LINGUAGEM ADA: ESTADO DA ARTE, TECNOLOGIAS ASSOCIADAS E MECANISMOS DE DEPENDABILIDADE

Trabalho de Conclusão de Curso

Engenharia da Computação

Aluno: Marcos Aurélio Silva de Souza
Orientador: Prof. Dr. Sergio Murilo Maciel Fernandes

Universidade de Pernambuco
Escola Politécnica de Pernambuco
Graduação em Engenharia de Computação

MARCOS AURÉLIO SILVA DE SOUZA

LINGUAGEM ADA: ESTADO DA ARTE, TECNOLOGIAS ASSOCIADAS E MECANISMOS DE DEPENDABILIDADE

Monografia apresentada como requisito parcial para obtenção do diploma de Bacharel em Engenharia de Computação pela Escola Politécnica de Pernambuco – Universidade de Pernambuco.

Recife, novembro de 2011

Escola Politécnica de Pernambuco
Graduação em Engenharia de Computação
Projeto de Final de Curso

POLI
ESCOLA POLITÉCNICA
DE PERNAMBUCO

MONOGRAFIA DE FINAL DE CURSO

Avaliação Final (para o presidente da banca)*

No dia 19 de Dezembro de 2011, às 11:00 horas, reuniu-se para deliberar a defesa da monografia de conclusão de curso do discente MARCOS AURELIO SILVA DE SOUZA, orientado pelo professor Sérgio Murilo Maciel Fernandes, sob título Linguagem ADA: Estado da arte, tecnologias associadas e mecanismos de dependabilidade, a banca composta pelos professores:

Joabe Bezerra de Jesus Júnior

Sérgio Murilo Maciel Fernandes

Após a apresentação da monografia e discussão entre os membros da Banca, a mesma foi considerada:

☒ Aprovada ☐ Aprovada com Restrições* ☐ Reprovada

e foi-lhe atribuída nota: 8,8 (Oito, oito)

*(Obrigatório o preenchimento do campo abaixo com comentários para o autor)

O discente terá 07 dias para entrega da versão final da monografia a contar da data deste documento.

Joabe Bezerra de Jesus Júnior
JOABE BEZERRA DE JESUS JÚNIOR

Sérgio Murilo Maciel Fernandes
SÉRGIO MURILO MACIEL FERNANDES

* Este documento deverá ser encadernado juntamente com a monografia em versão final.

Esse trabalho dedico a minha querida mamãe, Maria do Carmo, in memoria.

Agradecimentos

Primeiramente ao meu tio-avô, José Xavier de Morais, e minha tia-avó, Maria José Xavier de Morais, por terem sempre me incentivado e buscado dar condições, que me proporcionaram toda a base educacional para que eu chegasse até aqui e sem os quais eu não seria quem sou. Também a minha querida Priscila Tamar por ter me apoiado com seu amor, trazendo harmonia e tranquilidade, me ensinando esses fundamentos importantes que levarão a minha realização profissional.

Ao professor Sergio Murilo Maciel Fernandes, por ter me dado uma excelente orientação durante os anos em que fui seu aluno na graduação e pelo estímulo e credibilidade dada durante a realização deste trabalho. Também gostaria de agradecer aos docentes do curso de engenharia da computação pela excelente formação ministrada durante esses anos. Também agradeço a todos os profissionais que durante meus estágios compartilharam conhecimento, valores éticos e senso profissional, são eles meus colegas do LINCS-CETENE, SERPRO, e Divisão de Tecnologia da Informação da Escola Politécnica de Pernambuco, onde pude conviver no dia a dia com um grande homem e profissional, o professor Pedro Alcântara, diretor da POLI.

Agradeço também aos meus colegas de Politécnica pelo excelente ambiente que proporcionaram durante a graduação, tornando essa a melhor fase da minha vida. Em especial a meus amigos-irmãos: Bruno Melo, David Edson Riberio, Rômulo Jales, Antônio Bezerra e Murilo Rebelo Pontes. A todos os outros colegas que compartilharam comigo dificuldades, desafios e alegrias durante esses anos.

Resumo

Esse trabalho apresenta algumas das técnicas de dependabilidade, e a facilidade em se desenvolver essas técnicas em software usando a linguagem de programação ADA. Muitos dos mecanismos da linguagem ADA foram utilizados na experimentação realizada. A hipótese que a técnica Consensus Recovery Block é mais confiável que as técnicas de blocos de recuperação e de programação diversitária foi provada, pelo índice de confiabilidade. Um outro objetivo foi apresentar as tecnologias associadas com ADA, mostrando a flexibilidade e extensibilidade. Com esse trabalho espera-se impulsionar a formação de massa crítica na área de dependabilidade, juntos aos estudantes do curso de engenharia da computação da Universidade de Pernambuco.

.

Abstract

This paper presents some techniques for dependability, and easy availability these techniques to develop software using the programming language ADA. Many of the ADA language mechanisms were used in experimentation carried. The hypothesis that the Consensus Recovery Block technique is more reliable than the techniques of recovery blocks and n-versions programming has been proven by the reliability index. Another objective was to present the technologies associated with ADA, where he saw the great extensibility of the ADA. With this work is expected to boost the formation of critical mass in the area of dependability together the students of the course of computer engineering at the University of Pernambuco.

Sumário

Capítulo 1 Introdução **1**

1.1 Motivação e Caracterização do problema 2

1.2 Hipóteses e Objetivos 4

1.2.1 Objetivos Gerais 5

1.2.2 Objetivos Específicos 5

1.3 Organização do documento 6

Capítulo 2 Dependabilidade e Tolerância a Falhas **7**

2.1 Dependabilidade 7

2.1.1 Confiabilidade, disponibilidade e sistemas dependáveis críticos 10

2.2 Tolerância a falhas 11

2.2.1 Redundância de *Hardware* 14

2.2.2 Redudância de *Software* 16

2.2.3 Recovery Blocks 19

2.2.4 NVP 21

2.2.5 CRB 22

Capítulo 3 Linguagem ADA **24**

3.1 Subprogramas 26

3.1.1 Composição de um progama ADA 26

3.1.2 Distinção de um unit progama 26

3.1.3 Programas units 27

3.2 Operadores e operações 28

3.3 Escopo e Visibilidade 28

3.3.1 Escopo 29

3.3.2 Visibilidade 29

3.4 Tipos em ADA 30

3.4.1 Tipos Elementares 32

3.4.2 Tipos Compostos 33

3.5 Tipo Access (Ponteiros) 34

3.6 Estruturas de controle e repetição 35

3.7 Exceções 35

3.8 Pragmas 35

3.9 Generics Packages 36

3.10 Child packages 36

3.11 Concorrência e *Tasks* 36

3.12 Mais da ALRM 37

3.12.1 Ada.Unchecked_Conversion 37

3.12.2 Ada.Strings.Unbounded 37

3.12.3 Ada.Real_Time 38

3.12.4 Ada.Numerics.Discrete_Random 38

Capítulo 4 ADA com tecnologias associadas 39

4.1 ADA e Sistemas em Tempo Real 39

4.1.1 *Ravenscar* profile 41

4.1.2 SPARK ADA. 42

4.2 ADA e Sistemas Embarcados 42

4.3 ADA com FPGA 43

4.3.1 FPGA 43

4.3.2 Arquitetura do FPGA 44

4.3.3 FPGA Tolerante a falhas 45

4.3.4 ADA como HDL e HLL 46

4.4 ADA com HW/SW Co-design 47

4.5 ADA com Redes de Petri 47

4.6 ADA com UML 48

Capítulo 5 Experimentação e Resultados 49

5.1 Caracterização do experimento 49

5.1.1 Consensus Recovery Block 50

5.1.2 NVP 50

5.1.3 Recovery Block 51

5.2 Dificuldades durante o desenvolvimento 51

5.3 Discussão dos Resultados 51

5.3.1 CRB Procedural 52

5.3.2 CRB paralelo 53

5.3.3 NVP 53

5.3.4 Bloco de recuperação 54

5.4 Conclusões dos resultados 54

Capítulo 6 Conclusão e Trabalhos Futuros 55

6.1 Trabalhos Futuros 55

Bibliografia 57

Apêndice A Tabelas dos resultados da Experimentação I

Apêndice B Codificação do experimento V

Índice de Figuras

Figura 2.1 Modelo dos 3 universos........ 8
Figura 2.2 Recuperação por retrocesso e por avanço........ 13
Figura 2.3 TMR – Tiple Module Redundancy. 15
Figura 2.4 Blocos de recuperação........ 18
Figura 2.5 Estrutura do Bloco de Recuperação proposto por Randell 20
Figura 2.6 Funcionamento da programação de n-versões 22
Figura 2.7 Consensus Recovery Blocks 23
Figura 3.1 Hierarquia de Tipos em ADA 32
Figura 4.1 Arquitetura do FPGA 44

Índice de Tabelas

Tabela 2.1 Resumo dos principais atributos de dependabilidade..........................9

Tabela 3.1 Visibilidade em ADA 27

Tabela 5.1 Resumo dos dados do CRB procedural...52

Tabela 5.2 Resumo dos dados do CRB paralelo...53

Tabela 5.3 Resumo dos dados do NVP..53

Tabela 5.4 Resumo dos dados do Bloco de Recuperação..................................54

Tabela de Símbolos e Siglas

ALRM	Ada Language Reference Manual
CDL	CO-Design Language
CRB	Consensus Recovery Block
CMOS	Complementary metal-oxide-semiconductor
FPGA	Field Programmable Gate Array
HDL	Hardware Description Language
HLL	High-Level Language
HW	Hardware
NVP	N-Version Programming
RTS	Run-time system
SW	Software
SWFT	Software Fault Tolerence
UML	Unified Modeling Language

Capítulo 1
Introdução

A evolução tecnológica e científica em nossa sociedade tem estimulado o desenvolvimento de novos equipamentos, quer seja pela redução de seus custos, quer seja pela agregação de novas funcionalidades. Alguns desses dispositivos exigem além de um custo baixo, atributos não funcionais que implicam em uma maior qualidade. Atributos como performance, confiabilidade e segurança são importantes e dão uma maior qualidade aos equipamentos das indústrias aeronáuticas, aeroespaciais, nucleares, médicas, além de outras onde a não garantia de funcionamento do produto pode resultar em grandes perdas econômicas ou mesmo de vidas humanas. A demanda por qualidade dos produtos justifica-se pelo rápido e permanente aumento das responsabilidades que o ser humano transfere aos produtos manufaturados, em particular, aqueles controlados por computadores e softwares. Diante dessa situação é necessário que softwares sejam desenvolvidos utilizando-se linguagens de programação que tenham como foco o desenvolvimento de sistemas seguros como, por exemplo, a linguagem ADA.

No decorrer desse trabalho, é apresentado um estudo dirigido dos conceitos de dependabilidade da linguagem de programação ADA e algumas tecnologias com que se relacionam. O trabalho mostra a perspectiva de uma melhor qualidade dos produtos de software através da área de dependabilidade (segurança de funcionamento) e espera estimular novos profissionais a conhecerem uma linguagem segura para desenvolvimento de sistemas.

Na Seção 1.1, a caracterização do problema será abordada, bem como as motivações para a realização deste trabalho. Em seguida, na Seção 1.2, são apresentados os objetivos gerais e específicos, e as hipóteses que foram abordadas pela experimentação. Por fim, na Seção 1.3, é descrita a estrutura do trabalho.

1.1 Motivação e Caracterização do problema

Cada vez mais a complexidade dos sistemas computacionais demandam que os requisitos não-funcionais de disponibilidade, confiabilidade e segurança sejam atendidos. Sistemas de operações financeiras e sistemas de controle de tráfego aéreo são alguns exemplos desses sistemas críticos e dependáveis [2]. Uma falha nesse tipo de sistema pode ocasionar perda de vidas humanas, perdas econômicas e ambientais. Então, como evitar falhas nesses sistemas computacionais? A resposta é que falhas são inevitáveis, mas suas consequências devem ser evitadas por meios de mecanismos que tornem as falhas toleráveis, isto é, espera-se que o sistema esteja operante e não entre em colapso, ou em outras palavras que o sistema vá para um estado seguro.

Até então, o conceito de dependabilidade [1] era apenas preocupação exclusiva de projetistas de sistemas críticos, como aviões, controle de tráfego aéreo e controle de plantas industriais em tempo real. Dependabilidade é uma tradução literal do termo inglês *dependability*, que indica a qualidade do serviço fornecido por um dado sistema e a confiança depositada no serviço fornecido [2]. Entre seus principais atributos estão a disponibilidade, que é um requisito não-funcional importante nos sistemas de telecomunicações e a confiabilidade nos sistemas aeronáuticos, aeroespaciais e controle de tráfego aéreo, dentre outros. Como visto pelos exemplos, para alguns sistemas a confiabilidade tem que ser garantida, enquanto para outros sistemas a disponibilidade é algo crítico.

Para que tais sistemas críticos funcionem de acordo com o especificado, técnicas de tolerância a falhas devem ser usadas para fornecer o serviço esperado mesmo na presença de falhas [3]. Para o desenvolvimento de sistemas dependáveis temos alguns desafios e motivações [1]:

1. Como gerenciar a altíssima complexidade dos sistemas atuais de computação construídos com dezenas de chips de milhões de transistores e com software de centenas de milhares de linhas de código?

2. Como evitar, detectar e contornar bugs no projeto de software?
3. Como desenvolver computadores móveis e sistemas embarcados, garantindo confiabilidade e segurança nesses dispositivos, e assegurando simultaneamente baixo consumo de potência, sem recorrer as técnicas usuais de replicação de componentes que aumentam peso e volume?
4. Como explorar paralelismo para aumentar o desempenho sem comprometer a qualidade dos resultados mesmo em caso de falha de um ou mais componentes do sistema?
5. Como aproveitar, para aplicações críticas e para operação em tempo real, o modelo de sistemas distribuídos construídos sobre plataformas não confiáveis de redes, contornando os problemas de perdas de mensagens, particionamento de rede e intrusão de hackers?
6. Como conciliar alta confiabilidade e alta disponibilidade com as crescentes demandas por alto desempenho?

Como linguagem de programação de alto nível, criada para atender ao desenvolvimento de sistemas embarcados e críticos do departamento de defesa dos Estados Unidos (DOD), a linguagem ADA fornece construtores intrínsecos que podem ser usados nos mecanismos de tolerância a falhas, não só facilitando o desenvolvimento de tais sistemas como também associando-se ao estado da arte de outras tecnologias de sistemas [4]. Entre esses mecanismos podemos citar os *restrictions pragmas*" como o *ravenscar profille* [5], "*streams*" [6], SPARK [7] entre outros [8].

A linguagem de programação ADA além de seus mecanismos inerentes pode ser usada também com as seguintes tecnologias para prover dependabilidade aos sistemas:

1. Redes de Petri [9];

2. linguagem de *codesign* de HW/SW (*Hardware/Software)* (Metodologia de desenvolvimento de sistemas embarcados) [10];

3. o uso de ADA como linguagem de descrição de Hardware, HDL (*Hardware Description Language*) [11] e HLL (*High-Level Language*) em FPGA (*Field Programmable Gate Array*) [12];

4. UML (*Unified Modeling Language*) para geração automática de código [13];

5. desenvolvimento de sistemas de tempo real [14] e embarcados [15];

6. para desenvolvimento de sistemas de controle e robótica [16];

7. e uso com técnicas de computação inteligente [17].

1.2 Hipóteses e Objetivos

Para a demonstração de desempenho, de confiabilidade, de estruturas e de mecanismos intrínsecos da linguagem ADA na implementação de mecanismos de tolerância a falhas, foi utilizada uma experimentação das técnicas CRB (*Consensus Recovery Block*) [18], implementados de modo a executar paralelamente e seqüencialmente (procedural); NVP (*N-Version Programming*) [19] e *Recovery Blocks* [20].

Com os dados obtidos da experimentação, podem ser vistos aspectos de confiabilidade das três técnicas. O índice de confiabilidade será dado pela frequência relativa entre o números de sucessos e o total de iterações da execução da técnica. A hipótese de que o CRB produz um software mais confiável será provada, ou não. Para fins estatísticos, será usado a média dos índice de confiabilidade resultantes de 30 amostras.

Nas subseções a seguir são apresentados os objetivos gerais e específicos deste trabalho.

1.2.1 Objetivos Gerais

- Apresentar o uso de ADA para desenvolvimento de sistemas dependáveis com as diversas tecnologias selecionadas;

- Mostrar, por meio da linguagem ADA, o desenvolvimento experimental de programas que simulam as técnicas de tolerância a falhas NVP, *Recovery Blocks* e CRB em *software*.

1.2.2 Objetivos Específicos

- Apresentar através de uma revisão bibliográfica os trabalhos publicados e direcionados a linguagem de programação ADA;

- Apresentar o conceito de sistemas dependáveis e os conceitos inerentes as técnicas de dependabilidade;

- Mostrar o uso de ADA e diferentes técnicas e metódos;

- Mostrar, através de experimentação, resultados que demonstrem o uso eficiente de alguns mecanismos de tolerância a falhas implementados em ADA.

1.3 Organização do documento

O trabalho está organizado em 6 capítulos. No Capítulo 2 serão abordados aspectos relativos a dependabilidade e as técnicas de tolerência a falhas, especificamente o CRB [18], o NVP [19] e o *Recovery Blocks* [20] implementadas na experimentação. Em seguida, no Capítulo 3, são abordados os conceitos e estruturas da linguagem de programação ADA [4]. No Capítulo 4, é apresentado o uso de ADA com algumas tecnologias associadas. O Capítulo 5, descreverá a experimentação realizada, na qual as técnicas de tolerância a falhas foram implementados em ADA e executadas sob algumas condições, além dos resultados da experimentação e algumas conclusões sobre eles. Por fim, no Capítulo 6 as principais conclusões deste trabalho são apresentadas, bem como propostas para trabalhos futuros.

Capítulo 2
Dependabilidade e Tolerância a Falhas

Nesse capítulo é apresentada a base necessária para entendimento dos conceitos da dependabilidade, sua importância no contexto de sistemas críticos e complexos, reforçado através de exemplos de sistemas dependáveis. Também são apresentadas as técnicas de tolerâncias a falhas, com ênfase nas que foram utilizadas na experimentação do trabalho. Na Seção 1.1 será descrito os conceitos da dependabilidade. Em seguida na Sub-seção 1.1.1, será mostrado alguns exemplos de sistemas dependáveis críticos e os atributos de confiabilidade e disponibilidade. Por fim, na Seção 1.2, será abordado as técnicas de tolerâncias a falhas, e as sub-seções 1.2.1, 1.2.2, 1.2.3, 1.2.4 apresentam os conceitos de redudância de *hardware* e de *software* e as técnicas *Recovery Blocks*, NVP e CRB, respectivamente.

2.1 Dependabilidade

Falhas em sistemas computacionais são inevitáveis, mas é desejável e até imprescindível em alguns casos, que as consequências não levem a um colapso do sistema ou em sua indisponibilidade. A área da computação que engloba problemas dessa natureza é conhecida como dependabilidade, sendo esse termo o mais amplamente utilizado nos dias atuais [1]. Sistemas dependáveis, tolerantes a falhas, ou com segurança de funcionamento não podem ser entendidos como tolerante a toda e qualquer falha em qualquer situação, porque pode ser uma algo impraticável, gerando uma expectativa falsa nos usuários. Os conceitos chaves e termos

apresentados a seguir, foram derivados dos trabalhos de Laprie [2] e de Anderson e Lee [22].

Segundo essa terminologia, um defeito (*failure*) é definido como um desvio da especificação. Esses desvios não podem ser tolerados, e devem ser evitados. Define-se que um sistema está em estado errôneo, ou em erro, se o processamento posterior, a partir desse estado, pode levar a um defeito. Finalmente define-se falha ou falta (*fault*) como a causa física ou algorítmica do erro. Na Figura 2.1, o modelo dos 3 universos, uma simplificação proposta por Barry W. Johnson [21], ilustam estes conceitos.

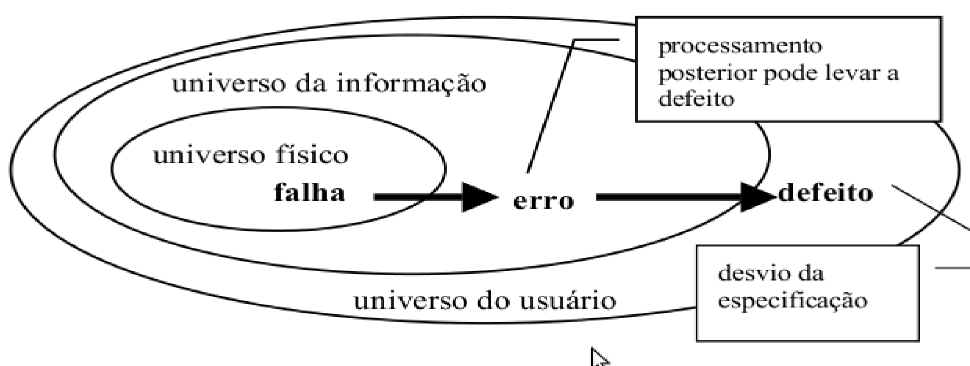

Figura 2.1. Modelo dos 3 universos

Dependabilidade é uma tradução literal do termo inglês *dependability*, que indica a qualidade do serviço fornecido por um dado sistema e a confiança depositada no serviço fornecido.

A dependabilidade não é uma propriedade de um sistema que se pode atribuir valores numéricos diretamente, mas todos os seus atributos correspondem a medidas numéricas. Na Tabela 2.1 é apresentado um resumo dos atributos principais [2].

Tabela 2.1. Resumo dos principais atributos de dependabilidade

Atributo	Significado
Confiabilidade (*reliability*)	capacidade de atender a especificação, dentro de condições definidas, durante certo período de funcionamento e condicionado a estar operacional no início do período
Disponibilidade (*availability*)	probabilidade do sistema estar operacional num instante de tempo determinado; alternância de períodos de funcionamento e reparo

Outros atributos são a segurança (security), a segurança do funcionamento (safety), a mantenabilidade, a testabilidade, o comprometimento do desempenho (*performability*), a confidencialidade (*confidentiality*) e a integridade (*integrity*), para ver mais detalhes sobre essas medidas consultar [2][21].

Sistemas dependáveis podem ser definidos como aqueles em que a dependabilidade é requerida e alcançada pelo uso de um conjunto combinados de técnicas e metódos que podem ser classificados [2]:

1. Prevenção de falhas: como prevenir a introdução e a ocorrência de falhas.

2. Tolerância a falhas: como garantir que um serviço cumpra a função do sistema na presença de falhas.

3. Remoção de falhas: como reduzir a presença (quantidade e gravidade) de falhas.

4. Previsão de falhas: como estimar o presente número, a incidência no futuro, e as consequências de falhas.

O foco desse trabalho são as técnicas de tolerância a falhas, que, assim como as outras, tem como objetivo alcançar dependabilidade.

2.1.1 Confiabilidade, disponibilidade e sistemas dependáveis críticos

O atributo da confiabilidade mede a continuidade do serviço correto de um sistema. Esse atributo é probabilístico, pois a ocorrência de falhas é um fenômeno aleatório, além de ser o atributo de dependabilidade mais usado em sistemas críticos [1], ou seja, nos seguintes sistemas:

- sistemas em que mesmo curtos períodos de operação incorreta são inaceitáveis;
- sistemas em que reparo é impossível.

Exemplos desses sistemas são: exploração espacial, controle tráfego áereo, e sistemas em tempo real onde a não confiabilidade pode levar a perdas de vidas humanas, ou desastres ambientais. As medidas de confiabilidade mais usadas são: taxa de defeitos, MTTF (*mean time to failure)*, MTTR (*mean time to repair)* e MTBF (*mean time between failure).* Para mais informações sobre essas medidas, consultar [1].

Já o atributo da disponibilidade [1] mensura a probabilidade da operacionalidade do sistema em um tempo determinado, isto é, o sistema pode apresentar inoperabilidade, quando está sendo reparado, desde que esses períodos sejam curtos e não comprometam a qualidade do serviço. Quanto menor for o tempo entre os reparos, maior a disponibilidade do sistema.

Apesar dos atribuitos de disponibilidade e confiabilidade representarem medidas diferentes, eles não são excludentes, e muitos sistemas requerem as duas características. Exemplos desses sistemas dependáveis são:

1. aplicações críticas de sistemas de tempo real como medicina;

2. controle de processos e transportes aéreos;
3. aplicações seguras de tempo real como transportes urbanos;
4. aplicações em sistemas de tempo real de longo período de duração sem manutenção, como em viagens espaciais, satélites e sondas;
5. aplicações técnicas como telefonia e telecomunicações;
6. aplicações comerciais de alta disponibilidade como sistemas de transação e servidores de redes.

2.2 Tolerância a falhas

Dentre as técnicas para alcançar a dependabilidade, tolerância a falhas deve ser empregada para sistemas que requerem alta confiabilidade e alta disponibilidade. Na ocorrência de falhas em um sistema, e para garantir o seu funcionamento correto, empregam-se essas técnicas que são baseadas tanto em redundância de *software*, como de *hardware*, ou ambas. Na Sub-seção anterior 2.1.1, foram apresentados alguns exemplos de sistemas dependáveis, e todos eles são áreas tradicionais nas quais se emprega tolerância a falhas. Embora nos sistemas computacionais a exigência para disponibilidade e confiabilidade podem ser encontradas em qualquer área, sistemas tolerantes a falhas tem um alto custo associado, portanto, são empregados apenas nas situações em que a não utilização acarretaria em prejuízos irrecuperáveis.

Em geral, as técnicas de tolerância a falhas são implementadas por detecção de erro seguido de recuperação do sistema. O processo de recuperação restaura o sistema para um estado livre das falhas e erros que são condições indesejáveis. As técnicas são classificadas de acordo com sua aplicação, no contexto da taxonomia que Anderson e Lee [22] propuseram: detecção, confinamento, recuperação e tratamento.

Os mecanismos usados na classe de detecção de erros são: replicação; testes de limites de tempo; cão de guarda (*watchdog timers*); testes reversos; codificação: paridade, códigos de detecção de erros (*Hamming*); teste de razoabilidade, de limites e de compatibilidades; testes estruturais e de consistência; além de diagnóstico. Essa primeira fase ocorre logo após a primeira manifestação da falha em um estado errôneo, sendo detectada por alguns desses mecanismos citados. Entretanto, vale salientar que uma falha pode ficar latente durante toda vida útil do sistema sem necessariamente se manifestar como erro.

Já os mecanismos do confinamento são: ações atômicas; operações primitivas auto encapsuladas; isolamento de processos; regras como tudo que não é permitido é proibido; hierarquia de processos e controle de recursos. Após a ocorrência da falha até a manifestação do erro, pode ter ocorrido o espalhamento de dados inválidos, sendo, portanto, a fase do confinamento importante para evitar a propagação do dano, colocando limites nos fluxos de informação para evitar fluxos acidentais, além de estabelecer interfaces de verificação para detecção de erros.

A fase de recuperação de erros usa os mecanismos de blocos de recuperação por retrocesso ou por avanço, e os mecanismos de tratamento de falhas são a detecção e o reparo. Após a detecção do erro, a recuperação pode levar de um estado errôneo para um estado de funcionamento livre daquela falha. A recuperação pode se dar em duas formas: através de técnicas de de recuperação por retrocesso (*backward error recovery ou rollback recovery*) e técnicas de recuperação por avanço (*forward error recovery*). Na Figura 2.2 se pode visualizar essas técnicas. As técnicas de recuperação funcionam de maneira simples em sistemas monoprocessados, mas pode ter grande complexidade nos sistemas distribuídos, já que ao desfazer uma computação realizada, o processo deixa mensagens órfãs na rede [53]. A complexidade está em estabelecer que os outros processos que recebam essas mensagens também desfaçam a computação realizada, e assim sucessivamente com os processos que recebam mensagens destes. Uma solução nesse caso é impor restrições a comunicação entre processos.

As técnicas de recuperação por retrocesso (*rollback recovery*) são inadequadas em sistemas de tempo real, e nesses sistemas a recuperação por avanço deve ser usada.

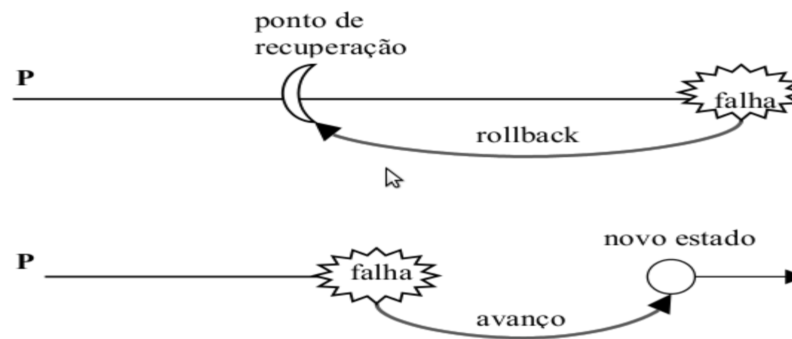

Figura 2.2. Recuperação por retrocesso e por avanço

Na última fase proposta por Anderson e Lee [22], o tratamento de falhas tem os seguintes mecanismos: localização da falha, que feita de duas formas, rápida e precisa; e o reparo da falha, recuperação do restante do sistema. A localização rápida ocorre em um módulo ou subsistema e a precisa ocorre de forma mais lenta, já que deve determinar o componente falho. O mecanismo de localização usa um teste de comparação entre os resultados obtidos e os resultados previstos, sendo feitos de forma automática ou manual. O diagnóstico automático deve ser feito por um componente do sistema que seja de alta confiabilidade. Além disso, no mecanismo de reparo remove-se o componente falho, manualmente ou automaticamente. O reparo automático pode envolver uma reconfiguração do sistema para operar com menos componentes ou a substituição do componente por um outro disponível no sistema, sendo este empregado quando há impossibilidade de reparo manual. Para esse caso, temos como exemplos, os satélites e as sondas espaciais.

Além dessas fases que se complementam, alguns autores como [1] consideram a fase extra de mascaramento, que garante resposta correta do sistema mesmo na presença da falhas. Com os mecanismos de mascaramento, o estado

errôneo não se manifesta para aquela falha, com isso não se precisa ter o custo com as fases de detecção, confinamento e recuperação. Mas, há casos que as falhas são permanentes e, portanto, é necessário usar os mecanismos da fase de localização. Alguns dos mecanismos do mascaramento são: redundância de *hardware, o*u seja, replicação de componentes*;* diversidade de *software*, também conhecida como programação de n-versões; códigos de correção de erro; e os blocos de recuperação.

O uso de redundância tanto de *hardware* como de *software* para aumentar a confiabilidade de um sistema tem um custo associado, podendo torná-los maiores e mais caros, e como todas as técnicas de tolerância a falhas usam algum tipo de redundância, faz-se necessário avaliar o uso delas no desenvolvimento de algum sistema. Como parte deste trabalho, vamos explicar a redundância de *software,* já que a complexidade dos softwares desenvolvidos hoje em dia faz com que suas falhas, e também falhas de especificação, sejam consideradas graves problema em computação crítica. Também, nas sub-seções a seguir aborda-se a redundância de *hardware*.

2.2.1 Redundância de *Hardware*

A redundância de hardware é provavelmente a técnica mais comum de proteção usada em sistemas [24] [25]. Existem basicamente 3 formas básicas de redundância de hardware: redundância estática, que alcança a tolerância a falhas sem que para isto seja necessário detectar uma falha; redundância dinâmica, que utiliza o conceito de detecção e posterior recuperação de falhas; e redundância híbrida, que utiliza características das duas.

2.2.1.1 Redundância estática

A mais comum técnica de mascaramento de falhas é a chamada TMR - *Triple Module Redundancy* [24] [25]. A técnica foi proposta por Von Neumann (1956) e está representada na Figura 2.3.

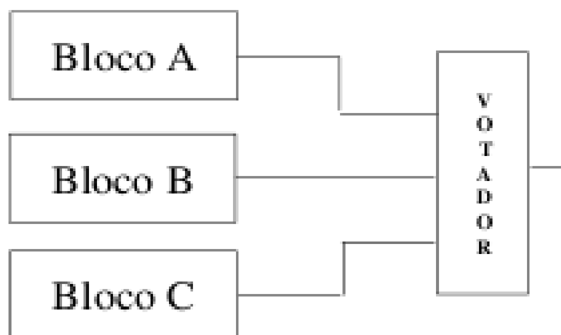

Figura 2.3. TMR – *Triple Module Redundancy*

A utilização de TMR como técnica de proteção implica em algumas desvantagens. *A* mais clara desvantagem reside no acréscimo de recursos: mais de 200%. Outra desvantagem é referente ao circuito votador, que se constitui no único ponto vulnerável a falhas, ou seja, dada uma falha neste circuito a técnica não garante mais a corretude dos resultados. Por fim, outro problema que não pode ser ignorado diz respeito à possível ocorrência de uma falha em mais de um bloco por vez.

A fim de evitar a última desvantagem citada no parágrafo anterior, o conceito de TMR pode ser expandido para incluir qualquer número de blocos redundantes, dando origem, assim, aos chamados NMR - *N Modular Redundancy* - redudância modular múltipla.

2.2.1.2 Redundância Dinâmica

Outra categoria de técnica de proteção são as chamadas técnicas ativas, também conhecidas como técnicas dinâmicas. Estas, ao contrário das técnicas passivas, tem por objetivos a detecção de uma falha no sistema e, se necessário, a substituição de partes que não estejam funcionando corretamente. Em outras

palavras, o sistema que faz uso de técnicas deste gênero deve estar apto a ser reconfigurado [24] [25].

Um sistema que faz uso de redundância dinâmica utiliza vários módulos idênticos, porém apenas um operando a cada vez. Se uma falha é detectada no módulo que está operando, este é substituído por um módulo reserva. Está técnica pode gerar problemas se a falha persistir e atingir também o módulo reserva.

2.2.1.3 Redundância Híbrida

Técnicas de tolerância a falhas baseadas em redundância híbrida combinam características de técnicas estáticas e dinâmicas. Geralmente o mascaramento de falhas é usado como forma de prevenir o sistema de produzir resultados errôneos, enquanto que, a detecção, localização e recuperação de falhas são utilizadas a fim de reconfigurar o sistema na presença de uma falha.

A grande desvantagem de técnicas desta natureza é a quantidade de hardware extra despendida. Como exemplo, para a implementação de um TMR é necessário no mínimo 3 blocos redundantes extras, um detector de erros, um circuito de chaveamento e mais um circuito votador [24] [25].

A maioria dos sistemas com redundância híbrida usam o conceito de NRM com reserva (*N-modular redundancy with spares*), que combina as técnicas NMR e *Standby Sparing*, ou seja, consiste em compor um conjunto de módulo operando em NMR, os quais, na ocorrência de falhas, podem ser substituídos por módulos de reserva [24] [25].

2.2.2 Redudância de *Software*

Diferentemente da redundância de hardware [1], a redundância de software não usa componentes iguais na replicação, já que seria uma estratégia inútil, pois o componente de software falho que for idêntico, apresentaria o mesmo estado errôneo. Portando colocar em paralelo, ou executar o mesmo programa duas vezes

não resolveria a condição de tolerar a falha. Para garantir confiabilidade no *software*, usa-se os seguintes mecanismos de redundância: diversidade e blocos de recuperação.

2.2.2.1 Diversidade

Também conhecida como programação diversitária, ou ainda programação n-versões, como citado por Algirdas Avizienis e John P. J. Kelly [23]: "é uma a abordagem em que o *hardware* e os componentes de *software* que estão sendo utilizados em várias computações, não são apenas cópias destes, mas são independentemente projetados para atender os requisitos de um sistema, de acordo com o especificado".

Após pesquisa realizada na UCLA (*University of California, Los Angeles*), que iniciou em 1976, a adaptação da redundância modular múltipla usado em *hardware*, que usa mecanismo de votação da maioria, resultou no primeiro experimento realizado na programação de n-versões [23]. A técnica apresenta a desvantagem de não considerar se erros em programas alternativos apresentam a mesma causa. Com isso, para serem detectados, os erros têm que se manifestar de forma diferente nas diversas versões do *software*. Para que os erros possam ser detectados, eles devem necessariamente se manifestar de forma diferente nas diversas versões, ou seja, devem ser estatisticamente independentes.

Outra desvantagem da técnica, é o aumento do custos de desenvolvimento e manutenção, a complexidade de sincronização das versões e problemas na determinação da correlação das fontes de erro [1].

Uma das vantagem da programação diversitária é de poder ser utilizada em todas as fases do desenvolvimento de um programa, desde sua especificação até os testes, dependendo do tipo de erro que se deseja detectar (erro de especificação, de projeto, ou de implementação). Ela é conhecida como projeto diversitário pois, quando usada, o desenvolvimento de sistemas é feito de forma diversitária. Já o

termo programação em n-versões é usado quando se restringe à implementação do sistema. Outras vantagens são: A facilidade no reconhecimento de erros na fase de teste do sistema, a tolerância tanto de falhas intermitentes quanto de permanentes e a atuação potencial também contra erros externos ao sistema (como, por exemplo, erros do compilador, do sistema operacional e até mesmo falhas de hardware) [1].

2.2.2.2 Blocos de recuperação

O segundo experimento da UCLA teve como resultado o mecanismo de bloco de recuperação. Ele se assemelha à programação diversitária, com a diferença que os programas secundários só irão executar quando detectado um erro no programa principal. Para isso, a técnica utiliza um teste de aceitação. Os programas são executados e testados um a um, até que um deles passe no teste, considerando n-1 testes para as n-versões do programa, e com falhas independentes entre elas. Na Figura 2.4. visualiza-se o funcionamento do bloco de recuperação.

Nas próximas sub-seções são apresentados com mais detalhes os conceitos das técnicas utilizadas na experimentação, inclusive, com mais profundidade, os blocos de recuperação e a programação de n-versões, além do *Consensus Recovery Block* (CRB).

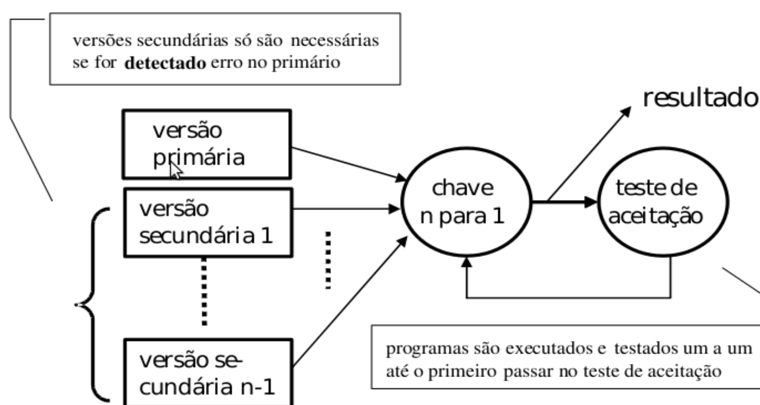

Figura 2.4. Blocos de recuperação

2.2.3 Recovery Blocks

Como descrito anteriormente, os blocos de recuperação permitem aos desenvolvedores inserir testes de aceitação em fases intermediárias da execução dos programas, alterando o fluxo da execução quando os testes mostram que o trecho testado do programa tem um comportamento inadequado.

O bloco de recuperação pode agir por retrocesso ou por avanço. O *backward error recovery* utiliza um ponto de recuperação para armazenar dados e informações que serão usados numa posterior recuperação em casos de erro. O ponto de recuperação (*recovery point*) fica ativo a partir do momento que é estabelecido até o momento que é descartado. Esse esquema de recuperação não só permite reparos em dados inválidos, como também em comportamentos de procedimentos. Os fluxos de execução alternativos devem ser funcionalmente equivalentes e, para garantir maior confiabilidade, a independência entre eles é um fator importante para que sejam evitadas falhas de origem comum entre eles.

O *backward error recovery ou rollback recovery* desfaz ações executadas e retorna para o ponto de recuperação de modo a seguir a execução a partir deste. Uma das vantagens dessa técnica é a possibilidade de recuperação do sistema das condições de erro, sem que haja a preocupação de saber a causa da falha, já que em software, são em sua maior parte de natureza não-adiantada. Já a desvantagem desta técnica está relacionada ao desempenho e a utilização de recursos do sistema para armazenamento das informações necessárias à recuperação. Além de como dito anteriormente, na Seção 1.2 deste capítulo, seu uso é inadequado para sistemas em tempo real. Alguns autores como Randell [26], assumiu que os processos podem ser sincronizados com relação ao descarte dos pontos de recuperação e sugeriu um mecanismo denominado "*conversations*" entre os processos. Com isso seria possível atender aos requisitos temporais do sistemas críticos de tempo real.

O bloco de recuperação por avanço, *rollforward recovery block*, é mais adequado em sistema críticos de tempo real. Esse mecanismo leva o sistema para

um estado a frente, para que fique livre da falha. Como será visto na experimentação, a utilização de tratamento de exceções pode ser classificado como um *rollforward recovery block.*

*O*s blocos de recuperação por retrocesso e por avanço não são mutuamente exclusivos. O mecanismo de retrocesso pode ser usado em primeiro lugar e, se o erro persistir, o mecanismo de avanço pode então ser então usado.

A proposição feita por Randell [26] da estrutura de um bloco de recuperação é visualizada na Figura 2.5 a seguir:

```
ensure  Acceptance  Test
by  Alternate1
else by  Alternate2
. . . . . . . . . . .
else by  Alternaten
else  error.
```

Figura 2.5. Estrutura do Bloco de Recuperação proposto por Randell.

Como percebesse na Figura 2.5, a verificação da validade de cada uma das alternativas será realizada por meio de um teste de aceitação, o qual pode ser único para todas as alternativas, ou específico para cada uma delas. Os testes de aceitação não garantem a correção, mas a validação dos resultados produzidos por cada fluxo alternativo. A estrutura dos blocos de recuperação também pode incorporar outras estruturas de blocos de recuperação recursivamente. E como visto na Figura 2.5, a estrutura proposta por Randel é um construtor de programa que visa colocar o sistema para um estado coerente para que a execução normal ainda possa prosseguir usando pontos de verificação e recuperação combinados com recuperação por retrocesso e diversidade na codificação dos fluxo de execução. Isto permite a diversidade e a redundância, ocultando-as dentro dos blocos do programa.

2.2.4 NVP

Define-se como N-versões de software, versões que são projetadas independentemente, executadas simultaneamente e devendo satisfazer uma mesma especificação inicial [19]. Essa especificação inicial deve ser correta e completa, e não gere ambigüidade, pois requisitos imprecisos ao serem gerados, podem levar os programadores a cometerem falhas de projetos semelhantes. O objetivo do NVP é minimizar a probabilidade de acontecer erros similares.

Segundo Fernandes. S. M. M. [20]: "As N versões são ditas independentes pois são produzidas utilizando-se diferentes algoritmos, desenvolvidas por profissionais de diferentes perfis profissionais e usando diferentes linguagens de programação, compiladores e sistemas operacionais, entre outros recursos diversos. Os resultados produzidos pela execução concorrente dos N maior ou igual a 2 programas, funcionalmente equivalentes, são comparados por um mecanismo de decisão, um teste de replicação que considera um resultado como válido quando a maioria das saídas forem suficientemente similares. O mecanismo de decisão se torna cada vez mais complexo, quanto mais idênticos forem os resultados a serem validado. Baseado no voto de maioria o votador pode mascarar resultados errôneos, e liberar um resultado válido, de acordo com as versões majoritárias, para o restante do sistema. Se não houver acordo da maioria das N-versões, um resultado inválido é produzido para o sistema". Na Figura 2.6. visualiza-se o funcionamento do NVP.

Os mecanismos de decisão, podem ser exato ou inexato. Também segundo Fernandes [20]: "a votação exata é usada para comparações aritméticas ou entre strings, onde a votação majoritária de valores iguais é predominante, isto é, os resultados são idênticos bit-a-bit". Como será visto no Capítulo 6, foi esse o mecanismo de decisão utilizado na experimentação do NVP.

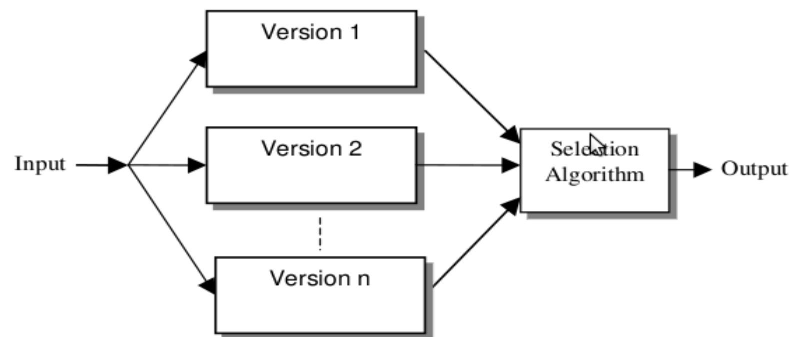

Figura 2.6. Funcionamento da programação de n-versões

Complementado a estrutura do NVP, ainda usando [2o] para conceituar a votação inexata, temos que: “é usada em aplicações onde a saída dos módulos da N-versões são valores numéricos, de natureza continua. Também se utiliza a votação inexata em aplicações onde discrepâncias entre resultados são admitidas, em virtude de deficiências do hardware ou do algorítmo utilizado, desde que a diferença entre os resultados das versões estejam dentro de um intervalo de aceitação. Um programa supervisor é usado para supervisionar todas as interações entre as N versões de software e o votador, além de manusear parte do mecanismo de sincronização, o qual coloca as versões independentes em diferentes estados de operação”.

Como será visto na experimentação, o NVP, pode gerar resultados falsos-positivos, no qual a votação acorde um resultado correto baseado na maioria resultante de resultados errôneos.

2.2.5 CRB

O *Consensus Recovery Block* foi introduzido conceitualmente por R.K. Scott no ano de 1983 [18] (ver Figura 2.7). Ele combina em sua abordagem o mecanismo de NVP e *Recovery Blocks* para melhorar a confiabilidade.

De acordo com Scott [18], os testes de aceitação nos Blocos de Recuperação sofrem com a falta de diretrizes para seu desenvolvimento. Já o uso de votador em NVP pode não ser apropriado em todas as situações, especialmente quando várias saídas corretas são possíveis. Nesse caso, um votador, por exemplo, iria declarar uma falha na seleção de uma saída apropriada.

Consensus Recovery Blocks usa um algoritmo de decisão semelhante NVP, como uma primeira camada de decisão. Se esta primeira camada identifica uma falha, uma segunda camada utilizando, testes de aceitação semelhantes aos utilizados na abordagem dos blocos de recuperação, é invocado. Embora, obviamente, muito mais complexa do que qualquer uma das técnicas individuais, os modelos de confiabilidade indicam que esta abordagem combinada tem o potencial de produzir um software mais confiável [18].

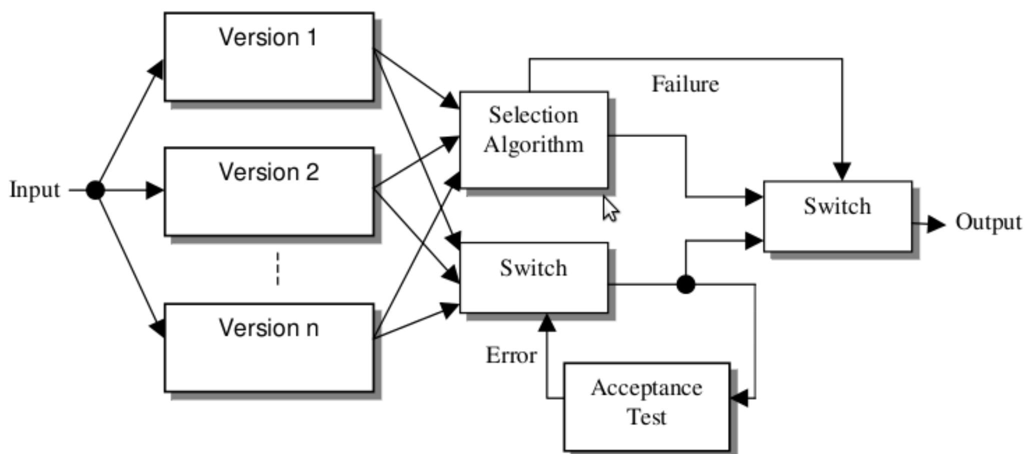

Figura 2.7. Consensus Recovery Blocks

Capítulo 3
Linguagem ADA

"ADA foi originalmente concebida com três preocupações primordiais: confiabilidade e manutenção do programa, a programação como uma atividade humana, e eficiência. A revisão de 1995 da linguagem foi desenvolvida para proporcionar maior flexibilidade e extensibilidade, controle adicional sobre o gerenciamento de armazenamento e sincronização, e os pacotes padronizados orientado para apoiar as áreas de aplicação importantes, enquanto ao mesmo tempo mantendo a tônica original na confiabilidade, eficiência e manutenção. Esta nova versão, prevê maior flexibilidade e acrescenta mais pacotes padronizados dentro do quadro previsto pela revisão de 1995", trecho retirado da ALRM (*Ada Language Reference Manual*) [4].

A linguagem ADA tem uma gama de compiladores suportando muitas arquiteturas. O compilador GNAT é livre, open source e pode ser baixado pela web [54]. Existem compiladores ADA e ferramentas comerciais, como o *Ada Core Technologies (*GNAT*)* [55], DDC-I [56], *Rational da IBM* [57], *RR Software* [58], *Irvine Compiler Corporation* [59], *Green Hills* [60], *SofCheck* [61], *Aonix* [62], *e OC Systems* [63].

O uso de ADA tem alguns objetivos:

- Alta confiabilidade e segurança para ambientes de segurança crítica,
- Manutenção durante um longo período por quem nunca viu o código antes,
- Ênfase na legibilidade do programa, em vez de capacidade de escrita do programa,

- Capacidade de desenvolvimento eficiente de software utilizando componentes reusáveis.

Em resumo, ADA foi projetada para maximizar a verificação de erros que um compilador pode fazer no início do processo de desenvolvimento. Cada construção sintática se destina a ajudar o compilador atingir esse objetivo. O padrão para cada construtor ADA é seguro, mas é permitido relaxar esse padrão, quando necessário. O padrão seguro de ADA, contrasta com a maioria da família de linguagens C onde o padrão é, normalmente, inseguro. O padrão seguro, dos construtores da linguagem ADA, é uma das mais importantes contribuições de ADA na área de engenharia de software [27].

O *software* em sistemas de controle de tráfego aéreo, aviônicos, dispositivos médicos, estradas de ferro, foguetes, satélites e comunicações seguras de dados é freqüentemente escrito em ADA [27]. A linguagem tem suporte a arquiterturas *multicore*, multiprocessadores, e *multithreaded* desde de seu início. Enquanto alguns preveram que levará mais de uma década antes que haja um modelo de programação para sistemas *multi-core*, os programadores têm usado com sucesso o modelo de ADA durante anos, segundo é dito em [27].

Algumas informações para o início do estudo sobre ADA:

- ADA tem uma lista total de 72 palavras reservadas que pode se encontrada na Seção 2.9 do seu manual de referência [4].
- ADA fornece todas as estruturas de controle esperada em uma linguagem de programação de alto nível.
- Como será visto na Sub-seção 1.14, pode-se inserir instruções de código de máquina diretamente em programas ADA.

3.1 Subprogramas

Um programa em ADA é composto de *library units*. A *library unit* é uma unidade que pode ser referenciada dentro da cláusula with. Como já dito o nome técnico da cláusula **with** é cláusula de contexto, e ela é como a diretiva de compilador #include de outras linguagens, mas com importantes diferenças.

Cada *library unit* deve compilar com sucesso antes do compilador reconhecê-la em uma cláusula contexto. Cada unidade compilada é colocada em uma biblioteca. Ao contrário de um #include, a cláusula de contexto não torna os elementos de uma *library unit* visível. Em vez disso, uma cláusula de contexto simplesmente a coloca no escopo, tornando-a potencialmente visível.

3.1.1 Composição de um progama ADA

Um programa em ADA é compostos por um ou mais “units”. Programas units podem ser: “subprogramas” que define algoritmos executáveis; “packages” no qual define coleções de entidades; “task units” no qual define a computação concorrente; “protected units” no qual define operações para coordenar compartilhamento de dados entre as “tasks”; “generics units” no qual define formas parametrizadas de “packages” e “subprogramas”.

3.1.2 Distinção de um unit progama

Cada unit programa normalmente consiste em duas partes: a especificação contendo a informação o que pode ser visível para outros programas units; e o “body” corpo contendo detalhes da implementação, no qual não necessita de estar visível para outros “progamas units”. A Vantagem da distinção da especificação e do corpo da implementação e da habilidade de compilar separadamente programas units, permite desenvolver, escrever e testar, como um conjunto de componentes de software independentes.

3.1.3 Programas units

3.1.3.1 Subprogramas

O mecanismo básico de expressão de algoritmos são as classes de subprogramas que são funções e procedimentos. Os procedimentos são mecanismos que envolve uma série de ações, como exemplo ele pode ler dados, atualizar variáveis, ou produzir alguma saída. Ele pode ter parâmetros que é uma forma de controlar a a passagem de informação entre o procedimento e o seu ponto de chamada. Já função e um mecanismo que envolver a computação de um valor. É similar ao procedimento mas em adição retorna algum valor como resultado.

3.1.3.2 Packages units

Unidade básica que define uma coleção de entidades logicamente relacionadas. Por exemplo um “package” pode ser usado para definir um conjunto de declarações de tipos e operações associadas. Partes de um “package” pode ser ocultas dos usuários, assim permitindo acesso somente as propriedades lógicas expressas pela especificação do “package”.

3.1.3.3 Tasks units

Unidade básica para definir uma tarefa cuja seqüência de ações podem se executadas concorrentemente com outras tarefas.

3.1.3.4 Protected units

Unidade básica que define operações protegidas para coordenar o uso de dados compartilhados entres tarefas. Uma simples exclusão mutua é provida automaticamente, e protocolos mais elaborados de compartilhamento podem ser definidos. Uma operação protegida pode ser tanto um subprograma ou uma entrada. Uma entrada protegida especifica uma expressão booleana (uma barreira de entrada) que deve ser Verdadeira antes do corpo da entrada ser executado. Uma

"*protected unit*" pode definir um único objeto protegido ou tipos protegidos que permitam a criação de vários objetos semelhantes.

3.2 Operadores e operações

Métodos em Ada são subprogramas (procedimentos / funções) e incluem tanto os operadores e operações. Os operadores incluem os símbolos: =, / =, <,>, <=,> =, &, +, -, /, *. Outros operadores são as palavras reservadas, **and**, **or**, **xor**, **not**, **abs**, **rem**, **mod**. Um aspecto de ADA que pode criar dificuldade inicial a um novo programador, são as regras de visibilidade associadas com os operadores. Será discutido as regras de visibilidade e as formas de usá-las na Seção 1.3 desse capítulo. Para ter mais informações referentes aos operadores, consultar a Seção 4.5 do capítulo 4 da ALRM [4].

Todos os outros métodos são chamados de operações. "Uma operação de atribuição usa o símbolo composto: ":=" . Na linguagem ADA, não se pode diretamente sobrecarregar a operação de atribuição. A operação de atribuição é pré-definida para a maioria dos tipos ADA, mas ela é proibida para os tipos limitados. Veremos mais sobre Tipos de ADA mais adiante na Seção 1.5.

ADA dar possibilidade de o programador, definir métodos para tipos específicos, além de pode sobrecarregar e/ou substituir métodos existentes, quando esses estão declarados em alguma especificação de **package**.

3.3 Escopo e Visibilidade

A incapacidade de compreender a diferença entre escopo e visibilidade, causa mais problemas para os programadores novatos de ADA do que qualquer outro aspecto da linguagem. Se trata de uma idéia central para o desenvolvimento de todos os softwares em ADA. Existindo capítulo inteiro dedicado à visibilidade no

manual de referência ALRM[4], onde em seu capítulo 8 se encontra muitos exemplos de uso.

A cláusula **with,** que é uma palavra-chave da linguagem**,** coloca uma unit libray no escopo, mas nenhum dos recursos dela são diretamente visíveis a um cliente, diferentemente do “#include” da família da linguagem C. Separar escopo de visibilidade é um importante conceito de engenharia de *software*. ADA tem diversas técnicas para fazer no escopo elementos diretamente visíveis.

3.3.1 Escopo

Cada declaração de ADA tem um delimitador do escopo, sendo às vezes, fácil de ver no código fonte. Existe um ponto de entrada (declare, identificador de subprograma , identificador de tipo composto, identificador de package, etc) e um ponto explícito de termino. Terminações explícitas são codificadas com uma declaração final. Toda vez que se encontra uma cláusula **end**, você sabe que está fechando um escopo. O escopo pode ser aninhado. Por exemplo, um procedimento pode vim ser declarado dentro de outro procedimento. A cláusula de contexto **with** não é tão óbvia. O contexto da cláusula coloca todas os recursos de uma *library unit* no escopo, mas não deixa elas diretamente visíveis. Mas adiante na Seção 1.4, desse capítulo, abordasse as *librarys units*.

3.3.2 Visibilidade

Uma entidade pode estar no escopo, mas não diretamente visível. Este conceito é melhor desenvolvido em ADA que na maioria das linguagens de programação. Na Tabela 3.1 tem-se as regras de visibilidade.

Tabela 3.1 Visibilidade em ADA

Clausula **use**	Todas os dados públicos de um **package** ficam visíveis.
Clausula **use type**	Faz todas as operadores estarem diretamente visíveis para um tipo designado.
Notação de ponto da entidade	A entidade na notação é diretamente visível, usualmente é uma opção mais indicada [27].
Renomear localmente operadores e operações	Opção usualmente mais indicada para fazer operadores diretamente visíveis [27].

3.4 Tipos em ADA

Um tipo é definido por um conjunto de valores possíveis (seu domínio) e um conjunto de operações primitivas sobre esses valores. Exemplos de tipos definidos na linguagem, são tais como Integer, Float e Boolean. ADA nos permite definir os nossos próprios tipos simples e complexos. O modelo de tipagem de ADA é uma característica significativa da linguagem. Ela permite criar modelos precisos do mundo real e fornecer informações valiosas para o compilador para que possa identificar erros antes que o programa ser executado. Quatro princípios regem o sistema de tipagem de ADA [27, pág 35], são eles: tipagem forte, tipagem estática, abstração e equivalência de nomes.

A melhor maneira de perceber a tipagem forte é pensar em termos de segurança de tipo. Como já foi dito anteriormente em outra Seção nesse capítulo, os construtores da linguagem são por padrão seguros, logo por padrão os tipos de ADA são seguros. Para muitas linguagens o tipo seguro não é o padrão. Os desenvolvedores em ADA declaram os tipos de dados, usualmente na especificação

de um package. A declaração de tipos incluem o conjunto de valores e operações apropriadas para a resolução dos problemas abordados. Isso garante um contrato rigoroso entre o cliente de um tipo e o *package* em que o tipo é definido.

Tipos ADA podem ser categorizados em: limitados, e não limitados. Um tipo limitado não pode ser usado em uma operação de atribuição com a expressão ":=". Outros tipos podem usar com a operação de atribuição, desde de que sejam compatíveis entre si ou ocorra conversão entre tipos. ADA define alguns tipos como sempre limitados. Estes incluem *task types*, *protected types*, e *record types.*

Tipos em ADA pode ser considerada em termos de sua visão. Um tipo pode ser definido com uma exibição pública que pode ser visto por um cliente do tipo, e uma visão não-pública que é visto pela implementação do tipo. Às vezes nós falamos da visão parcial do tipo. Uma visão parcial é uma exibição pública de uma visão não-pública correspondente. Visões parciais são geralmente definidas como privadas ou privadas limitada . Além disso, a exibição pública de um tipo pode ser limitado quando a visão de implementação do mesmo tipo podem ser não-limitado. ADA não define uma visão protegida diretamente análoga ao C + + ou Java. No entanto, algumas das propriedades essenciais dessa visão estão disponíveis conforme necessário.

Outra categoria importante é o tipo privado versus tipos não-privado. Um tipo limitado também pode ser privado. Um tipo com uma exibição privada também podem ter uma visão que não é privada. As operações pré-definidas para um tipo não-limitado privado incluem: operação de atribuição, operador =, operador /= . Quaisquer outras operações de um tipo privado deve ser declarado explicitamente pela especificação do pacote em que o tipo é declarado publicamente. Para mais detalhes e visualização de exemplos, consultar a Seção 7.3 do capítulo 7 da ALRM[4]. Na figura 3.1 é apresentada a hierarquia de tipos de ADA.

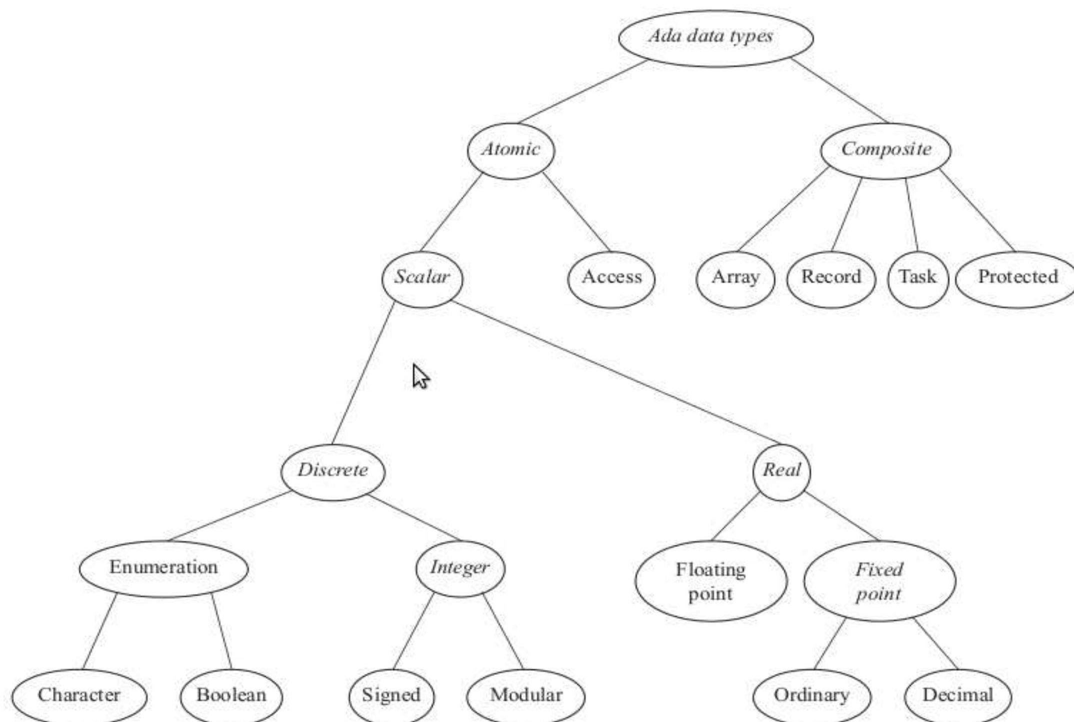

Figura 3.1. Hierarquia de Tipos em ADA

Algumas características de tipos limitados são importantes ressaltar. Nos tipos limitados, como foi dito, além da operação de atribuição, o teste de igualdade também não está disponível, e são também tipos para os quais a cópia não é permitida. Um tipo limitado é um tipo que inclui uma das seguintes palavras reservadas **limited**, **synchronized**, **task**, ou **protected** na sua definição ou na definição de um componente de um tipo composto.

3.4.1 Tipos Elementares

Tipos elementares são de duas categorias principais, escalar e de acesso. Um tipo de acesso é uma espécie de ponteiro e será discutido na Sub-seção 1.6 deste capítulo. Tipos escalares são discretos e reais. Tipos discretos são tipos enumerados e tipos inteiros. Tecnicamente, tipos inteiros também são enumerados com a adição da funcionalidade dos operadores aritméticos. Tipos numéricos discretos são inteiros com e sem sinal.

Tipos escalares não-discretos são os números reais, que inclui os pontos flutuantes, os fixos ordinários e os fixos decimais.

Todos os tipos escalares podem ser definidos em termos de precisão e em intervalo de valores aceitáveis. Ao desenvolvedor ainda é dada a permissão para especificar a representação interna (número de bits) para um valor escalar. Para ver exemplos, consultar a Seção 3.5 do capítulo 3 da ALRM[4].

3.4.2 Tipos Compostos

Tipos compostos contêm objetos / valores de outro tipo. Existem quatro categorias gerais de tipos compostos: **arrays**, **records**, **task types**, e **protected types**.

3.4.2.1 Arrays

Um array pode ter componentes de qualquer tipo, desde que todos tenham o mesmo tamanho de armazenamento. ADA tem três opções principais para a definição de array: anonymous, baseados em tipo ***unconstrained***, baseados em tipo ***constrained***. ADA permite arrays multidimensionais, bem como arrays of arrays [27 , pág 55] . Para ver exemplos, consultar a Seção 3.6 do capítulo 3 da ALRM[4].

3.4.2.2 Arrays Constrained e Unconstrained

O array ***constrained*** é um tipo para qual existe uma restrição no *range* de indexação. Já o tipo de array ***unconstrained*** a definição prover apenas o tipo da indexação, mas não especifica o *range* do tipo [27, pág 52]. Para ver exemplos, consultar a Seção 3.6 do capítulo 3 da ALRM[4].

3.4.2.3 Records Types

Records são tipos compostos de dados heterogêneos - os componentes são de tipos diferentes. Enquanto no array se acessa um componente específico pela

posição na coleção, nos **records** acessa-se a um componente específico pelo seu nome dado. Para ver exemplos do tipo **records**, consultar a Seção 3.8 do Capítulo 3 da ALRM[4].

3.4.2.4 Tipos derivados

Os Subtipos permitem definir subconjuntos de tipos existentes, cujos valores podem ser combinado com qualquer outro subtipo que compartilha o mesmo tipo base. Às vezes, precisasse criar um novo tipo que é semelhante a um tipo existente, e ainda ser um tipo distinto. Para ver exemplos de tipos derivados, ver a Seção 3.4 do Capítulo 3 da ALRM[4].

3.5 Tipo Access (Ponteiros)

Não existe aritmética de ponteiros em ADA. Quando nos referimos a ponteiros em ADA nos estamos referindo a um conjunto padrão de regras seguras e sem operadores aritméticos.

Há um perigo em potencial com atribuição direta de ponteiros, presente o tempo todo nas linguagens da família C. O que acontece quando um item de dados sai do escopo e ainda tem alguma outra variável apontando para ele? A resposta é que ADA tem regras do compilador para evitar isso. Além disso, essas regras, faz com que uma variável que não esteja mais no escopo não tenha um ponteiro tentando fazer referência a ela. As regras podem ser resumidas em termos de tempo de vida do tipo de acesso em si.

Afim de evitar os riscos do uso de ponteiros, se usa uma declaração de bloco, quem proporciona alocação dinâmica de memória em tempo de execução [27, pág 62]. Para ver mais informações e exemplos de tipos access, consultar a Seção 3.10 do Capítulo 3 da ALRM[4].

3.6 Estruturas de controle e repetição

ADA oferece duas estruturas de controle para a tomada de decisões: o **if** e o **case**. A Seção 5.3 do manual de referência da linguagem (ALRM)[4] dar mais detalhes do uso do **if** e a Seção 5.4 detalhes do **case**. ADA provê estruturas de iteração, **loop**, com diferentes esquemas de iteração. Estes são descritos em detalhes na Seção 5.5 do ALRM.

Um laço interativo incremental como um “**for loop**”, é uma estrutura interativa segura, não permitindo a alteração do índice pela operação de atribuição. O índice se mantém constante durante o “**loop**”. Para ver sobre “**while loop”** consultar a Secção 5.5 da ALRM [4].

Para ver exemplos, consultar o manual de referência de ADA [4].

3.7 Exceções

Algumas vezes a execução de um programa ADA entrar em um estado errôneo, e então uma exceção é lançada [27, págs 63-64]. Uma exceção é uma condição anormal que requer tratamento especial. Exceções permitem separar o processamento normal do processamento de erro. Para ver mais informações sobre tratamento de exceções em ADA, consultar a Seção 11.4 do capítulo 11 da ALRM[4].

3.8 Pragmas

Mecanismos que permite os programadores selecionar exatamente quais características são necessárias para o programa que está sendo desenvolvido. O **pragma** é uma diretiva do compilador. Existes muitos pragmas predefinidos em ADA como instruções de otimização, etc. Para ver mais detalhes sobre essas diretivas de compilação, consultar a Seção 10.1.5 do Capítulo 10 da ALRM[4].

3.9 Generics Packages

Um ***generic unit*** é uma unidade de programa que pode ser um subprograma ou um ***generic package*** [4, Seção 12, Capítulo 12]. Um ***generic unit*** é um modelo que pode ser parametrizado, e a partir do seu correspondente (não genérico) subprogramas ou ***packages*** podem ser obtidos. Os *units* resultantes são instâncias dos *generics units*. Os *generics* em ADA é uma ferramenta poderosa para criar código reutilizável e seguro.

3.10 Child packages

Todos os *packages* padrões de ADA são organizados sobre três *packages* pais: Ada, *Interfaces*, e *System* [27, pág 76]. O *package* Ada serve como o pai da maioria das outras biblioteca padrão de *packages*. Recursos para combinar código ADA com código escrito em outra linguagem de programação são encontrados no *package Interfaces* e seus filhos. Finalmente, *package System* e seus filhos fornece definições das características do ambiente particular em que o programa é executado.

3.11 Concorrência e *Tasks*

ADA é uma linguagem de programação concorrente — que fornece intrisicamente os construtores necessários para suportar a criação de programas com múltiplas *threads* de controle. O ADA task é uma implementação de processos de Dijkstra's [28]. Um progamar ADA é composto por uma ou mais **tasks** que executam concorrentemente para resolução dos problemas.

Existem dois modelos de concorrência em ADA: *multitasking*, e objetos distribuídos. Este último, objetos distribuídos, está além do escopo deste trabalho. O Foco da discussão será em *multitasking*. Em ADA este é simplesmente chamado

tasking. Tasking é implementado usando a sintaxe padrão da linguagem ADA e semanticamente juntamente com dois tipos adicionais: ***task types*** e ***protected types***. A sintaxe e a semântica dos ***task types*** e ***protected types*** é descrito no capítulo do Ada *Language Reference Manual* (ALRM)[4]. Para mais informações sobre a semântica, consulta o Annex D e Annex C do ALRM.

Cada ***task*** é uma entidade sequencial que podem operar simultaneamente com, e comunicar com, outras tasks. Uma objeto task pode ser de um tipo anônimo ou um objeto de um tipo ***task****.*

3.12 Mais da ALRM

Nessa Seção é apresentada algumas da *units librarys* usadas na experimentação, como será visto no capítulo 5.

3.12.1 Ada.Unchecked_Conversion

A biblioteca Ada inclui uma função *generic* chamada Ada.*Unchecked_Conversion*, que pode ser usada para copiar partes de objetos de um tipo, para objetos de outros tipos [4, Seção 13.9, cap 13].

3.12.2 Ada.Strings.Unbounded

O *package Strings.Unbounded* é uma biblioteca pré-definida na linguagem que fornece um tipo privado *Unbounded_String* e um conjunto de operações sobre ele [4, anexo A.4.5]. Um objeto do tipo *Unbounded_String* representa uma *String* que o limite mais baixo é 1 e o tamanho pode variar entre 0 e conceitualmente o último número Natural. Os subprogramas de tamanho fixo para manuseio da string são sobrecarregado diretamente para *Unbounded_String*, ou são modificados conforme necessário para refletir a flexibilidade de tamanho. Como o *Unbounded_String* é um tipo privado, os construtores relevantes e as operações de seleção são fornecidas.

3.12.3 Ada.Real_Time

Essa *unit library* foi utilizada para prover a temporização da execução de cada iteração do experimento, como será visto no capítulo 5 deste trabalho. Ela se encontra definida na Anexo D da ALRM[4] que especifica características adicionais de implementação ADA voltada para sistemas de software em tempo real.

3.12.4 Ada.Numerics.Discrete_Random

Facilidades para a geração de números pseudo-aleatórios de ponto flutuante são fornecidos no ***package*** *Numerics.Float_Random*; o ***generic package*** *Numerics. Discrete_Random* fornece estruturas similares para a geração de valores inteiros pseudo-aleatórios e dos tipos enumerados. [4, anexo A.5.2].

Para resumir, valores pseudo-aleatórios de qualquer um desses tipos são chamados de números aleatórios. Algumas das facilidades oferecidas são básicas para todas as aplicações de números aleatórios. Estes incluem um tipo privado limitado e cada um de seus objetos serve como gerador de uma sequência (possivelmente distintos) de números aleatórios; uma função para obter o próximo número "next" aleatório de uma dada sequência de números aleatórios (isto é, do seu gerador); e subprogramas para inicializar ou reinicializar um gerador dado a um estado dependente do tempo ou um estado indicado por um único número inteiro.

Outras facilidades são fornecidas especificamente para aplicações avançadas. Estas incluem subprogramas para salvar e restaurar o estado de um gerador de dados; um tipo privado cujos objetos podem ser usados para armazenar o estado salvo de um gerador; e subprogramas para obter uma representação de cadeia de um estado gerador de dado, ou, dada como uma representação de String, o estado correspondente.

Capítulo 4
ADA com tecnologias associadas

Este capítulo apresenta a linguagem ADA com algumas tecnologias.

4.1 ADA e Sistemas em Tempo Real

Sistemas de tempo real geralmente têm requisitos de confiabilidade muito alta e, portanto, é dos principais candidatos para a inclusão de técnicas de tolerância a falhas. A fim de fornecer tolerância a falhas de *software*, alguma forma “de restauração de estado” é geralmente defendida como um meio de recuperação. Restauração de estado pode ser caro e o custo é agravado para sistemas que utilizam processos simultâneos. A simultaneidade presente na maioria dos sistemas de tempo real e as dificuldades ainda introduzidas por restrições de tempo sugerem que o fornecimento de tolerância para os erros de *software* ser excessivamente caros ou complexos.

ADA é provavelmente a linguagem mais adequada para o desenvolvimento de aplicações paralelas e em tempo real. Desde sua primeira padronização em 1983, os três objetivos principais se mantiveram:

- Confiabilidade e manutenção do programa
- Programação como uma atividade humana
- Eficiência

Há muitas definições de um sistema de tempo real, mas, como seria de esperar, todos os incluir o conceito de tempo. A atividade específica deve ser concluída dentro de um limite de tempo especificado, o prazo final. Um sistema que

não é em tempo real, é considerado correto quando a saída está de acordo com a especificação do programa. Você provavelmente já passou uma boa parte do tempo verificando os programas que você escreveu, testando-as com vários valores de entrada. Você pode com ADA utilizar verificadores de prova como a que está disponível com SPARK ADA [15] para sistemas em tempo real verificar um programa matematicamente. Um programa em tempo real que calcula a resposta correta depois do prazo, passa a não ser correto.

Sistemas em tempo real podem ser classificados como:

- Um *deadline* é um ponto no tempo pelo qual uma atividade deve ser concluída.
- Um sistema de tempo real suave só precisa produzir as respostas corretas; perder um prazo, é apenas uma inconveniência.
- Um sistema de tempo real rígido falha quando um *deadline* é perdido.

Denomina-se um sistema cujo desempenho só é degradado por um prazo não cumprido, um sistema de tempo real suave. Um sistema de tempo real suave só precisa produzir as respostas corretas; perde *deadline* é apenas um inconveniente. Um sistema de tempo real rígido é aquele em que um prazo único perdido, leva a falha do sistema.

Segundo [27], ADA é sem dúvida a linguagem mais adequada para o desenvolvimento de aplicações em tempo real e paralelas. ADA requer uma nova concepção de métodos para sistemas de tempo real, uma vez que difere das linguagens convencionais/combinações de execução de duas maneiras importantes: (I) concorrência e comunicação de processo são inerentes à linguagem e não requerem uma biblioteca *run-time system* (RTS) separada; e (II) a natureza do modelo de ***tasking*** é diferente das abordagens corrente usados para agendamento de tarefas, comunicação e sincronização [27].

Nas Sub-seções seguintes é apresentado dois mecanismos de ADA que são poderosos aliados no desenvolvimento de sistemas de tempo real críticos.

4.1.1 *Ravenscar* profile

O modelo ***tasking*** em ADA é extremamente poderoso, mas sempre foi reconhecido que, no caso de sistemas de missão crítica, é adequado escolher um subconjunto dos mecanismos de ***tasking***, porque a análise exata do sincronismo é difícil de conseguir. Avanços nos métodos de análise de tempo em sistemas de tempo real abriram caminho para um modelo de ***tasking*** confiável em ADA, podendo fazer uma análise precisa dos comportamento em tempo real, onde é possível com uma escolha cuidadosa dos métodos de *scheduling/dispatching*, juntamente com restrições adequadas sobre as interações permitidas entre as tarefas.

Uma das conquistas mais importantes da versão ADA 2005 foi a padronização do profile Ravenscar ***tasking.*** Este *profile* [30] define um subconjunto dos recursos ***tasking*** de Ada, que é passível de análise estática para a certificação de alta integridade do sistema, e que pode ser apoiado por um pequeno sistema de em tempo de execução confiável.

Este profile é estabelecido sobre o estado da arte, construindo concorrência determinística que seja adequada para a construção da maioria dos tipos de software em tempo real [31].

Os principais benefícios deste modelo são:

- Melhora a eficiência da memória e do de tempo de execução, através da remoção de alta sobrecarga ou recursos complexos.
- Maior confiabilidade e previsibilidade, através da remoção de não-determinismo e características não analisáveis.

- Redução de custos de certificação através da remoção de características complexas da linguagem, simplificando assim a geração de provas de previsibilidade, confiabilidade e segurança.

Esse Sub-conjunto de *Restrictions pragma*, também deve ser utilizado para uso em sistemas embarcados de tempo-real que tem restrição de alta integridade. É modelo padronizado de concorrência que atende ao requisito de confiabilidade [14].

pragma profile (Ravenscar) ;

O profile *Ravenscar* é descrito na Seção D.13.1 do Manual de Referência Ada. ALRM[4].

4.1.2 SPARK ADA.

SPARK é uma sub-linguagem de Ada, que é sem ambiguidades e adequada para a análise estática rigorosa [32]. Tem sido amplamente utilizado em aplicações industriais onde a segurança de funcionamento e segurança são fundamentais, como aeroespacial militar, sinalização ferroviária, e sistemas criptográficos de alto grau, como exemplos[33].

A concepção de SPARK está bem alinhada com as abordagens rigorosas de engenharia, tais como PSP/TSP, SixSigma, e os princípios do movimento *Lean Engineering* levando uma "tolerância zero" para a redução de defeitos [32].

Com a linguagem SPARK ADA [32] um *software* correto pode ser criado.

4.2 ADA e Sistemas Embarcados

Sistemas embarcados interagir com o mundo através de sensores e atuadores. ADA é uma das poucas linguagens de programação para fornecer operações de alto nível para controlar e manipular os registros e interrupções desses dispositivos de entrada e de saída [27].

A maioria dos programas escritos para sistemas embarcados são programas em tempo real, e como vimos nas seções anteriores, a linguagem ADA tem características adequadas para desenvolvimento desses sistemas [14][15][30][31].

4.3 ADA com FPGA

4.3.1 FPGA

O avanço da tecnologia CMOS (*complementary metal-oxide-semiconductor)* estado-da-arte, bem como a constante diminuição das dimensões dos transistores, têm tornado o comportamento dos circuitos integrados cada vez mais imprevisível. Desta forma, a implementação de técnicas de tolerância a falhas nos projetos de sistemas digitais têm sido cada vez mais necessárias. Entre esses problemas temos o da falha transiente SEU [34] .

Atualmente, a maioria das técnicas que têm sido propostas para a proteção de arquiteturas implementadas em FPGAs se remetem aos efeitos causados por SEUs. Esses dispositivos programáveis, se constituem em uma opção muito utilizada para a implementação física de circuitos e sistemas eletrônicos. Tais dispositivos, por serem fabricados com tecnologia CMOS estado-da-arte, estão vulneráveis a falhas transientes. Por conta disso algumas empresas desenvolveram FPGA's que utilizam algumas técnicas de proteção, notadamente nas células de memória, com vistas a aplicações espaciais e militares [35]. Entretanto, em função do acréscimo de hardware e da baixa demanda, o preço por peça de tais componentes é muito elevado,

A maioria das técnicas de alto nível utilizadas atualmente, tanto para proteção contra SEUs quanto para proteção contra SETs, baseia-se em redundância de *hardware* [38], redundância temporal [39] ou em uma combinação de ambas [40].

Hoje em dia, dispositivos FPGAs utilizam dezenas de milhões de portas lógicas, interface de entrada e saída de alta velocidade e muitos deles possuem

pequenos processadores embutidos, com intuito de aumentar o desempenho [36]. Estão disponíveis no mercado três principais tipos de tecnologias de FPGAs: baseados em SRAM (“SRAM-*based*”), baseados em anti-fusível (“*anti- fuse-based*”) e baseados em memória FLASH (FLASH-*based*) [36] [37].

4.3.2 Arquitetura do FPGA

O FPGA [41] é constituído por três recursos de Hardware: Bloco lógico: que configuram a lógica do usuário ; Blocos de entrada e sáida: responsáveis por interconectar a estrutura interna do FPGA aos pinos de saída ; Interconexões configuráveis: que têm o objetivo de realizar a ligação entre as estruturas de hardware internas ao dispositivo. Na Figura 4.1, visualiza-se a arquitetura do FPGA.

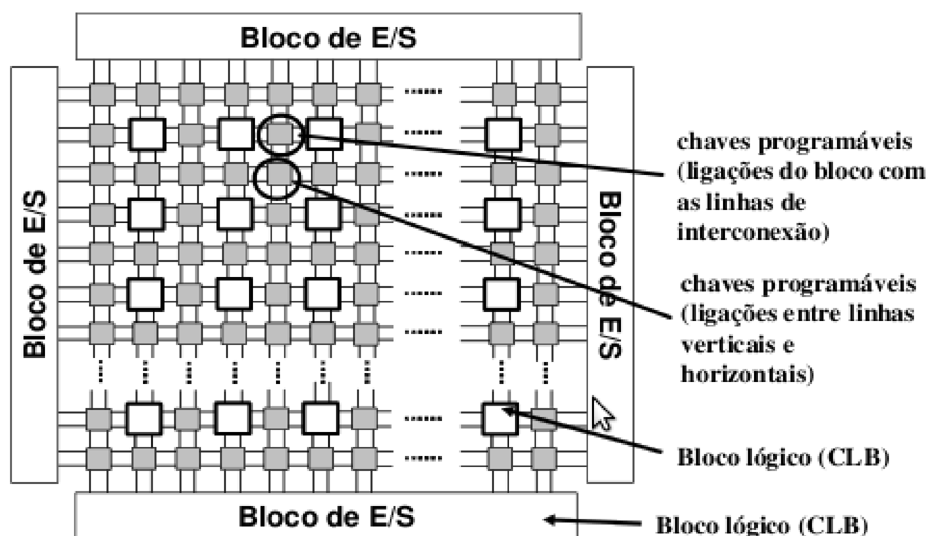

Figura 4.1. Arquitetura do FPGA

Os Blocos lógicos, geralmente, apresentam algumas estruturas básicas. Dentre elas podemos citar a LUT (*Look Up Table*) que utiliza células de memória SRAM para implementação de pequenas funções lógicas, um registrador que pode funcionar tanto como flip-flop quanto latch e um multiplexador. Embora a utilização de LUTs de três entradas seja uma realidade, existe um consenso entre os

fabricantes de FPGAs que LUTs de quatro entradas suprem a maioria das aplicações e portanto, são as mais utilizadas [36] .

Com relação às matrizes de chaveamento, a configurabilidade dessas conexões, assim como nas ALUTs, é realizada utilizando-se memórias SRAM. O processo de carregamento do arquivo (upload) é conhecido como configuração do bitstream [36].

4.3.3 FPGA Tolerante a falhas

A fim de que possam ser reduzidas as taxas de suscetibilidade a falhas provocada por radiação a valores tais quais os exigidos, empresas, tais como Xilinx e Actel, têm investido na construção de FPGAs protegidos para aplicações críticas [35]. Sendo que o alto custo exigido para implementação de FPGAs como os citados anteriormente, devido à necessidade de investimentos em desenvolvimento, teste e fabricação, faz com que o preço por peça de um FPGA protegido seja demasiadamente alto.

Devido ao avanço da tecnologia CMOS, cada vez mais os circuitos operando na superfície da terra estão se tornando vulneráveis a falhas provocadas pela radiação. Assim, soluções que dispensem a utilização de FPGAs protegidos, começam a ser estudas a fim de proteger estes dispositivos operando ao nível do mar. Uma destas soluções é a aplicação de técnicas de mais alto nível às arquiteturas que serão implementadas dentro dos FPGAs.

Uma das técnicas mais utilizadas para proteger FPGA's de SEU's são uma combinação de TMR com scrubbing [42] [38].

A técnica de *scrubbing* [43] consiste em carregar o frame de configuração do FPGA em um dado intervalo de tempo. Este intervalo varia conforme a taxa de SEU's esperada para determinada aplicação. Assim, TMR é responsável por proteger a lógica do usuário enquanto que os *bits* de configuração ficam protegidos pela técnica de *scrubbing*. Alguns FPGAs podem ser configurados parcialmente.

Desta forma, se detectada uma falha em algum bit de configuração do FPGA, apenas o frame que possui um bit errôneo será carregado. Esta técnica é conhecida como reconfiguração parcial [42].

Outras técnicas arquiteturais utilizadas usam a redudância de hardware juntamente com a redudância temporal. A DWC-CED, que é encontrada em [40], e DWC+TR (DWC - duplication with comparison, TR – Time redundancy). Tanto a a duplicação com comparação como a redudância temporal podem detectar falhas transientes na lógica combinacional.

Entretanto, a combinação da redundância temporal com DWC provê uma avaliação interessante das falhas, pois pode não somente detectar a presença de falhas, como também reconhecer em qual dos blocos redundantes a falha ocorreu [44].

4.3.4 ADA como HDL e HLL

Uma pesquisa bibliográfica foi realizada e encontrou estudos de ADA como linguagem ADA como HDL (linguagem de descrição de *Hardware*) [11] e HLL (linguagem de alto-nível) para FPGA [12].

Um outro trabalho publicado [45] discute a viabilidade do uso de um subconjunto de ADA como uma linguagem de descrição de hardware. Métodos são apresentados para realizar os recursos adicionais necessários para descrição de hardware dentro da sintaxe da Ada, permitindo que o programa ADA compilado aja como um simulador funcional. No contexto particular para descrição de hardware. é como uma linguagem fonte para um hardware compilador. Regras são apresentadas para traduzir um circuito descrito no subconjunto de ADA em um gráfico de controle/ fluxo de dados (CDFG), usando um formulário de nível intermediário.

Por fim, foram encontrados muitos trabalhos relacionados ao uso de ADA como linguagem de descrição de *hardware*. A possibilidade de ADA como HDL,

facilitar a implementação de técnicas de tolerância a falhas em FPGA é estudada em alguns trabalhos[46].

4.4 ADA com HW/SW Co-design

Na pesquisa sobre ADA como uma linguagem de descrição de hardware, encontrou-se o uso de ADA na metodologia de *co-design* de *hardware/software* [47]. No artigo [47], ADA é usada como linguagem de *co-design* (CDL), onde é utilizado subconjunto da linguagem ADA ISO/ANSI. O interessante é que o artigo informa que a CDL existe desde de 1980 quando VHDL iniciou e está contudo no IEEE 1076VHDL-1993 com apenas pequenas diferenças. ADA é usada como HDL usando expressões booleanas ou implementações alternativas a niveís de porta, e testes de programas e ferramentas utilitárias quando necessário. O artigo também se refere a deficiência de VHDL, que seria devido a confiar apenas nas expressões booleanas, uma abstração, para descrever o desenvolvimento de chips. Ele defende no artigo que a CDL de ADA usada em VHDL, tem mais expressividade para descrever componentes, além de ser muito mais simples. Ele apresenta a evolução de VHDL junto com ADA mostrando o paralelo sobre a CDL. Por fim mostra os exemplos comparativo da sintaxe de VHDL e CDL/ADA.

4.5 ADA com Redes de Petri

Redes de Petri são métodos formais que modelam eficientemente sistemas com concorrência, entre eles os sistemas distribuídos, caracterizados por concorrência, sincronização, exclusão mútua e conflito. Estas redes incorporam a noção de estado e regras para mudança de estados, além do conceito do princípio da localidade, o que as permite capturar características estáticas e dinâmicas de um sistema de tempo real. Como a linguagem de programação ADA tem concorrência intrínseca nela, as ***tasks*** ADA deve ter seu comportamento capturado pela modelagem de redes de petri.

O artigo [48] mostra o uso de redes de petri para detecção de deadlocks em ***tasks*** ADA, fazendo a tradução entre ADA e redes de Petri para análise as propriedades desta, com isso detectando deadlocks.

Um outro artigo [49], mostrar as abstrações do desenvolvimento que decorre da interação entre ***tasks*** ADA, onde se pode analisar sua corretude lógica usando redes de petri.

4.6 ADA com UML

O aumento da complexidade dos sistemas em tempo real, concorrente e embarcado, levam a um maior cuidado para a construção de modelos adequados de software e analisar esses projetos antes do desenvolvimento do código. O uso da modelagem desses sistemas usando UML (*Unified Modeling Language*) versão 2.0, pode ser vista nesse artigo [50], que apresentar as diretrizes para essa modelagem. No final, ele apresenta como proceder a partir de um modelo UML a codificação em ADA 2005.

Como boa prática de engenharia de *software,* o artigo [51] apresenta o uso de UML com ADA, mostrando que é necessário usar regras para mapear os modelos UML em ADA, mas todos os desenvolvedores terão que seguir o mesmo conjunto de regras de mapeamento. Isso também implica em algumas considerações sobre geração de código ADA através do modelo UML.

Capítulo 5
Experimentação e Resultados

Este capítulo apresenta a experimentação e os resultados obtidos.

5.1 Caracterização do experimento

Foi realizado um experimento do uso de ADA na implementação de três técnicas de tolerância a falhas. O experimento consistiu na execução de três algoritmos de ordenação: bubblesort, mergesort e quicksort. Estes receberam como entrada, uma mesma estrutuda de dados do tipo array de Strings, sendo executados em paralelo, e na sua saída, injetada uma falha. O erro provocado pela falha, alterará os bits s última string do array ordenado, dentro de uma probabilidade de 10% de erro de uma range de 256 possibilidades, gerado por um algoritmo de números aleatórios já implementado como uma unit library de ADA 95, o package "Ada.Numerics.Discrete_Random", que foi visto no capítulo 3 deste trabalho.

Para o algoritmo bubblesort ocorrendo o erro maior que 0% e menor ou igual do que 10%, se altera o bit menos significativo.

Para o algoritmo quicksort ocorrendo:

1. O erro maior que 0% e menor ou igual do que 5% se altera o segundo bit menos significativo.
2. O erro maior que 5% e menor ou igual do que 10% se altera o bit menos significativo.

Para o algoritmos insertionsort ocorrendo:

1. O erro maior que 0% e menor ou igual do que 3,3% se altera o terceiro bit menos significativo.

2. O erro maior que 3,3% e menor ou igual do que 6,6% se altera o segundo bit menos significativo.

3. O erro maior que 6,6% e menor ou igual do que 10% se altera o bit menos significativo.

O experimento foi executado para cada mecanismo de tolerância a falhas durante 1 minuto de cada iteração do experimento, tendo um total de 30 iterações, que gerou resultados para ser usado na estatística do experimento.

O ambiente computacional usado na experimentação, foi um computador com arquitetura do processador, Intel Pentium Dual Core T2390 1.86GHz, e quantidade de memória RAM de 2Gb; sistema operacional GNU/Linux na versão do *kernel* 2.6.35-27-generic na distribuição Ubuntu 10.10. Como ambiente de desenvolvimento foi utilizado o GNAT Programming Studio, da empresa Ada Core, e sua instalação já se encontra disponível nas fontes de instalação de pacotes do Ubuntu.

5.1.1 Consensus Recovery Block

O experimento do mecanismo de tolerância a falhas Consensus Recovery Block foi realizado, em versão procedural e paralela. No algoritmo paralelo, para a sincronização entre a NVP e o Recovery Block, foi implementado o mecanismo de semáforos em Tasks ADA, sendo de forma simples e boa legibilidade.

5.1.2 NVP

Aplicou-se um teste de replicação para analisar a validade dos resultados, no caso os CRC's das saídas dos algoritmos de ordenação. O teste verifica se a existência de dois ou mais valore similares, através da votação por maioria, os resultados são "válidos". Os algoritmos seguiram uma especificação inicial, que foi

ordenar uma lista de um mesmo tipo. A comparação se realizou entre valores numéricos de natureza contínua, no caso os CRC's.

5.1.3 Recovery Block

O bloco de recuperação por retrocesso, usou o algoritmo implementado no teste de aceitação [52] seguindo o exemplo no artigo. O bloco de recuperação por avanço usa o mecanismo de tratamento de exceções de ADA.

5.2 Dificuldades durante o desenvolvimento

Algumas dificuldades inicialmente ocorreram, na implementação devido ao entendimento de visibilidade e escopo em ADA, no uso da tipagem forte de ADA, e no uso do tipo String. O tipo String ele é um array "*unconstrained*" de Character [1 , pág 54], ou seja, é muma string de tamanho fixo, então não é possível usar a operação de atribuição que não esteja com valores restrito ao tamanho determinado exatamente. Para utilizar um array de Strings com tamanho variável, usamos o tipo Unbounded_String da biblioteca padrão de ADA. Esse tipo permite flexibilizar o tamanho da representação de uma String.

5.3 Discussão dos Resultados

No resultado do experimento, calculamos o índice de confiabilidade. Definiu-se como medida desse índice o total de sucessos na execução para cada técnica implementada pelo total de iterações realizadas no tempo estipulado de 1 minuto.

Como resultado chegou a conclusão que o maior índice de confiabilidade é do CRB paralelo e procedural, que na média das 30 amostras utilizadas teve os seguintes resultados respectivamente: 0,980398306 e 0,973444776. Com um resultado muito próximo teve o bloco de recuperação que na média obteve 0,958501394. O NVP teve índice de confiabilidade de 0,9658526455.

Fazendo uma análise mais detalhada, avalia-se que o índice de confiabilidade tanto do CRB como do NVP, têm que leva em consideração o problemas dos falsos positivos do mecanismo do NVP. Sendo assim, a média do índice de confiabilidade do CRB paralelo, do CRB procedural e do NVP, deve ser a média do índice de confiabilidade obtido menos a média da freqüência relativa da taxa de falsos positivos, que é calculado pelo números de falsos positivos dividido pelo total de iterações.

Outros dados estatísticos obtidos foram o desvio padrão e o coeficiente de variação dos dados obtidos. Nas sub-seções a seguir é feita a análise de cada mecanismo com detalhes. O Apêndice A apresenta as tabelas com os dados completos do experimento.

5.3.1 CRB Procedural

A avaliação do algoritmo baseia-se nos dados na tabela 5.1. Fazendo a diferença entre a média do índice de confiabilidade obtido e a freqüência relativa dos resultados falsos positivos, o índice de confiabilidade resultante é 0,90611742, sendo menor do que o índice de confiabilidade do bloco de recuperação. Na tabela abaixo mais dados estatísticos são apresentados.

Tabela 5.1 Resumo dos dados do CRB procedural

	Índice de confiabilidade	Freqüência Relativa dos resultados falsos positivos	Freqüência Relativa dos resultados falhos
Média	0,973444776	0,067327356	0,02655522
Desvio padrão	0,000593575	0,001083967	0,00059358
Variância	3,52332e-07	1,17498e-06	3,5233e-07

5.3.2 CRB paralelo

A avaliação do algoritmo baseia-se nos dados na tabela 5.2. Fazendo a diferença entre a média do índice de confiabilidade obtido e a freqüência relativa dos resultados falsos positivos, o índice de confiabilidade resultante é 0,916564616, sendo menor do que o índice de confiabilidade do bloco de recuperação Na tabela abaixo mais dados estatísticos são apresentados.

Tabela 5.2 Resumo dos dados do CRB paralelo

	Índice de confiabilidade	Freqüência Relativa dos resultados falsos positivos	Freqüência Relativa dos resultados falhos
Média	0,980398306	0,06383369	0,019601694
Desvio padrão	0,000605491	0,001168349	0,000605491
Variância	3,66619E-07	1,36504E-06	3,66619E-07

5.3.3 NVP

A avaliação do algoritmo baseia-se nos dados na tabela 5.3. Fazendo a diferença entre a média do índice de confiabilidade obtido e a freqüência relativa dos resultados falsos positivos, o índice de confiabilidade resultante é 0,898413874, sendo menor do que o índice de confiabilidade do bloco de recuperação e do CRB. Na tabela abaixo mais dados estatísticos são apresentados.

Tabela 5.3 Resumo dos dados do NVP

	Índice de confiabilidade	Freqüência Relativa dos resultados falsos positivos	Freqüência Relativa dos resultados falhos
Média	0,965852645	0,067438771	0,034147355
Desvio padrão	0,000767627	0,000837699	0,000767627
Variância	5,89251E-07	7,0174E-07	5,89251E-07

5.3.4 Bloco de recuperação

A avaliação do algoritmo baseia-se nos dados na tabela 5.4. O índice de confiabilidade é mais preciso já que resultados falsos positivos não são obtidos, e o índice de confiabilidade resultante é 0,958501394, sendo maior entre os mecanismos avaliados. Na tabela abaixo mais dados estatísticos são apresentados.

Tabela 5.4 Resumo dos dados do Bloco de Recuperação

	Índice de confiabilidade	Freqüência Relativa dos resultados falhos
Média	0,958501394	0,04149861
Desvio padrão	0,000564341	0,00056434
Variância	3,1848E-07	3,1848E-07

5.4 Conclusões dos resultados

Avaliando o índice de confiabilidade sob o efeito dos resultados de falsos positivos resultantes do mecanismo de votação do NVP, a confiabilidade do CRB é menor do que o bloco de recuperação e maior do que o NVP.

Capítulo 6
Conclusão e Trabalhos Futuros

Como demonstrado durante esse trabalho, através do estudo bibliográfico e da experimentação o uso da linguagem ADA nos mecanismos de tolerância a falhas, foi visto que para sistemas complexos e sistemas críticos de tempo real, a linguagem ADA é a mais adequada. Conclui-se também que a implementação das técnicas de tolerância a falhas é facilitada em ADA devido a seus mecanismos intrínsecos, como ***tasks***, tratamento de exceções, ***unit libray*** para sistemas de tempo real.

Além disso, ao apresentar a linguagem Ada com algumas tecnologias de estado da arte no desenvolvimento de sistemas críticos, dependáveis e em tempo real, ADA mostrasse sempre atualizada como linguagem de programação de alto nível.

6.1 Trabalhos Futuros

Espera-se com esse trabalho impulsionar a formação de um grupo de pesquisa de sistemas dependáveis no âmbito da Escola Politécnica de Pernambuco.

Como estudo inicial, se propõe:

1. Construir como projeto futuro software em ADA, que se reconfigure de forma dinâmica componente de Hardware redundante em FPGA .
2. Analisar para projetos futuros a metodologia de co-design de HW/SW usando ADA para descrever HW e implementar SW embarcado, encontrando um estudo de caso específico de aplicabilidade.

3. Analisar para projetos futuros o reaproveitamento de pedaços de hardware em software, fazendo o FPGA um componente reconfiguŕavel de redundância de Hardware.
4. Definir projetos futuros usando ADA com técnicas de tolerância falhas.
5. Definir projetos futuros usando ADA com técnicas de computação inteligente para usar técnicas de previsão de falhas.
6. Definir projetos futuros usando ADA como HHL e HDL para prover tolerância a falhas em FPGA;.
7. Estudar a computação autonômica na visão da dependabilidade.

Bibliografia

[1] Weber, T. *Tolerância a falhas: conceitos e exemplos* - Programa de Pós-Graduação em Computação - Instituto de Informática – UFRGS.

[2] Laprie, J. C. *Dependable Computing: Concepts, Limits, Challenges. In: 25th IEEE International Symposium on Fault-Tolerant Computing,Pasadena, California, USA, June 27-30, 1995, Special Issue, p.42-54.*

[3] Avizienis, A.;Laprie, J. C.; RandellB.; Landwehr, C. Basic Concepts and Taxonomy of Dependable and Secure Computing . IEEE Transactions on dependable and secure computing, vol. 1, n. 1. January-March 2004 .

[4] *Ada Reference Manual*, ISO/IEC 8652:2007(E) Ed. 3.

[5] Lundqvist, K.; Asplund, L. *Formal Model of a Run-Time Kernel for Ravenscar. In: Sixth International Conference on Real-Time Computing Systems and Applications* — RTCSA'99A.

[6] Kienzle, J.; Romanovsky, A. *On Persistent and Reliable Streaming in Ada. In: International Conference on Reliable Software Technologies – ADA EUROPE 2000.*

[7] Amey, P. *A Language for Systems not Just Software. ACM SIGAda Ada Letters December 2001, v.21, n.4, p.3-11.*

[8] Morris, D.S.*; Stevens Inst. of Technol., Hoboken, NJ. Packaging fault-tolerant software with Ada. In: Third International IEEE Conference on Ada Applications and Environments, 1988.*

[9] Burns, A.; Wellings, A.J.; Koelmans, A.M. ; Koutny, M.; Romanovsky, A.; Yakovlev, A. *On Developing and Verifying Design Abstractions for Refiable*

Concurrent Programming in Ada. In: IRTAW '00 Proceedings of the 10th international workshop on Real-time Ada workshop.

[10] Wong, S. Hardware/Software Co-Design Language Compatible with VHDL. In: Wescon/98, 1998.

[11] Mahani,N.; Mokri,P.; Sedghi, M.; Navabi, Z. *SystemAda: An Ada Based System-Level Hardware Description Language.* ACM SIGAda Ada Letters August 2009, v.29, n.2, p.15-19.

[12] Ward, M.; Audsley, N. C. *Hardware Compilation of Sequential Ada. In: CASES '01 Proceedings of the 2001 international conference on Compilers, architecture, and synthesis for embedded systems.*

[13] Lapping, A. *Model Driven Development with Ada.* ACM SIGAda Ada Letters December 2004, v.24, n.4, p.19-22.

[14] Gregertsen, K.N.; Skavhaug, A. *A real-time framework for Ada 2005 and the Ravenscar profile.* In: 2009 35th *Euromicro Conference on Software Engineering and Advanced Applications.*

[15] Loseby,C.; Chapin, P.;Brandon, C. *Use of SPARK in a Resource Constrained Embedded System.* ACM SIGAda Ada Letters - SIGAda '09 December 2009, v.29, n.3, p.87-90.

[16] Tardieu, S.; Polti, A. *Complementing Ada with Other Programming Languages.* ACM SIGAda Ada Letters - SIGAda '09 December 2009, v.29, n.3, p.105-114

[17] Carlisle, M.C.; Baird III, L.C. *Timing neural networks in C and ada.* ACM SIGAda Ada Letters - SIGAda '07' December 2007, v.27, n.3, p.71-74.

[18] Scott, R. K.; Gault,J.W.; McAllister,D. F. . *Fault-Tolerant Software Reliability Modeling*, IEEE *Transactions on Software Engineering*, Vol. SE-13, n. 5, m*ay* 1987, p. 582 – 592.

[19] Avizienis, A. *The N-Version Approach to Fault-Tolerant Software.* IEEE *Transactions on Software Engineering*, Vol. SE-I 1, n. 12, *december* 1985 .

[20] Fernandes. S. M. M. Tese apresentada para a obtenção do título de Doutor em Ciências da Computação pela Universidade Federal de Pernambuco - Centro de Informática. *Avaliação de Dependabilidade de Sistemas com Mecanismos Tolerantes a Falha: Desenvolvimento de um Método Híbrido Baseado em EDSPN e Diagrama de Blocos, p.54-58.*

[21] Pradhan, D. *Fault-Tolerant Computer Design.* Englewood Cliffs,Prentice-Hall, 1996.

[22] A*nderson*, T.; Lee, P. A. *Fault tolerance - principles and practice*. Englewood Cliffs, Prentice-Hall, 1981 .

[23] Avizienis, A. e Kelly,J.P.J. " *Fault Tolerance by Design Diversity: Concepts and Experiments*," Computer, vol. 17, no. 8, pp. 67-80, Aug. 1984.

[24] Pradhan, D. K. . *An Introduction To Desigin and Analysis of Fault-Tolerant Systems. In: Faut-Tolerant Computer System Design*. New Jersey: Prentice Hall PTR, 1996. p. 1-8

[25] Lala, P. K. . *Fault-Tolerant Design. In: Self-Checking and Fault-Tolerant Digital Design*. San Francisco: Academic Press, 2001. p. 161-201.

[26] Randell, B.: '*System structure for software fault tolerance*', IEEE Trans. Soft. Eng., 1975, SE-1, (2), pp.220-232

[27] Mccormick,J.W.; Singhoff,F.; Hugues, J. *In book*: *Building Parallel, Embedded, and Real-time Applications with ADA. Cambridge University Press.*

[28] Dijkstra, E. 1968. *Cooperating sequential processes*. Pages 43–112 of: Genuys, F. (ed), Programming Languages. Academic Press. Reprinted from the original Technological University, Eindhoven, The Netherlands, September 1965.

[29] Gregertsen, K.N.; Skavhaug, A. *A real-time framework for Ada 2005 and the Ravenscar profile.* In: 2009 35th *Euromicro Conference on Software Engineering and Advanced Applications.*

[30] *Burns, A. "The Ravenscar Profile," Ada Letters, vol. 19, no. 4, pp. 49-52, 1999.*

[31] *Burns,A.; Dobbing,B. e Vardanega, T. Guide for the use of the Ada Ravenscar Profile in high integrity systems. Technical Report YCS-2003-348, University of York, 2003.*

[32] Jennings , T. SPARK – *The Libre Language and Toolset for High-Assurance Software Engineering . Tutorial in:* SIGAda'09, *November* 1–5, 2009, *Saint Petersburg, Florida*, *USA* .

[33] Chapman , R. *Industrial Experience with SPARK . Ada Letters, December* 2000 . vol 20, n 4.

[34] Dodd, P. E.; Massengill, L. W. *Basic Mechanisms and Modeling of Single-Event Upset in Digital Microelectronics*. IEEE *Transactions on Nuclear Science*, v. 50, n. 3, p. 583–602, June, 2003.

[35] ACTEL. Actel Corporation. RTSX-SU RadTolerant FPGAs (UMC). Mar. 2006.

[36] Maxfield, C. *The Design Warrior's guide to FPGAs Devices, Tools and Flows.* Amsterdam: Elsevier, 2004. MAXFIELD, C. FPGAs, RTSX-SU

[37] Roosta, R. *A Comparison of Radiation-Hard and Radiation - Tolerant FPGAs for Space Applications. NASA Electronic Parts and Packaging Program. Dec.* 2004.

[38] Carmichael, C. *Triple Module Redundancy Design Techniques for Virtex FPGA. Xilinx Application Notes 197. San Jose, USA:* Xilinx*, 2001*

[39] Nicolaidis, M. *Time Redundancy Based Soft-Error Tolerance to Rescue Nanometer Technologies. In: IEEE VLSI TEST SYMPOSIUM, 17., 1999. Proceedings... Los Alamitos: IEEE Computer Society,* 1999. p. 86-94.

[40] Lima, F.; Carro, L; Reis, R. techniques for reconfigurable logic applications: designing fault tolerant systems into sram-based fpgas. In: international design automation conference, dac, 2003. proceedings... New York: ACM, 2003. p. 650-655.

[41] Carmichael, C.; Caffrey, M.; Salazar, A. *Correcting Single-Event Upsets Through Virtex Partial Configuration*. Xilinx Application Notes 216. San Jose, USA: Xilinx, 2000.

[42] Deschamps, J-P.; Bioul, G. J. A.; Sutter, G. D. S*ynthesis of Arithmetic Circuits FPGA, ASIC anda Embedded Systems*. New Jersey: John Wiley e Sons, 2006

[43] Lima, F. *Designing Single Event Upset Mitigation Techniques for Large SRAM- based FPGA Components.* Tese de Doutorado. Porto Alegre: PPGC da UFRGS, 2003.

[44] Lima, F.; Kastensmidt, F.G.; *Designing Single Event Upset Mitigation Techniques for Large SRAM-based FPGA Components.* Tese de Doutorado. Porto Alegre: PPGC da UFRGS, 2003

[45] Ghosh , S. *Using ADA as an HDL*. *IEEE Design & Test of Computers*, 1988.

[46] Ashenden, P.J. *Reuse Through Genericity in SUAVE. VIUF '97 Proceedings of the 1997 VHDL International User's Forum (VIUF '97)*

[47] *Adaware , Dr. S. W. Hardware/Software Co-Design Language Compatible with VHDL (IEEE -1998).*

[48] *Shenker, B. ; Murata,T.; Shatz,S-M. Use of petri net invariants to detect static deadlocks in ADA program. Journal IEEE Transactions on Software Engineering, vol 15, issue 3, March 1989.*

[49] *Burns,A.; Wellings,A.J.; Koelmans,A.M.; Koutny,M.; Romanovsky,A.;Yakovlev ,A. On Developing and Verifying Design Abstractions for Refiable Concurrent Programming in Ada. IRTAW '00 Proceedings of the 10th international workshop on Real-time Ada workshop.*

[50] *Pettit,R. G. Designing real-time, concurrent, and embedded software systems using UML and Ada. SIGAda'10, October 24–28, 2010, Fairfax, Florida, USA.*

[51] *Lapping, A. Model Driven Development with Ada.SIGAda'04, November 14-18, 2004, Atlanta, Georgia, USA*

[52] *Romanovsky, A.; e Strigin, L. Backward error recovery via conversations in Ada*.

[53] Jansch-Porto,I.E.S; Weber, T.S. Recuperação em Sistemas Distribuídos. In: XVI Jornada de Atualização em Informática, XVII Congresso da SBC, Brasília, 2-8 Agosto de 1997. anais. P. 263-310.

[54] http://www.gnu.org/software/gnat acessado no dia 22/12/2011 às 20:32 horas.

[55] http://www.adacore.com/home/products/gnatpro/ acessado no dia 22/12/2011 ás 20:40 horas.

[56] http://www.ddci.com/DDC-I_ada_compiler_system.php acessado no dia 22/12/2011 às 21:00 horas.

[57] http://www-01.ibm.com/software/awdtools/developer/ada/enterprise/ acessado no dia 22/12/2011 às 21:20 horas.

[58] http://www.rrsoftware.com/html/prodinf/janus95/j-ada95.htm acessado no dia 22/12/2011 às 21:33 horas.

[59] http://www.irvine.com/ acessado no dia 22/12/2011 às 21:45 horas.

[60] http://www.ghs.com/products/ada_optimizing_compilers.html acessado no dia 22/12/2011 às 21:50 horas.

[61] http://www.sofcheck.com/products/adamagic.html acessado no dia 22/12/2011 às 22:03 horas.

[62] http://www.atego.com/products/aonix-objectada/ acessado no dia 22/12/2011 às 22:10 horas.

[63] http://www.ocsystems.com/prod_powerada.html acessado no dia 22/12/2011 às 22:20 horas.

Apêndice A Tabelas dos resultados da Experimentação

Experimentação do Recovery Block implementado em ADA					
N° de vezes	**Sucesso**	**Resultados Falhos**	**Total Iterações**	**Indice de confiabilidade**	**FR Falhos**
1	585768	25700	611468	0,9579700001	0,0420299999
2	581926	25625	607551	0,9578224709	0,0421775291
3	582267	25556	607823	0,9579548651	0,0420451349
4	582932	24832	607764	0,9591420354	0,0408579646
5	581521	25210	606731	0,9584494611	0,0415505389
6	582636	24092	606728	0,9602919265	0,0397080735
7	582982	25168	608150	0,9586154732	0,0413845268
8	583193	24924	608117	0,959014466	0,040985534
9	582254	25369	607623	0,9582487826	0,0417512174
10	580023	25777	605800	0,9574496534	0,0425503466
11	580124	25055	605179	0,9585990261	0,0414009739
12	573567	25061	598628	0,9581359375	0,0418640625
13	570364	24569	594933	0,9587029128	0,0412970872
14	571740	24276	596016	0,9592695498	0,0407304502
15	577075	24670	601745	0,9590025675	0,0409974325
16	576060	25078	601138	0,9582824576	0,0417175424
17	567215	25005	592220	0,9577775151	0,0422224849
18	580353	25161	605514	0,9584468732	0,0415531268
19	578465	25563	604028	0,9576791142	0,0423208858
20	580364	25061	605425	0,958605938	0,041394062
21	578990	25241	604231	0,9582262413	0,0417737587
22	580072	24908	604980	0,958828391	0,041171609
23	578530	24965	603495	0,9586326316	0,0413673684
24	580083	24773	604856	0,9590431442	0,0409568558
25	578101	25038	603139	0,9584871812	0,0415128188
26	578449	25007	603456	0,958560359	0,041439641
27	576121	25380	601501	0,9578055564	0,0421944436
28	578524	24810	603334	0,9588784985	0,0411215015
29	577603	25048	602651	0,9584369726	0,0415630274
30	578738	24943	603681	0,9586818204	0,0413181796
Média	578868	25062,1666666667	603930,166666667	0,9585013941	0,0414986059
Desvio padrao	3966,9431304217	375,4839617821	4109,6478120259	0,0005643407	0,0005643407
Variancia	15736637,8	140988,205555556	16889205,1388889	3,1848047356557E-007	3,18480E-007
Máximo Valor	585768	25777	611468	0,9602919265	0,0425503466
Mínimo Valor	567215	24092	592220	0,9574496534	0,0397080735

Experimentação do CRB Parallel implementado em ADA									
N° de vezes	**Corretos**	**Falsos Positivos**	**Resultados Falhos**	**CRB Correto**	**Falhas lançam exceção**	**Total Iterações**	**Indice de confiabilidade**	**FR Falsos positivos**	**FR Falhos**
1	57466	4063	2433	1192	1241	63962	0,980597855	0,0635220912	0,019402145
2	58073	4024	2391	1138	1253	64488	0,9805700285	0,0623992061	0,0194299715
3	57551	3954	2500	1109	1391	64005	0,9782673229	0,0617764237	0,0217326771
4	56716	4020	2381	1135	1246	63117	0,9802588843	0,0636912401	0,0197411157
5	58008	4189	2391	1112	1279	64588	0,9801975599	0,064857249	0,0198024401
6	57444	4103	2388	1150	1238	63935	0,980636584	0,0641745523	0,019363416
7	57517	4110	2423	1155	1268	64050	0,9802029664	0,0641686183	0,0197970336
8	57092	4111	2359	1186	1173	63562	0,9815455775	0,0646770083	0,0184544225
9	56686	3961	2423	1164	1259	63070	0,980038053	0,0628032345	0,019961947
10	57234	4112	2418	1206	1212	63764	0,9809924095	0,0644877988	0,0190075905
11	57541	3995	2488	1185	1303	64024	0,9796482569	0,0623984756	0,0203517431
12	57749	4042	2466	1177	1289	64257	0,9799399287	0,0629036525	0,0200600713
13	57392	4191	2387	1172	1215	63970	0,9810067219	0,0655150852	0,0189932781
14	57733	4190	2414	1154	1260	64337	0,980415624	0,0651258218	0,019584376
15	57381	4114	2425	1136	1289	63920	0,9798341677	0,0643617021	0,0201658323
16	57977	4313	2400	1173	1227	64690	0,9810326171	0,0666718194	0,0189673829
17	57153	4187	2511	1227	1284	63851	0,979890683	0,0655745407	0,020109317
18	57789	3998	2371	1109	1262	64158	0,9803298108	0,0623149101	0,0196701892
19	57478	4061	2308	1074	1234	63847	0,9806725453	0,0636051811	0,0193274547
20	57128	4155	2435	1214	1221	63718	0,98083744	0,0652092031	0,01916256
21	57478	4071	2378	1122	1256	63927	0,9803525897	0,0636820123	0,0196474103
22	57694	4117	2509	1206	1303	64320	0,9797419154	0,0640080846	0,0202580846
23	57171	4016	2397	1207	1190	63584	0,9812845999	0,0631605435	0,0187154001
24	57341	4097	2396	1146	1250	63834	0,9804179591	0,0641820973	0,0195820409
25	57657	3956	2503	1226	1277	64116	0,9800829746	0,0617006675	0,0199170254
26	57352	4010	2423	1214	1209	63785	0,9810457004	0,0628674453	0,0189542996
27	56916	4110	2377	1128	1249	63403	0,9803006167	0,0648234311	0,0196993833
28	57726	4107	2399	1173	1226	64232	0,9809129406	0,0639400922	0,0190870594
29	57783	4080	2425	1161	1264	64288	0,9803384769	0,0634644102	0,0196615231
30	58071	4066	2460	1204	1256	64597	0,9805563726	0,0629440996	0,0194436274
Média	57476,5666666667	4084,1	2419,3	1165,16666667	1254,1333333333	63979,9666666667	0,9803983061	0,0638336899	0,0196016939
Desvio padrao	356,9385272315	80,1004993742	47,5633962903	38,634253656	40,0014444184	387,4476724525	0,0006054907	0,0011683489	0,0006054907
Variancia	127405,112222222	6416,09	2262,2766666667	1492,60555556	1600,1155555556	150115,698888889	3,66618936370685E-007	0,000001365	3,6661894E-007
Máximo Valor	58073	4313	2511	1227	1391	64690	0,9815455775	0,0666718194	0,0217326771
Mínimo Valor	56686	3954	2308	1074	1173	63070	0,9782673229	0,0617006675	0,0184544225

Experimentação do CRB Procedural implementado em ADA									
N° de vezes	Corretos	Falsos Positivos	Resultados Falhos	CRB Correto	Falhas lançam exceção	Total Iterações	Indice de confiabilidade	FR Falsos positivos	FR Falhos
1	149065	11335	5626	1307	4319	166026	0,9739860022	0,0682724393	0,0260139978
2	153086	11575	6082	1347	4735	170743	0,9722682628	0,0677919446	0,0277317372
3	164558	12052	6095	1319	4776	182705	0,9738595003	0,0659642593	0,0261404997
4	165973	12931	6349	1321	5028	185253	0,9728587391	0,0698018386	0,0271412609
5	162580	11866	5990	1247	4743	180436	0,9737136713	0,0657629298	0,0262863287
6	164912	12079	6293	1295	4998	183284	0,9727308439	0,0659031885	0,0272691561
7	165874	12303	6402	1338	5064	184579	0,9725645929	0,0666543865	0,0274354071
8	165697	12575	6327	1355	4972	184599	0,9730659429	0,068120629	0,0269340571
9	167013	12391	6136	1310	4826	185540	0,9739894362	0,0667834429	0,0260105638
10	162746	11724	6042	1316	4726	180512	0,9738189151	0,0649485907	0,0261810849
11	165969	12269	6247	1264	4983	184485	0,972989674	0,0665040518	0,027010326
12	162756	12169	6142	1325	4817	181067	0,973396588	0,0672071664	0,026603412
13	164150	12605	6230	1355	4875	182985	0,973358472	0,0688854278	0,026641528
14	144400	10965	5673	1334	4339	161038	0,9730560489	0,0680895192	0,0269439511
15	161123	12109	6452	1391	5061	179684	0,9718338862	0,06739053	0,0281661138
16	163221	12092	5960	1266	4694	181273	0,9741053549	0,066706018	0,0258946451
17	159948	12037	6075	1333	4742	178060	0,9733685275	0,0676008087	0,0266314725
18	163693	12454	6159	1321	4838	182306	0,9734622009	0,0683137143	0,0265377991
19	140260	10478	5441	1306	4135	156179	0,9735239693	0,0670896856	0,0264760307
20	165787	12089	6076	1299	4777	183952	0,974031269	0,0657182308	0,025968731
21	164584	12596	6115	1341	4774	183295	0,9739545541	0,0687198232	0,0260454459
22	148635	11196	5524	1338	4186	165355	0,9746847691	0,0677088688	0,0253152309
23	164654	12365	6247	1331	4916	183266	0,9731756027	0,0674702345	0,0268243973
24	162584	12286	6074	1296	4778	180944	0,9735940401	0,0678994606	0,0264059599
25	156122	11551	5969	1338	4631	173642	0,9733301851	0,0665219244	0,0266698149
26	154371	11881	5852	1333	4519	172104	0,9737426207	0,06903384	0,0262573793
27	154943	11503	5883	1367	4516	172329	0,973794312	0,0667502278	0,026205688
28	148028	11053	5613	1363	4250	164694	0,9741945669	0,0671123417	0,0258054331
29	164054	12245	6179	1323	4856	182478	0,9733885729	0,0671039797	0,0266114271
30	158874	12029	6017	1329	4688	176920	0,9735021479	0,0679911825	0,0264978521
Média	159655,333333333	11960,1	6042,3333333333	1323,6	4718,7333333333	177657,766666667	0,9734447756	0,0673273562	0,0265552244
Desvio padrao	7161,972881515	543,0699371781	252,9435158731	30,3200703605	252,6338501116	7889,5048838455	0,0005935755	0,0010839667	0,0005935755
Variancia	51293855,5555556	294924,956666667	63980,4222222222	919,3066666667	63823,8622222222	62244287,3122222	3,52331867218453E-007	0,000001175	3,52332E-007
Máximo Valor	167013	12931	6452	1391	5064	185540	0,9746847691	0,0698018386	0,0281661138
Mínimo Valor	140260	10478	5441	1247	4135	156179	0,9718338862	0,0649485907	0,0253152309

Experimentação do NVP implementado em ADA							
N° de vezes	**Corretos**	**Falsos Positivos**	**Resultados Falhos**	**Total Iterações**	**Indice de confiabilidade**	**FR dos falsos positivos**	**FR dos Resultados Falhos**
1	157094	11889	5954	174937	0,9659648902	0,067961609	0,0340351098
2	165615	12693	6454	184762	0,9650685747	0,0686991914	0,0349314253
3	166377	12217	6023	184617	0,9673757021	0,0661748376	0,0326242979
4	166713	12578	6327	185618	0,9659138661	0,0677628247	0,0340861339
5	165662	12337	6090	184089	0,9669181754	0,0670164975	0,0330818246
6	164387	12253	6347	182987	0,9653144759	0,0669610409	0,0346855241
7	143774	10841	5675	160290	0,9645954208	0,067633664	0,0354045792
8	164391	12507	6175	183073	0,9662702856	0,0683170102	0,0337297144
9	165596	12536	6066	184198	0,9670680463	0,0680571993	0,0329319537
10	166029	12326	6550	184905	0,9645764041	0,0666612585	0,0354235959
11	166003	12659	6211	184873	0,9664039638	0,0684740335	0,0335960362
12	166502	12364	6122	184988	0,9669059615	0,0668367678	0,0330940385
13	153503	11512	5969	170984	0,9650903008	0,0673279371	0,0349096992
14	154347	11588	5739	171674	0,9665703601	0,0675000291	0,0334296399
15	165644	12324	6463	184431	0,9649570842	0,0668217382	0,0350429158
16	164408	12164	6300	182872	0,9655496741	0,0665164705	0,0344503259
17	141186	10467	5413	157066	0,9655367807	0,0666407752	0,0344632193
18	161617	11836	6101	179554	0,966021364	0,0659188879	0,033978636
19	164405	12119	6286	182810	0,9656145725	0,0662928724	0,0343854275
20	156987	11831	6195	175013	0,9646026295	0,0676006925	0,0353973705
21	151773	11529	5911	169213	0,9650676957	0,0681330631	0,0349323043
22	154308	11421	5692	171421	0,9667952001	0,0666254426	0,0332047999
23	149654	11500	5588	166742	0,9664871478	0,0689688261	0,0335128522
24	155106	11912	5922	172940	0,9657569099	0,0688793801	0,0342430901
25	163956	12487	6139	182582	0,9663767513	0,0683911886	0,0336232487
26	156159	11906	5823	173888	0,9665129279	0,0684693596	0,0334870721
27	157038	11814	5942	174794	0,9660056981	0,0675881323	0,0339943019
28	157094	11657	6010	174761	0,9656101762	0,0667025252	0,0343898238
29	149425	11149	5750	166324	0,9654289219	0,0670318174	0,0345710781
30	163025	12199	6314	181538	0,9652194031	0,0671980522	0,0347805969
Média	159259,266666667	11953,8333333333	6051,7	177264,8	0,9658526455	0,0674387708	0,0341473545
Desvio padrao	7069,3686938007	531,8020987381	271,3170531561	7813,6673267636	0,0007676271	0,0008376991	0,0007676271
Variancia	49975973,7288889	282813,472222222	73612,9433333333	61053397,0933333	5,89251299798379E-007	7,01739775548756E-007	5,8925129979838E-007
Máximo Valor	166713	12693	6550	185618	0,9673757021	0,0689688261	0,0354235959
Mínimo Valor	141186	10467	5413	157066	0,9645764041	0,0659188879	0,0326242979

Apêndice B

Codificação do experimento

Os códigos serão anexados nas folhas a seguir.

```ada
with Ada.Text_IO; use Ada.Text_IO;
with main;


procedure Gmain is
   use main;
   count : Integer;
   Output_Procedural, Output_Parallel : File_Type;

begin

   count := 0;

   while count < 30 loop
      put("Execucao procedural do experimento do Consensus Recovery Block");
      main.crb_procedural;

      count:= count + 1;

   end loop;

   New_Line;New_Line;New_Line;
   count := 0;

   while count < 30 loop

      put("Execucao paralela do experimento do Consensus Recovery Block");
      main.crb_parallel;

      count:= count + 1;

   end loop;

   New_Line;New_Line;New_Line;
   count := 0;

   while count < 30 loop

      put("Execucao do experimento do mecanismo de N-Programing Versions");
      main.nvp;

      count:= count + 1;

   end loop;

   New_Line;New_Line;New_Line;
   count := 0;

   while count < 30 loop

      put("Execução do experimento do mecanismo de Recovery Block");
      main.recovery_block;

      count:= count + 1;

   end loop;

end Gmain;
```

```
with sorting;
with Ada.Containers;
with Ada.Numerics.Discrete_Random;
with Ada.Strings;
with Gnat.CRC32;
with Interfaces;
use sorting;


package main is

   -- Procedimento de desmostração do mecanismo do Consensus Recovery Block executado
   sequencialmente
   procedure crb_procedural;
   -- Procedimento de desmostração do mecanismo do Consensus Recovery Block usando tasks
   procedure crb_parallel;
   -- Procedimento de desmostração do mecanismo de N-Programing Versions
   procedure nvp;
   -- Procedimento de desmostração do mecanismo de Bloco de Recuperação
   procedure recovery_block;

   procedure gerar_erro_saida_bs ( LN1 : in out sorting.List_Names_Sorting );

   procedure gerar_erro_saida_qs ( LN1 : in out sorting.List_Names_Sorting );

   procedure gerar_erro_saida_is ( LN1 : in out sorting.List_Names_Sorting );

   function calculo_ckecksum ( LN1 : in List_Names_Sorting ) return Interfaces.Unsigned_32;


end main;
```

```ada
with sorting; -- This could be any Application package
with Ada.Strings.Hash;
with Ada.Containers;
with Ada.Numerics.Discrete_Random;
with Ada.Strings;
with Interfaces;

package body main is

   use sorting;
   use Ada;


   -- Procedimento de desmostração do mecanismo do Consensus Recovery Block executado
   sequencialmente
   procedure crb_procedural is separate;

   -- Procedimento de desmostração do mecanismo do Consensus Recovery Block usando tasks
   procedure crb_parallel is separate;

    -- Procedimento de desmostração do mecanismo de N-Programing Versions
   procedure nvp is separate;

   -- Procedimento de desmostração do mecanismo de Bloco de Recuperação
   procedure recovery_block is separate;


   -- Procedimento que gera um erro no resultado da saí-da dos algoritmo de ordenação
   Bubblesort.
   procedure gerar_erro_saida_bs ( LN1 : in out sorting.List_Names_Sorting ) is separate;


   -- Procedimento que gera um erro no resultado da saí-da dos algoritmo de ordenação
   Quicksort.
   procedure gerar_erro_saida_qs ( LN1 : in out sorting.List_Names_Sorting ) is separate;

   -- Procedimento que gera um erro no resultado da saí-da dos algoritmo de ordenação
   Quicksort.
   procedure gerar_erro_saida_is ( LN1 : in out sorting.List_Names_Sorting ) is separate;

   -- Função que verifica os erros na saida e calcula o checksum
   function calculo_ckecksum ( LN1 : in sorting.List_Names_Sorting ) return Interfaces.
   Unsigned_32 is separate;

end main;
```

```
with Ada.Text_IO;
with Ada.Integer_Text_IO;
with Gnat.CRC32;
with Interfaces;
with Ada.Strings.Unbounded;
with Ada.Exceptions;
with Ada.Text_IO.Unbounded_IO;
with Ada.IO_Exceptions;
with Ada.Integer_Text_IO;
with Ada.Real_Time; use Ada.Real_Time;


separate ( main )
procedure crb_procedural is
   use sorting;
   use Ada;
   use Ada.Text_IO;
   use Gnat.CRC32;
   use Ada.Strings.Unbounded;
   use Interfaces;
   use Ada.Text_IO.Unbounded_IO;
   use Ada.Integer_Text_IO;

   package CRC_IO is new Ada.Text_IO.Modular_IO(Interfaces.Unsigned_32);


   List : List_Names_Sorting := ( Ada.Strings.Unbounded.To_Unbounded_String("Marcos Aurelio"),
                                  Ada.Strings.Unbounded.To_Unbounded_String("Marcos Arnaldo"),
                                  Ada.Strings.Unbounded.To_Unbounded_String("Marcela
                                  Cristina"),
                                  Ada.Strings.Unbounded.To_Unbounded_String("Zuleica Florez"),
                                  Ada.Strings.Unbounded.To_Unbounded_String("Roberta Maria"),
                                  Ada.Strings.Unbounded.To_Unbounded_String("Adriana Celia"),
                                  Ada.Strings.Unbounded.To_Unbounded_String("Orlando Silva
                                  Moraes Sales"),
                                  Ada.Strings.Unbounded.To_Unbounded_String("Orlando Silva
                                  Moraes Silva"),
                                  Ada.Strings.Unbounded.To_Unbounded_String("Adriana Celia
                                  Cavalcanti"),
                                 Ada.Strings.Unbounded.To_Unbounded_String("Bruna Adriana"));

   List_NVP_Result, TEMP_LN_B, TEMP_LN_Q, TEMP_LN_I : List_Names_Sorting := List;

   -- CRC's que será utilizado após o resultado de cada ordenação;
   crc_result_bubble, crc_result_quick, crc_result_insertion : Interfaces.Unsigned_32;
   -- CRC usado na checkagem do teste de aceitação
   crc_check : Interfaces.Unsigned_32;
   -- Contador geral para quantizar as vezes que ocorrer o teste de aceitação
   count_geral : Integer;
   -- Contador dos resultados do teste de aceitação com pelos menos dois CRC's
   correspondendo ao original.
   count_corretos : Integer := 0;
   -- Contador dos resultados do teste de aceitação com pelos menos dois CRC's de Falso
   Positivo.
   count_falso_positivo : Integer := 0;
   -- Contador dos resultados do teste de aceitação com pelos menos dois CRC's de Falso
   Positivo.
   count_falso_negativo : Integer := 0;
```

```ada
    -- Contadores de demostração do uso do teste de aceitação
    count_TA_OK : Integer := 0;
    count_TA_FAILURE : Integer := 0;

    -- Variáveis de temporização
    temp_inicial, temp_final : Ada.Real_Time.Time;

   -- Função e procedimentos do Bloco de Recuperação por Retrocesso
    function teste_de_aceitacao ( crc1 : Interfaces.Unsigned_32; crc2 : Interfaces.Unsigned_32
     ) return Boolean is
       teste : Boolean;
    begin

          if ( crc1 = crc2 ) then

             teste := True;
          else

             teste := False;
          end if;

       return teste;
    end  teste_de_aceitacao;


    procedure recovery_block is


          -- Implementação do Teste de Aceitação
             type ALTERNATE_RANGE is range 1..4;
             alternate: ALTERNATE_RANGE := 1;
             -- Variavel do teste de aceitação
             alternatesuccess : Boolean ;
             teste_FAILURE : Boolean := FALSE;

          begin
             loop
                case alternate is
                   when 1 => alternatesuccess := teste_de_aceitacao(crc1 => crc_check,
                                                                     crc2 => crc_result_bubble
                                                                     );
                         teste_FAILURE := FALSE;

                   when 2 => alternatesuccess := teste_de_aceitacao(crc1 => crc_check,
                                                                     crc2 => crc_result_quick );
                             teste_FAILURE := FALSE;

                   when 3 => alternatesuccess := teste_de_aceitacao(crc1 => crc_check,
                                                                     crc2 =>
                                                                     crc_result_insertion );
                             teste_FAILURE := FALSE;

                   when others => alternatesuccess := TRUE;  teste_FAILURE := TRUE;

                end case;

                exit when alternatesuccess;

                alternate := alternate + 1;
```

```
            end loop;

--            if testedeaceitacao = False then

               if teste_FAILURE then

                  count_TA_FAILURE :=  count_TA_FAILURE + 1;

               else
                  count_TA_OK := count_TA_OK + 1;

               end if;


--            end if;


     end recovery_block;

     -- Procedimento de nvp é uma função pois tem que retorna o valor da lista ordenada.
     procedure nvp_votador_exato is

     begin


        if ( crc_result_insertion = crc_result_bubble ) or
          ( ( crc_result_bubble = crc_result_quick ) or (crc_result_quick =
          crc_result_insertion) ) then



           if ( crc_result_bubble = crc_check and crc_result_quick = crc_check ) or
              ( ( crc_result_insertion = crc_check and crc_result_bubble = crc_check ) or
                 ( crc_result_quick = crc_check and crc_result_insertion = crc_check ) ) then



              count_corretos :=  count_corretos + 1;

           else

              count_falso_positivo := count_falso_positivo + 1;

           end if;

        else

           count_falso_negativo := count_falso_negativo + 1;



           -- RollBack aqui
           recovery_block;

        end if;
```

```
   end nvp_votador_exato;



begin

   -- Primeira ordenação onde gerar o CRC Original dos Dados Ordenados.

   sorting.bubblesort( TEMP_LN_B );
   crc_check := calculo_ckecksum ( TEMP_LN_B );



   -- Inicialiação do contador geral do número de vezes que ocorrerá a execução
   count_geral := 0;

   -- Inicialização dos temporizadores inicial e final.
   temp_inicial := Ada.Real_Time.Clock;

   temp_final :=  Ada.Real_Time."+"(Left  => temp_inicial,
                                    Right => Ada.Real_Time.Seconds(S => 60));


   -- Execução em um laço com condição de temporização para termino dentro de 60 segundos.
   -- No Termino do laço gerar as saidas do algoritmo, gravando em um arquivo.
   -- Implementar as tecnicas de tolerância a falhas:
   -- CRB
   -- Colocar os algoritmos de ordenação em concorrência dentro do laço.


   while temp_inicial < temp_final loop


      TEMP_LN_B := List;
      TEMP_LN_Q := List;
      TEMP_LN_I := List;



      declare
       -- Uso de tasks na paralelização das funções de ordenação.

        task type Bubble;
        task body Bubble is
        begin
           sorting.bubblesort( TEMP_LN_B );
           gerar_erro_saida_bs ( TEMP_LN_B );

           crc_result_bubble := calculo_ckecksum ( TEMP_LN_B );
        end Bubble;

        task type Quick ;
        task body Quick is
        begin
           sorting.quicksort( TEMP_LN_Q );
           gerar_erro_saida_qs ( TEMP_LN_Q );
           crc_result_quick := calculo_ckecksum ( TEMP_LN_Q );
        end Quick;
```

```
      task type Insertion;
      task body Insertion is
      begin
         sorting.insertionsort( TEMP_LN_I );
         gerar_erro_saida_is ( TEMP_LN_I );
         crc_result_insertion := calculo_ckecksum ( TEMP_LN_I );

       end Insertion;

      BB : Bubble;
      QQ : Quick;
      II : Insertion;

   begin

      null;

   end;



   -- Mecanismos de diagnósticos de erro.
   -- Teste de consistência pela verificação dos CRC's de cada saída dos algoritmos de
   ordenação
   -- Teste de replicação do mecanismos de NVP.
   -- Implementação do Votador da NVP: Valida 2 ou mais resultados similares.
   nvp_votador_exato;

   count_geral := count_geral + 1;

   temp_inicial := Ada.Real_Time.Clock;

end loop;


New_Line;
put( "Resultados: ");
New_Line;
put("Total de iteracoes realizadas da execucao: ");
put ( Item => count_geral,
     Base =>  10 );

New_Line;

put("Total de resultados Corretos: ");
put ( Item => count_corretos,
     Base =>  10 );

New_Line;
put("Total de Falsos Positivos: ");
put ( Item => count_falso_positivo,
     Base =>  10 );

New_Line;
put("Total de Falsos Negativos: ");
put ( Item => count_falso_negativo,
     Base =>  10 );

New_Line;
```

```
   put("Total vezes que o resultado do teste de aceitacao no CRB teve resultado correto");
   put ( Item => count_TA_OK,
        Base =>  10 );

   New_Line;
   put("Total vezes que o resultado do teste de aceitacao no CRB lancara uma excecao de
Falha!");
   put ( Item => count_TA_FAILURE,
        Base =>  10 );
   New_Line;
   New_Line;

   -- Inicializar o contador geral novamente
   count_geral := 0;


exception


   when Ada.IO_Exceptions.End_Error => put ( "Erro no programa. Finalizando temporizador, e
gravando os dados no arquivo.");
      -- procedimento para gravar os dados finais no arquivo.
      put("Contabilização da execução:");
      put ( Item => count_geral,
            Base =>  10 );

   when Program_Error => Put("Ocorreu um erro em alguma estrutura de controle. ");
      Put_Line("O programa será terminado.");
   when Constraint_Error => Put("Ocorreu um erro na atribuição de uma valor a algum tipo. ");
                           Put_Line("O programa será terminado.");

      raise;

end crb_procedural;
```

```
with Ada.Text_IO;
with Ada.Integer_Text_IO;
with Gnat.CRC32;
with Interfaces;
with Ada.Strings.Unbounded;
with Ada.Exceptions;
with Ada.Text_IO.Unbounded_IO;
with Ada.IO_Exceptions;
with Ada.Integer_Text_IO;
with Ada.Real_Time; use Ada.Real_Time;

separate ( main )
procedure crb_parallel is
   use sorting;
   use Ada;
   use Ada.Text_IO;
   use Gnat.CRC32;
   use Ada.Strings.Unbounded;
   use Interfaces;
   use Ada.Text_IO.Unbounded_IO;
   use Ada.Integer_Text_IO;

   package CRC_IO is new Ada.Text_IO.Modular_IO(Interfaces.Unsigned_32);


   List : List_Names_Sorting := ( Ada.Strings.Unbounded.To_Unbounded_String("Marcos Aurelio"),
                                  Ada.Strings.Unbounded.To_Unbounded_String("Marcos Arnaldo"),
                                  Ada.Strings.Unbounded.To_Unbounded_String("Marcela
                                  Cristina"),
                                  Ada.Strings.Unbounded.To_Unbounded_String("Zuleica Florez"),
                                  Ada.Strings.Unbounded.To_Unbounded_String("Roberta Maria"),
                                  Ada.Strings.Unbounded.To_Unbounded_String("Adriana Celia"),
                                  Ada.Strings.Unbounded.To_Unbounded_String("Orlando Silva
                                  Moraes Sales"),
                                  Ada.Strings.Unbounded.To_Unbounded_String("Orlando Silva
                                  Moraes Silva"),
                                  Ada.Strings.Unbounded.To_Unbounded_String("Adriana Celia
                                  Cavalcanti"),
                                  Ada.Strings.Unbounded.To_Unbounded_String("Bruna Adriana"));

   List_NVP_Result, TEMP_LN_B, TEMP_LN_Q, TEMP_LN_I : List_Names_Sorting := List;

   -- CRC's que será utilizado após o resultado de cada ordenação;
   crc_result_bubble, crc_result_quick, crc_result_insertion : Interfaces.Unsigned_32;
   -- CRC usado na checkagem do teste de aceitação
   crc_check : Interfaces.Unsigned_32;
   -- Contador geral para quantizar as vezes que ocorrer o teste de aceitação
   count_geral : Integer;
   -- Contador dos resultados do teste de aceitação com pelos menos dois CRC's
   correspondendo ao original.
   count_corretos : Integer := 0;
   -- Contador dos resultados do teste de aceitação com pelos menos dois CRC's de Falso
   Positivo.
   count_falso_positivo : Integer := 0;
   -- Contador dos resultados do teste de aceitação com pelos menos dois CRC's de Falso
   Positivo.
   count_falso_negativo : Integer := 0;

   -- Contadores de demostração do uso do teste de aceitação
```

```
count_TA_OK : Integer := 0;
count_TA_FAILURE : Integer := 0;

-- Variáveis de temporização
temp_inicial, temp_final : Ada.Real_Time.Time;

-- Variavel do teste de aceitação
protected testedeaceitacao is
   procedure Set ( Teste_B : Boolean );
   function Get return Boolean;
private
 teste : Boolean;
end testedeaceitacao;

protected body testedeaceitacao is
   procedure Set ( Teste_B : Boolean ) is

   begin
      teste := Teste_B;

   end Set;
   function Get return Boolean is
   begin
       return teste;

   end Get;
end testedeaceitacao;


-- Função e tipos protegidos das tasks do mecanismo de CRB
function teste_de_aceitacao ( crc1 : Interfaces.Unsigned_32; crc2 : Interfaces.Unsigned_32
 ) return Boolean is
   teste : Boolean;
begin

      if ( crc1 = crc2 ) then

         teste := True;
      else

         teste := False;
      end if;

   return teste;
end  teste_de_aceitacao;


protected type Semaphore ( InitialValue : Natural ) is
   procedure Signal;
   entry Wait;

private
   Count : Natural := InitialValue;
end Semaphore;


protected body Semaphore is
   procedure Signal is
 begin
```

```
    Count := Count + 1 ;
      end Signal;

      entry Wait when Count > 0 is
      begin

    Count := Count - 1 ;

      end Wait;

   end Semaphore;

   Sem : Semaphore ( InitialValue => 0 );


begin

   -- Primeira ordenação onde gerar o CRC Original dos Dados Ordenados.

   sorting.bubblesort( TEMP_LN_B );
   crc_check := calculo_ckecksum ( TEMP_LN_B );



   -- Inicialiação do contador geral do teste de aceitação
   count_geral := 0;

   -- Inicialização dos temporizadores inicial e final.
   temp_inicial := Ada.Real_Time.Clock;

   temp_final :=  Ada.Real_Time."+"(Left  => temp_inicial,
                                   Right => Ada.Real_Time.Seconds(S => 60));



   -- Execução em um laço com condição de temporização para termino dentro de 30 segundos.
   -- No Termino do laço gerar as saidas do algoritmo, gravando em um arquivo.
   -- Implementar as tecnicas de tolerância a falhas:
   -- TRM e Recovery Backward.
   -- Colocar os algoritmos de ordenação em concorrência dentro do laço.


   while temp_inicial < temp_final loop

      TEMP_LN_B := List;
      TEMP_LN_Q := List;
      TEMP_LN_I := List;



      declare
       -- Uso de tasks na paralelização das funções de ordenação.

        task type Bubble;
        task body Bubble is
        begin
           sorting.bubblesort( TEMP_LN_B );
           gerar_erro_saida_bs ( TEMP_LN_B );
           crc_result_bubble := calculo_ckecksum ( TEMP_LN_B );
```

```
  end Bubble;

  task type Quick ;
  task body Quick is
  begin
     sorting.quicksort( TEMP_LN_Q );
     gerar_erro_saida_qs ( TEMP_LN_Q );
     crc_result_quick := calculo_ckecksum ( TEMP_LN_Q );
  end Quick;

  task type Insertion;
  task body Insertion is
  begin
     sorting.insertionsort( TEMP_LN_I );
     gerar_erro_saida_is ( TEMP_LN_I );
     crc_result_insertion := calculo_ckecksum ( TEMP_LN_I );

   end Insertion;

  BB : Bubble;
  QQ : Quick;
  II : Insertion;

begin

   null;

end;


-- Mecanismos de diagnósticos de erro.
-- Teste de consistência pela verificação dos CRC's de cada saída dos algoritmos de
ordenação
-- Teste de replicação do mecanismos de NVP.
-- Implementação do Votador da NVP: Valida 2 ou mais resultados similares.

declare

   task type NVP_VOTADOR_EXATO;

   task body NVP_VOTADOR_EXATO is

   begin

      if ( crc_result_insertion = crc_result_bubble ) or
        ( ( crc_result_bubble = crc_result_quick ) or (crc_result_quick =
        crc_result_insertion) ) then


         testedeaceitacao.Set( True );
         Sem.Signal;

         if ( crc_result_bubble = crc_check and crc_result_quick = crc_check ) or
            ( ( crc_result_insertion = crc_check and crc_result_bubble = crc_check ) or
               ( crc_result_quick = crc_check and crc_result_insertion = crc_check ) )
               then

            count_corretos :=  count_corretos + 1;
```

```
         else

            count_falso_positivo := count_falso_positivo + 1;

         end if;

      else


         testedeaceitacao.Set( False );
         Sem.Signal;

         count_falso_negativo := count_falso_negativo + 1;

         end if;


   end NVP_VOTADOR_EXATO;

   task type RECOVERY_BLOCK;
   task body RECOVERY_BLOCK is
       -- Implementação do Teste de Aceitação
      type ALTERNATE_RANGE is range 1..4;
      alternate: ALTERNATE_RANGE := 1;
      -- Variavel do teste de aceitação
      alternatesuccess : Boolean ;
      teste_FAILURE : Boolean := FALSE;

   begin

      loop
         case alternate is
            when 1 => alternatesuccess := teste_de_aceitacao(crc1 => crc_check,
                                                           crc2 => crc_result_bubble
                                                           );
                      teste_FAILURE := FALSE;

            when 2 => alternatesuccess := teste_de_aceitacao(crc1 => crc_check,
                                                           crc2 => crc_result_quick );
                      teste_FAILURE := FALSE;

            when 3 => alternatesuccess := teste_de_aceitacao(crc1 => crc_check,
                                                           crc2 =>
                                                           crc_result_insertion );
                      teste_FAILURE := FALSE;

            when others => alternatesuccess := TRUE;  teste_FAILURE := TRUE;

         end case;

         exit when alternatesuccess;

         alternate := alternate + 1;

      end loop;


     -- Ação atômica protegida pelo Semaforo.
     Sem.Wait;
```

```
        if testedeaceitacao.Get = False then

            if teste_FAILURE then

               count_TA_FAILURE :=  count_TA_FAILURE + 1;

            else
               count_TA_OK := count_TA_OK + 1;

            end if;
         end if;


      end RECOVERY_BLOCK;

      RB : RECOVERY_BLOCK;
      NVP : NVP_VOTADOR_EXATO;

   begin

      null;

   end;


   count_geral := count_geral + 1;


   temp_inicial := Ada.Real_Time.Clock;

end loop;


New_Line;
put( "Resultados: ");
New_Line;
put("Total de iteracoes realizadas da execucao: ");
put ( Item => count_geral,
     Base =>  10 );



New_Line;
put("Total de resultados Corretos: ");
put ( Item => count_corretos,
     Base =>  10 );

New_Line;
put("Total de Falsos Positivos: ");
put ( Item => count_falso_positivo,
     Base =>  10 );

New_Line;
put("Total de Falsos Negativos: ");
put ( Item => count_falso_negativo,
     Base =>  10 );


New_Line;
```

```
   put("Total vezes que o resultado do teste de aceitacao no CRB teve resultado correto");
   put ( Item => count_TA_OK,
        Base =>  10 );

   New_Line;
   put("Total vezes que o resultado do teste de aceitacao no CRB lancara uma excecao de
   Falha!");
   put ( Item => count_TA_FAILURE,
        Base =>  10 );
   New_Line;
   New_Line;
   -- Inicializar o contador geral novamente
   count_geral := 0;

exception


   when Ada.IO_Exceptions.End_Error => put ( "Erro no programa. Finalizando temporizador, e
   gravando os dados no arquivo.");
      -- procedimento para gravar os dados finais no arquivo.
      put("Contabilização da execução:");
      put ( Item => count_geral,
            Base =>  10 );

   when Program_Error => Put("Ocorreu um erro em alguma estrutura de controle. ");
      Put_Line("O programa será terminado.");
   when Constraint_Error => Put("Ocorreu um erro na atribuição de uma valor a algum tipo. ");
                              Put_Line("O programa será terminado.");

      raise;

end crb_parallel;
```

```
with Ada.Text_IO;
with Ada.Integer_Text_IO;
with Gnat.CRC32;
with Interfaces;
with Ada.Strings.Unbounded;
with Ada.Exceptions;
with Ada.Text_IO.Unbounded_IO;
with Ada.IO_Exceptions;
with Ada.Integer_Text_IO;
with Ada.Real_Time; use Ada.Real_Time;


separate ( main )
procedure nvp is
   use sorting;
   use Ada;
   use Ada.Text_IO;
   use Gnat.CRC32;
   use Ada.Strings.Unbounded;
   use Interfaces;
   use Ada.Text_IO.Unbounded_IO;
   use Ada.Integer_Text_IO;

   package CRC_IO is new Ada.Text_IO.Modular_IO(Interfaces.Unsigned_32);

   List : List_Names_Sorting := ( Ada.Strings.Unbounded.To_Unbounded_String("Marcos Aurelio"),
                                  Ada.Strings.Unbounded.To_Unbounded_String("Marcos Arnaldo"),
                                  Ada.Strings.Unbounded.To_Unbounded_String("Marcela
                                  Cristina"),
                                  Ada.Strings.Unbounded.To_Unbounded_String("Zuleica Florez"),
                                  Ada.Strings.Unbounded.To_Unbounded_String("Roberta Maria"),
                                  Ada.Strings.Unbounded.To_Unbounded_String("Adriana Celia"),
                                  Ada.Strings.Unbounded.To_Unbounded_String("Orlando Silva
                                  Moraes Sales"),
                                  Ada.Strings.Unbounded.To_Unbounded_String("Orlando Silva
                                  Moraes Silva"),
                                  Ada.Strings.Unbounded.To_Unbounded_String("Adriana Celia
                                  Cavalcanti"),
                                 Ada.Strings.Unbounded.To_Unbounded_String("Bruna Adriana"));

   List_NVP_Result, TEMP_LN_B, TEMP_LN_Q, TEMP_LN_I : List_Names_Sorting := List;

   -- CRC's que será utilizado após o resultado de cada ordenação;
   crc_result_bubble, crc_result_quick, crc_result_insertion : Interfaces.Unsigned_32;
   -- CRC usado na checkagem do teste de aceitação
   crc_check : Interfaces.Unsigned_32;
   -- Contador geral para quantizar as vezes que ocorrer o teste de aceitação
   count_geral : Integer;
   -- Contador dos resultados do teste de aceitação com pelos menos dois CRC's
   correspondendo ao original.
   count_corretos : Integer := 0;
   -- Contador dos resultados do teste de aceitação com pelos menos dois CRC's de Falso
   Positivo.
   count_falso_positivo : Integer := 0;
   -- Contador dos resultados do teste de aceitação com pelos menos dois CRC's de Falso
   Positivo.
   count_falso_negativo : Integer := 0;

   -- Variáveis de temporização
```

```
   temp_inicial, temp_final : Ada.Real_Time.Time;

   -- Procedimento de nvp é uma função pois tem que retorna o valor da lista ordenada.
   procedure nvp_votador_exato is

   begin

      if ( crc_result_insertion = crc_result_bubble ) or
         ( ( crc_result_bubble = crc_result_quick ) or (crc_result_quick =
         crc_result_insertion) ) then

         if ( crc_result_bubble = crc_check and crc_result_quick = crc_check ) or
            ( ( crc_result_insertion = crc_check and crc_result_bubble = crc_check ) or
              ( crc_result_quick = crc_check and crc_result_insertion = crc_check ) ) then

            count_corretos :=  count_corretos + 1;

         else

            count_falso_positivo := count_falso_positivo + 1;

         end if;

      else

         count_falso_negativo := count_falso_negativo + 1;

      end if;

   end nvp_votador_exato;

begin

   -- Primeira ordenação onde gerar o CRC Original dos Dados Ordenados.

   sorting.bubblesort( TEMP_LN_B );
   crc_check := calculo_ckecksum ( TEMP_LN_B );

   -- Inicialiação do contador geral do teste de aceitação
   count_geral := 0;
   -- Inicialização dos temporizadores inicial e final.
   temp_inicial := Ada.Real_Time.Clock;

   temp_final :=  Ada.Real_Time."+"(Left  => temp_inicial,
                                    Right => Ada.Real_Time.Seconds(S => 60));


   -- Execução em um laço com condição de temporização para termino dentro de 60 segundos.
   -- Implementar as tecnicas de tolerância a falhas:
   -- Recovery Block

   while temp_inicial < temp_final loop

      TEMP_LN_B := List;
      TEMP_LN_Q := List;
      TEMP_LN_I := List;

      declare
       -- Uso de tasks na paralelização das funções de ordenação.
```

```
      task type Bubble;
      task body Bubble is
      begin
         sorting.bubblesort( TEMP_LN_B );
         gerar_erro_saida_bs ( TEMP_LN_B );
            crc_result_bubble := calculo_ckecksum ( TEMP_LN_B );
      end Bubble;

      task type Quick ;
      task body Quick is
      begin
         sorting.quicksort( TEMP_LN_Q );
         gerar_erro_saida_qs ( TEMP_LN_Q );
         crc_result_quick := calculo_ckecksum ( TEMP_LN_Q );
      end Quick;

      task type Insertion;
      task body Insertion is
      begin
         sorting.insertionsort( TEMP_LN_I );
         gerar_erro_saida_is ( TEMP_LN_I );
         crc_result_insertion := calculo_ckecksum ( TEMP_LN_I );

       end Insertion;

      BB : Bubble;
      QQ : Quick;
      II : Insertion;

    begin

       null;

    end;

    -- Mecanismos de diagnósticos de erro.
    -- Teste de consistência pela verificação dos CRC's de cada saída dos algoritmos de
    ordenação
    -- Teste de replicação do mecanismos de NVP.

    -- Implementação do Votador da NVP: Valida 2 ou mais resultados similares.
    nvp_votador_exato;

    count_geral := count_geral + 1;

    temp_inicial := Ada.Real_Time.Clock;

 end loop;

 New_Line;
 put( "Resultados: ");
 New_Line;
 put("Total de iteracoes realizadas da execucao: ");
 put ( Item => count_geral,
      Base =>  10 );

 New_Line;
```

```
    put("Total de resultados Corretos: ");
    put ( Item => count_corretos,
         Base =>  10 );

    New_Line;
    put("Total de Falsos Positivos: ");
    put ( Item => count_falso_positivo,
         Base =>  10 );

    New_Line;
    put("Total de Falsos Negativos: ");
    put ( Item => count_falso_negativo,
         Base =>  10 );
    New_Line;
    New_Line;
    -- Inicializar o contador geral novamente
    count_geral := 0;


exception


    when Ada.IO_Exceptions.End_Error => put ( "Erro no programa. Finalizando temporizador, e
    gravando os dados no arquivo.");
       -- procedimento para gravar os dados finais no arquivo.
       put("Contabilização da execução:");
       put ( Item => count_geral,
             Base =>  10 );

    when Program_Error => Put("Ocorreu um erro em alguma estrutura de controle. ");
       Put_Line("O programa será terminado.");
    when Constraint_Error => Put("Ocorreu um erro na atribuição de uma valor a algum tipo. ");
                              Put_Line("O programa será terminado.");

       raise;

end nvp;
```

```
with Ada.Text_IO;
with Ada.Integer_Text_IO;
with Gnat.CRC32;
with Interfaces;
with Ada.Strings.Unbounded;
with Ada.Exceptions;
with Ada.Text_IO.Unbounded_IO;
with Ada.IO_Exceptions;
with Ada.Integer_Text_IO;
with Ada.Real_Time; use Ada.Real_Time;
with Gnat;
with Ada;
with System; use System;


separate ( main )
procedure recovery_block is
   use sorting;
   use Ada;
   use Ada.Text_IO;
   use Gnat.CRC32;
   use Ada.Strings.Unbounded;
   use Interfaces;
   use Ada.Text_IO.Unbounded_IO;
   use Ada.Integer_Text_IO;
   use Gnat;

   package CRC_IO is new Ada.Text_IO.Modular_IO(Interfaces.Unsigned_32);


   List : List_Names_Sorting := ( Ada.Strings.Unbounded.To_Unbounded_String("Marcos Aurelio"),
                                  Ada.Strings.Unbounded.To_Unbounded_String("Marcos Arnaldo"),
                                  Ada.Strings.Unbounded.To_Unbounded_String("Marcela
                                  Cristina"),
                                  Ada.Strings.Unbounded.To_Unbounded_String("Zuleica Florez"),
                                  Ada.Strings.Unbounded.To_Unbounded_String("Roberta Maria"),
                                  Ada.Strings.Unbounded.To_Unbounded_String("Adriana Celia"),
                                  Ada.Strings.Unbounded.To_Unbounded_String("Orlando Silva
                                  Moraes Sales"),
                                  Ada.Strings.Unbounded.To_Unbounded_String("Orlando Silva
                                  Moraes Silva"),
                                  Ada.Strings.Unbounded.To_Unbounded_String("Adriana Celia
                                  Cavalcanti"),
                                 Ada.Strings.Unbounded.To_Unbounded_String("Bruna Adriana"));

   List_Result, TEMP_LN_B, TEMP_LN_Q, TEMP_LN_I : List_Names_Sorting := List;

   -- CRC's que será utilizado após o resultado de cada ordenação;
   crc_result : Interfaces.Unsigned_32;
   -- CRC usado na checkagem do teste de aceitação
   crc_check : Interfaces.Unsigned_32;
   -- Contador geral para quantizar as vezes que ocorrer o teste de aceitação
   count_geral : Integer;

   -- Contador dos resultados do teste de aceitação com pelos menos dois CRC's
   correspondendo ao original.
   count_sucesso : Integer := 0;
   -- Contador dos resultados do teste de aceitação com pelos menos dois CRC's de Falso
   Positivo.
```

```
   count_falha : Integer := 0;

   -- Variáveis de temporização
   temp_inicial, temp_final : Ada.Real_Time.Time;

   -- Variavel do teste de aceitação
   alternatesuccess : Boolean ;

   FAILURE: exception;

   -- Funções e procedimentos do Bloco de Recuperação por Retrocesso
   function teste_de_aceitacao ( crc1 : Interfaces.Unsigned_32; crc2 : Interfaces.Unsigned_32
    ) return Boolean is
      teste : Boolean;
   begin

         if ( crc1 = crc2 ) then

            teste := True;
         else

            teste := False;
         end if;

      return teste;
   end  teste_de_aceitacao;

   procedure alternative1_bubble is
   begin
      sorting.bubblesort( TEMP_LN_B );
      gerar_erro_saida_qs ( TEMP_LN_B );
      crc_result := calculo_ckecksum ( TEMP_LN_B );
      alternatesuccess := teste_de_aceitacao(crc1 => crc_check, crc2 => crc_result );

   end alternative1_bubble;

   procedure alternative2_quick  is
   begin

      sorting.quicksort( TEMP_LN_Q );
      gerar_erro_saida_qs ( TEMP_LN_Q );
      crc_result := calculo_ckecksum ( TEMP_LN_Q );
      alternatesuccess := teste_de_aceitacao(crc1 => crc_check, crc2 => crc_result );


   end alternative2_quick;

   procedure alternative3_insertion  is
   begin

      sorting.insertionsort( TEMP_LN_I );
      gerar_erro_saida_qs ( TEMP_LN_I );
      crc_result := calculo_ckecksum ( TEMP_LN_I );
      alternatesuccess := teste_de_aceitacao(crc1 => crc_check, crc2 => crc_result );

   end alternative3_insertion;


begin
```

```ada
-- Primeira ordenação onde gerar o CRC Original dos Dados Ordenados.

sorting.quicksort ( TEMP_LN_Q );
crc_check := calculo_ckecksum ( TEMP_LN_Q );


-- Inicialiação do contador geral do teste de aceitação
count_geral := 0;

-- Inicialização dos temporizadores inicial e final.
temp_inicial := Ada.Real_Time.Clock;

temp_final :=  Ada.Real_Time."+"(Left  => temp_inicial,
                                 Right => Ada.Real_Time.Seconds(S => 60));


 while temp_inicial < temp_final loop

   TEMP_LN_B := List;
   TEMP_LN_Q := List;
   TEMP_LN_I := List;

   -- Implementação do Bloco de Recuperação por retrocesso
   -- Mecanismos de diagnósticos de erro.
   -- Teste de consistência pela verificação dos CRC's de cada saída dos algoritmos de
   ordenação
   -- Implementação do Teste de Aceitação
   declare

      type ALTERNATE_RANGE is range 1..4;
      alternate: ALTERNATE_RANGE := 1;

   begin

         loop
            case alternate is
               when 1 => alternative1_bubble;
                  if alternatesuccess = True then

                     count_sucesso := count_sucesso + 1;
                  end if;

               when 2 => alternative2_quick;
                   if alternatesuccess = True then

                     count_sucesso := count_sucesso + 1;
                   end if;


               when 3 => alternative3_insertion;
                  if alternatesuccess = True then

                     count_sucesso := count_sucesso + 1;
                  end if;

               when others => raise FAILURE;

            end case;
```

```
                exit when alternatesuccess;




                alternate := alternate + 1;

          end loop;

       exception
          when FAILURE => count_falha := count_falha + 1;
       end;

       count_geral := count_geral + 1;

       temp_inicial := Ada.Real_Time.Clock;

     end loop;

   New_Line;
   put( "Resultados: ");
   New_Line;
   put("Total de iteracoes realizadas da execucao: ");
   put ( Item => count_geral,
        Base =>  10 );

   New_Line;

   put("Total de Sucessos: ");
   put ( Item => count_sucesso,
        Base =>  10 );

   New_Line;
   put("Total de Falhas: ");
   put ( Item => count_falha,
        Base =>  10 );
   New_Line;
   New_Line;
   -- Inicializar o contador geral novamente
   count_geral := 0;

exception

   when Ada.IO_Exceptions.End_Error => put ( "Erro no programa. Finalizando temporizador, e
   gravando os dados no arquivo.");
      -- procedimento para gravar os dados finais no arquivo.
      put("Contabilização da execução:");
      put ( Item => count_geral,
            Base =>  10 );

   when Program_Error => Put("Ocorreu um erro em alguma estrutura de controle. ");
      Put_Line("O programa será terminado.");
   when Constraint_Error => Put("Ocorreu um erro na atribuição de uma valor a algum tipo. ");
      Put_Line("O programa será terminado.");

      raise;

end recovery_block;
```

```
with sorting;
with GNAT.Dynamic_Tables;
with Ada.Unchecked_Conversion;
with Ada.Strings;
with Ada.Strings.Unbounded;
with Ada.Text_IO;
with Ada.Integer_Text_IO;

separate ( main )

-- Gerar erro na saída do bubblesort
procedure gerar_erro_saida_bs ( LN1 : in out sorting.List_Names_Sorting ) is
   use sorting;
   use Ada;
   use Ada.Text_IO;
   use Ada.Integer_Text_IO;
   use Strings.Unbounded;
   num , index : Integer;

   subtype semente is Integer range 1 .. 256;
   package geraerrorandomico is new Ada.Numerics.Discrete_Random ( semente );
   use geraerrorandomico;
   G : Generator;
   S : semente;
   type Byte is mod 256;
   str_byte : Byte;
   package M_IO is new Ada.Text_IO.Modular_IO(Byte);
   str_byte_last : Byte;
   ch : Character;

begin

   Reset (Gen => G );
   S := Random (Gen => G);
   --  S varia de 1 a 256
   --  Ocorrendo o erro maior que 0% e menor ou igual do que 10%, se altera o bit menos
   significativo.
   if ( S > 0 ) and ( S < 27 ) then
      -- Gerar erro na saida do array ordenado, mudando o bit menos siginificativo do ultimo
      caracter da string.

      index := Ada.Strings.Unbounded.Length(Source =>LN1(LN1'Last));
      num := character'Pos( Ada.Strings.Unbounded.Element( Source => LN1(LN1'Last),
                                                            Index => index ));
      str_byte := Byte( num );

      str_byte_last := ( str_byte xor 1 );

      num := Integer( str_byte_last );
      -- Caracter alterado pela inversão do bit menos significativo
      ch := Character'Val(num);

      Ada.Strings.Unbounded.Replace_Element(Source => LN1(LN1'Last),
                                            Index  => index,
                                            By     => ch );

   end if;

end gerar_erro_saida_bs;
```

```
  -- Procedimento que gera um erro no resultado da saí-da dos algoritmos de ordenação
  quicksort
with sorting;
with GNAT.Dynamic_Tables;
with Ada.Unchecked_Conversion;
with Ada.Strings;
with Ada.Strings.Unbounded;
with Ada.Text_IO;
with Ada.Integer_Text_IO;

separate ( main )

-- Gerar erro na saída do Quicksort
procedure gerar_erro_saida_qs ( LN1 : in out sorting.List_Names_Sorting ) is
   use sorting;
   use Ada;
   use Ada.Text_IO;
   use Ada.Integer_Text_IO;
   use Strings.Unbounded;

   num , index : Integer;

   subtype semente is Integer range 1 .. 256;
   package geraerrorandomico is new Ada.Numerics.Discrete_Random ( semente );
   use geraerrorandomico;
   G : Generator;
   S : semente;
   type Byte is mod 256;
   str_byte : Byte;
   package M_IO is new Ada.Text_IO.Modular_IO(Byte);
   str_byte_last : Byte;
   ch : Character;

begin

   Reset (Gen => G );
   S := Random (Gen => G);
   --   S varia de 1 a 256
   --   O erro maior que 0% e menor ou igual do que 5% se altera o segundo bit menos
   significativo.
   --   O erro maior que 5% e menor ou igual do que 10% se altera o bit menos siginificativo.
   if ( S > 0 ) and ( S <= 26 ) then
      -- Gerar erro na saida do array ordenado, mudando o bit menos siginificativo do ultimo
      caracter da string.

      index := Ada.Strings.Unbounded.Length(Source =>LN1(LN1'Last));
      num := character'Pos( Ada.Strings.Unbounded.Element( Source => LN1(LN1'Last),
                                                           Index => index ));
      str_byte := Byte( num );

      if ( S > 0) and ( S <= 13 ) then
        --    O erro maior que 0% e menor ou igual do que 5% se altera o segundo bit menos
        significativo.
          str_byte_last := ( str_byte xor 10 );

          num := Integer( str_byte_last );
        -- Caracter alterado pela inversão do bit menos significativo
              ch := Character'Val(num);
```

```
        Ada.Strings.Unbounded.Replace_Element(Source => LN1(LN1'Last),
                                          Index  => index,
                                           By     => ch );
      else
        --    O erro maior que 5% e menor ou igual do que 10% se altera o bit menos
        siginificativo.
            str_byte_last := ( str_byte xor 1 );

          num := Integer( str_byte_last );
        -- Caracter alterado pela inversão do bit menos significativo
            ch := Character'Val(num);

          Ada.Strings.Unbounded.Replace_Element(Source => LN1(LN1'Last),
                                          Index  => index,
                                           By     => ch );

      end if;

   end if;

end gerar_erro_saida_qs;
```

```
 -- Procedimento que gera um erro no resultado da saí-da dos algoritmos de ordenação
 insertionsort
with sorting;
with GNAT.Dynamic_Tables;
with Ada.Unchecked_Conversion;
with Ada.Strings;
with Ada.Strings.Unbounded;
with Ada.Text_IO;
with Ada.Integer_Text_IO;

separate ( main )

-- Gerar erro na saída do insertionsort
procedure gerar_erro_saida_is ( LN1 : in out sorting.List_Names_Sorting ) is
   use sorting;
   use Ada;
   use Ada.Text_IO;
   use Ada.Integer_Text_IO;
   use Strings.Unbounded;

   num , index : Integer;

   subtype semente is Integer range 1 .. 256;
   package geraerrorandomico is new Ada.Numerics.Discrete_Random ( semente );
   use geraerrorandomico;
   G : Generator;
   S : semente;
   type Byte is mod 256;
   str_byte : Byte;
   package M_IO is new Ada.Text_IO.Modular_IO(Byte);
   str_byte_last : Byte;
   ch : Character;

begin

   Reset (Gen => G );
   S := Random (Gen => G);
   --   S varia de 1 a 256
   --   O erro maior que 0% e menor ou igual do que 3,3% se altera o segundo bit menos
   significativo.
   --   O erro maior que 3,3% e menor ou igual do que 6,6% se altera o segundo bit menos
   significativo.
   --   O erro maior que 6,6% e menor ou igual do que 10% se altera o bit menos
   siginificativo.
   if ( S > 0 ) and ( S <= 26 ) then
      -- Gerar erro na saida do array ordenado, mudando o bit menos siginificativo do ultimo
      caracter da string.

      index := Ada.Strings.Unbounded.Length(Source =>LN1(LN1'Last));
      num := character'Pos( Ada.Strings.Unbounded.Element( Source => LN1(LN1'Last),
                                                           Index => index ));
      str_byte := Byte( num );

      if ( S > 0 ) and ( S <= 8 ) then
        --    O erro maior que 0% e menor ou igual do que 3,3% se altera o segundo bit menos
        significativo.
          str_byte_last := ( str_byte xor 100 );

          num := Integer( str_byte_last );
```

```
      -- Caracter alterado pela inversão do bit menos significativo
            ch := Character'Val(num);

        Ada.Strings.Unbounded.Replace_Element(Source => LN1(LN1'Last),
                                              Index  => index,
                                               By     => ch );
    elsif ( S > 8 ) and ( S <= 17 ) then
      --    O erro maior que 3,3% e menor ou igual do que 6,6% se altera o segundo bit
      menos significativo.
            str_byte_last := ( str_byte xor 10 );

        num := Integer( str_byte_last );
      -- Caracter alterado pela inversão do bit menos significativo
            ch := Character'Val(num);

        Ada.Strings.Unbounded.Replace_Element(Source => LN1(LN1'Last),
                                              Index  => index,
                                               By     => ch );
    else
      --    O erro maior que 6,6% e menor ou igual do que 10% se altera o bit menos
      siginificativo.
            str_byte_last := ( str_byte xor 1 );

        num := Integer( str_byte_last );
      -- Caracter alterado pela inversão do bit menos significativo
            ch := Character'Val(num);

        Ada.Strings.Unbounded.Replace_Element(Source => LN1(LN1'Last),
                                              Index  => index,
                                               By     => ch );

    end if;

   end if;
end gerar_erro_saida_is;
```

```ada
with Ada.Strings.Hash_Case_Insensitive;
with Ada.Containers;
with Ada.Strings.Hash;
with Gnat.CRC32;
with Ada.Text_IO;
with Ada.Strings.Unbounded;
with Interfaces;
separate( main )

-- Verificação da soma usando Cyclic Redundancy Check.
function calculo_ckecksum ( LN1 : in sorting.List_Names_Sorting ) return Interfaces.
Unsigned_32 is
   use Gnat.CRC32;
   use sorting;
   use Ada;
   use Ada.Text_IO;
   use Ada.Strings.Unbounded;
   crc : Gnat.CRC32.CRC32;
   str : Unbounded_String;
   crc_unsigned_32 : Interfaces.Unsigned_32;

begin
   for Pass_Count in LN1'First .. LN1'Last loop

      str :=  str & LN1(Pass_Count);

   end loop;

   Initialize(C => crc);
   Update (C     => crc ,
           Value => Ada.Strings.Unbounded.To_String(Source => str));

   crc_unsigned_32 := Gnat.CRC32.Get_Value(crc);

   return crc_unsigned_32;

end calculo_ckecksum;
```

```
--------------------------------------------------------------
---- Especificação do package dos algoritmos de ordenação ----
------------- bubblesort, mergesort e quicksort --------------
---- para demostração das técnicas de tolerância a falhas ----
---- em ADA. Autor: Marcos Aurélio Silva de Souza ------------
--------------------------------------------------------------

with Ada;
with Ada.Strings.Unbounded;

package sorting is


   type List_Names_Sorting is array (Positive range <>) of Ada.Strings.Unbounded.
   Unbounded_String;

   procedure bubblesort ( LN : in out List_Names_Sorting );

   procedure quicksort ( LN : in out List_Names_Sorting );

   procedure insertionsort ( LN : in out List_Names_Sorting );



end sorting;
```

```
-------------------------------------------------------------------
-- Implementação do corpo do package dos algoritmos de ordenação--
--------------- bubblesort, mergesort e quicksort ----------------
---- para demostração das técnicas de tolerância a falhas --------
---- em ADA. Autor: Marcos Aurélio Silva de Souza ----------------
-------------------------------------------------------------------


with Ada;
use Ada;

package body sorting is

   --- serão compiladas como subunits.
   procedure bubblesort ( LN : in out List_Names_Sorting ) is separate;

   procedure quicksort ( LN : in out List_Names_Sorting ) is separate;

   procedure insertionsort ( LN : in out List_Names_Sorting ) is separate;


end sorting;
```

```
with Ada.Strings.Unbounded;

use Ada.Strings.Unbounded;

separate ( sorting )

procedure bubblesort ( LN : in out List_Names_Sorting ) is
 Finished : Boolean;
 Temp     : Ada.Strings.Unbounded.Unbounded_String;
begin
 loop
  Finished := True;
  for J in LN'First .. Positive'Pred (LN'Last) loop
   if LN (Positive'Succ (J)) < LN (J) then
    Finished := False;
    Temp := LN (Positive'Succ (J));
    LN (Positive'Succ (J)) := LN (J);
    LN (J) := Temp;
   end if;
  end loop;
  exit when Finished;
 end loop;
end bubblesort;
```

```ada
with Ada.Strings.Unbounded;
use Ada.Strings.Unbounded;

separate ( sorting )
procedure quicksort ( LN : in out List_Names_Sorting ) is

   procedure Swap(Left, Right : in out Ada.Strings.Unbounded.Unbounded_String) is
      Temp : Ada.Strings.Unbounded.Unbounded_String := Left;
   begin
      Left := Right;
      Right := Temp;
   end Swap;

   Pivot_Index : Positive;
   Pivot_Value : Ada.Strings.Unbounded.Unbounded_String;
   Right       : Positive := LN'Last;
   Left        : Positive := LN'First;

begin
   if LN'Length > 1 then
      Pivot_Index := Positive'Val((Positive'Pos(LN'Last) + 1 +
                                  Positive'Pos(LN'First)) / 2);
      Pivot_Value := LN(Pivot_Index);

      Left  := LN'First;
      Right := LN'Last;
      loop
         while Left < LN'Last and then LN(Left) < Pivot_Value loop
            Left := Positive'Succ(Left);
         end loop;
         while Right > LN'First and then LN(Right) > Pivot_Value loop
            Right := Positive'Pred(Right);
         end loop;
         exit when Left >= Right;
         Swap(LN(Left), LN(Right));
         if Left < LN'Last and Right > LN'First then
            Left := Positive'Succ(Left);
            Right := Positive'Pred(Right);
         end if;
      end loop;
      if Right > LN'First then
         quicksort(LN(LN'First..Positive'Pred(Right)));
      end if;
      if Left < LN'Last then
         quicksort(LN(Left..LN'Last));
      end if;
   end if;

end quicksort;
```

```ada
with Ada.Strings.Unbounded;
use Ada.Strings.Unbounded;
separate ( sorting )

procedure insertionsort ( LN : in out List_Names_Sorting ) is
   First : Positive := LN'First;
   Last  : Positive := LN'Last;
   Value : Ada.Strings.Unbounded.Unbounded_String;
   J     : Integer;
begin
   for I in (First + 1)..Last loop

      Value := LN(I);
      J := I - 1;

      while J in LN'range and then LN(J) > Value loop
         LN(J + 1) := LN(J);
         J := J - 1;
      end loop;

      LN(J + 1) := Value;

   end loop;

end insertionsort;
```