., 1996, Gramado. **Anais...** Gramado: IFAC/IFIP, 1996. p. 25-30.

[AN 86] AN, Jung Min; CHU, Wesley W. A Resilient Commit Protocol for Real-Time Systems. In: CHU, Wesley W. (Ed.). **Distributed Data Base Systems.** Dedham: Artech House, 1986. v. 2, p. 340- 344.

[AND 81] ANDERSON, T.; LEE, P. A. **Fault Tolerance - Principles and Practice.** Englewood Cliffs: Prentice-Hall, 1981.

[AND 81a] ANDERSON, T.; KNIGHT, J. C. **Practical Software Fault Tolerance for Real-Time Systems**. Hampton, Virginia: Institute for Computer Applications in Science and Engineering. NASA Langley Research Center, 1981. 40p. (Report, n. 81-10).

[AND 83] ANDERSON, Thomas; KNIGHT, John C. A Framework for Software Fault Tolerance in Real-Time Systems. **IEEE Transactions on Software Engineering**, Hoes Lane, v. SE-9, n.3, p. 355-364, May 1983.

[ARI 97] ARINC 629 - **Aeronautical Radio Inc. Protocol.** Disponível por WWW em: http://www.devsoft.com/ads/spec/spec.me-shs063.html. (jan. 1997).

[AUS 96] AUSSAGUES, J. et al. OASIS: a New Way to Design Safety Critical Applications. In: IFAC/IFIP WORKSHOP ON REAL-TIME PROGRAMMING, 21., 1996, Gramado. **Anais...** Gramado: IFAC/IFIP, 1996. p.11- 16.

[AVI 85] AVIZIENIS, Algirdas. The N-Version Approach to Fault-Tolerant Software. **IEEE Transactions on Software Engineering**, Hoes Lane, v. SE-11, n. 12, p. 1491-1501, Dec.1985.

[BEL 92] BELMONTE FILHO, Valdir Rossi. **Gerência de Processos em Sistemas Distribuídos Tolerantes a Falhas.** Porto Alegre: CPGCC da UFRGS, 1992. 186p. Dissertação de mestrado.

[BER 94] BERTOSSI, Alan A. et al. **Increasing Processor Utilization in Hard Real-Time Systems with Checkpoints.** [S.l.:s.n.], 1994.

[BUR 90] BURNS, A.; WELLINGS, J. The Notion of Priority in Real-Time Programming Languages. **Computer Languages**, Washington, v. 15, n. 3, p. 153-162, 1990.

[BUR 96] BURNS, A.; WELLINGS, J. Dual Priority Scheduling in Ada 95 and Real-Time Posix. In: IFAC/IFIP WORKSHOP ON REAL-TIME PROGRAMMING, 21., 1996, Gramado. **Anais...** Gramado: IFAC/IFIP, 1996. p. 45-50.

[CAR 68] CARTER, W. C.; SCHNEIDER, P. R. Design of Dynamically Checked Computers. In: IFIP CONGRESS, 1968, Amsterdam. **Proceedings...** Amsterdam: IFIP, 1968. p.878-883.

[CHE 79] CHERITON, David R. et al. Thoth, a Portable Real-Time Operating System. **Communications of the ACM**, New York, v.22, n.2, p. 105-115, Feb. 1979.

[CLE 93] CLEMATIS, Andrea; GIANUZZI, Vittoria. A Hierarchical Structure for Fault-Tolerant Reactive Programs. In: SYMPOSIUM ON APPLIED COMPUTING, 1993, Indiana. **Anais...** Indiana: SAC, 1993. p. 208- 214.

[CMX 96] CMX - **Real-Time Multi-Tasking Operating Systems for Microprocessors and Microcomputers.** CMX-RTX, 1996. Disponível por WWW em http://www.cmx.com/rtos.htm. (nov. 1996).

[CRI 91] CRISTIAN, Flaviu. Understanding Fault-Tolerant Distributed Systems. **Communications of the ACM**, New York, v.34, n.2, p.57-78, Feb. 1991.

[DAM 86] DAMM, A. The Efectiveness of Software Error-Detection Mechanisms in Real-Time Operating Systems. In: INTERNATIONAL FAULT-TOLERANT COMPUTING SYMPOSIUM, 16., 1986, Vienna. **Digest of Papers...** Vienna: IEEE, 1986. p. 171-176.

[DAV 93] DAVID, Philippe; GUIDAL, Claude. Development of a Fault-Tolerant Computer System for the HERMES Space Shuttle. In: INTERNATIONAL FAULT-TOLERANT COMPUTING SYMPOSIUM, 23., 1993, Toulouse. **Digest of Papers...** Toulouse: IEEE, 1993. p. 641- 646.

[DEN 96] DENARDIN, Fernanda Kruel. **Tolerância a Falhas em Sistemas Tempo Real Aplicados ao Controle de Processos Industriais.** Porto Alegre: CPGCC da UFRGS, 1996. 67p. (T.I. 524).

[DOD 92] DODD, Paul S.; RAVISHANKAR, Chinya V. Monitoring and Debugging Distributed Real-Time Programs. **Software - Practice and Experience**, London, v.22, n.10, p. 863-877, Oct. 1992.

[ESP 96] ESPINOSA, A. et al. Implementing Safe Real-Time Systems by Means of Automatic Prototyping. In: IFAC/IFIP WORKSHOP ON REAL-TIME PROGRAMMING, 21., 1996, Gramado. **Anais...** Gramado: IFAC/IFIP, 1996. p.95-99.

[EVE 95] EVERETT, William; HONIDEN, Shinichi. Reliability and Safety of Real-Time Systems. **IEEE Software**, Hoes Lane, v.12, n.3, p. 13-16, May 1995.

[FIE 96] **FielBus Industrial Protocols.** Disponível por WWW em: http://litsun.epfl.ch/~fieldbus/fieldbus.html. (nov. 1996).

[FRA 87] FRANCISCO, Maria de Fátima Mattielo. **Um Sistema de Supervisão Tolerante a Falhas.** São José dos Campos: INPE, 1987. 157p. Dissertação de mestrado.

[FRI 89] FRICKS, Ricardo Messias. **Projeto de um Sistema Operacional Tempo Real Baseado no MINIX.** Porto Alegre: CPGCC da UFRGS, 1989. 262p. Dissertação de mestrado.

[FUR 96] FURTADO, O.; FARINES, J.M. JAVA-RTR - Uma Linguagem Reflexiva para Programação de Aplicações Tempo Real. In: SIMPÓSIO BRASILEIRO DE LINGUAGENS DE PROGRAMAÇÃO, 1., 1996, Belo Horizonte. **Anais...** Belo Horizonte: SBC, 1996. p. 357-370.

[GUS 94] GUSTAFSSON, Jan. Calculation of Execution Times in RealTimeTalk - an Object-Oriented Language for Real-Time. In: WORKSHOP ON OBJECT-ORIENTED REAL-TIME DEPENDABLE SYSTEMS, 1., 1994, Dana Point. **Anais...** Dana Point: IEEE - WORDS, 1994. p.153- 162.

[HAC 93] HACHIGA, Akira et al. The Design Concepts and Operational Results of Fault-Tolerant Computer Systems for the Shinkansen Train Control. In: INTERNATIONAL FAULT-TOLERANT COMPUTING SYMPOSIUM, 23., 1993, Toulouse. **Digest of Papers...** Toulouse: IEEE, 1993. p. 78- 87.

[HEC 76] HECHT, H. Fault-Tolerant Software for Real-Time Applications. **ACM Computing Surveys**, New York, v.8, n. 4, p. 391-407, Dec. 1976.

[HEN 93] HENNEBERT, Claude; GUIHO, Gérard. SACEM: A Fault-Tolerant System for Train Speed Control. In: INTERNATIONAL FAULT-TOLERANT COMPUTING SYMPOSIUM, 23., 1993, Toulouse. **Digest of Papers...** Toulouse: IEEE, 1993. p. 624- 628.

[IEC 93] IEC - INTERNATIONAL ELECTROTECHNICAL COMMISSION. **International Standard Programmable Controllers.** Part 3: Programming Languages. [S. l.]: CEI/IEC, 1993.

[JAL 94] JALOTE, Pankaj. **Fault Tolerance in Distributed Systems.** Englewood Cliffs: Prentice Hall, 1994. 432p.

[KEL 91] KELLY, John P. J. et al. Implementing Design Diversity to Achieve Fault Tolerance. **IEEE Software**, Hoes Lane, v.8, n.4, p. 61- 71, July 1991.

[KIM 94] KIM, K. H. et al. Distinguishing Features and Potential Roles of the RTO.k Object Model. In: WORKSHOP ON OBJECT-ORIENTED REAL-TIME DEPENDABLE SYSTEMS, 1., 1994, Dana Point. **Anais...** Dana Point: IEEE - WORDS, 1994.

[KIM 95] KIM, K. H. Challenges in Integration of Major Design Techniques for Real-Time Fault-Tolerant Computer Systems. In: SIMPÓSIO DE COMPUTADORES TOLERANTES A FALHAS, 6., 1993, Canela. **Invited Talk...** Canela: SBC, 1995.

[KIM 96] KIM, K; BACELLAR, L.; SUBBARAMAN, C. Support for RTO.k Object Structured Programming in C++. In: IFAC/IFIP WORKSHOP ON REAL-TIME PROGRAMMING, 21., 1996, Gramado. **Anais...** Gramado: IFAC/IFIP, 1996. p. 17-22.

[KIR 96] KIRNER, T. G. Quality Requirements for Real-Time Safety-Critical Systems. In: IFAC/IFIP WORKSHOP ON REAL-TIME PROGRAMMING, 21., 1996, Gramado. **Anais...** Gramado: IFAC/IFIP, 1996. p. 101- 106.

[KOO 87] KOO, R.; TOUEG, S. Checkpointing and Rollback-Recovery for Distributed Systems. **IEEE Transactions on Software Engineering**, Hoes Lane, v. SE-13, n.1, p.23-31, Jan. 1987.

[KOP 93] KOPETZ, Hermann; VERÍSSIMO, Paulo. Real-Time and Dependability Concepts. In: MULLENDER, Sape J. (Ed.); VERÍSSIMO, Paulo (co-author). **Distributed Systems.** 2. ed. [S. l.]: Addison-Wesley ACM - Press Series, 1993. p. 411-445.

[KOP 94] KOPETZ, Hermann; GRÜNSTEIDL, Günter. TTP - A Protocol for Fault-Tolerant Real-Time Systems. **Computer**, New York, v.27, n.1, p. 14-23, Jan. 1994.

[KOP 94a] KOPETZ, Hermann. An Analisys of the Communication Service in Responsive Systems. In: WORKSHOP ON OBJECT-ORIENTED REAL-TIME DEPENDABLE SYSTEMS, 1., 1994, Dana Point. **Anais...** Dana Point: IEEE - WORDS, 1994. p.56-62.

[KRI 91] KRISHNA, C. M.; LEE, Y. H. Introduction Real-Time Systems. **Computer**, New York, v.24, n.5, p. 10-11, May 1991.

[KUR 94] KURKI, Reino Suonio. Real-Time: Further Misconceptions (or Half-Truths). **Computer**, New York, v.27, n.6, p. 71-76, June 1994.

[LAL 91] LALA, J. et al. A Design Approach for Ultrareliable Real-Time Systems. **Computer**, New York, v.24, n.5, p. 12-24, May 1991.

[LAP 85] LAPRIE, J. C. Dependable Computing and Faut-Tolerance: Concepts and Terminology. In: INTERNATIONAL FAULT-TOLERANT COMPUTING SYMPOSIUM, 15., 1985, Ann Arbor. **Digest of Papers...** Ann Arbor: IEEE, 1985. p. 2-11.

[LAP 90] LAPRIE, Jean-Claude et al. Definition and Analysis of Hardware - and Software - Fault-Tolerant Architectures. **Computer**, New York, v23, n.7, p. 39-51, July 1990.

[LIS 93] LISBÔA, Maria Lúcia Blanck; AZEREDO, Paulo Alberto de. **Paradigmas de Programação**: O Enfoque de Tolerância a Falhas. Brasília: [s. n.], 1993. Curso ministrado no Congresso Nacional de Informática e Telecomunicações, 26., 1993, Brasília.

[LIS 95] LISBÔA, Maria Lúcia Blanck. **MOTF: Meta Objetos para Tolerância a Falhas.** Porto Alegre: CPGCC da UFRGS, 1995. Tese de doutorado.

[LIU 90] LIU, Leo Y.; SHYAMASUNDAR, R.K. Exception Handling in RT-CDL. **Computer Languages**, Washington, v. 15, n. 3, p. 177-192, 1990.

[LOB 89] LOBIANCO JUNIOR, W. Portos-TF: Um Sistema Operacional Portátil de Tempo Real com Primitivas de Tolerância a Falhas. In: SIMPÓSIO DE COMPUTADORES TOLERANTES A FALHAS, 3., 1989, Rio de Janeiro. **Anais...** Rio de Janeiro: SBC, 1989. p. 90-109.

[LON 96] LON - **Local Area Network.** Disponível por WWW em: http://danville.res.utc.com/Mechatronicas/ads/barg/abstract/survey/survey.htm. (dez. 1996).

[MAR 91] MARTIN, Bruce E. et al. An Object-Based Taxonomy for Distributed Computing Systems. **Computer**, New York, v.24, n. 8, p. 17-27, Aug. 1991.

[MOR 95] MORON, Celio Estevan. A Real-Time Fault-Tolerant Scheduler. In: SIMPÓSIO DE COMPUTADORES TOLERANTES A FALHAS, 6., 1995, Canela. **Anais...** Canela: SBC, 1995. p. 313-331.

[MOS 90] MOSER, Louise E.; MELLIAR-SMITH, P. M. Formal Verification of Safety-Critical Systems. **Software - Practice and Experience**, London, v.20, n.8, p. 799-821, Aug. 1990.

[MUP 91] MUPPALA, J. et al. Real-Time Systems Performance in the Presence of Failures. **Computer**, New York, v.24, n.5, p. 37-47, May 1991.

[MUR 87] MURATA, Masao. **Aspectos Computacionais de Sistemas de Tempo Real.** São José dos Campos: INPE, 1987. 161p. Dissertação de mestrado.

[MUS 95] MUSLINER, David J. et al. The Challenges of Real-Time AI. **Computer**, New York, v. 28, n.1, p. 58-66, Jan. 1995.

[NAT 92] NATARAJAN, S.; ZHAO, W. Issues in Building Dynamic Real-Time Systems. **IEEE Software**, Los Alamitos, v. 9, n. 5, p. 16-21, Sept. 1992.

[NEL 87] NELSON, Victor P.; CARROLL, Bill D. **Tutorial:** Fault-Tolerant Computing. New York: IEEE,1987.

[NEL 90] NELSON, V. Fault-Tolerant Computing: Fundamental Concepts. **Computer**, New York, v. 23, n.7, p. 19-25, July 1990.

[PAU 89] PAULA JUNIOR, Alderico R. de et al. Fundamentos da Computação Tolerante a Falhas. In: SIMPÓSIO DE COMPUTADORES TOLERANTES A FALHAS, 3., 1989, Rio de janeiro. **Tutoriais...** Rio de Janeiro: SBC, 1989.

[PIL 88] PILAUD, D.; HALBWACHS, N. From a Synchronous Declarative Language toa a Temporal Logic Dealing with Multiform Time. In: FORMAL TECHNIQUES IN REAL-TIME AND FAULT-TOLERANT SYSTEMS, 1988, Warwick. **Proceedings...** Berlin: Springer-Verlag, 1988. p. 99-110.

[PRA 96] PRADHAN, Dhiraj K. **Fault-Tolerant Computer System Design.** Upper Saddle River: Prentice Hall, 1996.

[PRO 97] **Profibus Protocol.** Disponível por WWW em: http://www.usdata.com/spcc1002.htm. (jan. 1997).

[PUR 91] PURTILO, James M.; JALOTE, Pankaj. An Environment for Developing Fault-Tolerant Software. **IEEE Transactions on Software Engineering**, Hoes Lane, v. 17, n.2, p. 153-159, Feb. 1991.

[QNX 96] QNX SOFTWARE SYSTEMS LTDA. **QNX Operating System - System Architecture:** User's Guide. Ontario: QNX, 1996. 142p.

[RAN 75] RANDELL, R. System Structure for Software Fault Tolerance. **IEEE Transactions on Software Engineering**, Hoes Lane, v. SE-1, n. 1, p. 220-232, June 1975.

[RIP 93] RIPPS, D. L. **Guia de Implementação para Programação em Tempo Real.** Rio de Janeiro: Campus, 1993. 314p.

[ROS 94] ROOSMALEN, Onno S. van. DEAL: an Object-Oriented Language for Distributed Real-Time Systems. In: WORKSHOP ON OBJECT-ORIENTED REAL-TIME DEPENDABLE SYSTEMS, 1., 1994, Dana Point. **Anais...** Dana Point: IEEE - WORDS, 1994. p.146-152.

[RTX 96] RTX - **Real-Time Operating System.** CMX Company. Disponível por WWW em http://www.mwmedia.com/tpvs/cmxco/end/cmxrtos.htm. (dez. 1996).

[RTX 97] RTX - **Real-Time Operating System.** Disponível por WWW em: http://www.numedia.com/tpvs/cmx-co/xq/multitsk.htm. (jan. 1997).

[SAN 91] SANTOS, Jorge. Redes Locales en Tiempo-Real. In: ESCOLA BRASILEIRO-ARGENTINA DE INFORMÁTICA, 5., 1991, Rio de Janeiro. Rio de Janeiro: EBAI, 1991.

[SCH 95] SCHLICHTING, Richard D.; THOMAS, Vicraj T. Programming Language Support for Writing Fault-Tolerant Distributed Software. **IEEE Transactions on Computers**, New York, v.44, n. 2, p. 203-211, Feb. 1995.

[SCH 97] SCHOFIELD, Mike. **CAN - Controller Area Network.** Disponível por WWW em http://www.omegas.co.uk/CAN/index.html. (jan. 1997).

[SEL 96] SELIC, Bran. Modeling Real-Time - Distributed Software Systems. In: IFAC/IFIP WORKSHOP ON REAL-TIME PROGRAMMING, 21., 1996, Gramado. **Invited Talk...** Gramado: IFAC/IFIP, 1996.

[SHI 92] SHIN, K. et al. A Distributed Real-Time Operating System. **IEEE Software**, Los Alamitos, v. 9, n. 5, p. 58-68, Sept. 1992.

[SIE 82] SIEWIOREK, Daniel P.; SWARZ, Robert S. **The Theory and Practice of Reliable System Design.** Bedford: Digital, 1982.

[SIL 88] SILVA, José Carlos G. da; ASSIS, Fidelis Sigmaringa G. de. **Linguagens de Programação:** Conceitos e Avaliação. São Paulo: McGraw-Hill, 1988.

[SIN 94] SINGHAL, M.; SHIVARATRI, N. G. **Advanced Concepts in Operating Systems:** Distributed, Database and Multiprocessor Operating Systems. New York: McGraw-Hill, 1994.

[STA 88] STANKOVIC, John A. Misconceptions About Real-Time Computing - A Serious Problem for Next Generation Systems. **Computer**, New York, v.21, n.10, p. 10-19, Oct. 1988.

[STA 88a] STANKOVIC, John A.; RAMAMRITHAM, Krithi. Introduction. In: STANKOVIC, John A.; RAMAMRITHAM, Krithi. **Hard Real-Time Systems.** New York: IEEE, 1988. p. 1-11.

[STA 88b] STANKOVIC, John A. Real-Time Computing Systems: The Next Generation. In: STANKOVIC, John A.; RAMAMRITHAM, Krithi. **Hard Real-Time Systems.** New York: IEEE, 1988. p. 14-37.

[TOZ 83] TOZZI, Clésio L. et al. Introdução aos Sistemas Tempo Real para o Controle de Processos. In: CONGRESSO NACIONAL DE AUTOMAÇÃO INDUSTRIAL, 1., 1983, São Paulo. **Tutoriais...** São Paulo: [s. n.], 1983. p. 1-36

[VER 93] VERÍSSIMO, Paulo. Real-Time Communication. In: MULLENDER, Sape J. (Ed.); VERÍSSIMO, Paulo (co-author). **Distributed Systems.** 2. ed. [S. l.]: Addison-Wesley ACM - Press Series, 1993. p. 447-490.

[VER 93a] VERÍSSIMO, Paulo; KOPETZ, Hermann. Design of Distributed Real-Time Systems. In: MULLENDER, Sape J. (Ed.); VERÍSSIMO, Paulo (co-author). **Distributed Systems.** 2. ed. [S. l.]: Addison-Wesley ACM - Press Series, 1993. p. 511-530.

[VON 56] VON NEUMANN, J. Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components. In: SHANNON; McCARTHY (Eds.) **Automata Studies.** Princeton: Univ. Press, 1956. p. 43-98.

[ZAM 96] ZAMORANO, J. et al. Building Safety-Critical Real-Time Systems with Reusable Cyclic Executives. In: IFAC/IFIP WORKSHOP ON REAL-TIME PROGRAMMING, 21., 1996, Gramado. **Anais...** Gramado: IFAC/IFIP, 1996. p. 63-68.

"Quanto mais aumenta nosso conhecimento, mais evidente fica nossa ignorância."
(John Kennedy, 1917-1963)

Informática  UFRGS

CURSO DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Software Tolerante a Falhas para Aplicações Tempo-Real

por

Fernanda Kruel Denardin

Dissertação apresentada aos Senhores:

Profa. Dra. Taisy Silva Weber

Profa. Dra. Maria Lúcia Blanck Lisbôa

Prof. Dr. Alderico Rodrigues de Paula Júnior (UNIVAP)

Vista e permitida a impressão.
Porto Alegre, 14 / 05 / 97.

Profa. Dra. Ingrid Eleonora Schreiber Jansch Pôrto,
Orientador.