DISSERTAÇÃO DE MESTRADO

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM

CIÊNCIA DA COMPUTAÇÃO

"Um Modelo de Rede de Sensores sem fio Auto-Organizada e Tolerante a Falhas para Detecção de Incêndios"

ALUNO: Felipe Taliar Giuntini
ORIENTADOR: Prof. Dr. Delano Medeiros Beder

São Carlos
Agosto/2016

CAIXA POSTAL 676
FONE/FAX: (16) 3351-8233
13565-905 - SÃO CARLOS - SP
BRASIL

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UM MODELO DE REDE DE SENSORES SEM FIO AUTO-ORGANIZADA E TOLERANTE A FALHAS PARA DETECÇÃO DE INCÊNDIOS

FELIPE TALIAR GIUNTINI

ORIENTADOR: PROF. DR. DELANO MEDEIROS BEDER

São Carlos – SP

Agosto/2016

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UM MODELO DE REDE DE SENSORES SEM FIO AUTO-ORGANIZADA E TOLERANTE A FALHAS PARA DETECÇÃO DE INCÊNDIOS

FELIPE TALIAR GIUNTINI

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação, área de concentração: Engenharia de Software

Orientador: Prof. Dr. Delano Medeiros Beder

São Carlos – SP

Agosto/2016

Ficha catalográfica elaborada pelo DePT da Biblioteca Comunitária UFSCar
Processamento Técnico
com os dados fornecidos pelo(a) autor(a)

Giuntini, Felipe Taliar
G537m Um modelo de rede de sensores sem fio auto-organizada e tolerante a falhas para detecção de incêndios / Felipe Taliar Giuntini. -- São Carlos : UFSCar, 2016.
94 p.

Dissertação (Mestrado) -- Universidade Federal de São Carlos, 2016.

1. Redes de sensores sem fio. 2. Auto-organização. 3. Tolerância a Falhas. 4. Detecção de fogo. I. Título.

UNIVERSIDADE FEDERAL DE SÃO CARLOS

Centro de Ciências Exatas e de Tecnologia

Programa de Pós-Graduação em Ciência da Computação

Folha de Aprovação

Assinaturas dos membros da comissão examinadora que avaliou e aprovou a defesa de Dissertação de Mestrado do candidato Felipe Taliar Giuntini, realizada em 30/08/2016.

Prof. Dr. Delano Medeiros Beder
(UFSCar)

Prof. Dr. Jo Ueyama
(ICMC-USP)

****

Prof. Dr. Roberto Sadao Yokoyam
(UTFPR)

Certifico que a sessão de defesa foi realizada com a participação à distância do membro Prof. Dr. Roberto Sadao Yokoyam . Depois das arguições e deliberações realizadas, o participante à distância está de acordo com o conteúdo do parecer da comissão examinadora redigido no relatório de defesa do aluno Felipe Taliar Giuntini.

Prof. Dr. Delano Medeiros Beder
Coordenador da Comissão Examinadora
(UFSCar)

AGRADECIMENTOS

Este trabalho é resultado de muito esforço e perseverança dispensados nestes últimos anos. Quando escrevo estas palavras, é inevitável o sentimento de orgulho de que cheguei ao fim desta etapa e de gratidão àqueles que contribuíram durante todo o processo. Gostaria de expressar minha sincera gratidão,

- a minha família pelo apoio em todos os sentidos e compreensão de que eu precisava buscar os meus objetivos distante de casa;
- ao professor Dr. Delano Medeiros Beder, pela orientação, sabedoria e coerência com que guiou este trabalho. Além disso, pela enorme paciência e amizade;
- aos meus amigos André, Augusto, Amandia, Mirela, Rafael e Thiago, pelo apoio em diversos momentos e pelas divertidas quartas-feiras;
- aos meus amigos Crislaine, Filipe, Gregory e Lilian, pelas orações, orientações e pelos momentos que compartilhamos.
- aos colegas que conheci na sala de estudos da Pós-Graduação pelos sábados e domingos, que passamos estudando o conteúdo das disciplinas.
- aos colegas e amigos do laboratório WINDIS pela amizade e discussões em grupo.
- aos professores e funcionários do Departamento de Computação da UFSCar;
- a CAPES pelo apoio financeiro;
- e especialmente a Deus, por todos esses acima e todas as coisas na minha vida.

Obrigado!

“Use sua mente
Seu jeito, sua imaginação
Use força de vontade
Humildade e inovação.”

Pablo Coelho

RESUMO

Os incêndios causados pela ocupação humana é um dos fatores que mais contribui para o desmatamento das áreas de preservação ambiental, acarretando uma série de problemas aos sistemas ecológicos. A detecção precoce do fogo visa eliminar ou minimizar o dano que será causado por um incidente de fogo. Redes de Sensores sem Fio (RSSFs) tem se mostrado uma boa alternativa para aplicações de monitoramento ambiental, visto que podem coletar e enviar informações em tempo real, como umidade, vento e temperatura de vários pontos da floresta. Devido a problemas como limitação de energia, falha na comunicação e perda de nós sensores, a topologia da rede muda constantemente, exigindo mecanismos que permitem alcançar a auto-organização e a tolerância a falhas. Este trabalho propõe o desenvolvimento de um modelo e uma aplicação em RSSFs auto-organizável e tolerante a falhas para detecção de fogo em áreas de preservação. Para alcançar a auto-organização e a tolerância a falhas incentiva-se as interações locais entre nós vizinhos que monitoram uma mesma região e a coordenação de tarefas, por meio de um nó coordenador equipado com um framework para desenvolvimento de aplicações tolerante a falhas baseado em componentes. Utilizando uma técnica de redundância de componentes com abordagem adaptativa, a solução de detecção de fogo foi implementada. Cada componente, ou seja, diferentes implementações de uma mesma especificação, é carregado e descarregado da memória em tempo de execução enquanto o nó assume o papel de coordenador. Os resultados são armazenados e após execução de todos componentes é obtido um consenso. Para análise e validação do modelo e da aplicação simulou-se 60 eventos na rede de sensores em um cenário real, utilizando o simulador Sinalgo. Os resultados foram classificados como Verdadeiros (parcial ou total) ou Falsos (parcial ou total). Em 45% dos consensos identificou-se uma possível falha na aplicação e somente em 35% houve um consenso total.

Palavras-chave: Redes de Sensores sem Fio, Auto-organização, Tolerância a Falhas, Detecção de Fogo

ABSTRACT

The wildfires caused by human occupation is one of the factors that most contributes to deforestation of conservation areas, resulting in a number of issues for ecological systems. Premature fire detection lead to the elimination or minimize the damage that will be caused by a fire incident. Wireless Sensor Networks (WSNs) has been shown to be a good alternative for environmental monitoring applications, as they can collect and send the information in real-time, such as humidity, wind and temperature of various parts of the forest. Due to problems such as power limitation, communication failure and loss of nodes, the network topology is constantly changing, requiring mechanisms to achieve self-organization and fault tolerance. This paper proposes the development of a model and application in self-organizing and fault-tolerant WSNs for fire detection in conservation areas. To achieve self-organization and fault tolerance is encouraged local interactions between neighboring nodes that monitor the same region and the coordination of tasks, through a supervisor node, equipped with a framework for developing fault-tolerant applications based on components. Using a component redundancy technique with adaptive approach, the fire detection solution was implemented. Each component, ie, different implementations of the same specification, it is loaded and unloaded from runtime memory while the node assumes the role of coordinator. The results are stored and after execution of all components is achieved a consensus. For analysis and validation of the model and the application was simulated to 60 events in the sensor network in a real scenario, using the Sinalgo simulator. The results were classified as True (partial or absolute) or False (partial or absolute). In 45% of consensus identified a possible fault in the application and in only 35% there was absolute consensus.

Keywords: Wireless Sensor Networks, Sel-Organization, Fault Tolerance, Fire Detection

LISTA DE FIGURAS

1.2 Comparação dos dados do ano atual com os valores máximos, médios e mínimos de 1998 até 21 de janeiro 2016. 14

2.1 Mapa de energia de uma rede 30

2.2 Modelo genérico de RSSF com um nodo *gateway* 32

2.3 Modelo de RSSF com um nodo *sink* 32

2.4 Estabelecimento da RSSF 33

2.5 Programação N-Versões 40

2.6 Blocos de Recuperação 40

2.7 FlexFT Framework (a) e (b) 42

2.8 Concepção da programação n-versões no FlexFT 43

2.9 Concepção do Blocos de Recuperação no FlexFT 44

2.10 Contraste entre redes convencionais e redes auto-organizáveis 47

2.11 Categorização Horizontal de Mecanismos de auto-organização em redes de ad hoc e de sensores 49

2.12 Categorização Vertical de Mecanismos de auto-organização em redes de ad hoc e de sensores 51

3.1 Notificação de eventos, Clusterização e Eleição de Coordenador. 58

3.2 Resumo do Modelo auto-organizável e tolerante a falhas para detecção de incêndios 62

3.3 Árvores de decisão para detecção de incêndios florestais (a), (b) e (c). 65

3.4 Detalhes de Acurácias das árvores de decisão por classe (a), (b) e (c). 66

3.5 Matriz de Confusão das árvores de decisão (a), (b) e (c). 66

3.6 Diagrama de Componentes da Solução 67
3.7 Diagrama de Classes da solução 68
3.8 Resultados da Simulação para 60 eventos. 73

LISTA DE TABELAS

2.1 Dimensões da Modelagem 24

2.2 Elementos que compõem o modelo de energia de um sensor 29

2.3 Requisitos das tarefas do modelo funcional 36

2.4 Propriedades de sistemas auto-organizáveis 45

3.1 Configurações iniciais do Sinalgo 74

GLOSSÁRIO

DAARP – *Data-Aggregation Aware Routing Protocol*

MAC – *Medium Access Protocol*

MANET – *Mobile Ad hoc Network*

RSSFs – *Redes de Sensores sem Fio*

SAS – *Situational Awareness System*

SGBDs – *Sistemas Gerenciadores de Banco de Dados*

SUO – *Small Units Operations*

TDM – *Time Division Multiplexing*

VANTs – *Veículos Aéreos não tripulados*

SUMÁRIO

GLOSSÁRIO

CAPÍTULO 1 – INTRODUÇÃO **13**

1.1 Contexto e Motivação 13

1.2 Solução Proposta 15

1.3 Contribuições 16

1.4 Organização da Dissertação 16

CAPÍTULO 2 – REVISÃO BIBLIOGRÁFICA **17**

2.1 Computação Autônoma 17

2.1.1 Situational Awareness System (SAS) 18

2.1.2 Propriedades de autogestão 18

2.1.3 Graus de automaticidade 20

2.2 Dimensões da Modelagem 21

2.2.1 Grupo *Goals* 21

2.2.2 Grupo *Change* 22

2.2.3 Grupo *Mechanisms* 22

2.2.4 Grupo *Effects* 23

2.3 Rede de Sensores sem Fio 25

2.3.1 Características das Redes de Sensores sem Fio 27

2.3.2 Componentes de uma RSSF 30

2.3.3 Modelo Funcional de RSSF 32
2.4 Tolerância a falhas 36
2.4.1 Falhas, erros e defeitos 37
2.4.2 Abordagens e técnicas para aplicação da tolerância a falhas 38
2.4.3 FlexFT: Um framework para desenvolvimento de aplicações tolerante a falhas 41
2.5 Auto-organização 44
2.6 Auto-organização em Rede de Sensores sem Fio 45
2.6.1 Propriedades e Objetivos 46
2.6.2 Categorização em duas dimensões 47
2.6.3 Métodos e aplicações 51
2.7 Trabalhos Relacionados 53
2.7.1 Auto-Organização e Tolerância a Falhas 53
2.7.2 Aplicações de detecção de incêndios em RSSFs 55
2.8 Considerações Finais 56

CAPÍTULO 3 – MODELO PROPOSTO **57**

3.1 O Modelo auto-organizável e tolerante a falhas 57
3.1.1 Fase A - Implementação e estabelecimento da rede 57
3.1.2 Fase B - Notificação de eventos, Clusterização e Eleição de Coordenador 58
3.1.3 Fase C - Tarefas de Coordenação e Detecção de Fogo 60
3.1.4 Fase D - Agregação e envio dos dados 61
3.2 Detalhes de implementação e Simulação 63
3.2.1 Protótipo da solução de detecção de incêndios 64
3.2.2 Inserção do protótipo no Nó Sensor 69
3.2.3 Resultados da Simulação 73
3.2.4 Categorização do modelo e solução 75

CAPÍTULO 4 – CONSIDERAÇÕES FINAIS **77**

4.1 Contribuições 78

4.2 Trabalhos Futuros 79

APÊNDICE A – CÓDIGO DA IMPLEMENTAÇÃO DA SOLUÇÃO **80**

REFERÊNCIAS BIBLIOGRÁFICAS **91**

Capítulo 1

INTRODUÇÃO

1.1 Contexto e Motivação

Um dos principais problemas ambientais no Brasil são as altas taxas de incêndios florestais causados pela ocupação humana, colocando o país entre os principais responsáveis pelo aumento das emissões de gases de efeito estufa no mundo. Além de contribuir para o aquecimento global e causar mudanças climáticas, os incêndios causam perdas econômicas, sociais e precipitam o processo de desmatamento e perda da biodiversidade do país. Embora as medidas do governo tentam minimizar este problema e punir os responsáveis por desastres ambientais, a demanda por terra é atualmente o maior motivo das intensas ocorrências de incêndios. Além disso, os incêndios também pode ocorrer naturalmente em áreas de conservação, devido ao aumento dos eventos de seca e temperaturas elevadas.

De acordo com estatísticas do Instituto Nacional de Pesquisas Espaciais (INPE)[1] apenas no ano de 2015 o satélite de referência detectou 236.371 focos de incêndios. Destes focos, 203.411 ocorreram em áreas de conservação ambiental, sendo a maior parte no bioma Amazônia (51%) e no bioma Cerrado (37%) - ver a distribuição completa na Figura 1.1. Os surtos anuais médios detectados desde o início do monitoramento (Julho de 1998) é 175,870 focos. Estes números podem ser ainda maiores, tendo em vista que os satélites detectam somente incêndios a partir de 30 metros de comprimento por 1 metro de largura. Ou seja, focos menores não entram nesta estatística.

[1] http://www.inpe.br/queimadas/estatisticas.php

Figura 1.1: Distribuição dos focos por bioma no Brasil[2]

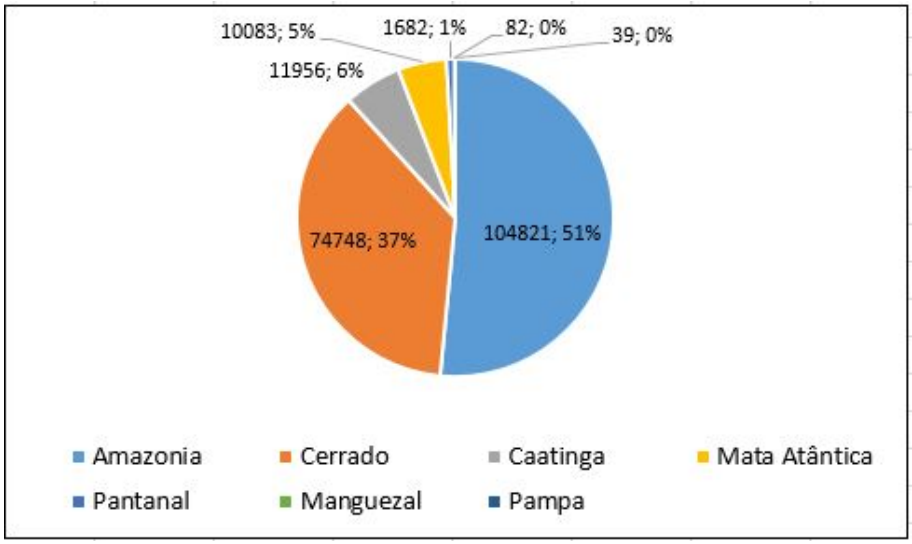
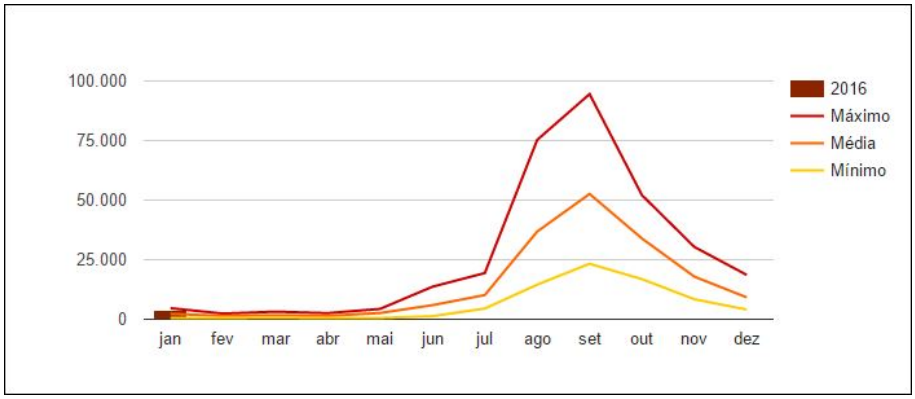


Figura 1.2 mostra uma comparação de dados do ano atual com os valores máximos, médios e mínimos, de julho de 1998 a 21 de janeiro de 2016. Durante os meses de julho a outubro percebe-se que há um maior número de focos, isto é devido à baixa umidade e altas temperaturas, muito comuns nesta época.

Figura 1.2: Comparação dos dados do ano atual com os valores máximos, médios e mínimos de 1998 até 21 de janeiro 2016.



Fonte:(INPE, 2016)

Embora existe a possibilidade de detecção de focos de incêndio por meio de satélites, são necessárias abordagens, tais como Redes de Sensores sem Fio (RSSFs), que realizam a detecção precoce dos focos e notificam rapidamente os gestores e equipes de combate da área monitorada, tendo em vista que rapidamente um foco pode tomar grandes proporções e tornar-se

[2]Dados obtidos a partir do Sistema de Informação Geográfica (GIS) e do banco de dados dos focos em áreas protegidas. Período:01/jan/2015 até 31/dez/2015. Disponível: http://www.dpi.inpe.br/proarco/bdqueimadas/bduc.php?LANGUAGE=PT

incontrolável.

As RSSFs têm sido empregadas nos mais diversos cenários, e resumem-se em uma grande quantidade de dispositivos sensores conectados por meio de um canal sem fio com o objetivo de realizar o monitoramento de uma área de forma cooperativa. Elas podem sofrer diversas mudanças durante seu tempo de vida por diversos motivos, dentre eles: (i) a destruição ou a falta de energia dos nós; (ii) por problemas na comunicação entre nós vizinhos; (iii) devido ao algoritmo de gestão da rede, como, por exemplo, nós podem desligados ou ligados de forma autônoma, seja para conservar a energia ou por causa de um nó vizinho já estar monitorando uma mesma região. Além disso, (iv) novos sensores podem ser inseridos na rede. No entanto, para gerenciar todas essas mudanças e continuar a desempenhar sua função, é essencial que uma RSSF possua mecanismos de auto-organização e tolerância a falhas em diversos níveis. Além disso, segundo (LOUREIRO et al., 2003) é peculiar de uma RSSF depender da auto-organização e de algoritmos adaptativos, visto que isso as torna menos dependentes de um controle centralizado, tornando-as mais escaláveis, adaptáveis e robustas.

1.2 Solução Proposta

Os trabalhos contemporâneos na área de RSSF, normalmente levam em consideração somente a auto-organização ou a tolerância a falhas. Quando reúnem os dois aspectos normalmente eles estão concentrados somente nas camadas de baixo nível, como, por exemplo, de roteamento. Quando trata-se do desenvolvimento de aplicações de soluções de detecção de incêndios, estas estão concentradas somente no lado servidor, desconsiderando todos aspectos anteriormente citados.

Este trabalho propõe um modelo de RSSF auto-organizável e tolerante a falhas para aplicações de detecção de incêndios em áreas de preservação ambiental. Diferentemente dos trabalhos encontrados na literatura, este modelo trata aspectos da auto-organização e tolerância a falhas tanto na camada de roteamento, quanto na camada de aplicação.

Para alcançar a auto-organização e a tolerância a falhas no nível de roteamento, aproveita-se do protocolo DAARP (VILLAS et al., 2009) com leves adaptações. Já no nível de aplicação utiliza-se de abordagens conceituadas de tolerância a falhas em software e frameworks baseados em componentes, que permitem uma abordagem autoadaptativa.

1.3 Contribuições

1. Um modelo auto-organizável e tolerante a falhas de RSSFs para detecção de incêndios;
2. Desenvolvimento de um protótipo de solução de software para detecção de incêndios;
3. Integração de trabalhos que tratam da auto-organização e tolerância a falhas em níveis diferentes;
4. Exploração de uma técnica de redundância de componentes em uma aplicação de rede de sensores;
5. Integração do Framework FlexFT (BEDER et al., 2013) ao Sinalgo (SINALGO, 2015);
6. Classificação de uma base de dados de focos de incêndios florestais;
7. Simulação sobre a área do Jardim Botânico de Brasília.

1.4 Organização da Dissertação

A dissertação é composta por 4 capítulos organizados da seguinte forma:

Capítulo 2: O capítulo trata dos principais conceitos e metodologias relacionados a este trabalho e tem por objetivo fornecer uma base teórica para o leitor. Diversos assuntos e implicações sobre computação autônoma, tolerância a falhas, RSSF e auto-organização em Redes de Sensores são tratados neste capítulo. Por fim, alguns trabalhos relacionados são descritos.

Capítulo 3: Este capítulo apresenta (i) um modelo de Rede de Sensores sem Fio auto-organizável e tolerante a falhas para detecção de incêndios; (ii) um protótipo de solução de detecção de incêndios; (iii) os detalhes de implementação e os resultados da simulação;

Capítulo 4: Este capítulo apresenta as considerações finais do trabalho, suas contribuições e trabalhos futuros.

Capítulo 2

REVISÃO BIBLIOGRÁFICA

Este capítulo tem por objetivo discutir conceitos, abordagens e técnicas no âmbito de sistemas autônomos, tolerância a falhas, redes de sensores e a auto-organização em Redes de Sensores sem Fio. Em síntese, a ideia é prover uma base teórica sobre os aspectos gerais relacionados a este trabalho.

2.1 Computação Autônoma

Sistemas autoadaptativos ou autônomos estão presentes nas mais diversas áreas. Com isso, a computação autônoma é um conceito que reúne diversas áreas da computação com o objetivo de criar sistemas computacionais de autogestão, reduzindo assim, o envolvimento humano com esses sistemas (HUEBSCHER; MCCANN, 2008).

Segundo Huebscher e McCann (2008), o termo ”autônomo” surgiu na biologia, pois é o sistema nervoso autônomo que cuida dos reflexos inconscientes do corpo humano, ou seja, ele é responsável por administrar todas funções corporais que não requerem tanto a nossa atenção. Um exemplo desses reflexos inconscientes são os ajustes corporais, tais como: tamanho da pupila, funções digestivas do estômago e dos intestinos, taxa e profundidade da respiração, e a dilatação ou constrição dos vasos sanguíneos. O ser humano estaria constantemente ocupado sem a atuação do sistema nervoso autônomo, tendo em vista que seria preciso, de modo consciente, adaptar o corpo às suas necessidades e ao meio ambiente.

Os sistemas de computação chegaram a um nível de complexidade, onde o esforço humano necessário para manter os sistemas em funcionamento e operacionais têm sido muito grande (HUEBSCHER; MCCANN, 2008). Um problema semelhante é o da telefonia em 1920, onde os quadros de distribuição eram manuais. Com o aumento do setor e a falta de pessoas treinadas

para operar esses quadros de distribuição, foi preciso introduzir centrais telefônicas automáticas, eliminando a intervenção humana.

2.1.1 Situational Awareness System (SAS)

Um dos primeiros notáveis sistemas de autogestão foi iniciado pela DARPA para uma aplicação militar em 1977 e foi chamado de *Situational Awareness System (SAS)*, que fazia parte do maior programa de pequenas unidades operacionais *Small Units Operations (SUO)*.

Segundo Huebscher e McCann (2008), o objetivo da aplicação era desenvolver uma comunicação pessoal e dispositivos de localização para os soldados no campo de batalha. O *SAS* permitia que os militares gerassem relatórios de *status* em seus dispositivos pessoais, como por exemplo, a descoberta de tanques inimigos. Essa informação se espalhava de forma autônoma a toda tropa, ou seja, todos tinham o mais recente relatório da situação ao entrar em uma área inimiga. Dentre os dados coletados e transmitidos estavam mensagens de voz, dados de sensores terrestres autônomos e veículos aéreos não tripulados (VANTs).

Mesmo em péssimas condições ambientais os dispositivos pessoais precisavam ser capazes de se comunicar uns com os outros, e além disso, era preciso ficar atento para que os dados não fossem interceptados pelo inimigo. Por meio de múltiplos saltos (*multi-hop*) a mensagem chegaria até o último ponto, desde que neste tipo de comunicação, um dispositivo somente enviaria seus dados para os vizinhos mais próximos, que iam fazendo o mesmo, até que todos recebessem os dados (HUEBSCHER; MCCANN, 2008).

Para Huebscher e McCann (2008) este é um modo de roteamento adaptativo descentralizado ponto a ponto móvel, que tem mostrado ser um problema de autogestão, já que, neste sistema, a intenção era manter a latência abaixo de 200 milissegundos, a partir do momento que um soldado começasse a falar até que a mensagem fosse recebida.

2.1.2 Propriedades de autogestão

A IBM sugere, em seu manifesto (HORN, 2001), que sistemas computacionais complexos também devem ter propriedades autonômicas, ou seja, devem ser capazes de tratar de modo independente as tarefas regulares de manutenção e otimização, reduzindo assim a carga sobre os administradores de sistemas. Assim, é definido quatro propriedades de autogestão, a saber:

Autoconfiguração: Um sistema computacional autônomo configura-se conforme suas metas de alto nível, ou seja, especificando o que é desejado, não necessariamente como

realizá-lo. Isto pode significar ser capaz de instalar e configurar-se com base nas necessidades da plataforma e do usuário (KEPHART; CHESS, 2003).

Auto-otimização: Um sistema de computação autônomo aprimora o uso de recursos. Com objetivo de melhorar o desempenho ou a qualidade do serviço, proativamente, ele pode decidir iniciar uma mudança (KEPHART; CHESS, 2003).

Autocura: Um sistema de computação autônomo detecta e realiza o diagnóstico dos problemas, que podem ser de baixo nível, como uma falha de hardware, ou alto nível, como uma falha de software. Embora o sistema deva tentar resolver o problema, como por exemplo, instalar automaticamente atualizações no sistema, é importante que o sistema não seja mais prejudicado nesse processo de cura, como por exemplo, a introdução de novos erros. Por isso um aspecto importante neste processo, é tolerância a falhas, ou seja, um sistema autônomo é dito ser reativo para falhas ou aos primeiros sinais de uma eventual falha (KEPHART; CHESS, 2003).

Autoproteção: O sistema automaticamente precisa se ajustar com objetivo de alcançar segurança, privacidade e proteção de dados, tendo em vista que a segurança é um aspecto crucial da autoproteção, tanto em software como no hardware. Ele também pode ser capaz de prever e impedir possíveis violações de segurança, uma vez que, desse modo, o sistema nervoso autônomo apresenta propriedades proativas (KEPHART; CHESS, 2003).

Segundo Huebscher e McCann (2008) as propriedades citadas acima são inspiradas nas características de agentes de *software*, que Wooldridge e Jennings (1995) apontaram como sendo:

Autonomia: Agentes atuam sem a mediação de humanos ou outros, e possuem controle sobre suas atividades e estado interno.

Habilidade social: Agentes exercem interação uns com os outros, até mesmo com humanos, por meio de alguma linguagem de comunicação.

Reatividade: Agentes constatam seu ambiente e rapidamente respondem às transformações que ocorrem nele.

Proatividade: Agentes tomam iniciativa, pois são capazes de apresentar um comportamento dirigido a objetivos.

Ainda há discussões sobre o que a autogestão de sistemas realmente representa, como por exemplo, sabe-se que Sistemas Gerenciadores de Banco de Dados (SGBDs) ou que sistemas

operacionais são capazes de se auto gerir, contudo a comunidade de sistemas de autogestão está chegando a um consenso que o termo computação autônoma não está sendo usado para descrever esses sistemas, mas sim aqueles em que a gestão de recursos ou a tomada de decisões são realizadas levando em consideração o contexto atual em que o sistema está inserido (HUEBSCHER; MCCANN, 2008).

Sistemas autoadaptativos contém alguns elementos dessas propriedades citadas por Wooldridge e Jennings (1995) e Huebscher e McCann (2008), especialmente para proporcionar a auto-otimização. Um exemplo pode ser visto em sistemas de *streaming* de mídia, em que o objetivo é manter a reprodução de música ou vídeo em uma qualidade tão alta quanto possível, mas adaptando-se à largura da banda.

2.1.3 Graus de automaticidade

Em IBM (2002) é proposto um conjunto de níveis de computação autônoma, que se estende do nível 1: básico, para o nível 5: autônomo.

1. Este primeiro nível define o estado em que elementos do sistema são geridos por pessoas qualificadas, que fazem uso de ferramentas de monitoramento, e posteriormente implementam manualmente as alterações. Acredita-se que a maioria dos sistemas de TI estão neste nível atualmente.

2. Conhecido como *Managed* ou Gerenciado, neste nível a coleta de informações é feita de forma inteligente, por meio de ferramentas de monitoramento, diminuindo a carga da administração de sistemas.

3. Já no nível 3 é realizado o reconhecimento de padrões de comportamento do sistema e a partir disso, são sugeridas ações a serem aprovadas e realizadas pela equipe de TI.

4. Neste nível, conhecido como nível de adaptação, o sistema faz uso das ferramentas disponíveis no nível 3, mas possui maior autonomia, sendo capaz tomar decisões. É reduzida a interação humana, e espera-se que o sistema realize o serviço com bom desempenho.

5. Já no nível autônomo completo, os componentes são geridos de forma dinâmica por regras e políticas de negócios, permitindo que a equipe fique focada nas necessidades de negócios.

2.2 Dimensões da Modelagem

Segundo Cheng et al. (2009) é preciso estabelecer as bases que permitem um desenvolvimento sistemático de sistemas autoadaptativos no futuro, e abordagens de engenharia de software adequadas para prover a autoadaptação.

A vista disso, Andersson et al. (2009) fornecem uma classificação das dimensões da modelagem, com o objetivo de resolver a dificuldade em representar a autoadaptação e estabelecer um apoio a partir do qual os aspectos-chaves de diferentes sistemas autoadaptativos podem ser identificados e comparados com mais facilidade.

Cada dimensão representa um aspecto particular do sistema que é relevante para autoadaptação, e elas estão divididas em quatro grupos que serão detalhados nesta seção.

2.2.1 Grupo *Goals*

Goals são os objetivos que o sistema deve alcançar, eles podem estar relacionados com o tempo de vida do sistema ou com os cenários relacionados a ele. Além disso, podem refletir os aspectos de autoadaptabilidade (ANDERSSON et al., 2009). De modo geral, as dimensões deste grupo tratam aspectos relacionados com a evolução, flexibilidade, duração, número de metas associadas e o nível acoplamento das mesmas, ou seja, o quanto uma meta depende da outra.

Evolução: Captura se as metas do sistema podem mudar durante o tempo de vida do sistema;

Flexibilidade: Captura se os objetivos são flexíveis na forma como são expressos;

Duração: Esta dimensão está preocupada com a validade de uma meta em todo o tempo de vida do sistema. Pode variar de temporário para persistente;

Multiplicidade: Esta dimensão está relacionada com o número de metas associadas com os aspectos da autoadaptabilidade de um sistema. Um sistema pode ter um único objetivo ou objetivos múltiplos;

Dependência: No caso de múltiplos objetivos, esta dimensão captura se eles estão relacionados uns com os outros. Eles podem ser independentes (não afetam uns aos outros) ou dependentes (podem ser complementares, ou até mesmo conflitantes).

2.2.2 Grupo *Change*

Segundo Andersson et al. (2009) o sistema deve-se autoadaptar quando há mudanças nos requisitos ou no ambiente em que o sistema está implantado. Essas alterações podem ser classificadas em termos de lugar, tipo, frequência, e se ela pode ou não ser prevista. Esses elementos do grupo *Change* são importantes, pois ajudam a identificar como o sistema deve reagir às mudanças que sucedem durante o tempo de execução.

Fonte: Esta dimensão identifica a origem da mudança, que pode ser tanto externa (ou seja, o seu meio ambiente) ou interna para o sistema, de acordo com o âmbito do sistema;

Tipo: Esta dimensão refere-se à natureza da mudança. Ela pode ser funcional, não funcional e tecnológica. Um exemplo de uma mudança funcional é quando o propósito do sistema muda e, consequentemente, o sistema precisa refletir esta mudança. Já a não funcional é quando, por exemplo o desempenho e a confiabilidade precisam ser melhorados. Já a mudança de origem tecnológica se refere a aspectos de hardware e software, como por exemplo, a plataforma que o sistema utiliza foi atualizada;

Frequência: Esta dimensão refere-se com a quantidade de vezes que ocorre uma determinada mudança, e pode variar de raro a frequente. Se, por exemplo, uma mudança acontece muitas vezes, isso pode afetar a capacidade de resposta da adaptação. Considera-se que mudanças não ocorrem frequentemente;

Antecipação: Nesta dimensão captura-se alterações previstas de antemão. Diferentes técnicas de autoadaptação são necessárias, dependendo do grau de antecipação: prevista (cuidado), previsível (prevista para um momento futuro), e imprevista (não planejado).

2.2.3 Grupo *Mechanisms*

As dimensões do grupo *Mechanisms* referem-se ao tipo de autoadaptação que é esperado, o nível de autonomia, como ela é controlada e o impacto dela em termos de tempo e espaço (ANDERSSON et al., 2009).

Tipo: Esta dimensão captura se a adaptação está relacionada com os parâmetros dos componentes do sistema ou com a estrutura do sistema. Com base nisso, a adaptação pode ser paramétrica ou estrutural, ou uma combinação destes. A adaptação estrutural também pode ser vista como composição, uma vez que depende de como os componentes são integrados;

Autonomia: Esta dimensão identifica o grau de intervenção externa durante a adaptação. O alcance desta dimensão vai de autônoma para assistida;

Organização: Essa dimensão capta se a adaptação é centralizada, e portanto, realizada por um único componente ou distribuída entre os vários componentes - descentralizada. Se a adaptação é descentralizada nenhum componente tem um controle total sobre o sistema;

Escopo: Esta dimensão identifica se a adaptação é localizada ou envolve todo o sistema;

Duração: Esta dimensão refere-se ao período de tempo em que o sistema encontra-se adaptando, ou seja, quanto tempo dura a adaptação;

Pontualidade: Esta dimensão captura se o período de tempo para realizar a adaptação é garantido;

Disparo: Esta dimensão identifica se a mudança que inicia a adaptação é um *event-trigger* ou *time-trigger*. Embora seja difícil de controlar como e quando ocorre a mudança, é possível controlar como e quando a adaptação deve reagir a uma certa alteração.

2.2.4 Grupo *Effects*

As dimensões do grupo *Effects* capturam o impacto da adaptação sobre o sistema e lidam com os efeitos dessa adaptação. Elas referem-se ao grau de importância da adaptação, seus impactos e consequências, e a capacidade de voltar ao estado anterior (ANDERSSON et al., 2009).

Criticidade: Esta dimensão captura o impacto sobre o sistema no caso da auto adaptação falhar;

Previsibilidade: Esta dimensão identifica se as consequências da autoadaptação podem ser previsíveis, tanto em valor, quanto em tempo;

Sobrecarga: Esta dimensão captura o impacto negativo da adaptação no desempenho do sistema;

Resiliência: Esta dimensão captura se o sistema é capaz de voltar ao seu estado natural, após a mudança. Em outras palavras, verifica se a entrega dos serviços continua confiável durante e após o processo de mudança.

Segundo Andersson et al. (2009) as dimensões de modelagem apresentadas aplicam-se a qualquer sistema de software autoadaptativo e são úteis em várias situações de desenvolvimento diferentes. Elas podem serem usadas como um guia para a engenharia de software tradicional, mas também podem ser utilizada no contexto de engenharia reversa, onde são compreendidas soluções já implementadas. Resumindo, o intuito dessa classificação é a criação de um vocabulário que pode ser usado por uma equipe de projeto. A tabela 2.1 apresenta um resumo de todas as dimensões em seus determinados grupos.

Tabela 2.1: Dimensões da Modelagem

Dimensões	**Grau**	**Definições**
Goals - são os objetivos que o sistema deve alcançar		
Evolução	estático ou dinâmico	se as metas podem mudar durante o tempo de vida do sistema.
Flexibilidade	rígido, constrangido, irrestrito	se os objetivos são flexíveis na forma como são expressos.
Duração	temporário ou persistente	validade de uma meta em todo tempo de vida do sistema.
Multiplicidade	único ou múltiplos	há muitos objetivos?
Dependência	independente ou dependente	como os objetivos estão relacionados entre si.
Change - mudanças são a causa da adaptação.		
Fonte	externa (ambiente) ou interna (aplicação, *middleware*, infraestrutura)	onde é a origem da mudança?
Tipo	funcional, não funcional e tecnológica	qual o tipo da mudança?
Frequência	raro ou frequente	quantas vezes uma determinada mudança ocorre?
Antecipação	previsto, previsível, imprevisto	se a mudança pode ser prevista
Mechanisms - qual a reação do sistema à mudança		
Tipo	paramétrico ou estrutural	se a adaptação está relacionada com os parâmetros dos componentes do sistema ou com a estrutura do sistema.

Tabela 2.1 – continua na próxima página

Tabela 2.1 – continuação da página anterior

Dimensões	Grau	Definições
Autonomia	autônomo ou assistido (sistema ou humano)	qual o grau de intervenção externa durante a adaptação.
Organização	centralizado ou descentralizado	se a adaptação é feita por um único componente ou distribuída entre vários componentes
Escopo	local ou global	se a adaptação é localizada ou se envolve todo sistema.
Duração	curto, médio ou longo prazo	quanto tempo dura a adaptação.
Pontualidade	melhor esforço ou garantido	se o período de tempo para a realização da autoadaptação pode ser garantido.
Disparo	evento acionador ou tempo acionador	se a mudança que desencadeia adaptação está associada com um evento ou um intervalo de tempo.
Effects - qual o impacto da adaptação sobre o sistema		
Criticidade	inofensiva, missão crítica, segurança crítica	impacto sobre o sistema caso a adaptação falhar
Previsibilidade	não determinístico ou determinístico	se a consequência da autoadaptação podem ser prevista
Sobrecarga	insignificante ou falha	o impacto da autoadaptação do sistema sobre a qualidade dos serviços do sistema.
Resiliência	resiliente ou vulnerável	se a entrega dos serviços continuam confiáveis diante do processo de mudança.

Fonte: (ANDERSSON et al., 2009)

2.3 Rede de Sensores sem Fio

Com o avanço das tecnologias nas áreas da eletrônica digital, dos sistemas microeletromecânicos e da comunicação sem fio, tem-se visto muitos estudos e aplicações de sensoriamento remoto, sendo que, uma boa parte estão concentrados no desenvolvimento de sensores

"inteligentes" e de Rede de Sensores sem Fio (RSSFs).

Esta seção tem por objetivo realizar uma introdução dos principais conceitos, abordagens e características de RSSF. Para isso, nos baseamos nos seguintes textos de Akyildiz et al. (2002), Loureiro et al. (2003), Lewis et al. (2004), Raghavendra, Sivalingam e Znati (2006), Samuel Madden and Kay Romer and Holger Karl and Friedemann Mattern (2006), Stankovic (2008), Akyildiz e Vuran (2010) e Yang (2014).

Segundo Loureiro et al. (2003), sensores inteligentes, de modo geral, são chips que contém um ou mais sensores com capacidade de processamento de sinais e de comunicação de dados. Diferentemente das redes de computadores tradicionais, as RSSFs normalmente possuem um grande número de nós distribuídos, sendo que cada nó pode ser equipado com uma variedade de sensores, tais como acústico e sísmico, infravermelho, câmera de vídeo, temperatura e pressão. E devido a problemas como limitação de energia, falhas de comunicação e perda de nós, eles devem possuir mecanismos de autoconfiguração e adaptação. Para executar as tarefas definidas pela rede, as RSSFs requerem um alto grau de cooperação e tendem a ser autônomas, ou seja, é necessário que algoritmos e protocolos de comunicação tradicionais sejam revistos, considerando este contexto.

Os nós podem trabalhar em grupos, onde pelo menos um dos sensores deve ser capaz de perceber uma ocorrência na região, processá-la e decidir se envia ou não o resultado para os outros nós na rede.

Diferentemente das redes tradicionais, onde a comunicação entre os elementos computacionais necessita de uma infraestrutura de comunicação, em uma RSSF, os nós trocam dados diretamente entre si, por meio de enlaces de comunicação, como em uma rede móvel *ad hoc* (*MANET - Mobile Ad hoc Network*). Embora *MANETs* e RSSFs sejam idênticas do ponto de vista da organização, uma função básica das *MANETs* é oferecer suporte à comunicação entre os elementos, que podem executar diferentes tarefas. Já as RSSFs são propensas a executar uma única tarefa de forma colaborativa, na qual os sensores fornecem dados, que são processados por nós sorvedouros, chamados de *sink nodes*.

As RSSFs podem ser homogêneas ou heterogêneas em relação aos tipos, dimensões e funcionalidades dos nós. Com isso, é possível desenvolver várias aplicações utilizando um ou vários tipos de nós sensores, como por exemplo, em uma aplicação de monitoramento de segurança, podem ser usados sensores de imagem e acústicos, que juntos capturam imagens, áudios e vídeos. Estes sensores podem estar inseridos em um único nó ou separadamente.

De modo geral, as RSSFs podem ser utilizadas para monitoramento e segurança de ambi-

entes internos e externos, controle, atuação e manutenção de sistemas complexos. Com isso, é possível perceber a presença dessas redes nas mais diversas áreas. Loureiro et al. (2003) apresenta algumas delas:

Controle: Para fornecer mecanismos de controle, como por exemplo, em ambientes industriais, em que sensores sem fio podem ser colocados em itens de uma linha de montagem para testar o processo de produção;

Ambiente: Para observar variáveis em diversos ambientes, sejam eles internos ou externos, como casas, prédios e florestas;

Tráfego: Para monitorar trafego de pessoas e veículos em determinados locais, como ruas e avenidas de grande movimentação;

Segurança: Para monitorar e oferecer certa segurança em locais de grande concentração de pessoas, como exposições, shows, shoppings, estacionamentos.

Medicina e Biologia: Para monitorar o funcionamento de partes do corpo e detectar possíveis falhas, como aumento de pressão arterial, níveis de açúcar, etc;

Militar: Para detectar presença de inimigos, materiais perigosos, explosões, dentre outras aplicações.

Como citado, as tarefas de uma RSSFs tendem a ser realizadas de forma colaborativa e de modo geral, seus objetivos dependem da aplicação. Com isso, diversas atividades são frequentemente encontradas neste tipo de rede, como por exemplo: determinar o valor de algum parâmetro num dado local; detectar a ocorrência de eventos de interesse e estimar valores de parâmetros em função do evento detectado; classificar um objeto detectado; e rastrear um objeto.

2.3.1 Características das Redes de Sensores sem Fio

RSSFs possuem peculiaridades que são apropriadas às áreas em que são aplicadas. Com isso, para melhor entendimento, se faz necessário discutir as principais características específicas deste tipo de rede.

Os sensores não necessariamente precisam ser endereçados unicamente, visto que a necessidade de se endereçar depende da aplicação, como por exemplo: uma aplicação que monitora

o ambiente de uma região externa. Neste caso, o mais interessante é saber o valor de uma determinada variável na região, ou seja, provavelmente os sensores não precisam ser endereçados unicamente para esta aplicação. Já em aplicações em que se deseja saber onde o dado está sendo coletado, como por exemplo, em uma linha de montagem, é necessário que os sensores sejam endereçados unicamente. Outro fator importante é que, dependendo da aplicação, os nós podem ser móveis ou estáticos.

As RSSFs podem ter uma funcionalidade chamada agregação de dados, isto é, a habilidade da rede em agregar ou sumarizar os dados coletados por meio dos sensores, antes dos dados chegarem na estação base. Assim sendo, é possível que a quantidade de mensagens transmitidas pela rede seja reduzida.

Outra especificidade é que os dados coletados por uma RSSFs podem ser restritos, como por exemplo, um período de tempo máximo para enviar seus dados para uma entidade de supervisão.

No caso de aplicações ambientais, como por exemplo, monitoramento de florestas, são previstos cerca de 10 mil a 100 mil sensores. Segundo Akyildiz et al. (2002) o número de sensores está associado com o contexto da aplicação e escalabilidade é uma propriedade importante para estabelecer protocolos de RSSF, tanto para transmissão de dados, quanto para gestão do consumo de energia, de modo que a rede mantenha suas funcionalidades independente da quantidade de nós sensores participantes da rede.

A escalabilidade retrata como muitos nós comunicando-se, são suportados pela rede, tendo como pior caso, um aumento linear no consumo de recursos. Conforme o cenário de aplicação, a escalabilidade pode ser o agente limitador em redes de sensores, como por exemplo, em redes com um grande número de nós (DRESSLER, 2007).

Já tempo de vida de uma rede de sensores nada mais é que a capacidade da rede em cumprir os requisitos da aplicação. De modo geral, o fato de que alguns nós podem falhar é ignorado, enquanto o objetivo geral é sempre respeitado. Além do mais, este parâmetro trata a questão de saber se a rede de sensores está sempre disponível, ou se existem períodos especiais de manutenção (DRESSLER, 2007).

Considerando que em muitas aplicações os sensores serão colocados em locais de difícil acesso, o que pode dificultar a manutenção, durante a escolha ou desenvolvimento de aplicações e protocolos para RSSFs, deve-se considerar o consumo, o modelo e o mapa de energia da rede, uma vez que o tempo de vida do sensor, e consequentemente da rede, dependem da quantidade de energia disponível.

Contudo, o modelo de energia é composto pelos recursos físicos específicos de um sensor

que utilizam energia. Ou seja, pode ser entendido como um provedor de energia para elementos consumidores, que são subordinados a uma bateria com capacidade limitada de energia. Esses elementos são os modelos de rádio, os sensores e o processador. Assim, cada item informa seu consumo de energia ao provedor, que, informa a quantidade de energia disponível para consumo. A Tabela 2.2 apresenta os elementos que compõem o modelo de energia.

Tabela 2.2: Elementos que compõem o modelo de energia de um sensor

Elemento	Descrição
Bateria	armazena a energia do nó sensor, que possui uma taxa de consumo e uma capacidade finita.
Rádio	é todo o sistema de transmissão e recepção, amplificador e antena. O consumo de energia depende da ação que é realizada, sendo que, normalmente a transmissão dos dados consome mais energia que a recepção.
Processador	é o elemento de processamento central do nó sensor, onde o consumo depende da velocidade do relógio e do modo de operação, sendo que, quanto menor a frequência menos energia é consumida. O consumo pode ser calculado pelo número de ciclos do relógio para tarefas distintas, como por exemplo, processamento de sinais, verificação de erro, etc. Este modelo é empregado em todos procedimentos que fazem parte do modelo de sensor.
Sensores	representa os dispositivos de sensoriamento, onde o consumo depende do modo de operação e da extensão medida.

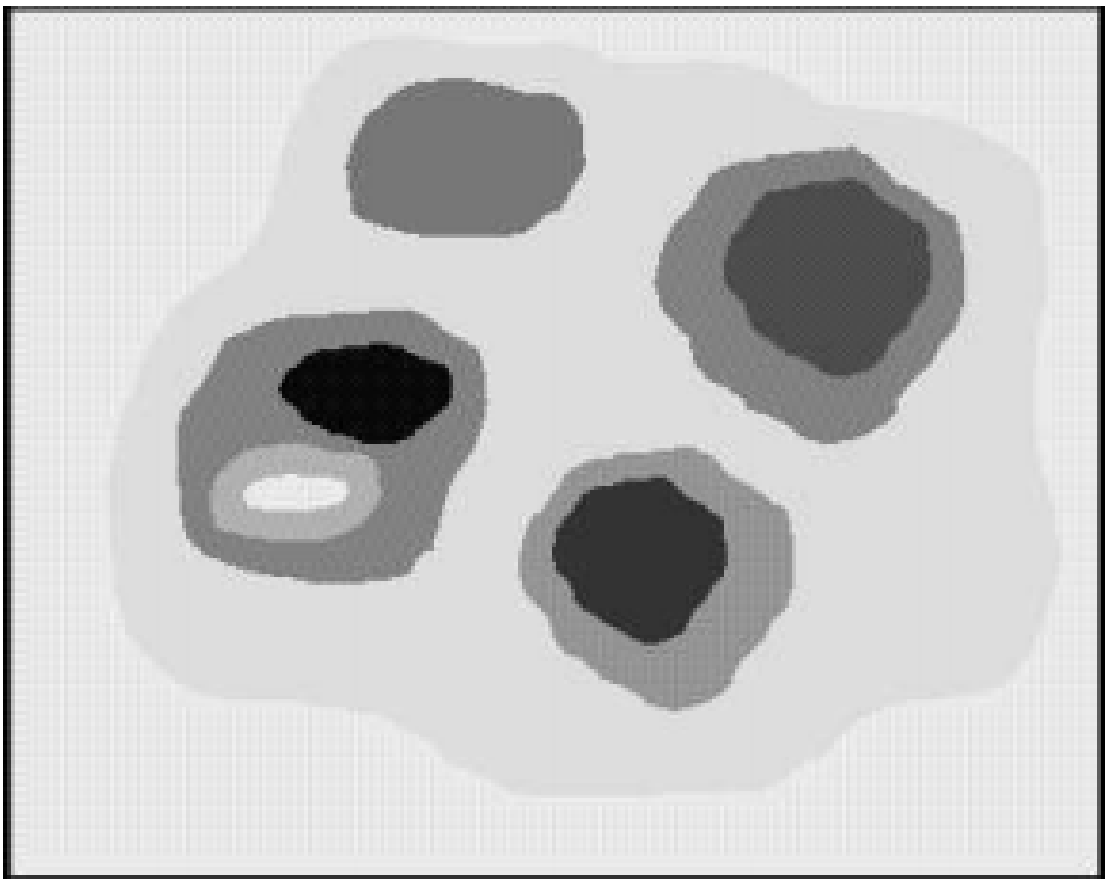
Fonte: (LOUREIRO et al., 2003)

Sabe-se que por meio de um processo de obtenção do modelo de energia de cada nodo, é possível fazer um levantamento do mapa de energia de toda rede, que uma vez obtido, pode ser usado para tomar decisões mais apropriadas, do que deve ou pode ser feito na rede. A figura 2.1 mostra um exemplo de um mapa de energia. Neste caso, quanto mais escura área do mapa, maior o nível de energia na área.

RSSFs podem sofrer várias mudanças, que podem ser causadas pela destruição ou falta de energia dos nós; por problemas no canal de comunicação entre os sensores; e pelo algoritmo de gerenciamento da rede, como por exemplo, para economizar energia ou por ter outro sensor que já coleta o mesmo dado desejado na mesma região. Além disso, sensores inativos podem tornar-se ativos e novos sensores podem ser inseridos na rede. Contudo, para gerenciamento de todas essas mudanças e para que a rede continue a exercer sua função, é necessário que haja mecanismos de auto-organização.

Por fim, entende-se que o objetivo primordial de uma RSSFs é realizar uma tarefa de forma colaborativa, na qual é necessário detectar e prever eventos de interesse e não somente prover mecanismos de comunicação. Por causa das limitações da rede, os dados geralmente são

Figura 2.1: Mapa de energia de uma rede



Fonte: (LOUREIRO et al., 2003)

agregados, com o objetivo de melhorar o desempenho da detecção de eventos, sendo que este processo de agregação depende da aplicação que está sendo executada. Além disso, dependendo do grau da agregação, pode não ser viável a transmissão dos dados até o nó sorvedouro, sendo necessário estabelecer outros nós sorvedouros, que coletam os dados de uma área e respondem as consultas referentes aos nós sob sua responsabilidade.

2.3.2 Componentes de uma RSSF

São muitos os elementos que podem fazer parte de uma RSSF, portanto, o principal objetivo desta seção é descrever os componentes mais relevantes deste tipo de rede, que são os nós sensores e os nós de interface com outras redes.

Nós sensores

Loureiro et al. (2003) define sensores como dispositivos autônomos capazes de sensoriar, processar e comunicar, que quando dispostos em rede, em modo *ad hoc*, estabelecem a rede de sensores.

Cada nó obtém os dados por meio de sensores, e pode processá-los localmente ou entre os nós vizinhos. Após processar, a informação é enviada geralmente para um *data sink*, que é o nó responsável por receber os dados de toda rede e se comunicar com o gerenciador de tarefas, por meio de uma interface de comunicação, como, por exemplo, utilizando internet ou satélite (subseção 2.3.2). Assim sendo, percebe-se que um nó possui tarefas distintas na rede, como sensoriamento do recinto, processamento dos dados e a incumbência de retransmitir a informação, por meio de múltiplos saltos(*multihop*). Com a utilização de *multihop* o consumo de energia durante a transmissão é menor, pois não é preciso que todos os nós da rede transmitam os dados diretamente ao nó *sink*, já que só precisam transmitir aos seus vizinhos, que fazem o mesmo, até que o *sink* seja alcançado (TELECO, 2014).

Os componentes básicos de um nó sensor são: transceptor, memória, processador, sensores e bateria. Contudo, os nós normalmente possuem características (físicas e teóricas) diferentes, tendo em vista que diversos modelos, de vários níveis de complexidade, podem ser construídos levando em consideração as necessidades das aplicações e as peculiaridades do dispositivo.

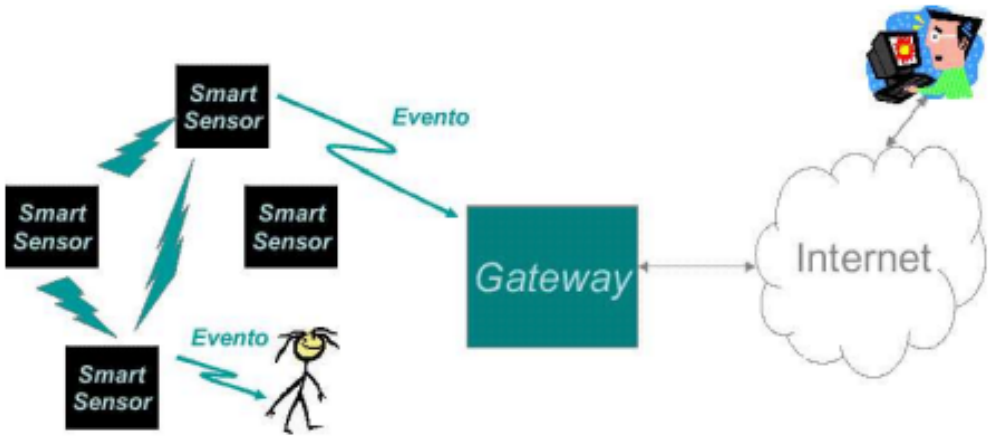
Todavia, duas características são comuns em grande parte dos modelos, uma é que quando a distância aumenta, a habilidade de sensoriamento diminui; outra é que a capacidade em sensoriar pode melhorar com o tempo de exposição, devido aos efeitos decrescentes dos ruídos nas medições.

Há também nodos denominados de atuadores, onde sua função é modificar valores, com a finalidade de corrigir falhas e controlar o objeto monitorado. Já quando um nodo possui ambas funções (sensoriar e atuar), ele é chamado de transdutor.

Nós de Interface com Outras Redes

É por meio de nós chamados *gateways* que as RSSFs se comunicam com outro tipo de rede. As mensagens percorrem a rede de sensores até chegar a um *gateway* que as encaminha, por uma rede como a Internet, até ao computador em que a aplicação está sendo executada. A Figura 2.2 apresenta um modelo genérico de uma RSSF conectada a uma rede fixa por meio de um nó *gateway*.

Figura 2.2: Modelo genérico de RSSF com um nodo *gateway*

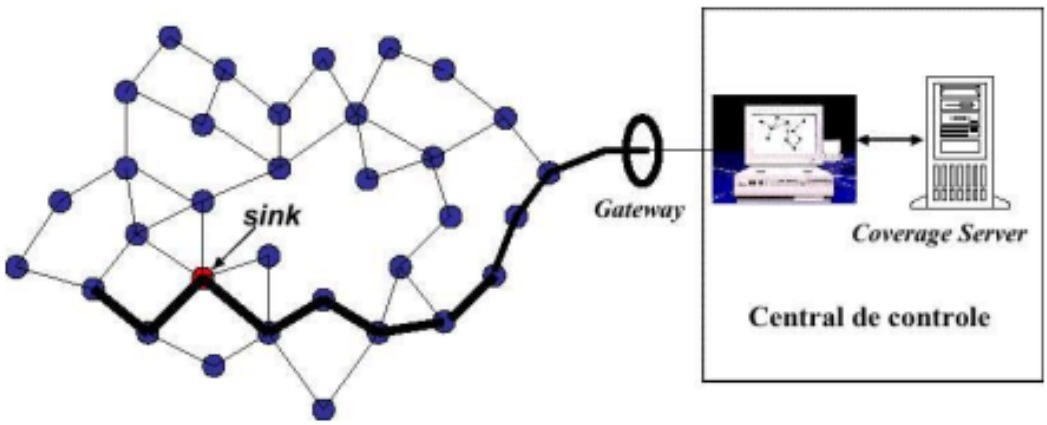


.

Fonte: (LOUREIRO et al., 2003)

Já a Figura 2.3 mostra uma RSSFs que também possui um nó *sink*, com objetivo de mostrar que são elementos diferentes.

Figura 2.3: Modelo de RSSF com um nodo *sink*



Fonte: (LOUREIRO et al., 2003)

2.3.3 Modelo Funcional de RSSF

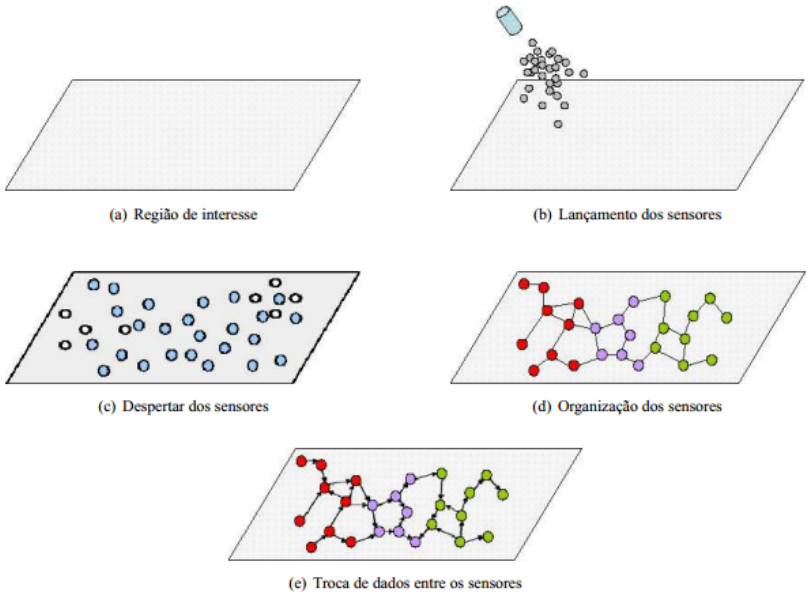
As funcionalidades de uma RSSF podem ser categorizadas em cinco fases, que podem ocorrer simultaneamente e estar ativas em diferentes momentos do tempo de vida da rede, são elas: estabelecimento da rede, manutenção, sensoriamento, processamento e comunicação (RUIZ, 2002).

Estabelecimento

O estabelecimento da RSSF compreende atividades de disposição dos nós e formação da rede, como mostrado na Figura 2.4. Geralmente, os nós são lançados sobre a área aleatoriamente e despertam para a formação da rede. Contudo, os nós podem realizar atividades de descobrimento da localização ou de formação de clusters, antes mesmo de iniciarem as operações de sensoriamento.

Redes de sensores são sistemas auto-organizados compostos por nós sensores que naturalmente podem estabelecer uma rede não premeditada, ajuntando-se e adaptando-se dinamicamente quando ocorrem falhas ou mesmo a destruição dos nós, gerindo a movimentação dos dispositivos e lidando com as mudanças de funcionalidade e requisitos da rede. Além do mais, nós também se organizam para aproveitar a redundância, causada pela alta densidade, a fim de estender o tempo de vida da rede (LOUREIRO et al., 2003).

Figura 2.4: Estabelecimento da RSSF



Fonte: (LOUREIRO et al., 2003)

Manutenção

A manutenção ocorre durante todo tempo de vida da rede e é útil em todas as fases citadas anteriormente. Sua meta é manter os requisitos da aplicação, diminuir a imprevisibilidade e estender o tempo de vida da rede, uma vez que com o tempo, alguns nós podem atingem níveis de energia que comprometem o seu funcionamento, de forma parcial ou total. Dessa forma, a

manutenção pode ser reativa, preventiva, corretiva ou adaptativa aos eventos que possam ocorrer.

Como já dito, as diversas fases da rede não necessariamente seguem uma ordem, e isto fica evidente quando acontecem falhas, que não são exceções. Com isso, mesmo que os nós sejam estáticos, acaba resultando em uma topologia de rede dinâmica. Portanto, mecanismos de manutenção precisam ser propostos, e essa manutenção pode demandar uma nova distribuição de nós, e consequentemente, uma nova organização da rede.

Sensoriamento

As tarefas de sensoriamento são associadas com a compreensão do ambiente e a coleta de dados, ou seja, dependendo da aplicação e os sensores compreendidos, essa fase inclui vários aspectos, como determinar a distância do alvo, os ruídos do ambiente, o tipo do dado coletado, a dimensão da informação e a frequência da amostragem. Do mesmo modo é a delimitação das áreas de sobreposição dos nós, visto que se há vários sensores e existe uma interseção na área de percepção, pode ocorrer duplicidade de dados, com isso, é necessário que haja uma correlação de informações antes da transmissão ou a alteração do estado de um dos sensores, de modo que apenas um transmita os dados.

Coleta de Dados: Sabe-se que a finalidade de uma RSSF é coletar dados de uma determinada região, processar e transmitir a informação, normalmente para um nó *sink* ou uma estação base. Assim, essa tarefa compreende o cálculo da área de alcance dos sensores e a exposição deles sobre os alvos.

Loureiro et al. (2003) define a exposição como a integral de uma função de sensoriamento, que depende da distância dos sensores sobre um caminho de um ponto inicial *ps* a um ponto final *pf*. Sendo que os parâmetros da função de sensoriamento dependem da natureza do dispositivo sensor. Contudo, informalmente, a exposição pode ser entendida como a capacidade de perceber um alvo na área atendida pelo sensor.

Sensoriamento Distribuído: Redes de sensores podem fazer o monitoramento de uma determinada área de dois modos: utilizando um sensor centralizado, como satélite ou radar, ou por meio de sensoriamento distribuído. A segunda maneira tem apresentado muitas vantagens, como propiciar maior tolerância a falhas por meio de redundância; atender uma grande área por meio da junção de muitos sensores; aprimorar a performance do sensoriamento com sensores variados; exceder os efeitos do ambiente posicionando os

nodos perto dos alvo; assegurar a qualidade do sensoriamento por meio da combinação de dados de diferentes perspectivas, dentre outras.

Processamento

Sabe-se que nas rede de sensores temos o processamento de suporte e o da informação. O primeiro é todo processamento funcional dos nós, isto é, todo processamento relacionado com a gestão, comunicação e a manutenção da rede, como as operações que implicam com os protocolos. Já no processamento da informação, os dados recebidos pelo nó podem ser processados em função da aplicação e/ou do envolvimento do sensor em relações de colaboração. Dependendo do que se deseja, os dados podem ser submetidos a compressão, criptografia, equiparação, e outros.

Comunicação

Diferentemente de outras redes, como *ad hoc* ou infraestruturada, nas RSSFs a topologia da rede é altamente variável devido ao recurso limitado de energia. O envio da informação envolve múltiplos saltos por meio de outros nós, em razão da limitação do alcance de transmissão. Além disso, as condições de ruído podem prejudicar o sensoriamento e a comunicação entre os nós, denotando um gasto desnecessário de energia.

Por fim, a Tabela 2.3 resume alguns requisitos das tarefas apresentadas no modelo funcional.

Tabela 2.3: Requisitos das tarefas do modelo funcional

Tarefas	Requisitos
Estabelecimento da Rede	distribuição dos nodos, despertar dos nós, dimensões envolvidas, densidades, tipos de sensores, área de cobertura, organização, topologia, conectividade, etc.
Manutenção	correção das situações de anormalidade provocadas por falhas nos nós, adaptação às condições de energia da rede, chegada de novos e/ou nós diferentes, etc.
Sensoriamento	tempo de exposição do alvo, tipos de dados, largura de banda e frequência de atualização.
Processamento	algoritmos de controle, compressão, segurança, criptografia, codificação e correção de erro.
Comunicação	quais as possíveis tecnologias de acesso (WLAN, Bluetooth), como estabelecer a topologia da rede, mobilidade dos nós.

Fonte: (LOUREIRO et al., 2003)

2.4 Tolerância a falhas

Embora os sistemas de computação nos permitem automatizar muitas tarefas, sabe-se que todos eles são sujeitos a falhas. Contudo, embora as falhas sejam inevitáveis, as suas consequências podem ser evitadas pelo uso adequado de técnicas que fornecem a tolerância a falhas.

Há algum tempo, a preocupação com falhas era exclusivamente dos projetistas de sistemas críticos, como sistemas de controles industriais, de aviões e aeroespaciais. Contudo, com a popularização dos serviços oferecidos em rede, o mais simples dos computadores deve apresentar o mínimo de confiabilidade.

O principal objetivo da tolerância a falhas é alcançar a dependabilidade[1], que indica a qualidade e a confiança depositada no serviço entregue por um sistema (PRADHAN, 1996). O conceito de tolerância falhas foi primeiramente apresentado por Avizienis (1967), que diz o seguinte: um sistema é tolerante a falhas se seus programas podem ser executados corretamente, apesar da ocorrência de falhas lógicas. Ou seja, com a tolerância a falhas é possível entregar o serviço esperado mesmo na presença de falhas.

Com a finalidade de discutir os principais conceitos, abordagens e técnicas de tolerância a

[1]Em inglês: *dependability.*

falhas, esta seção está organizada em três partes: Primeiro, introduzimos os conceitos de falha, erro e defeito no contexto da área de tolerância a falhas (2.4.1). Em seguida apresentamos as principais abordagens e técnicas desenvolvidas (2.4.2). Por fim, apresenta-se um framework que implementa algumas das abordagens discutidas (2.4.3).

Vários autores tem se dedicado ao estudo dos conceitos e terminologias utilizadas na área de tolerância a falhas. Com isso, os conceitos apresentados nesta seção são baseados nos trabalhos de Laprie (1985), Laprie (1995), Anderson e Lee (1990), JALOTE (1994) e PRADHAN (1996).

2.4.1 Falhas, erros e defeitos

Muitas vezes considerados como sinônimos por muitos na escrita leiga, no contexto da tolerância a falhas eles possuem significados bem diferentes. O mais importante é observar a relação falha $\rightarrow$ erro $\rightarrow$ defeito[2] que é adotada na área.

Estando interessados no atendimento da especificação dos sistemas de computação. O defeito em um sistema ocorre quando o comportamento do sistema desvia do especificado (ANDERSON; LEE, 1990). Ou seja, o resultado (saída) do sistema não está correto, ou o sistema interrompeu a execução anormalmente, ou o resultado correto foi entregue, mas após ao limite de tempo especificado. Como não podem ser tolerados, é preciso evitar que o sistema apresente defeito.

Além disso, as causas de um defeito podem estar relacionadas somente às próprias atividades internas do sistema, ou seja, não estão relacionadas à especificação. Ou seja, o sistema se encontra em um estado errôneo, ou seja, seu processamento posterior pode levar a um defeito. Um erro é a parte do estado errôneo que constitui a diferença em relação ao estado válido (ANDERSON; LEE, 1990).

Por fim, define-se falha como a causa física ou algorítmica do erro. Isso é, será denominado como falha um erro em um componente ou no projeto do sistema, tendo em vista que uma falha no componente de um sistema causa um erro no estado interno de um componente. Já uma uma falha no projeto de um sistema é um erro no estado do projeto (ANDERSON; LEE, 1990).

Note que falhas levam os sistemas a apresentarem erros, que podem ser manifestados como defeitos. Isto é, a falha é a origem de um erro, e um erro é a causa de um defeito.

[2]Em inglês: *fault*, *error* e *failure*, respectivamente.

2.4.2 Abordagens e técnicas para aplicação da tolerância a falhas

De modo geral, as técnicas para tolerância a falhas podem ser divididas em duas classes: **(I) mascaramento** ou **(II) detecção, localização e reconfiguração**. Adotada para sistemas de tempo real críticos, a primeira normalmente aplica mais redundância que a segunda, uma vez que não envolve os tempos gastos para a detecção, localização e reconfiguração.

Anderson e Lee (1990) dividiram a tolerância a falhas em quatro fases de aplicação, na seguinte ordem: detecção, confinamento, recuperação e tratamento. As fases são explicadas a seguir:

Detecção: Uma falha só será detectada caso ela se manisfeste como um erro, pois ao se manisfestar algum mecanismo fará a detecção. Algumas falhas podem não comprometer o funcionamento e a entrega do serviço do sistema, tendo em vista que não se manifestam como erro, e por não serem detectadas podem permanecer um longo período no sistema.

Confinamento: Até que o erro seja detectado pode haver espalhamento de dados inválidos, então o confinamento estabelece limites para a propagação do dano. Contudo, para que isso aconteça é preciso que restrições de fluxo de informações estejam incluídas no projeto, o que não é muito comum.

Recuperação: Após a detecção, ocorre a fase de recuperação de erros, que envolve a troca do estado atual com falhas por um estado livre de falhas. A recuperação pode ocorrer de duas formas, por meio de técnicas de recuperação por retorno a um estado anterior correto *(backward error recovery)* ou por avanço a um novo estado livre de falhas *(forward error recovery)*. Técnicas de recuperação por retorno não são indicadas para sistemas de tempo real, visto que retroceder ao estado anterior podem deixar alguns dados órfãos no sistema, causando um efeito dominó. Isso se ao fato que os processos que receberam esses dados também terão que realizar um retrocesso. Com isso, neste caso é mais viável a recuperação por avanço.

Tratamento: Esta última fase consiste em localizar a origem do erro, localizar precisamente a falha, reparar a falha e recuperar o restante do sistema. Resumindo, após a localização, a falha é reparada por meio da remoção do componente danificado e a recuperação é realizada de forma manual ou automática.

Percebe-se que após a falha ser detectada, primeiramente é estabelecido os limites da propagação do dano. Após isso, o sistema passa a se recuperar, voltando a um estado ante-

rior ou avançando para novo estado livre de falhas. Somente após a recuperação do sistema, é que a origem do erro é localizada e reparada.

Mascaramento de falhas

O objetivo do mascaramento de falhas é garantir a entrega do serviço de forma correta, mesmo com presença de falhas no sistema. Neste caso, como a falha não se revela como um erro, o sistema não entra no estado errôneo e com isso, não há necessidade de detectar, confinar e realizar recuperação dos erros. Dentre os mecanismos mais usuais de mascaramentos de falhas estão a redundância, como a diversidade (Programação n-versões) e blocos de recuperação.

Redundância de *Software*

Sabe-se que todas as técnicas de tolerância a falhas envolve alguma forma de redundância, haja vista que na indústria o termo mais usado para indicar se um sistema é tolerante a falhas é a redundância. A redundância para tolerância a falhas pode aparecer de diversas formas, como por exemplo, redundância de hardware, software, informação e tempo.

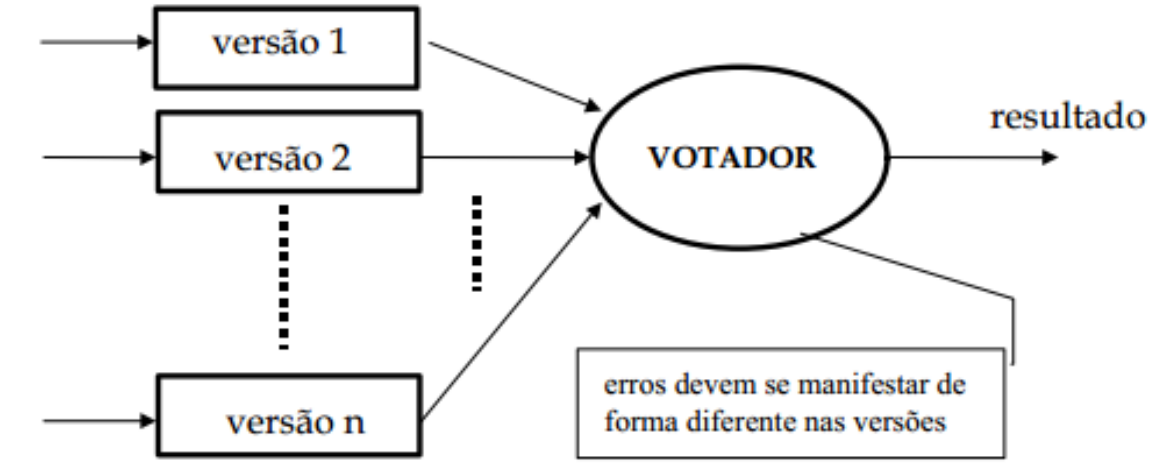
No caso da redundância de software, a replicação de componentes idênticos não é uma estratégia inteligente, pois a utilização de componentes de software idênticos resultará nos mesmos erros. Assim sendo, as formas mais utilizadas para prover a tolerância a falhas são as abordagens programação n-versões[3] e blocos de recuperação[4].

Programação N-Versões Uma das técnicas de redundância utilizadas para alcançar a tolerância a falhas de software é a programação n-versões,também conhecida como programação diversitária. Essa abordagem proposta por (AVIZIENIS, 1985) consiste na implementação de diferentes alternativas de soluções que resolvem um mesmo problema, isto é, atendem a uma especificação. Assim, a resposta do sistema é definida por votação, conforme apresenta a Figura 2.5.

Essa técnica não leva em consideração que erros nas soluções alternativas podem originar da mesma causa, como por exemplo, erro na interpretação da especificação. Com isso, para que os erros sejam detectados, é preciso que se manisfestem de uma forma diferente das outras implementações.

[3]Em inglês: *N-Version Programming.*
[4]Em inglês: *Recovery Blocks.*

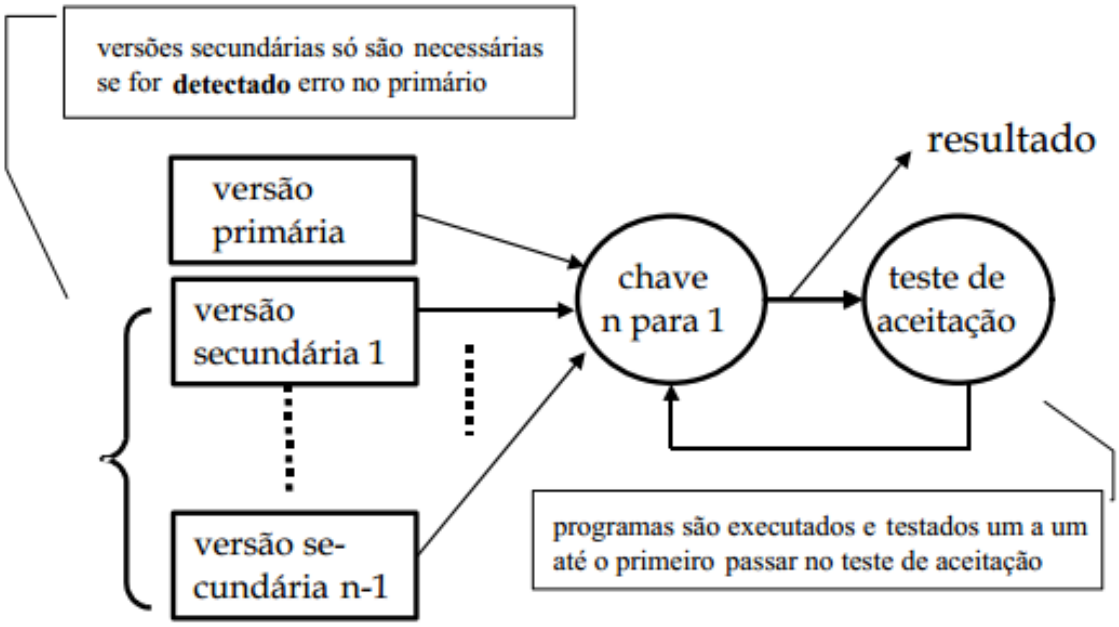
Figura 2.5: Programação N-Versões



Fonte: (WEBER, 2001)

Blocos de Recuperação Diferentemente da programação n-versões, a abordagem Blocos de Recuperação, desenvolvida por Randell (1975) e evoluída em Randell e Xu (1994), envolve um teste de aceitação, isto é, as versões secundárias só serão executadas caso seja detectado um erro na versão primária. Ou seja, os algoritmos são executados um a um até que o primeiro passe no teste de aceitação. A Figura 2.6 ilustra essa execução.

Figura 2.6: Blocos de Recuperação



Fonte: (WEBER, 2001).

A seção 2.4.3 apresenta um framework para desenvolvimento de aplicações tolerante a falhas que implementa as abordagens n-versões e blocos de recuperação.

2.4.3 FlexFT: Um framework para desenvolvimento de aplicações tolerante a falhas

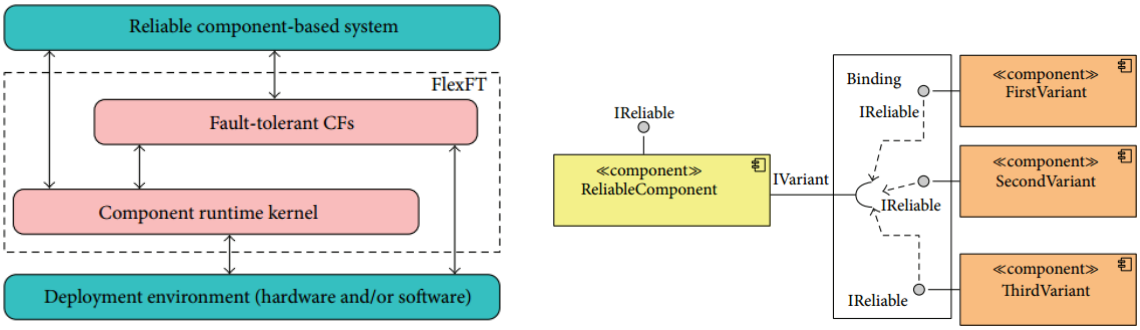
O framework FlexFT, desenvolvido por Beder et al. (2013), constitui-se de uma estrutura genérica baseada em componentes e escrita em Java para a construção de sistemas adaptáveis tolerante a falhas, que se integra a dispositivos e tecnologias diferentes, tanto software, quanto hardware. O framework consiste de um pequeno núcleo, onde as políticas de tolerância a falhas são inseridas conforme a demanda, visto que elas são implementadas como componentes e são instanciadas conforme a necessidade. Os seguintes benefícios são apontados pelos autores: flexibilidade, reusabilidade, transferências de habilidades, suporte a tecnologias independentes e interoperabilidade.

O FlexFT é construído sobre o framework para desenvolvimento baseado em componentes OpenCOM (COULSON et al., 2008), que executa as seguintes operações em tempo de execução e de forma adaptativa: **carregar** (*load*), **instanciar** (*instantiate*), **descarregar** (*unload*) e **destruir**(*destroy*).

Conforme a Figura 2.8(a), a arquitetura do FlexFT é composta por duas camadas, `Fault-Tolerant Component Frameworks` e `Component Runtime Kernel Layer`. A primeira fornece mecanismos para desenvolvimento de sistemas baseados em componentes confiáveis. Já a segunda oferece suporte para desenvolvimento de sistemas confiáveis baseado em componentes. A Figura 2.8(b) mostra que a abordagem implementa diferentes versões de componentes confiáveis (BEDER et al., 2013).

As n-versões (`FirtVariant, Second,...`) são componentes funcionalmente semelhantes, que são construídas de modo independente tendo por base a mesma especificação inicial. Já o Componente confiável (`ReliableComponent`) tem por função coordenar a execução das versões e chamar as operações pertinentes, utilizando testes de aceitação ou votação, como nas técnicas blocos de recuperação e Programação N-versões, explicados na seção 2.4.2. Já a função do mecanismo de ligação (`Binding`) é ligar as interfaces fornecidas e necessárias (BEDER et al., 2013).

Figura 2.7: FlexFT Framework (a) e (b)



(a) FlexFT Arquitetura (b) FlexFT Componente Confiável

Fonte: (BEDER et al., 2013).

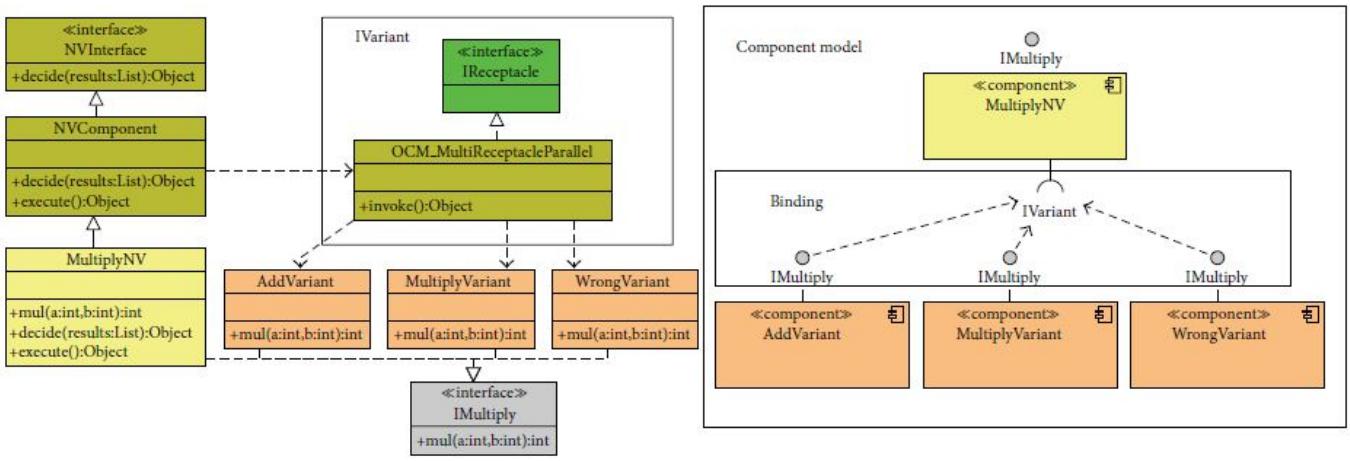
Implementação da técnica n-versões no FlexFT

Beder et al. (2013) por meio do framework FlexFT, exploraram a implementação de componentes confiáveis utilizando a técnica de programação n-versões, introduzida na seção 2.4.2.

A Figura 2.8 apresenta o componente `MultiplyNV` e as suas variantes `AddVariant`, *MultiplyVariant*, e `WrongVariant`. Cada uma das variantes representa uma implementação da funcionalidade de retornar o produto de dois inteiros. Já a classe *MultiplyNV* representa o `Reliable-Component` (Figura 2.7) e estende a classe abstrata `NVComponent`, que implementa o método `execute()`, levando em consideração as peculiaridades inerentes a técnica de programação N-Versões. Além de ser responsável por executar as variantes, o método execute() é responsável pela obtenção do resultado da decisão, dado pelo método `decide()` da classe `NVInterface`.

Beder et al. (2013) ressaltam que o FlexFT oferece um algoritmo padrão na classe `NVComponent`, que tem por função oferecer a decisão. No caso, o resultado é a maioria dos votos. Contudo, por meio de subclasses da `NVComponent` é possível reimplementar esse método, isto é, utilizar um método de decisão diferente. Além disso, a interface `IMultiply` é implementada pela classe `MultiplyNV`. Esta implementação consiste apenas de invocar o método `execute()`.

Figura 2.8: Concepção da programação n-versões no FlexFT



Fonte: (BEDER et al., 2013).

Implementação da técnica Blocos de Recuperação no FlexFT

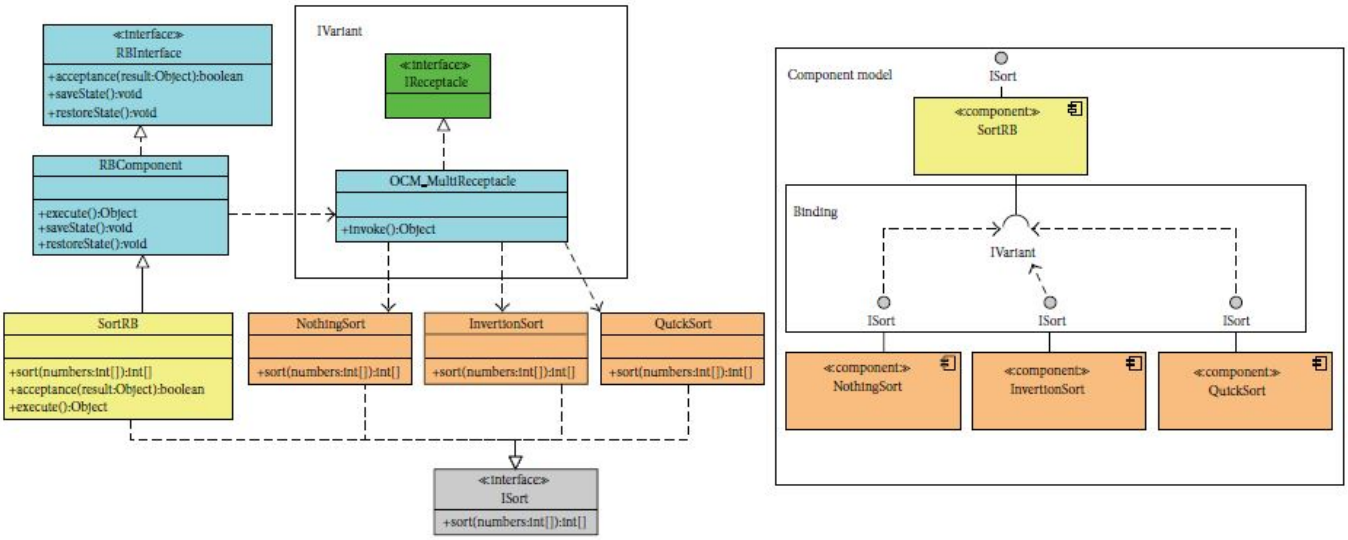
Beder et al. (2013) oferecem um exemplo de como o FlexFT pode ser usado para implementar componentes confiáveis utilizando a técnica blocos de recuperação. Para isso, é apresentado um componente chamado `SortRB` e suas variações `NothingSort`, `InvertionSort` e `QuickSort`. Cada variante tem como funcionalidade a ordenação de um vetor de inteiros em ordem ascendente.

Neste caso, o `ReliableComponent` apresentado na Figura 2.7 é representado pela classe `SortRB`, que estende a classe `RBComponent` e implementa as operações inerentes da técnica bloco de recuperação, que são: teste de aceitação, armazenamento do estado do componente e restauração. No exemplo dado, o teste de aceitação é capaz de verificar a seguinte condição: V é um vetor de inteiros, $V[i+1] \geq V[i]$, para $i = 1$ até $n-1$.

Beder et al. (2013) ressaltam que o FlexFT já possui implementado um algorítimo padrão, baseado na serialização de objetos, que tem por funcionalidade o armazenamento do estado atual e restauração do componente, por meio dos métodos `saveState()` e `restoreState()`, ambos presentes na classe `RBComponent`. Contudo, é possível sobrescrever o algorítimo utilizando subclasses.

Além disso, a interface `ISort` é implementada pela classe `SortRB`, cuja funcionalidade consiste apenas em chamar e executar o método discutido. Lembrando que, essas classes são estendidas da classe `OpenCOMComponent`.

Figura 2.9: Concepção do Blocos de Recuperação no FlexFT



Fonte: (BEDER et al., 2013).

2.5 Auto-organização

A auto-organização é um conceito que possibilita que sistemas formados por muitos subsistemas atuem de modo autônomo para executar uma tarefa em grupo. Além disso, sistemas auto-organizáveis demonstram um comportamento global, que dificilmente podem ser previstos ou pré-programado de modo escalável. As investigações de propriedades da auto-organização em sistemas naturais iniciou-se por volta dos anos 1960 e desde então, foram desenvolvidas muitas técnicas intrínsecas aos conceitos fundamentais da auto-organização (DRESSLER, 2007).

Segundo Yates (1987) os sistemas tecnológicos de modo geral se tornam organizados por comandos exteriores, como quando intenções humanas levam à construção de estruturas e máquinas. Contudo, sistemas auto-organizáveis são aqueles que tornam-se estruturados por meio de seus próprios processos internos, em que o surgimento de ordem dentro de si é um evento complexo que intriga cientistas de diversas áreas.

Mais tarde, Camazine et al. (2001) definiram auto-organização como um processo padrão em que o nível global de um sistema surge somente a partir de muitas interações entre os elementos de nível inferior do sistema. Além disso, é especificado, por meio de regras, que as interações entre os componentes do sistema somente sejam executadas utilizando apenas informações locais, isto é, sem referência ao padrão global.

Já Dressler (2007) resume a ideia da auto-organização como a criação de um comporta-

Tabela 2.4: Propriedades de sistemas auto-organizáveis

Propriedades	Descrição
Sem controle central	Não há sistema de controle ou informação global disponível, uma vez que cada subsistema deve operar de forma completamente autônoma.
Estruturas emergentes	O comportamento global ou o funcionamento do sistema surge sob a forma de padrões ou estruturas observáveis.
Complexidade Resultante	Mesmo que os subsistemas individuais sejam simples e realizem regras sucintas, o resultante de todo sistema torna-se complexo e muitas vezes imprevisível.
Alta escalabilidade	Se mais subsistemas são adicionados ao sistema, o desempenho não é comprometido, uma vez que o sistema deve desempenhar sua função, independentemente do número de subsistemas.

Fonte: (DRESSLER, 2007)

mento global a partir de interações locais entre os elementos do sistema.

Em síntese, a auto-organização é um paradigma para execução e controle de sistemas distribuídos em massa. É possível encontrar vários exemplos de auto-organização no meio ambiente, sendo que, algumas técnicas propostas no domínio foram inspiradas em mecanismos biológicos, e, são denominadas de bioinspirada. Todavia, a auto-organização é apenas um exemplo de algoritmos inspirado na biologia, e nem todas as técnicas de auto-organização têm relação com a investigação bioinspirada (DRESSLER, 2007).

Até então, a auto-organização foi discutida de modo superficial. No entanto, algumas propriedades de auto-organização são resumidos na Tabela 2.4.

A grosso modo, a auto-organização pode estar presente em todos os tipos de sistemas de controle, sempre que as propriedades descritas da Tabela 2.4 são consideradas. De modo geral, a auto-organização é sempre empregada para a execução e controle de sistemas complexos, que são formados por vários subsistemas (DRESSLER, 2007).

A auto-organização está presente em diversos cenários e aplicações, contudo neste trabalho será tratado apenas da auto-organização no aspecto de Redes de Sensores sem Fio (Seção 2.6).

2.6 Auto-organização em Rede de Sensores sem Fio

Como visto na seção 2.3, as RSSFs podem ser empregadas em diversos cenários e aplicações, sendo que os sensores podem ser distribuídos de forma manual em locais pré-determinados ou jogados na área que se deseja monitorar. Nos casos em que a rede contém centenas de sensores ou em casos que o local monitorado é remoto e de difícil acesso, a última forma deve ser mais

aplicada (LOUREIRO et al., 2003).

As RSSFs podem mudar sua topologia constantemente, por diversos motivos, como por exemplo: falta de bateria ou destruição dos nós, novos sensores podem ser inseridos, sensores podem alternar entre ligado e desligado para economizar energia, dentre outros. Assim sendo, essas redes precisam ser auto-organizáveis, ou seja, é necessário ter a habilidade de se ajustar às alterações sem interferência humana.

Segundo Loureiro et al. (2003) as principais dificuldades da auto-organização de uma rede de sensores seriam facilmente solucionadas com uma visão global da rede, tendo em vista que assim seria possível contabilizar os grupos de sensores que precisam interagir, estabelecer rotas de transmissão de dados mais inteligentes e verificar os sensores que podem ser desligados em determinados instantes, com o objetivo de se economizar energia e consequentemente permitir que a rede mantenha a cobertura desejada por um período maior. Contudo, no que tange a maioria da aplicações em RSSF, na prática é exigido que elas possuem autonomia, isto é, funcionem sem intervenção humana o maior tempo possível, além de serem escaláveis e robustas. Com isso, consideram ser mais proveitoso utilizar algoritmos localizados no projeto dessas redes, em que os sensores interajam somente entre si e de forma coletiva para conquistar a meta global.

Loureiro et al. (2003) definem a auto-organização de RSSFs como a habilidade em executar alterações na sua estrutura sem intermédio de ação humana, de modo torná-las escaláveis e robustas perante as características dinâmicas intrínsecas ao tipo dessa rede.

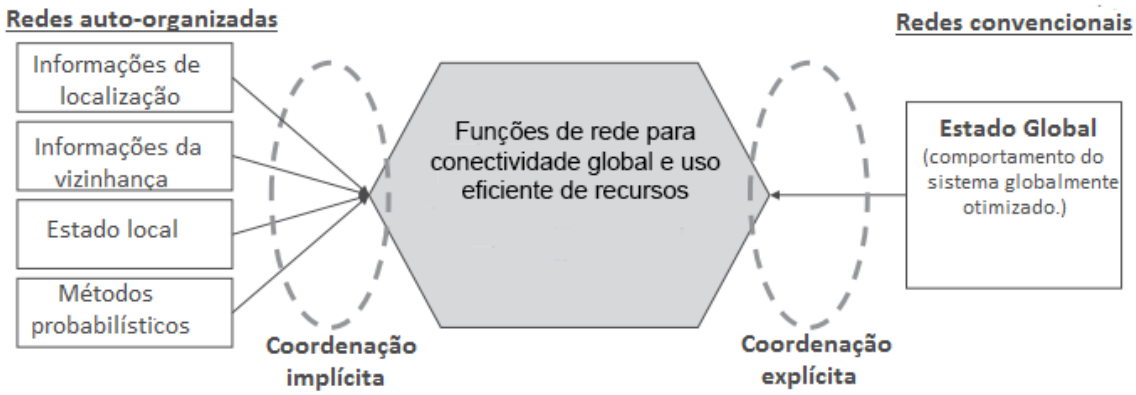
2.6.1 Propriedades e Objetivos

Sabe-se que realizar uma tarefa específica é o objetivo geral de toda rede de sensores. Contudo, para alcançar a meta global precisa-se entender como controlar os nós individuais na rede.

De modo geral, as metodologias de auto-organização tentam reduzir, ou mesmo extinguir o controle centralizado, levando em consideração as restrições de energia, poder de processamento, armazenamento e mobilidade (DRESSLER, 2007). Entretanto as preocupações mais relevantes, que devem ser resolvidas, são escalabilidade e o tempo de vida da rede.

Para melhor entendimento, a Figura 2.10 apresenta as abordagens que são utilizadas com o objetivo de manter as funcionalidades mais importantes das redes convencionais e auto-organizáveis, isto é, a conectividade global. No lado direito, a organização de redes convencionais fundamentadas no estado global é mostrada. Neste caso, é possível perceber que o controle entre os nós da rede é explícito e realizado por meio de algoritmos de otimização determinísticos. Já no lado esquerdo, é retratado o comportamento de redes auto-organizáveis,

Figura 2.10: Contraste entre redes convencionais e redes auto-organizáveis



Fonte: adaptado de Dressler (2007).

onde a coordenação é implícita, uma vez que é conseguida por meio do estado local, interações entre os nós vizinhos e métodos probabilísticos. Neste último caso, a otimização global já não é mais possível, todavia a capacidade de expansão da rede é bem maior (DRESSLER, 2007).

2.6.2 Categorização em duas dimensões

Dressler (2007) classifica as metodologias de auto-organização em duas dimensões, dimensão horizontal e vertical. A primeira trata questões da auto-organização no estado local dos nós, já a segunda preocupa-se com suas funções na pilha de protocolos. As duas categorias são tratadas a seguir.

Dimensão Horizontal

Segue abaixo as principais metodologias, apontadas por Dressler (2007), que compõem a dimensão horizontal.

Informações de localização: Esta dimensão diz respeito às posições geográficas, isto é, posições inerentes a uma área de interesse ou de uma área cercada por um grupo nodos sensores, que coletam informações do estado local para tomada de decisões de roteamento e sincronização. Um exemplo, é o controle da topologia e *clustering*, que confia na constatação de informações sobre a posição dos nós na rede. Para ambos os casos podem ser utilizados mecanismos centrados em rede ou GPS - Sistema de Posicionamento Global.

Informações da vizinhança: É possível alcançar uma redução adicional do estado, por

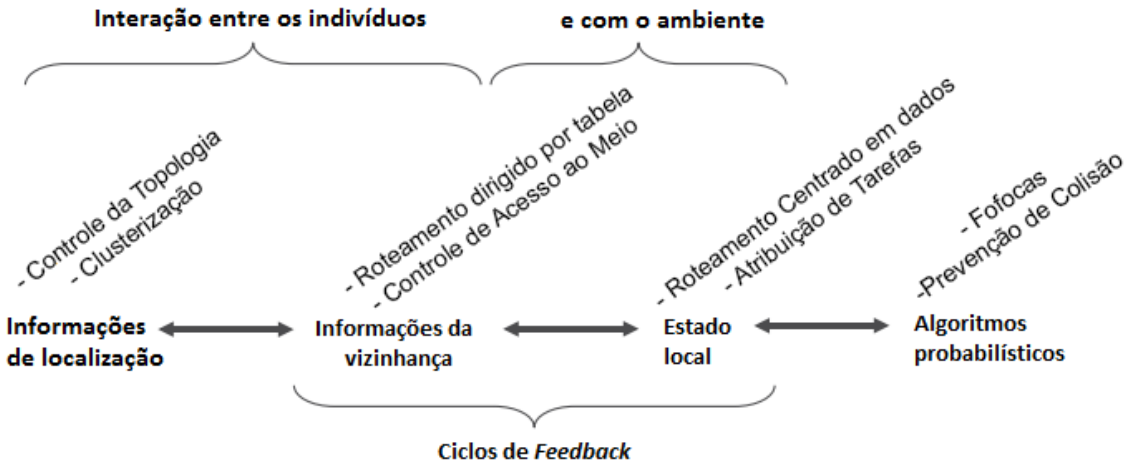
meio da diminuição do tamanho dos *clusters* para apenas o diâmetro de um salto. Assim, somente os dados da vizinhança estarão disponível para a tomada de decisão. Com o objetivo de manter as informações da vizinhança atualizada, os nodos trocam mensagens de "Olá" em certos períodos de tempo. Deste modo, a troca de medidas de desempenho torna-se possível, como por exemplo, a distribuição da carga atual do sistema. Por fim, o comportamento global pode ser obtido, por meio da troca de informações com outros grupos de vizinhos.

Estado local: No caso de redes de sensores, as características do sistema podem ser adequadas às condições ambientais, observadas por sensores instalados. Um exemplo de técnica que utiliza informações do estado local é o roteamento centrado em dados, que não lida com a topologia global, mas aplica programas locais simples às mensagens recebidas, encaminhando-as ou tomando as ações necessárias. Além disso, as tarefas locais podem ser acionadas conforme as leituras dos sensores ou respeitando um calendário fixo. Por fim, a situação atual precisa estar sempre disponível.

Algoritmos probabilísticos: Para Dressler (2007) em alguns casos, acaba sendo útil não armazenar informações do estado como um todo. Exemplo disso, é que por meio de métodos probabilísticos é possível alcançar bons resultados, mesmo quando a troca de mensagens é escassa ou quando os nós se movimentam muito. Por meio de medidas estatísticas é possível descrever o desempenho de todo o sistema em termos de carga prevista e operações realizadas, contudo não é possível garantir que um objetivo desejado será sempre alcançado. No caso de redes *ad hoc* e de sensores, um exemplo é o roteamento baseado em fofocas (*gossip-based routing*), que ao invés de inundar toda rede com mensagens, sobrecarregando a rede por causa de duplicatas, são utilizados esquemas probabilísticos para decidir se enviam ou não a mensagem. Comparativamente, desse modo, é possível alcançar um bom desempenho. Um exemplo de aplicação acontece na camada MAC - Media Access Control (Controle de Acesso ao Meio), visto que técnicas de anticolisão fazem uso de probabilidades, como por exemplo, se as estações começam a enviar mensagens ao acaso, a probabilidade de colisões pode ser reduzida.

Por fim, tem-se a Figura 2.11, que sintetiza os principais mecanismos da dimensão horizontal. Percebe-se que os mecanismos evitam lidar com as informações do estado global, uma vez que tem por objetivo aumentar a capacidade de expansão da abordagem particular. Além disso, os ciclos de *feedback* também são representados na figura. Contudo, a escolha da metodologia requer um certo cuidado, uma vez que que é necessário considerar o cenário e os requisitos da aplicação.

Figura 2.11: Categorização Horizontal de Mecanismos de auto-organização em redes de ad hoc e de sensores



Fonte: adaptado de Dressler (2007).

Dimensão Vertical

Nesta dimensão (DRESSLER, 2007) trata da arquitetura do sistema em camadas, tendo em vista que dependendo da camada há diferentes aspectos relacionados a auto-organização. Nesta seção é dado ênfase nas camadas MAC, Rede e Aplicação, por representar a base principal para pesquisa em redes de sensores. A dimensão vertical é ilustrada pela Figura 2.12.

Camada MAC: A principal função desta camada é gerenciar o acesso ao link de rádio sem fio e efetuar a troca de mensagens entre os nós que estão conectados na mesma faixa de transmissão. Sabe-se que mecanismos baseados em contenção dominam esta camada, permitindo que todos os nós vizinhos sustentam equitativamente o canal. Além disso, por meio de técnicas de *Carrier-sensing* é possível diminuir as colisões, enquanto que os intervalos de sono aleatórios garantem que dois nós não privem um ao outro. Além de tudo, a sincronização entre nós vizinhos pode ser utilizada para otimizar o link compartilhado, como por exemplo, esquemas de Time Division Multiplexing (TDM) garantem acesso exclusivo em intervalos de tempo selecionados. Enquanto técnicas de *overhearing* permitem que mensagens desnecessárias sejam transmitidas pelos nós, a utilização de ciclos de trabalho, geridos entre a própria vizinhança, podem diminuir o consumo de energia do receptor de rádio, além de melhorar o desempenho do sistema como um todo. Por fim, mecanismos de auto-organização ajudam a realizar o acesso simultâneo, para sincronizar os nós e manter ciclos de trabalho de modo distribuído, sem a necessidade de uma gestão

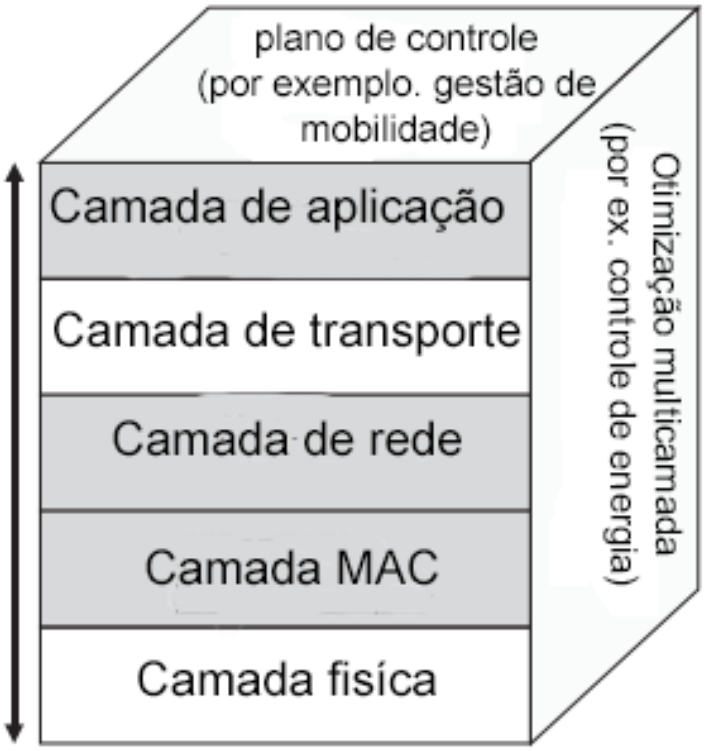
central ou pré-configuração.

Camada de rede: Roteamento e encaminhamento de dados são as tarefas realizadas por esta camada. A preocupação do roteamento é a identificação da topologia atual. Por meio da troca de informações entre nós vizinhos as tabelas de roteamento são mantidas. Para isso tem-se diversas técnicas como por exemplo, *table-driven* e roteamento *link-state*, além de modalidades diferentes, como roteamento pró-ativo e sob demanda. Logo, percebe-se que todos mecanismos dependem de informações de estado sincronizados com outros nós na rede.

Por meio da análise do endereço de destino e a verificação com o correspondente na tabela de roteamento, as mensagens recebidas são encaminhadas. De modo geral, enquanto o roteamento envolve uma quantidade grande de informações em nível global, o encaminhamento preocupa-se com a melhora do desempenho, gestão de erros e tratamento de congestionamento. Muitas abordagens têm sido desenvolvidas com o objetivo de realizar as duas tarefas de modo auto-organizado, como por exemplo, o roteamento centrado em dados, em que a troca, o processamento e o encaminhando das mensagens é realizado de acordo com a semântica da mensagem.

Camada de aplicação: Segundo Dressler (2007) muitas tarefas de coordenação devem ser auto-organizadas, além dos conteúdos da própria aplicação. Para isso, regimes de atribuição de tarefas têm sido estudados em diversos tipos de redes ad hoc, que são baseadas em sistemas auto-organizados ou em abordagens probabilísticas puras. Além dessas questões de coordenação, a camada de aplicação define os requisitos básicos de qualquer comunicação na rede. A cobertura, a conectividade e a disponibilidade podem ser apontadas como as principais características em um RSSF, além de estar diretamente relacionadas com o tempo de vida da rede como um todo. Contudo, além de requisitos, a camada de aplicação provê informações essenciais acerca dos padrões de comunicação que podem ser esperados a partir da aplicação específica, em execução na rede de sensores.

Figura 2.12: Categorização Vertical de Mecanismos de auto-organização em redes de ad hoc e de sensores



Fonte: adaptado de Dressler (2007).

2.6.3 Métodos e aplicações

Até agora foram discutidas as principais abordagens de auto-organização em RSSF e *ad-hoc*. Nesta seção, primeiramente é apresentado um mapeamento dos principais métodos de auto-organização. Posteriormente alguns trabalhos são apresentados, com o objetivo de evidenciar a utilização dessas abordagens e os cenários de aplicação.

Mapeamento das principais metodologias de auto-organização

Podendo ser utilizadas em qualquer contexto de sistemas complexos e massivamente distribuídos, abordagens como ciclos de *feedback*, interações entre componentes do sistema e técnicas probabilísticas podem ser aplicadas para alcançar a auto-organização (DRESSLER, 2007). Considerando que o domínio de RSSF e rede ad-hoc é muito específico, tem-se uma breve descrição de possíveis aplicações dessas metodologias.

Ciclos de *feedback*: Utilizados para atualizar os parâmetros operacionais dos sistemas

ou grupos formados por diversos subsistemas de acordo com o estado atual, algoritmos adaptativos podem ser construídos usando ciclos de *feedback* negativo e positivo, sendo o primeiro utilizado para a amplificação e o segundo para o controle do processo de adaptação (DRESSLER, 2007).

De modo geral, em RSSF o *feedback* é dado por meio da observação e avaliação de diversas parâmetros do sistema. Caso o foco seja mudança de condições ambientais, sensores podem ser empregados localmente para realizar a medição dessas condições. Deste modo, a partir da análise periódica da leitura dos sensores pode-se iniciar a adequação dos parâmetros, caso seja necessário. Além disso, o *feedback* também pode ser dado por meio de nós vizinhos, que mediram ou detectaram alguma alteração ambiental, na qual o sistema, ou um grupo de sistemas, deve se adaptar. Logo, informações da vizinhança e do estado local são associadas para a aplicação de algoritmos adaptativos, que monitoram o desempenho do nodo sensor individual (DRESSLER, 2007).

Interações entre os componentes do sistema e o ambiente: Dressler (2007) divide as interações em três tipos, sendo elas: interações com o nós remotos, com vizinhos diretos ou por meio interações indiretas. As interações com nós remotos podem ser necessárias para alcançar um objetivo comum. Em rede de sensores, exemplos que encaixam nessa categoria são os algoritmos de roteamento ou mecanismos de otimização de cobertura, uma vez que dependem de interações com nós remotos. Já as interações realizadas com vizinhos diretos são altamente necessárias para o uso eficiente dos recursos de rádio, como por exemplo, sistemas de anticolisão empregam esquemas de *handshake*[5] e de sincronização. Além disso, atualizações localizadas podem ser usadas para reduzir as redundâncias desnecessárias e aumentar a tolerância falhas. Também existem as interações com o ambiente ou interações indiretas com outros nós conforme mudanças no ambiente, visto que os nós podem explorar o ambiente e o estado local pode ser adaptado conforme a medição dos fenômenos físicos, selecionando parâmetros de funcionamento e algoritmos em conformidade.

Técnicas probabilísticas: Técnicas probabilistas têm sido utilizadas em diversos contextos. Dois exemplos em redes de sensores são os roteamentos probabilísticos, como o baseado em fofocas (*gossiping*), e a alocação de tarefas probabilísticas. Além disso, a aleatoriedade é utilizado em diversos mecanismos e protocolos para evitar os efeitos indesejados de sincronização. Métodos probabilísticos também podem ser utilizados para

[5]Muito utilizado em protocolos de comunicação, *handshake* (ou aperto de mão) é o processo em que dois dispositivos afirmam que reconheceram um ao outro e que estão prontos para iniciar a comunicação (KUROSE; ROSS, 2010).

evitar gargalos na rede (DRESSLER, 2007).

2.7 Trabalhos Relacionados

A fim de mostrar a utilização das metodologias citadas na seção 2.6.3, apresenta-se alguns trabalhos de pesquisa propostos.

2.7.1 Auto-Organização e Tolerância a Falhas

Chen, Tao e Ma (2008) apresentaram um modelo de uma aplicação de RSSF para monitoramento de eventos de emergência. O modelo opera em dois estados, **vigilante** e de **detecção**, sendo que o sistema permanece no primeiro modo enquanto não há ocorrência. Para economizar energia, o sistema funciona em um modo alternativo, uma vez que alguns nós são selecionados para operar com baixa frequência de amostragem em um intervalo de tempo determinado, enquanto que outros nós continuam dormindo. A cada intervalo de tempo os papéis são trocados. Assim que um evento de emergência é detectado, o sistema passa a funcionar no estado **detecção**, uma vez que o nó imediatamente transmite a informação por toda rede, de modo que todos nós despertem e tenham conhecimento da ocorrência. Em tempo, o sistema descobre e marca o ponto exato de onde o evento iniciou e prevê a abrangência, ou seja, a possível área infectada. Por fim, a sala de controle recebe os dados do nó sorvedouro, que mantém contato com a situação, permitindo assim, que socorristas sejam enviados para a área infectada, aos passos que os nós relatam a situação em tempo real por meio de múltiplos saltos.

Villas et al. (2009) apresentaram um protocolo de roteamento ciente de agregação de dados denominado DAARP (Data-Aggregation Aware Routing Protocol). Esse protocolo seleciona rotas com alto grau de agregação e realiza a transmissão confiável dos dados agregados. Nele, os sensores podem assumir quatro papéis diferentes: Colaborador, Coordenador, *Sink* e *Relay*. Resumidamente, pode-se entender sua execução em três partes: Primeiro, o nó Sink envia uma mensagem em *broadcast* para todos nós da rede com a finalidade de construir a árvore de roteamento. Já a segunda fase consiste na formação de um cluster com os nós que detectaram um mesmo evento na rede. Inicialmente, todos nós no cluster possuem o papel de colaborador. Posteriormente, há a eleição de um nó coordenador para o cluster considerando o número de saltos do Sink e do evento. Na eleição, todos nós no cluster são elegíveis e somente um nó é eleito. Este nó coordenador é o nó responsável por coletar os dados dos nós colaboradores, agregá-los e enviar o resultado para o *sink*. Já a última fase é responsável pela criação de uma nova rota para transmissão confiável dos dados e atualização da árvore de roteamento. Neste

caso, o nó Relay é responsável pelo encaminhamento desses dados até o *sink*.

O protocolo se mostrou muito eficiente em comparação com outros protocolos na literatura. Foi reduzido o número de mensagens necessárias para configurar a árvore de roteamento, maximiza o número de rotas, seleciona rotas com um alto grau de agregação e possui entrega confiável dos dados.

Tatara, Lee e Chong (2011) propõem uma solução descentralizada para tratar o problema da auto-organização em diversas aplicações compostas por uma multidão de sensores robóticos autônomos. Essa solução descentralizada permite que cada robô construa e mantenha a rede, por meio de interações de comunicação locais com seus vizinhos, permitindo que robôs escolhem como vizinhos àqueles que possuem maior conectividade, adaptando às mudanças topológicas da rede através de movimentos do robô. A proposta foi validada por meio de simulações realistas e extensivas, considerando atrasos de comunicação e falhas do robô.

Somando-se a isso, o trabalho de Vilela e Araujo (2012) apresenta uma solução de roteamento para uma RSSF móvel, composta por nós fixos e móveis. A solução tem por objetivo coletar dados de nós sensores que enviam os dados para o sink, por meio de nós intermediários e de mobilidade diferentes. Basicamente a solução é composta por três fases: Na fase de formação da rede, todos os nós recebem a informação de localização do sink. Já na segunda fase, uma eleição do *cluster-head* é realizada quando ocorre um evento, considerando os níveis de mobilidade, a distância do sink e energia. Assim sendo, todos os dados coletados pelos sensores são enviados para o *cluster-head*. Na terceira fase, os dados recebidos pelo *cluster-head* são entregues ao sorvedouro. Segundo Vilela e Araujo (2012) a solução tem se mostrado eficiente em termos de mensagens de *overhead* e transmissão de pacotes de dados. Além disso, o objetivo principal é que essa solução seja adaptável a diferentes domínios.

Joshi e Younis (2012) apresentam um algoritmo distribuído para Reparação Autônoma (*Autonomous Repair -AuR*) de topologias de RSSF danificadas por causa das falhas dos nós sensores. Primeiramente é assumido que os nós estão cientes da sua posição e da área monitorada, por meio de esquemas de localização contemporâneos. Além disso, também assume-se que todos nós tem o mesmo alcance de transmissão, que é bem menor do que a área monitorada. A falha de um nó da rede, pode resultar na perda de conectividade e da cobertura. A partir disso, o algoritmo concentra-se em reestabelecer a conectividade, sendo que o processo de recuperação é iniciado localmente nos vizinhos dos nós falhos, movendo-os na direção da perda, para se reconectar com os outros nós. Além disso, para restaurar um pouco da cobertura é utilizado nós otimizados de autopropagação, movendo-os para o centro da zona monitorada. Segundo Joshi e Younis (2012) os resultados das simulações mostram que o *AuR* supera as soluções atuais em

termos de recuperação de falhas nós de grau médio, sobrecarga e cobertura.

2.7.2 Aplicações de detecção de incêndios em RSSFs

Hefeeda e Bagheri (2007) apresentam um projeto de RSSF para a detecção precoce de incêndios a partir de uma análise do sistema Fire Weather Index (FWI), uma dos sistemas mais abrangentes para notificações de risco de incêndios florestais na América do Norte. Deste, aparece os principais aspectos que devem ser considerados na modelagem de incêndios e como os diferentes componentes do FWI podem ser usados em projetos de sistemas eficientes de detecção de incêndio. O problema da detecção de incêndios é resolvido com um algoritmo distribuído com K-cobertura para um K-cobertura problema. Além disso, propõe-se um esquema de agregação de dados, que prolonga a vida útil da rede. O projeto é validado por simulação.

Liu et al. (2011) propõem um sistema de detecção de incêndio que utiliza uma RSSF e a aplicação de múltiplos critérios para aumentar a precisão da detecção de um incêndios, tendo em vista que uma decisão de alarme precisa de muitos atributos de um incêndio. A detecção multi-critério é implementada por uma rede neural artificial que combina os dados detectados de vários atributos de um incêndio em uma decisão de alarme. O uso da rede neural permite a autoaprendizagem e a baixa sobrecarga da rede. Eles assumem que cada nó está consciente da sua localização, fornecidos pela estação de base ou por GPS, e possuem diversos sensores, tais como umidade e temperatura. Usando o protótipo desenvolvido, os experimentos foram realizados para avaliar o desempenho do sistema proposto. Os resultados mostram que o protótipo ativa alarmes de incêndios com uma alta precisão. Além disso, uma bateria solar desenvolvida provou ser capaz de atender a rede por anos.

Yoon et al. (2012) apresentam um sistema de monitoramento confiável baseado em um RSSF implantada em condições adversas, com conexões sem fio não confiáveis, assimétricas, irregulares, com problemas de roteamento, dentre outros. É usado sistemas de comunicações baseados em uma topologia tolerante a falhas, assegurando o fornecimento de dados para a estação base e permitindo a exibição de dados em tempo real. Neste trabalho, 20 sensores foram implantados para coletar dados de temperatura e umidade. Os nós são projetados para desligar seus transceptores para economizar energia, e sincronizar seus ciclos, de sono ou vigilante, para simplificar a comunicação. Nós que estão despertos podem operar em modo normal ou em modo de alerta. O modo alerta implica que o fogo foi detectado, o que requer que os dados sensoriais devem ser entregues tão rapidamente quanto possível, mesmo que consome mais energia. O atraso de entrega na transmissão dos dados para o nó *sink* é reduzido por meio da utilização de um algoritmo de encaminhamento PISA-I modificado.

2.8 Considerações Finais

O presente capítulo apresentou os principais conceitos e características da área de computação autônoma, redes de sensores, tolerância a falhas e auto-organização. Pode-se notar, que durante os últimos anos muitas abordagens foram propostas e que ainda há muito ser feito. Em síntese, também ficou evidente que no contexto de Redes de Sensores Fio, as principais abordagens propostas defendem a utilização de mecanismos de interação local, localização, temporalização e *feedback* para alcançar a auto-organização e consequentemente, a tolerância a falhas.

Além disso, observando os trabalhos relacionados, percebe-se a ausência de um trabalho que trata em conjunto a auto-organização e tolerância a falhas em níveis diferentes, como por exemplo, na camada de aplicação e tarefas de roteamento. Normalmente, a auto-organização e a tolerância a falhas estão concentrados somente nas tarefas de baixo nível.

Já no contexto de aplicação em RSSF para detecção de incêndios, percebe-se nos trabalhos que as aplicações não estão distribuídas na rede de sensores, mas alocadas somente no servidor, sendo necessário o envio de todos os dados para análise pela central de controle. Como a transmissão dos dados é o fator que mais consome energia na rede, talvez, neste contexto, isso não seja recomendado, tendo em vista que o tempo de vida da rede é menor e a troca de bateria dos sensores é dificultada, tendo em vista que normalmente são locais de difícil acesso.

Além do mais, não foi possível a replicação das soluções de detecção de incêndios apresentadas nos trabalhos relacionados, tendo em vista que os trabalhos não tratam em detalhes aspectos referentes a construção e implementação das soluções.

Portanto, fica evidente a necessidade de um trabalho que trate a auto-organização e a tolerância a falhas nos níveis de roteamento e de aplicação ao mesmo tempo, em especial, no contexto de detecção de incêndios. Com isso, no capítulo 3 apresentados um modelo e uma aplicação que buscam resolver esse problema.

Capítulo 3

MODELO PROPOSTO

Este capítulo apresenta o modelo para desenvolvimento de aplicações tolerante a falhas e auto-organizáveis em Redes de Sensores sem Fio. Além disso, apresenta-se os detalhes de implementação e os resultados de uma simulação de uma aplicação de detecção de fogo em uma área de preservação ambiental.

3.1 O Modelo auto-organizável e tolerante a falhas

Esta seção apresenta o modelo desenvolvido para alcançar a auto-organização e a tolerância a falhas em aplicações de Rede de Sensores sem Fio densas, especialmente àquelas aplicadas no monitoramento de áreas de preservação ambiental. O modelo relaciona diversas camadas de uma RSSF e consiste das seguintes fases: (A) Implementação e estabelecimento da rede; (B) Anúncio de eventos, Clusterização e Eleição de Coordenação; (C)Tarefas de Coordenação e Detecção de incêndios; (D)Agregação e envio dos dados para o *Sink*. Cada etapa será explanada a seguir.

3.1.1 Fase A - Implementação e estabelecimento da rede

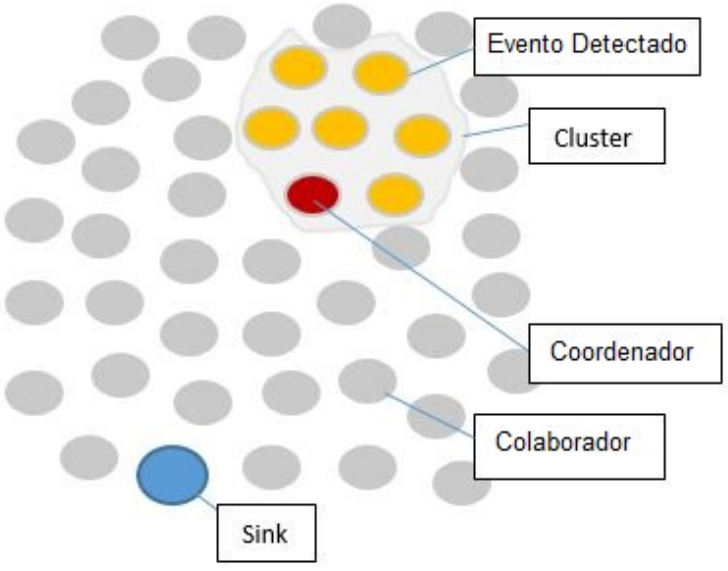
Inicialmente, os nós são inseridos aleatoriamente na área de interesse e despertam para a formação da rede. Para isso, o nó sink envia uma mensagem de controle em modo *broadcast* para toda rede com a finalidade de construir a árvore de roteamento. Assim sendo, cada nó ao

receber a mensagem calcula e armazena o número de saltos. Feito isso, repassa a mensagem aos seus nós vizinhos. Eles vão fazendo isso até que toda rede seja alcançada. De acordo com Bulusu et al. (2001) a densidade recomendada da rede é próxima de 21.7 e a sua distribuição, isto é, o número de nós requeridos para monitorar uma região pode ser calculado como $(N\pi R^2)/A$. Onde N é o número de nós sensores disseminados na região A; e R, é a taxa de transmissão.

3.1.2 Fase B - Notificação de eventos, Clusterização e Eleição de Coordenador

O disparo de um evento pode ocorrer de várias formas dependendo dos requisitos da aplicação. Um exemplo em uma aplicação de detecção de fogo é quando há um aumento considerável de temperatura. Uma característica de RSSFs aplicadas em monitoramento ambiental é ser redes densas. Isso aumenta a confiabilidade da rede e garante a entrega de pacotes, mesmo em caso de falha de alguns nós. No entanto, é comum que vários nós detectam um mesmo evento. Assim sendo, uma abordagem para evitar o envio de dados duplicados para o *sink* é o agrupamento desses nós (clusterização) e a eleição de um nó com o papel de coordenador. Assim, após a coleta dos dados pelos nós participantes do *cluster*, os dados são enviados ao coordenador para análise (Figura 3.1).

Figura 3.1: Notificação de eventos, Clusterização e Eleição de Coordenador.



A eleição do coordenador pode ser dada de diversas maneiras. O modelo propõe o algoritmo 1 para eleição do coordenador. Este algoritmo é baseado no algoritmo de eleição de líder proposto pelo protocolo DAARP (VILLAS et al., 2009). Foi realizado uma modificação para considerar além do número de saltos, o nível de energia dos nós para um melhor balanceamento da rede.

Algorithm 1 :Coordinator election

Input: S // Set of nodes that detect the event
Output: u // A node of cluster is elected coordinator
Initialisation :
for each u in S **do**
  $role_u \leftarrow$ Coordinator
  u sends message in brodcast to neighbors(w) belonging to the cluster
  if $EventNum = 1$ **then**
    if $HopstoSink_u > HopstoSink_w$ **then**
      $role_u \leftarrow$ Collaborator
      Node u retransmits the message received from node w
    else if $HopstoSink_u = HopstoSink_w$ and $Energy_u < Energy_w$ **then**
      $role_u \leftarrow$ Collaborator
      Node u retransmits the message received from node w
    else if $HopstoSink_u = HopstoSink_w$ and $Energy_u = Energy_w$ and $ID_u > ID_w$ **then**
      $role_u \leftarrow$ Collaborator
      Node u retransmits the message received from node w
    end if
  else
    if $EventNum > 1$ **then**
      if $HopstoEvent_u > HopstoEvent_w$ and $Energy_u < Energy_w$ **then**
        $role_u \leftarrow$ Collaborator
        Node u retransmits the message received from node w
      else if $HopstoEvent_u = HopstoEvent_w$ and $Energy_u \leq Energy_w$ and $ID_u > ID_w$ **then**
        $role_u \leftarrow$ Collaborator
        Node u retransmits the message received from node w
      end if
    end if
  else
    Node u discards the message received from w
  end if
end for

Inicialmente, o algoritmo tem como entrada todos os nós que pertencem ao cluster, ou seja, nós que detectaram o mesmo evento, uma vez que todos são elegíveis. Se é o primeiro evento da rede, será eleito Coordenador o nó que tem a menor distância do Sink. Se o número de saltos

forem iguais, o nó que possuí maior nível de energia será eleito. Finalmente, se houver um empate em ambos os casos, a ID será usado para desempate. Já a partir da segunda ocorrência do evento, será eleito coordenador do grupo o nó que possui o menor número de saltos da árvore de roteamento existente e maior nível de energia. No caso de empate, o desempate será feito pelo ID. Todos os outros nós que detectaram o mesmo evento permaneceram como colaboradores. Os nós colaboradores transmitem os dados detectados para avaliação do coordenador.

Desta forma, é possível garantir um melhor equilíbrio de energia de rede. Tendo em vista que as tarefas de coordenação geralmente consomem mais energia e que o nível de energia é considerado na eleição. Dificilmente, um nó será eleito coordenador mais de uma vez em seguida.

3.1.3 Fase C - Tarefas de Coordenação e Detecção de Fogo

Em muitos trabalhos de pesquisa encontrados na literatura, as funções de coordenação são resumidas em apenas coletar os dados dos nós colaboradores, agregá-los e enviá-los ao nó sink, para só então realizar a análise dos dados. Isto é devido à aplicação estar alocada somente no *sink* ou na estação de base (servidor), não distribuída pela rede de sensores. Consequentemente, muitos dados acabam sendo enviados para o *sink*, consumindo energia desnecessária, uma vez que nem todos os eventos anunciados são considerados realmente fogo. Alarmes falsos podem ocorrer e a aplicação deve estar preparada para isso.

Além das tarefas comumente relevantes para o coordenador, o modelo sugere que a detecção de fogo seja realizada no coordenador, isto é, a solução estaria alocada nos sensores, não na estação de base. Para fazer isso, assume-se que a solução esteja previamente implantada em todos os sensores. No entanto, só será instanciada quando o sensor assumir o papel de coordenador. Uma vez que o sensor volta ao papel de colaborador, a aplicação será desalocada da memória. Vale a pena dizer que isso acontece em uma abordagem adaptativa.

Afinal, para que tudo isso seja possível sem sobrecarregar a rede e manter um baixo consumo de energia, o modelo encoraja que a solução seja construída utilizando componentização de software. Entende-se que as soluções são propensas a falhas e, devido a isso, a tolerância a

falhas deve ser considerada desde a concepção da aplicação. Contudo, a principal abordagem de se implementar a tolerância a falhas em software é por meio da redundância. Ou seja, diferentes soluções alternativas que buscam resolver um mesmo problema.

Contudo, propõem-se que a detecção de incêndios seja efetuada por várias soluções, em que cada solução seja construída como um componente, que seria **carregada** e **descarregada** em tempo de execução. Para gerir esta dinâmica, alcançar a tolerância a falhas e oferecer um resultado preciso, implementa-se a técnica de programação N-Versões citada na seção 2.4.2.

Ou seja, diversas soluções de detecção de incêndio propostas por especialistas do domínio são executadas em tempo de execução, gerando resultados **verdadeiro** ou **falso** e uma **probabilidade de erro**. Após isso, uma votação é realizada, isto é, um consenso de todos os resultados. Por exemplo, há três árvores de decisão e os resultados de cada foram respectivamente Verdadeiro 6%, Falso 10% e Verdadeiro 7%. A nossa abordagem conta o número de verdadeiros e falsos e calcula a média das probabilidades. Ou seja, neste caso, o resultado do consenso seria Verdadeiro com probabilidade de erro de 6,5% ((6 + 7) / 2). O resultado é verdadeiro, pois duas soluções forneceram verdadeiro como resultado, contra uma que apresentou resultado falso. O resultado da probabilidade de erro é 6,5%, pois é a média das probabilidades das árvores de decisão que proveram verdadeiro como resultado.

3.1.4 Fase D - Agregação e envio dos dados

Após a conclusão do consenso, ou seja, a decisão final, o coordenador pode ou não enviar os dados para o *sink*. O especialista de domínio pode configurar como desejado. Assim, ele pode enviar apenas os resultados positivos, em caso em que se deseja economizar energia e consequentemente, aumentar o tempo de vida da rede, uma vez que não há gasto de energia com transmissão de alarmes falsos. No entanto, pode-se também optar por enviar todos os resultados (verdadeiros e falsos), tendo em vista que podem ser úteis para estudos estatísticos.

Além disso, resultados idênticos podem ser agregados para reduzir o número de transmissões duplicadas. Por fim, a Figura 3.2 resume as principais atividades das etapas que compõe o modelo proposto.

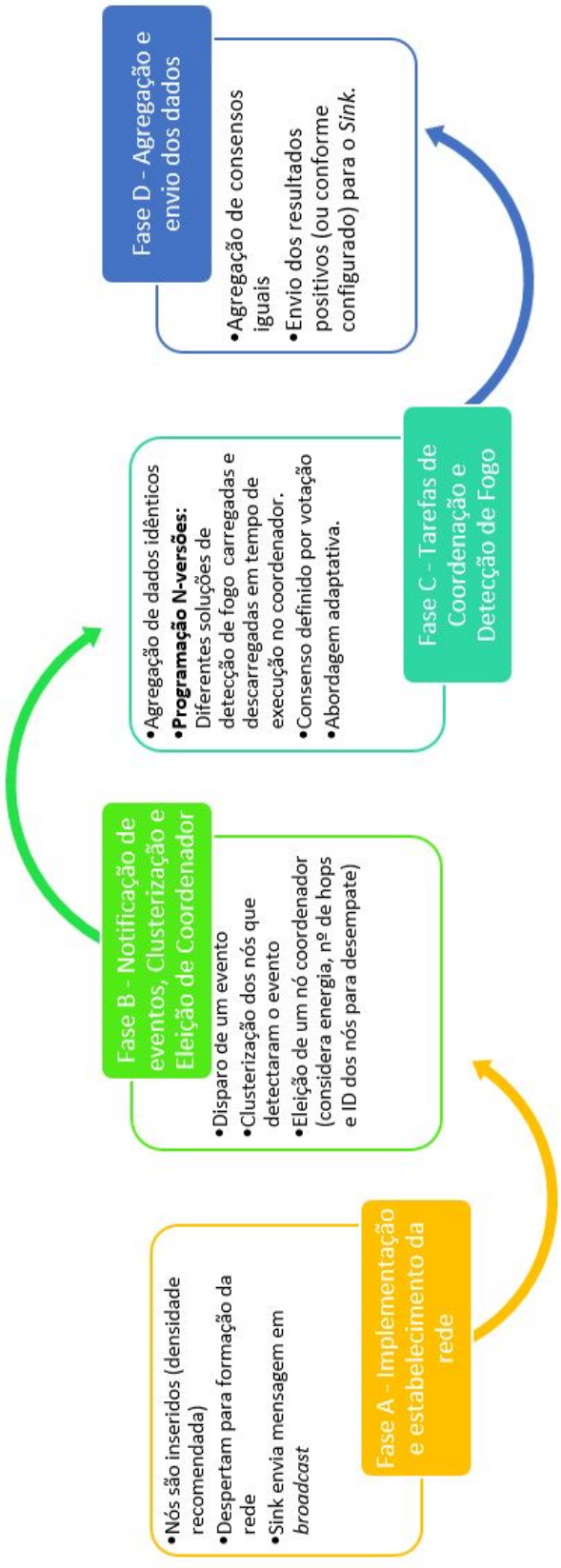


Figura 3.2: Resumo do Modelo auto-organizável e tolerante a falhas para detecção de incêndios

3.2 Detalhes de implementação e Simulação

Para avaliar e validar o modelo foi implementado e realizado uma simulação no Sinalgo v. 0.75.3 (SINALGO, 2015), um framework simulador escrito na linguagem Java[1] para testar e validar algoritmos de redes.

A simulação foi realizada sobre a área de preservação ambiental do Jardim Botânico de Brasília[2], que é uma unidade de conservação da natureza ligado à Secretaria de Meio Ambiente e Recursos Hídricos do Distrito Federal. A instituição possui uma propriedade de 500 hectares de visitação pública, abrangendo vários tipos de vegetação do bioma Cerrado. Ela também possui 4.500 hectares de reserva ecológica para a pesquisa e conservação do Cerrado. Aplicando o cálculo da distribuição, apresentado na fase A da seção 3.1.1, para alcançar uma densidade de 21,7 no monitoramento de uma área de 4.500 hectares, com comunicação via rádio de 100 metros cada nó. O cálculo mostrou ser necessários cerca de 31.082 nós sensores para contemplar toda reserva ecológica.

Os sensores foram modelados para serem capazes de monitorar e coletar dados de umidade, temperatura e vento. Eles foram distribuídos sobre área conforme o cálculo da distribuição da densidade.

Para as tarefas de baixo nível, como roteamento, comunicação e clusterização utilizou-se o protocolo de roteamento ciente da agregação de dados DAARP (Data-Aggregation Aware Routing Protocol) (VILLAS et al., 2009) com algumas modificações. O DAARP já foi descrito na seção 2.7. Conforme mencionado na seção 3.1, além da modificação no algoritmo de eleição de líder para considerar o nível de energia dos nós, foi alterado a estrutura das mensagens para suportar os dados detectados dos sensores e da aplicação.

Já na fase de tarefas de coordenação, implementou-se a aplicação de detecção de fogo em forma de componentes e utilizando a programação n-versões, por meio do FlexFT (BEDER et al., 2013).

[1] https://www.java.com/
[2] http://www.jardimbotanico.df.gov.br/

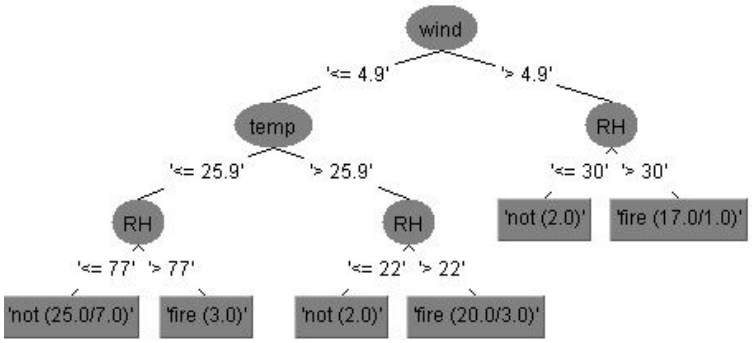
3.2.1 Protótipo da solução de detecção de incêndios

A solução de detecção de fogo é composta por três árvores de decisão que foram implementadas como componentes: AD1 - Figura 3.4(a), AD2 - Figura 3.4(b), AD3 - Figura 3.4(c), que são carregadas somente quando o nó assume o papel de coordenador, conforme mencionado na seção 3.1. Para desenvolvimento das árvores de decisão foi utilizado o *Forest Fires Data Set* com 517 instancias do *UCI Machine Learning Repository* (LICHMAN, 2013) e a ferramenta de mineração de dados Weka 3.6 (HALL et al., 2009).

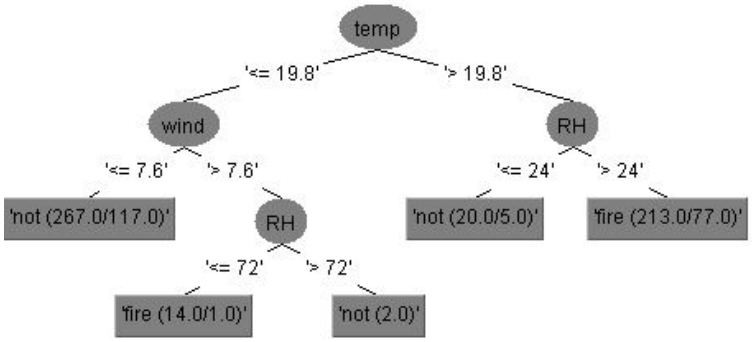
Primeiro, classificou-se a base de dados e fez a validação com um especialista em detecção de fogo, responsável pela equipe de combate a queimadas no Jardim Botânico de Brasília. De todos atributos da base de dados, utilizou-se somente os que o nó sensor modelado seria capaz de detectar. No caso, vento (*wind*), umidade relativa do ar (*RH*) e temperatura (*temp*). O atributo chuva(*rain*) não fui utilizado pois demonstrou não causar nenhum impacto nas árvores de decisão. As árvores de decisão foram geradas com quantidade de instâncias diferentes, com a finalidade de representar níveis de conhecimento e equipes diferentes. A primeira(AD1) com 69, a segunda(AD2) com 516 e a terceira (AD3) com 243 instâncias. Com isso, elas obtiveram diferentes taxas de classificação (*cross-validation*),que são respectivamente 84.05%, 61.24% e 73.66%. Os detalhes de acurácia de cada árvore de decisão podem ser conferidos pelas Figura 3.4 e a Matriz de confusão pela Figura 3.5.

O objetivo é demonstrar que podem existir diferentes soluções com diferentes taxas de classificação e construídas por grupos com características diferentes. Sem dúvida, a melhor solução é àquela construída por um conjunto de especialistas do domínio. Por isso, destaca-se que o modelo proposto é eficaz se as soluções forem construídas por esses especialistas. Contudo, esses modelos de detecção podem trabalhar com conjunto com o objetivo de aumentar a confiabilidade depositada na aplicação.

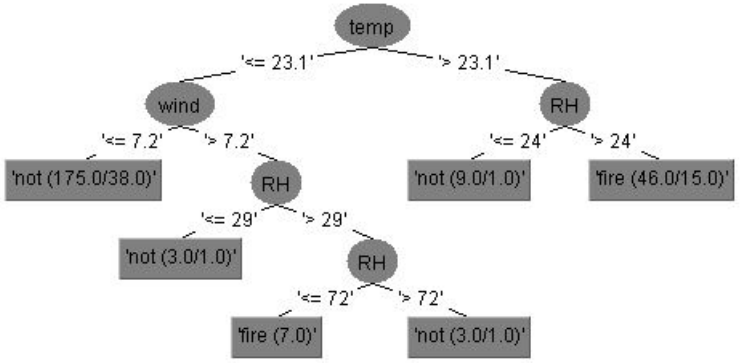
Figura 3.3: Árvores de decisão para detecção de incêndios florestais (a), (b) e (c).



(a) AD1



(b) AD2



(c) AD3

Figura 3.4: Detalhes de Acurácias das árvores de decisão por classe (a), (b) e (c).

	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
	0.846	0.163	0.759	0.846	0.8	0.873	not
	0.837	0.154	0.9	0.837	0.867	0.873	fire
Weighted Avg.	0.841	0.157	0.847	0.841	0.842	0.873	

(a) AD1

	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
	0.682	0.45	0.578	0.682	0.625	0.632	not
	0.55	0.318	0.656	0.55	0.598	0.632	fire
Weighted Avg.	0.612	0.381	0.619	0.612	0.611	0.632	

(b) AD2

	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
	0.872	0.544	0.769	0.872	0.817	0.634	not
	0.456	0.128	0.632	0.456	0.529	0.634	fire
Weighted Avg.	0.737	0.409	0.724	0.737	0.724	0.634	

(c) AD3

Figura 3.5: Matriz de Confusão das árvores de decisão (a), (b) e (c).

a	b	<-- classified as
22	4	a = not
7	36	b = fire

(a) AD1

a	b	<-- classified as
167	78	a = not
122	149	b = fire

(b) AD2

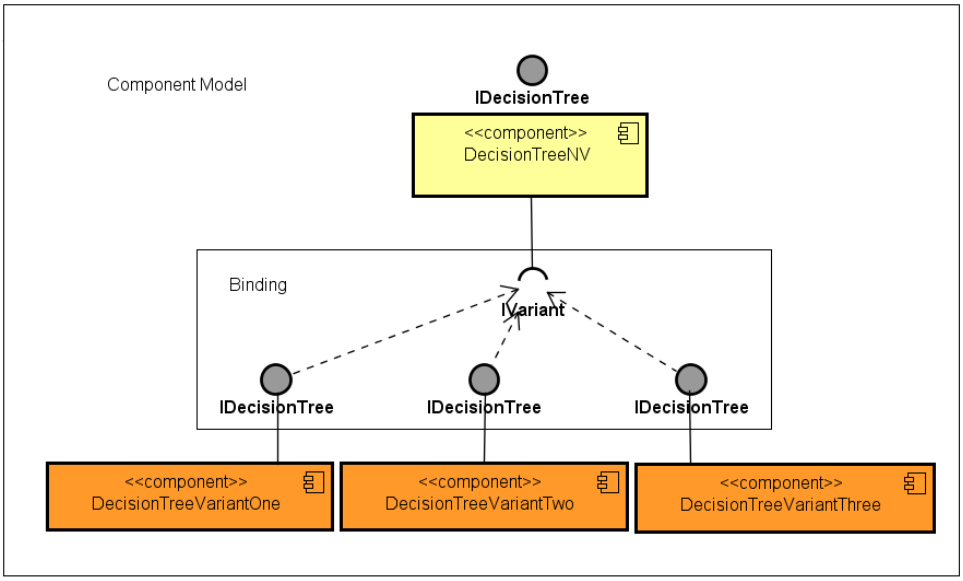
a	b	<-- classified as
143	21	a = not
43	36	b = fire

(c) AD3

A Figura 3.6 apresenta o diagrama de componentes. É possível verificar o componente desenvolvido `DecisionTreeNV` e suas variantes `DecisionTreeVariantOne`, `DecisionTreeVariantTwo` e `DecisionTreeVariantThree`. Sendo que, cada uma dessas variantes é composta pela implementação de uma solução de detecção de fogo, isto é, respectivamente as árvores de decisão AD1, AD2, AD3. É possível conferir todo código na apêndice A.

Já no diagrama de classes (Figura 3.7) , a classe `DecisionTreeNV` representa o componente confiável (`Reliable-Component` do FlexFT - Figura 2.7) e estende a classe abstrata `NVComponent`, que levando em consideração as peculiaridades inerentes a técnica de programação N-Versões, implementa o método `execute()`. Além de ser responsável por executar as variantes(operações *load* e *unload* no OpenCOM), o método `execute()` é responsável pela obtenção do resultado da decisão, dado pelo método *decide()* da classe *NVInterface*.

Figura 3.6: Diagrama de Componentes da Solução



Como o FlexFT oferece um algoritmo padrão na classe `NVComponent`, para atender os requisitos da aplicação sobrescreveu-se a função `decide()` na classe`DecisionTreeNV`, que é responsável por chamar o método estático `decide()` na classe de interface `IDecisionTree` passando uma lista de resultados. Essa lista de resultados é construída pelo algoritmo `decision()` em `DecisionTreeNV`, que recebe os dados dos sensores e realiza o método `execute()` repassando os dados para cada variante. Os resultados retornados são inseridos em uma lista, que posteriormente, é repassada como parâmetro para o método `decide()`. Neste caso, este último método conta o número de resultados verdadeiros e falsos (respectivamente, *true* e *false*) e calcula a média das probabilidades de erro. Por fim, retorna como consenso o que teve maior número de ocorrências e a média das probabilidades. O procedimento está descrito no código fonte 3.1. É evidente que a interface `IDecisionTree`, apresentada no diagrama de componentes é implementada pela classe `DecisionTreeNV`.

Código: 3.1: Implementação do decide() da classe IDecisionTree

```
1 static public Object decide(List<?> results) {
2
3    int contador[] = new int[2];
```

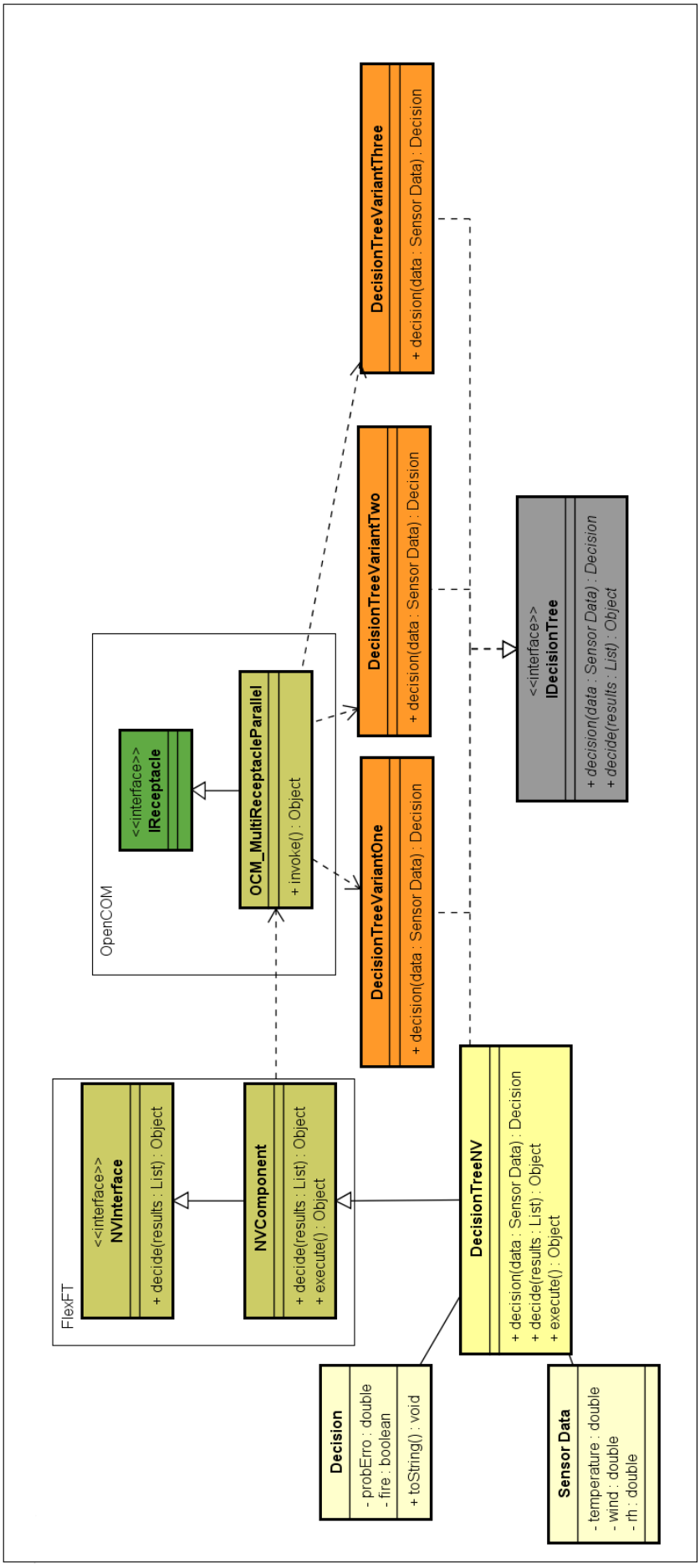


Figura 3.7: Diagrama de Classes da solução

```
4     double probabilidade[] = new double[2];
5
6     for (Object result : results) {
7        Decision d = (Decision) result;
8        if (d.isFire()) {
9           contador[1]++;
10          probabilidade[1] += d.getProbErro();
11       } else {
12          contador[0]++;
13          probabilidade[0] += d.getProbErro();
14       }
15    }
16    if (contador[1] >= contador[0]) {
17       return new Decision(true, probabilidade[1] / contador[1]);
18    } else {
19       return new Decision(false, probabilidade[0] / contador[0]);
20    }
21 }
```

3.2.2 Inserção do protótipo no Nó Sensor

Para cumprimento dos requisitos da aplicação, reimplementou-se o método encontrado na classe do nó sensor, do protocolo DAARP. Anteriormente, o nó somente recebia e encaminhava os dados para o Sink, por meio de uma rota estabelecida pelo protocolo.

Como a aplicação está alocada no nó sensor, quando o método `SentInformationEvents` é chamado, verifica-se se o nó possuí o papel de coordenador e se ele já recebeu todos os dados, isto é, quando a variável `senddata` é igual a *true*. Caso seja verdeiro, inicia-se a execução da aplicação. Como é possível ver no código 3.2, primeiramente, cria-se o ambiente de execução `OpenCOM` e solicitasse a interface IOpenCOM. Logo em seguida, é criado o componente `DecisionNV` e os componentes de decisão, isso é, as árvores de decisão. Posteri-

ormente é requerido a interface `IDecisionTree` e finalmente é executado as decisões com os valores de vento, umidade e temperatura, fornecidos pelos sensores. Nesse caso, a mensagem foi passada por parâmetro para o método.

Posteriormente, os resultados dos consensos são armazenados em uma lista e somente os resultados positivos são enviados para o nó *Sink*. Note, que isso deve ser definido pelo especialista de domínio. Em caso, que deseja-se receber todos os resultados, basta retirar a verificação (`if`), ou adaptar conforme orientações do especialista.

Código: 3.2: Execução da aplicação no nó sensor

```
1  if ((this.senddata) && (this.myrole == Roles.COORDINATOR)) {
2
3      senders.add(ID);
4
5      // Create the OpenCOM runtime & Get the IOpenCOM interface
6      // reference
7      OpenCOM runtime = new OpenCOM();
8      IOpenCOM pIOCM = (IOpenCOM) runtime
9      .QueryInterface(IOpenCOM.class.getName());
10
11     // Create the DecisionNV component
12
13     IUnknown pDecNVIUnk = (IUnknown) pIOCM.createInstance(
14     Sample.DecisionTree.DecisionTreeNV.class.getName(),
15     "DecisionNV");
16
17     ILifeCycle pILife = (ILifeCycle) pDecNVIUnk
18        .QueryInterface(ILifeCycle.class.getName());
19     pILife.startup(pIOCM);
20
21     // Create the Decision Components
22
```

```
23    for (int i = 1; i <= 3; i++) {
24       IUnknown pDecIUnk;
25       if (i == 1) {
26          pDecIUnk = (IUnknown) pIOCM.createInstance(
27                DecisionTreeVariantOne.class.getName(),
28                "Decision" + i);
29       } else if (i == 2) {
30          pDecIUnk = (IUnknown) pIOCM.createInstance(
31                DecisionTreeVariantTwo.class.getName(),
32                "Decision " + i);
33       } else {
34          pDecIUnk = (IUnknown) pIOCM.createInstance(
35          DecisionTreeVariationThree.class.getName(),
36          "Decision " + i);
37       }
38
39       pILife = (ILifeCycle) pDecIUnk
40             .QueryInterface(ILifeCycle.class.getName());
41       pILife.startup(pIOCM);
42       runtime.connect(pDecNVIUnk, pDecIUnk,
43             IDecisionTree.class.getName());
44       }
45
46       // Get the Decision Interface
47       IDecisionTree pIDecision = (IDecisionTree) pDecNVIUnk
48             .QueryInterface(IDecisionTree.class.getName());
49
50       // Get the IDecision Interface
51       pIDecision = (IDecisionTree) pDecNVIUnk
52             .QueryInterface(IDecisionTree.class.getName());
53
```

```
54          //Running decision components
55          Decision d;
56
57          d = pIDecision.decision(new SensorData(mdata.getTemp(),
                mdata.getWind(), mdata.getRh()));
58
59
60          decisions.add(d);
61
62
63          cont = cont + 1;
64
65
66          Tools.appendToOutput("Consensus: "+ IDecisionTree.decide(
                decisions) + "n°" + cont);
67
68
69          mdata.setDecisions(decisions); //add result of the decision
                 in the message
70
71          //only sends the positive results. Comment on this check,
                you want to send all the data.
72          if(decisions.get(decisions.size()-1).isFire()){
73              this.senddata=true;
74          }else{
75              this.senddata=false;
76          }
77
78  }
```

3.2.3 Resultados da Simulação

A simulação foi replicada 10 vezes com a ocorrência de 6 eventos aleatórios cada, garantindo um total de 60 eventos. Os valores de vento, umidade e temperatura de cada nó também foram gerados aleatoriamente, considerando um intervalo de valores mínimos e máximos da base de dados.

A simulação foi realizada em um Notebook com Processador Intel i5 de 2.4 GHz e 8 GB de RAM. Devido ao grande número de sensores foram dedicados 4 GB de RAM para a máquina virtual Java e a simulação ocorreu via console, ou seja, sem utilizar a interface gráfica do Sinalgo. Na tabela 3.1 é possível verificar os parâmetros de configuração da simulação.

Para fins de análise, foi optado pelo envio de todos os resultados. Os consensos de cada evento foram analisados e classificados como:

Falso Total : Fogo não detectado e há um consenso absoluto de todas árvores de decisão.

Verdadeiro Total: Fogo detectado e há um consenso absoluto de todas árvores de decisão.

Falso Parcial: Fogo não detectado e há um consenso parcial das árvores de decisão.

Verdadeiro Parcial: Fogo detectado e há um consenso parcial das árvores de decisão.

Figura 3.8: Resultados da Simulação para 60 eventos.

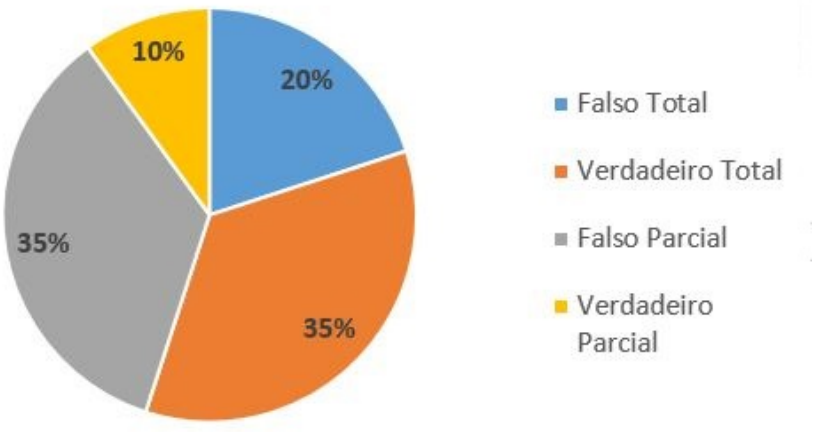


Tabela 3.1: Configurações iniciais do Sinalgo

Campo	Valor
Simulation Area	
dimensions	2
dimX	6708
dimY	6708
dimZ	50
Simulation	
asynchronousMode	true
mobility	false
interference	false
interferenceIsAdditive	true
canReceiveWhileSending	true
canReceiveMultiplePacketsInParallel	true
edgeType	BidirectionalEdge
exitOnTerminationInGUI	false
initializeConnectionsOnStartup	true
refreshRate	1
generateNAckMessages	true
handleEmptyEventQueue	true
javaCmd	java
javaVMmaxMem	4096
Random number generators	
useSameSeedAsInPreviousRun	false
useFixedSeed	true
fixedSeed	40
Logging	
logFileName	logfile.txt
outputToConsole	true
logToTimeDirectory	false
logConfiguration	true
eagerFlush	true
GUI	
extendedControl	true
drawArrows	false
Background map in 2D	
useMap	true
map	images/map.bmp

Os resultados apresentados na Figura 3.8 mostram que somente 35% dos eventos inicialmente anunciados foram identificados verdadeiramente como incêndio. Do mesmo modo, 20% foram alarmes falsos. Nestes eventos não foram detectados nenhuma falha nas aplicações de detecção. Ou seja, em ambos os casos houve um consenso total de fogo ou não fogo. No entanto, 45% (isto é, 35% de Falso Parcial somado com 10% de Verdadeiro Parcial) dos consensos identificaram uma possível falha. Ou seja, uma das três soluções não esteve em acordo com outras duas. Isso mostra que o modelo proposto é tolerante a falhas, caso as soluções sejam construídas por diferentes grupos de especialistas em detecção de incêndios.

Além disso, caso tenha sido optado que o nó coordenador transmita para o sink somente os eventos detectados como verdadeiro total, haveria uma economia de 65% no número de transmissões. Como a literatura aponta a transmissão dos dados, como o principal fator consumidor de energia em uma RSSF, consequentemente o tempo de vida da rede seria maior. Mesmo no caso, em que permitissem também a transmissão dos eventos com consenso verdadeiro parcial para análise de um especialista, haveria uma economia de 55% na transmissão de mensagens. Em casos em que se desejasse enviar todos os resultados para análise pessoal de um especialista, exceto o Falso Total, haveria economia de 20% no número de transmissões.

3.2.4 Categorização do modelo e solução

Observado as dimensões da modelagem apresentada por Andersson et al. (2009)(Seção 2.2) classifica-se a solução proposta da seguinte forma:

Grupo Goals: Neste grupo, cujas dimensões definem os objetivos do sistema, o consideramos como estático, rígido, persistente e único. Visto que o único objetivo do sistema é realizar a detecção de incêndios. De modo geral, não há múltiplos objetivos.

Grupo Change: Neste grupo, em que são classificadas as adaptações do sistema, considera-se a fonte das mudanças como externa e interna, visto que temos adaptação quando ocorre um evento e também por meio dos frameworks e protocolos por si só. Considera-se que esta adaptação é não-funcional, tendo em vista que estão relacionadas ao desempenho e

confiabilidade do sistema. Entende-se também que a abordagem adaptativa ocorre somente quando é anunciado um evento, logo, neste caso, não é possível definir a mudança como rara ou frequente, visto que depende de muitos fatores, como por exemplo, o clima e estação do ano, no caso de incêndios. Por fim, toda abordagem adaptativa da solução já está prevista nos seus requisitos.

Mechanisms: As dimensões deste grupo classificam o solução em relação a reação do sistema à mudança. Neste caso, considera-se como a adaptação como estrutural, autônoma, descentralizada e global, tendo em vista que uma rede de sensores é auto-organizável e sua topologia muda constantemente. Do ponto de vista da aplicação, ela também pode ser autoadaptativa e autônoma, visto que faz o carregamento e descarregamento dos componentes em tempo de execução. Além de estar distribuída na rede, não requer ação humana após os nós serem inseridos no ambiente. Além disso, como o sistema segue essa abordagem, pode-se dizer que a pontualidade da adaptação é garantida e que o disparo é feito por um evento.

Effects: Esta dimensão refere-se ao impacto da adaptação sobre o sistema e lidam com os efeitos dessa adaptação. Do ponto de vista da criticidade, considera-se a abordagem adaptativa no nível de aplicação como de missão crítica, visto que se houver falha na execução, como por exemplo, deixar de executar alguma variante, a detecção pode ficar comprometida. Além disso, entende-se que a adaptação é determinística, não causa sobrecarga no sistema e a entrega dos serviços continuam confiáveis diante do processo de mudança.

Capítulo 4

CONSIDERAÇÕES FINAIS

O presente trabalho apresentou um modelo auto-organizável e tolerante a falhas para detecção de incêndios em áreas de preservação ambiental. Para alcançar diferentes níveis de auto-organização e tolerância a falhas, foi necessário integrar diversas abordagens e soluções anteriormente propostas. Dentre eles, o protocolo DAARP (VILLAS et al., 2009) e os framewroks FlexFT (BEDER et al., 2013), OpenCOM (COULSON et al., 2008) e Sinalgo(SINALGO, 2015).

Por meio da simulação sobre a área do Jardim Botânico de Brasília, foi possível mostrar a viabilidade de implementação do modelo em um cenário real. Utilizando a técnica de programação n-versões implementada no FlexFT, foi possível demonstrar que há possíveis indícios de falhas nas árvores de decisão, o que é verdade, já que as taxas de classificação não são muito altas. Além disso, foi considerado que as soluções de modo geral podem ser imperfeitas, e que a organização de diferentes soluções podem aumentar a confiança depositada na aplicação.

Percebeu-se também que muitos dos eventos detectados eram apenas alarmes falsos. Assim sendo, com a não transmissão desses alarmes, pode-se ter uma economia considerável de energia e consequentemente o aumento do tempo de vida da rede.

4.1 Contribuições

1. Um modelo auto-organizável e tolerante a falhas de RSSFs para detecção de incêndios. Pelo que é conhecido, até então, não foi relatado na literatura um modelo (ou trabalho) no contexto de detecção de incêndios, que sistematize o processo de uma RSSF.

2. Um protótipo de solução de software para detecção de incêndios. A solução proposta se destaca pelos seguintes motivos: (i) esta alocada nos nós sensores, não no servidor como os trabalhos na literatura, permitindo a detecção precoce e economia de transmissões, tendo em vista que alarmes falsos não são enviados para o *sink* e estação base; (ii) utiliza dados de vento, umidade e temperatura, o que torna o processamento menos custoso do que as soluções que utilizam vídeo, que no caso são a maioria na literatura; (ii) construída utilizando uma abordagem adaptativa.

3. Reúne aspectos de auto-organização e tolerância a falhas em níveis diferentes. Pelo que é conhecido na literatura não há trabalhos que tratem em conjunto aspectos de auto-organização e tolerância a falhas em diferentes níveis em RSSF para detecção de incêndios. O modelo proposto trata tanto esses aspectos no nível de roteamento, quanto no nível de aplicação.

4. Aplicação da técnica de redundância de componentes, em específico, a programação N-Versões, cujo desenvolvimento foi baseado em componentes, em uma aplicação de rede de sensores. Por meio do consenso de diferentes soluções implementadas como componentes foi possível identificar possíveis indícios de falhas no nível de aplicação, aumentando a confiança depositada no serviço.

5. Integração do Framework FlexFT (BEDER et al., 2013) ao Sinalgo (SINALGO, 2015), podendo ser utilizado na construção de novas soluções no Sinalgo. A partir da integração de do FlexFT ao Sinalgo novos algoritmos de rede tolerante a falhas podem ser construídos utilizando uma abordagem adaptativa. Isso vale tanto para o nível de aplicação quanto para o nível de roteamento.

6. Classificação de uma base de dados de focos de incêndios florestais. As instâncias do *Forest Fires Data Set* foram classificadas como ”Fogo”e ”Não Fogo”com o acompanhamento e validação de um especialista do domínio. No caso, o gestor de uma equipe de combate de incêndios.

7. Simulação sobre a reserva ambiental do Jardim Botânico de Brasília. A simulação ocorreu sobre uma área de 4.500 hectares. Isso permitiu ter ciência da quantidade de nós necessários, por meio do cálculo de distribuição da densidade.

4.2 Trabalhos Futuros

1. Sugere-se que o modelo proposto seja aplicado também com outros protocolos para verificar seu comportamento;

2. Uma análise de viabilidade econômica de implementação da RSSF no Jardim Botânico de Brasília;

3. Desenvolver outros modelos para a aplicação de detecção de incêndios em conjunto com uma equipe de especialistas no domínio;

4. Construir uma solução que verifique o lado de crescimento do fogo, visto que ajuda as equipes de combate a definir suas estratégias;

5. Integrar outras soluções de detecção de incêndios propostas por outros pesquisadores no modelo;

6. Aplicar o modelo para outras aplicações de monitoramento, como, por exemplo, detecção de enchentes;

7. Desenvolver uma interface do lado servidor para aplicação.

Apendice A

CÓDIGO DA IMPLEMENTAÇÃO DA SOLUÇÃO

Código: A.1: Classe Decision

```
1  package Sample.DecisionTree;
2
3  public class Decision{
4
5      private double probErro;
6      private boolean fire;
7
8
9      public Decision(boolean fire, double probErro) {
10         super();
11         this.probErro = probErro;
12         this.fire = fire;
13     }
14
15     public double getProbErro() {
16         return probErro;
17     }
18     public void setProbErro(double probErro) {
```

```
19          this.probErro = probErro;
20      }
21      public boolean isFire() {
22          return fire;
23      }
24      public void setFire(boolean fire) {
25          this.fire = fire;
26      }
27
28      public String toString(){
29          return "[" + fire+","+ probErro + "]";
30
31      }
32  }
```

Código: A.2: Classe DecisionTreeNV

```
1   package Sample.DecisionTree;
2
3   import java.util.ArrayList;
4   import java.util.List;
5
6   import FlexFT.NVParalelComponent;
7   import OpenCOM.IConnections;
8   import OpenCOM.ILifeCycle;
9   import OpenCOM.IMetaInterface;
10  import OpenCOM.IUnknown;
11
12  public class DecisionTreeNV extends NVParalelComponent<
        IDecisionTree>
13          implements IConnections, ILifeCycle, IUnknown,
                IMetaInterface, IDecisionTree {
```

```
14
15      List<Decision> lista;
16
17      public DecisionTreeNV(IUnknown binder) {
18          super(binder);
19          lista = new ArrayList<>();
20      }
21
22      public Decision decision(SensorData data) {
23          Decision d = (Decision) this.execute("decision", data);
24          lista.add(d);
25          return d;
26      }
27
28      @Override
29      public Object decide(List<?> results) {
30          return IDecisionTree.decide(results);
31      }
32
33  }
```

Código: A.3: Classe IDecisionTree

```
1   package Sample.DecisionTree;
2
3   import java.util.List;
4
5   public interface IDecisionTree {
6
7       public Decision decision(SensorData data);
8
9       static public Object decide(List<?> results) {
```

```
10
11        int contador[] = new int[2];
12        double probabilidade[] = new double[2];
13
14        for (Object result : results) {
15           Decision d = (Decision) result;
16           if (d.isFire()) {
17              contador[1]++;
18              probabilidade[1] += d.getProbErro();
19           } else {
20              contador[0]++;
21              probabilidade[0] += d.getProbErro();
22           }
23        }
24
25        if (contador[1] >= contador[0]) {
26           return new Decision(true, probabilidade[1] / contador
                [1]);
27
28        } else {
29           return new Decision(false, probabilidade[0] / contador
                [0]);
30        }
31     }
32  }
```

Código: A.4: Classe DecisionTreeVariantOne

```
1  package Sample.DecisionTree;
2
3  import OpenCOM.ILifeCycle;
4  import OpenCOM.IMetaInterface;
```

```
5  import OpenCOM.IUnknown;
6  import OpenCOM.OpenCOMComponent;
7
8  public class DecisionTreeVariantOne extends OpenCOMComponent
9        implements IUnknown, IDecisionTree, IMetaInterface,
           ILifeCycle {
10
11    public DecisionTreeVariantOne(IUnknown pRuntime) {
12       super(pRuntime);
13    }
14
15    public Decision decision(SensorData data) {
16       boolean fire = false;
17       double probErro = 0;
18
19       if (data.getWind() <= 4.9) {
20          if (data.getTemperature() <= 25.9) {
21             if (data.getRh() <= 77) {
22                fire = false;
23                probErro = 3.57;
24             } else {
25                fire = true;
26                probErro = 3;
27             }
28          } else {
29             if (data.getRh() <= 22) {
30                fire = false;
31                probErro = 2;
32             } else {
33                fire = true;
34                probErro = 6.66;
```

```
35              }
36
37          }
38      } else {
39          if (data.getRh() <= 30) {
40              fire = false;
41              probErro = 2;
42          } else {
43              fire = true;
44              probErro = 17;
45          }
46      }
47
48      System.out.println("[" + this + "]executando decision: Fire
            :" + fire + " com ProbErro:" + probErro);
49      return new Decision(fire, probErro);
50  }
51
52  public boolean startup(Object pIOCM) {
53      return true;
54  }
55
56  public boolean shutdown() {
57      return true;
58  }
59 }
```

Código: A.5: Classe DecisionTreeVariantTwo

```
1 package Sample.DecisionTree;
2
3 import OpenCOM.ILifeCycle;
```

```
4  import OpenCOM.IMetaInterface;
5  import OpenCOM.IUnknown;
6  import OpenCOM.OpenCOMComponent;
7
8  public class DecisionTreeVariantTwo extends OpenCOMComponent
9        implements IUnknown, IDecisionTree, IMetaInterface,
           ILifeCycle {
10
11    public DecisionTreeVariantTwo(IUnknown pRuntime) {
12       super(pRuntime);
13    }
14
15    public Decision decision(SensorData data) {
16       boolean fire = false;
17       double probErro = 0;
18
19       if (data.getTemperature() <= 19.8) {
20          if (data.getWind() <= 7.6) {
21             fire = false;
22             probErro = 2.28;
23          } else {
24             if (data.getRh() <= 72) {
25                fire = true;
26                probErro = 14;
27             } else {
28                fire = false;
29                probErro = 2;
30             }
31          }
32       } else {
33          if (data.getRh() <= 24) {
```

```
34              fire = false;
35              probErro = 4;
36          } else {
37              fire = true;
38              probErro = 2.76;
39          }
40      }
41
42      System.out.println("[" + this + "]executando decision: Fire
            :" + fire + " com ProbErro:" + probErro);
43      return new Decision(fire, probErro);
44  }
45
46  public boolean startup(Object pIOCM) {
47      return true;
48  }
49
50  public boolean shutdown() {
51      return true;
52  }
53 }
```

Código: A.6: Classe DecisionTreeVariantThree

```
1 package Sample.DecisionTree;
2
3 import OpenCOM.ILifeCycle;
4 import OpenCOM.IMetaInterface;
5 import OpenCOM.IUnknown;
6 import OpenCOM.OpenCOMComponent;
7
8 public class DecisionTreeVariantThree extends OpenCOMComponent
```

```
implements IUnknown, IDecisionTree,
    IMetaInterface, ILifeCycle {

public DecisionTreeVariantThree(IUnknown pRuntime) {
    super(pRuntime);
}

public Decision decision(SensorData data) {
    boolean fire = false;
    double probErro = 0;

    //AD criada com 243 instancias
    if(data.getTemperature() <=23.1){
        if(data.getWind()<=7.2){
            fire = false;
            probErro = 4.60;
        }else{
            if(data.getWind()>7.2){
                if(data.getRh()<=29){
                    fire = false;
                    probErro = 3;
                }else{
                    if(data.getRh()>29){
                        if(data.getRh()<=72){
                            fire = true;
                             probErro = 7;
                        }else{
                            if(data.getRh()>72){
                                fire = false;
                                 probErro = 3;
                            }
```

```
39                          }
40                      }
41                  }
42              }
43              }
44
45          }if(data.getTemperature()>23.1){
46              if(data.getRh()<=24){
47                  fire = false;
48                   probErro = 9;
49              }else{
50                  if(data.getRh()>24){
51                      fire = true;
52                       probErro = 3.06;
53                  }
54              }
55          }
56
57          System.out.println("[" + this + "]executando decision: Fire
                :" + fire + " com ProbErro:" + probErro);
58          return new Decision(fire, probErro);
59      }
60
61      public boolean startup(Object pIOCM) {
62          return true;
63      }
64
65      public boolean shutdown() {
66          return true;
67      }
68  }
```

Código: A.7: Classe SensorData

```
1  package Sample.DecisionTree;
2
3  public class SensorData {
4
5      private Double temperature;
6      private Double wind;
7      private Double rh;
8
9
10     public SensorData(Double temperature, Double wind, Double rh)
           {
11         super();
12         this.temperature = temperature;
13         this.wind = wind;
14         this.rh = rh;
15     }
16
17     public Double getTemperature() {
18         return temperature;
19     }
20     public Double getWind() {
21         return wind;
22     }
23     public Double getRh() {
24         return rh;
25     }
26
27
28 }
```

REFERÊNCIAS BIBLIOGRÁFICAS

AKYILDIZ, I. F.; SU, W.; SANKARASUBRAMANIAM, Y.; CAYIRCI, E. Wireless Sensor Networks: A Survey. *Comput. Netw.*, Elsevier North-Holland, Inc., New York, NY, USA, v. 38, n. 4, p. 393–422, mar. 2002. ISSN 1389-1286.

AKYILDIZ, I. F.; VURAN, M. C. *Wireless sensor networks*. [S.l.]: John Wiley & Sons, 2010.

ANDERSON, T. T.; LEE, P. A. *Fault tolerance: principles and practice*. 2nd. ed. New York, NY, USA: Springer-Verlag/Wien, 1990.

ANDERSSON, J.; LEMOS, R.; MALEK, S.; WEYNS, D. Software Engineering for Self-Adaptive Systems. In: CHENG, B. H.; LEMOS, R.; GIESE, H.; INVERARDI, P.; MAGEE, J. (Ed.). Berlin, Heidelberg: Springer-Verlag, 2009. cap. Modeling Dimensions of Self-Adaptive Software Systems, p. 27–47. ISBN 978-3-642-02160-2.

AVIZIENIS, A. Design of Fault-tolerant Computers. In: *Proceedings of the November, Fall Joint Computer Conference*. New York, NY, USA: ACM, 1967. p. 733–743.

AVIZIENIS, A. The n-version approach to fault-tolerant software. *IEEE Transactions on Software Engineering*, SE-11, n. 12, p. 1491–1501, Dec 1985.

BEDER, D. M.; UEYAMA, J.; ALBUQUERQUE, J. P. d.; CHAIM, M. L. FlexFT: A Generic Framework for Developing Fault-Tolerant Applications in the Sensor Web. *International Journal of Distributed Sensor Networks*, v. 2013, 2013. ISSN 1550-1477.

BULUSU, N.; ESTRIN, D.; GIROD, L.; HEIDEMANN, J. Scalable coordination for wireless sensor networks: self-configuring localization systems. In: *International Symposium on Communication Theory and Applications (ISCTA 2001), Ambleside, UK*. [S.l.: s.n.], 2001.

CAMAZINE, S.; FRANKS, N. R.; SNEYD, J.; BONABEAU, E.; DENEUBOURG, J.; THERAULA, G. *Self-Organization in Biological Systems*. Princeton, NJ, USA: Princeton University Press, 2001. ISBN 0691012113.

CHEN, D.; TAO, Z.; MA, G. Application of Wireless Sensor Networks for Monitoring Emergency Events. In: *Wireless Communications, Networking and Mobile Computing, 2008. WiCOM '08. 4th International Conference on*. [S.l.: s.n.], 2008. p. 1–4.

CHENG, B. H.; LEMOS, R. de; GIESE, H.; INVERARDI, P.; MAGEE, J.; ANDERSSON, J.; BECKER, B.; BENCOMO, N.; BRUN, Y.; CUKIC, B.; SERUGENDO, G. D. M.; DUSTDAR, S.; FINKELSTEIN, A.; GACEK, C.; GEIHS, K.; GRASSI, V.; KARSAI, G.; KIENLE, H.; KRAMER, J.; LITOIU, M.; MALEK, S.; MIRANDOLA, R.; MLLER, H.; PARK, S.;

SHAW, M.; TICHY, M.; TIVOLI, M.; WEYNS, D.; WHITTLE, J. Software Engineering for Self-Adaptive Systems: A Research Roadmap. In: CHENG, B. H.; LEMOS, R.; GIESE, H.; INVERARDI, P.; MAGEE, J. (Ed.). *Software Engineering for Self-Adaptive Systems*. [S.l.]: Springer Berlin Heidelberg, 2009, (Lecture Notes in Computer Science, v. 5525). p. 1–26. ISBN 978-3-642-02160-2.

COULSON, G.; BLAIR, G.; GRACE, P.; TAIANI, F.; JOOLIA, A.; LEE, K.; UEYAMA, J.; SIVAHARAN, T. A generic component model for building systems software. *ACM Trans. Comput. Syst.*, ACM, New York, NY, USA, v. 26, n. 1, p. 1:1–1:42, mar. 2008. ISSN 0734-2071.

DRESSLER, F. *Self-Organization in Sensor and Actor Networks*. [S.l.]: Wiley, 2007. ISBN 978-0-470-02820-9.

HALL, M.; FRANK, E.; HOLMES, G.; PFAHRINGER, B.; REUTEMANN, P.; WITTEN, I. H. *The WEKA Data Mining Software: An Update*. 2009.

HEFEEDA, M.; BAGHERI, M. Wireless sensor networks for early detection of forest fires. In: *Mobile Adhoc and Sensor Systems, 2007. MASS 2007. IEEE International Conference on*. [S.l.: s.n.], 2007. p. 1–6.

HORN, P. *Autonomic computing: IBM's Perspective on the State of Information Technology*. [S.l.]: IBM, 2001. `http://people.scs.carleton.ca/~soma/biosec/readings/autonomic_computing.pdf`.

HUEBSCHER, M. C.; MCCANN, J. A. A Survey of Autonomic Computing: Degrees, Models, and Applications. *ACM Comput. Surv.*, ACM, New York, NY, USA, v. 40, n. 3, p. 7:1–7:28, ago. 2008. ISSN 0360-0300.

IBM. *The Tivoli software implementation of autonomic computing guidelines*. 2002. ftp://ftp.software.ibm.com/software/tivoli/brochures/br-autonomic-guide.pdf.

INPE. *Monitoramento dos Focos Ativos no Brasil*. 2016. Acesso em: 21 jan. 2016. Disponível em: <http://www.inpe.br/queimadas/estatisticas.php>.

JALOTE, P. *Fault tolerance in distributed systems*. New Jersey, USA: Prentice Hall, 1994.

JOSHI, Y.; YOUNIS, M. Autonomous recovery from multi-node failure in Wireless Sensor Network. In: *Global Communications Conference (GLOBECOM), 2012 IEEE*. [S.l.: s.n.], 2012. p. 652–657. ISSN 1930-529X.

KEPHART, J. O.; CHESS, D. The Vision of Autonomic Computing. *Computer*, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 36, n. 1, p. 41–50, jan. 2003. ISSN 0018-9162.

KUROSE, J.; ROSS, K. W. *Redes de computadores e a Internet: uma abordagem top-down*. 5. ed. New York: Addison-Wesley, 2010. ISSN 978-85-88639-97-3.

LAPRIE, J. Dependable computing and fault tolerance: Concepts and terminology. In: IEEE. *Fault-Tolerant Computing, 1995, Highlights from Twenty-Five Years., Twenty-Fifth International Symposium on*. [S.l.], 1995. p. 2.

LAPRIE, J.-C. Dependable computing and fault-tolerance. *Digest of Papers FTCS-15*, p. 2–11, 1985.

LEWIS, F. L. et al. Wireless sensor networks. *Smart environments: technologies, protocols, and applications*, New York: Wiley, p. 11–46, 2004.

LICHMAN, M. *UCI Machine Learning Repository*. 2013. Disponível em: <http://archive.ics.uci.edu/ml>.

LIU, Y.; GU, Y.; CHEN, G.; JI, Y.; LI, J. A novel accurate forest fire detection system using wireless sensor networks. In: *Mobile Ad-hoc and Sensor Networks (MSN), 2011 Seventh International Conference on*. [S.l.: s.n.], 2011. p. 52–59.

LOUREIRO, A. A.; NOGUEIRA, J. M. S.; RUIZ, L. B.; NAKAMURA, E. F.; FIGUEIREDO, C. M. S. Redes de Sensores sem Fio. In: *XII Simpósio Brasileiro de Redes de Computadores*. [S.l.: s.n.], 2003. Acesso em: 18 nov. 2014.

PRADHAN, D. K. *Fault-Tolerant System Design*. New Jersey, USA: Prentice Hall, 1996.

RAGHAVENDRA, C. S.; SIVALINGAM, K. M.; ZNATI, T. *Wireless sensor networks*. [S.l.]: Springer, 2006.

RANDELL, B. System structure for software fault tolerance. *IEEE Transactions on Software Engineering*, SE-1, n. 2, p. 220–232, June 1975.

RANDELL, B.; XU, J. The evolution of the recovery block concept. *Software Fault Tolerance, Trends in Software*, Wiley, p. 1–22, 1994.

RUIZ, L. B. *Uma arquitetura para o gerenciamento de redes de sensores sem fio*. 2002. Echnical Report DCC/UFMG RT.005/2002.

Samuel Madden and Kay Romer and Holger Karl and Friedemann Mattern. *Wireless Sensor Networks: Third European Workshop, EWSN 2006, Zurich, Switzerland, February 13-15, 2006. Proceedings*. 1. ed. [S.l.]: Springer-Verlag Berlin Heidelberg, 2006. (Lecture Notes in Computer Science 3868 : Computer Communication Networks and Telecommunications). ISBN 3540321586,9783540321583.

SINALGO. Distributed Computing Group at ETH Zurich., 2015. Acesso em: 01 mar. 2015. Disponível em: <http://disco.ethz.ch/projects/sinalgo/index.html>.

STANKOVIC, J. A. Wireless sensor networks. *IEEE Computer*, v. 41, n. 10, p. 92–95, 2008.

TATARA, K.; LEE, G.; CHONG, N. Y. Self-organizing ad-hoc robotic sensor networks based on locally communicative interactions. In: *Ubiquitous Robots and Ambient Intelligence (URAI), 2011 8th International Conference on*. [S.l.: s.n.], 2011. p. 485–490.

TELECO. *Rede de Sensores Sem Fio: Arquitetura e Gerência*. 2014. Acesso em: 09 nov. 2014. Disponível em: <http://www.teleco.com.br/tutoriais/tutorialrssf/pagina_4.asp>.

VILELA, M.; ARAUJO, R. RAHMoN: Routing Algorithm for Heterogeneous Mobile Networks. In: *Critical Embedded Systems (CBSEC), 2012 Second Brazilian Conference on*. [S.l.: s.n.], 2012. p. 24–29.

VILLAS, L. A.; BOUKERCHE, A.; ARAUJO, R. B.; LOUREIRO, A. A. A reliable and data aggregation aware routing protocol for wireless sensor networks. In: *Proceedings of the 12th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*. New York, NY, USA: ACM, 2009. (MSWiM '09), p. 245–252. ISBN 978-1-60558-616-8. Disponível em: <http://doi.acm.org/10.1145/1641804.1641846>.

WEBER, T. S. *Tolerância a falhas: conceitos e exemplos*. 2001. Acesso em: 09 nov. 2014. Disponível em: <http://www.inf.ufrgs.br/taisy/disciplinas/textos/ConceitosDependabilidade.pdf>.

WOOLDRIDGE, M.; JENNINGS, N. Intelligent agents: Theory and Practice. Knowledge Engineering Review, 1995.

YANG, S.-H. *Wireless Sensor Networks: Principles, Design and Applications*. 1. ed. [S.l.]: Springer-Verlag London, 2014. (Signals and Communication Technology). ISBN 978-1-4471-5504-1,978-1-4471-5505-8.

YATES, F. E. (Ed.). *Self-Organizing Systems: The Emergence of Order*. [S.l.]: Springer US, 1987. Copyright Holder: Plenum Press, New York.

YOON, I.; NOH, D. K.; LEE, D.; TEGUH, R.; HONMA, T.; SHIN, H. Reliable wildfire monitoring with sparsely deployed wireless sensor networks. In: *Advanced Information Networking and Applications (AINA), 2012 IEEE 26th International Conference on*. [S.l.: s.n.], 2012. p. 460–466. ISSN 1550-445X.