UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
CIÊNCIA DA COMPUTAÇÃO

IMPLEMENTAÇÃO DE TÉCNICAS DE RESOLUÇÃO DE CONFLITOS PARA SISTEMAS UBÍQUOS COM SUPORTE DE COMUNICAÇÃO DE GRUPO

TRABALHO DE GRADUAÇÃO

Felipe Silvano Perini

Santa Maria, RS, Brasil

2013

IMPLEMENTAÇÃO DE TÉCNICAS DE RESOLUÇÃO DE CONFLITOS PARA SISTEMAS UBÍQUOS COM SUPORTE DE COMUNICAÇÃO DE GRUPO

Felipe Silvano Perini

Trabalho de Graduação apresentado ao Ciência da Computação da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para a obtenção do grau de

Bacharel em Ciência da Computação

Orientadora: Profª. Dr. Marcia Pasin

Trabalho de Graduação Nº 360
Santa Maria, RS, Brasil

2013

Universidade Federal de Santa Maria
Centro de Tecnologia
Ciência da Computação

A Comissão Examinadora, abaixo assinada,
aprova o Trabalho de Graduação

IMPLEMENTAÇÃO DE TÉCNICAS DE RESOLUÇÃO DE CONFLITOS PARA SISTEMAS UBÍQUOS COM SUPORTE DE COMUNICAÇÃO DE GRUPO

elaborado por
Felipe Silvano Perini

como requisito parcial para obtenção do grau de
Bacharel em Ciência da Computação

COMISSÃO EXAMINADORA:

Marcia Pasin, Dr.
(Presidente/Orientadora)

Andrea Schwertner Charão, Dr. (UFSM)

Patricia Pitthan Barcelos, Dr. (UFSM)

Santa Maria, 26 de julho de 2013.

RESUMO

Trabalho de Graduação
Ciência da Computação
Universidade Federal de Santa Maria

IMPLEMENTAÇÃO DE TÉCNICAS DE RESOLUÇÃO DE CONFLITOS PARA SISTEMAS UBÍQUOS COM SUPORTE DE COMUNICAÇÃO DE GRUPO

AUTOR: FELIPE SILVANO PERINI
ORIENTADORA: MARCIA PASIN
Local da Defesa e Data: Santa Maria, 26 de julho de 2013.

A utilização de dispositivos móveis vem se tornando cada vez mais popular. Sistemas embarcados vem ganhando maior poder de processamento, e com isso, possibilitando um aumento no seu número de funções. A integração destes computadores, em suas mais variadas formas, é uma tendência para o futuro que já começa a ser aplicada em nosso cotidiano. Interligar estes aparelhos e tornar o sistema transparente ao usuário é uma tarefa da computação ubíqua. Porém, características desses dispositivos (tais como, a heterogeneidade de seus sistemas, a dependência de bateria, mobilidade e instabilidade da conexão) ocasionam sérios problemas para um sistema ubíquo. Neste trabalho, são implementadas soluções com a utilização de comunicação em grupos para resolver questões decorrentes do dinamismo, mobilidade e instabilidade de conexão de um sistema ubíquo. Através da comunicação em grupo é possível prover uma maior transparência diante dos problemas ocasionados pela utilização de dispositivos móveis e embarcados e, ainda, aumentar a confiabilidade do sistema através de tolerância a falhas.

Palavras-chave: Comunicação em Grupo. Resolução de Conflitos. Sistemas Ubíquos.

ABSTRACT

Undergraduate Final Work
Undergraduate Program in Computer Science
Federal University of Santa Maria

IMPLEMENTATION OF CONFLICT RESOLUTION TECHNIQUES FOR UBIQUITOUS COMPUTING WITH GROUP COMMUNICATION SUPPORT

AUTHOR: FELIPE SILVANO PERINI
ADVISOR: MARCIA PASIN
Defense Place and Date: Santa Maria, February 26th, 2013.

The use of mobile devices is becoming increasingly popular. Embedded computing is becoming more powerful and, consequently, increasing their functions. The integration of these computers is a tendency for the future. Connecting these devices and building a transparent environment for the user is the mission of the ubiquitous computing. However, characteristics of these devices (such as heterogeneity, battery dependency, mobility, and connection instability) cause serious issues for an ubiquitous system. In this work, solutions using group communication to solve issues related to dynamism, mobility and connection instability are implemented. With the use of group communication it is possible to provide a better transparency against the issues caused by the use of mobile devices and, also, increase reliability through fault tolerance.

Keywords: Conflict Resolution Techniques, Group Communication, Ubiquitous Systems.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

FIFO *First In First Out*

JMS *Java Message Service*

QoS *Quality of Service*

TCP *Transmission Control Protocol*

UDP *User Datagram Protocol*

XML *eXtensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

1.1 Contextualização

O uso de novas tecnologias na vida cotidiana, notadamente dipositivos móveis, está se ternando cada vez mais popular. Aplicações distribuídas que suportam estes dispositivos e tecnologias precisam se adaptar para prover qualidade contínua de serviço apesar de mudanças no contexto, tais como variações na largura de banda de rede, consumo de bateria, conectividade, mobilidade, acessibilidade de serviços e hosts, etc. Neste cenário, a pesquisa em computação ubíqua (WEISER, 1993) tem ganhado força.

A implementação de soluções para sistemas ubíquos sobre as infraestruturas heterogêneas de *software* e *hardware* existentes não é uma tarefa trivial. De fato, decisões adaptativas (TIGLI et al., 2009) as quais sistemas ubíquos estão sujeitos podem levar a ações conflitantes. Gerenciamento de conflito em sistemas ubíquos com qualidade de serviço representa um grande desafio e poucas soluções tem sido implementadas.

Técnicas tradicionais para a resolução de conflitos (FIFO, uso de prioridades, escolhas randômicas, votação, etc.) estão sendo constantemente adaptadas para acomodar cenários dinâmicos e distribuídos, como aqueles requeridos em sistemas ubíquos. Destaca-se a implementação dessas estratégias a nível de *middleware* (CAPRA; EMMERICH; MASCOLO, 2002; HASAN et al., 2006; CHAE et al., 2007; SILVA; RUIZ; LOUREIRO, 2010). *Middleware* para sistemas ubíquos deve ter a capacidade de resolver conflitos em tempo de execução levando em conta exigências de uma aplicação em determinados cenários e/ou contexto adaptativo, incluindo mobilidade de processos, heterogeneidade, alta demanda, distância geográfica, etc.

De forma geral, em presença de ambientes distribuídos, soluções existentes pecam pela falta de descentralização e, portanto, não são escalares nem adequadas para muitos ambientes reais (por exemplo gerenciamento de trânsito de veículos, gerenciamento de dispositivos em um campus ou aeroporto). A implementação de mecanismos para realizar detecção e resolução de conflito de forma mais eficiente e ampla, que leve em conta distribuição e necessidades de sistemas ubíquos, são altamente desejáveis. Neste sentido, acredita-se que comunicação de grupo e o conceito de sincronia virtual (BIRMAN; JOSEPH, 1987) podem prover suporte adequado para mapear processos interessados em resolver conflitos como grupos.

A comunicação de grupo é utilizada para comunicação entre dois ou mais computadores

presentes em um sistema distribuído. Provê maior transparência e, ao mesmo tempo, garante uma comunicação confiável entre os nodos operacionais do sistema.

1.2 Objetivos

Este trabalho de graduação tem como objetivo principal realizar a avaliação experimental de técnicas para resolução de conflitos em sistemas ubíquos com suporte de comunicação de grupo.

Para concluir o objetivo deste trabalho será seguida a seguinte metodologia:

- Estudo de conceitos de comunicação de grupo;
- O estudo de algoritmos para resolução de conflitos em sistemas ubíquos;
- A implementação de soluções para resolução de conflitos em sistemas ubíquos;
- O estudo da robustez das soluções dos algoritmos sobre pequenas perturbações dos parâmetros;
- Avaliação da escalabilidade das soluções implementadas através da variação do número de processos no sistema.

1.3 Estrutura do texto

Este texto esta estruturado como segue:

- O capítulo 2 apresenta uma revisão bibliográfica sobre os assuntos necessários para o desenvolvimento do trabalho, o que inclui sistemas ubíquos, possíveis conflitos em computação ubíqua e comunicação em grupo aplicada a estes sistemas;
- O capítulo 3 introduz diferentes técnicas para resolução de conflitos em sistemas ubíquos, discute estas técnicas e descreve de que forma elas foram utilizadas para implementação deste trabalho;
- Detalhes da implementação das técnicas para resolução de conflitos em conjunto com o sistema de comunicação de grupos são descritos no capítulo 4.
- O capítulo 5 apresenta os cenários de execução e os resultados obtidos através da implementação;

- Por fim, o capítulo 6 conclui este trabalho e sugere novos desafios para o futuro baseados nas experiências adquiridas durante o desenvolvimento.

2 REVISÃO BIBLIOGRÁFICA

Sistemas ubíquos são caracterizados por restrições dos recursos (bateria, largura de banda, entre outros), distribuição e heterogeneidade de $hardware$ e $software$. Este cenário favorece a existência de conflitos. Com o intuito de aumentar a confiabilidade do sistema o suporte de comunicação de grupo surge como ferramenta fundamental.

Neste capitulo, o item 2.1 descreve brevemente computação ubíqua, a seção 2.2 trata mais especificamente de conflitos e resolução de conflitos neste ambiente. O item 2.3 apresenta comunicação de grupo e de que forma ela traz maior confiabilidade ao sistema. O leitor familiarizado com estes assuntos poderá passar ao próximo capitulo.

2.1 Computação ubíqua

Sistemas computadorizados estão presentes em diferentes lugares na nossa vida cotidiana e nos oferecem variadas funcionalidades. O funcionamento integrado desses sistemas sem a percepção do usuário é chamada de computação ubíqua (WEISER, 1993).

A computação ubíqua é considerada a terceira era da computação moderna, seguida da era dos $mainframes$ (primeira era) e dos computadores pessoais (segunda era) (KRUMM, 2009). É resultado de um mundo onde uma única pessoa possui diversos dispositivos computadorizados.

De uma forma geral, sistemas de computação ubíqua caracterizam-se pela descentralização, diversidade e conectividade (ARAUJO, 2003).

A descentralização refere-se a capacidade de gerenciamento do próprio dispositivo. Ele é capaz de gerenciar a si mesmo e influenciar nas decisões do sistema através da cooperação.

Um único dispositivo dentro do sistema ubíquo apresenta diversas funcionalidades. Algumas destas funcionalidades sobrepõe-se a funções de outros aparelhos. A reprodução de vídeos, por exemplo, pode ser feita tanto pela televisão como por um $tablet$ ou $smartphone$.

Através de uma conexão comum os dispositivos são capazes de trocar informações entre si. Dispositivos e aplicações formam um sistema único, servindo como chave para a descentralização do ambiente.

Nesta nova era da computação, integram-se sistemas de computação móvel (como PDAs, $smartphones$, $tablets$) e embutidos (presentes em sistemas domésticos, em sistemas eletrôni-

cos para tratamento de saúde e em diversos tipos de redes de sensores (TANENBAUM; STEEN, 2007)). Estes eletrônicos possuem algumas particularidades, tais como: pequeno tamanho, mobilidade, instabilidade de conexão, dependência de bateria e heterogeneidade de recursos.

Interligar dispositivos tão diferenciados e apresentá-los ao usuário de forma transparente não é uma tarefa trivial. Devido as suas características peculiares, surgem problemas de diferentes complexidades.

A computação sensível ao contexto ($Context-aware\ Computing$) surge como aspecto-chave para implementação de questões como a transparência e disponibilidade de serviços sob demanda aos usuários dentro da computação ubíqua (MOURA BRAGA SILVA, 2010).

A utilização de sistemas cientes de contexto tem se tornado uma ótima solução para sistemas ubíquos (HUEBSCHER; MCCANN; HOSKINS, 2006), pois são sistemas que se adaptam de acordo com as mudanças que acontecem no ambiente ao longo do tempo. Através da utilização de contextos é possível construir um ambiente auto-gerenciável, onde cada dispositivo conectado a ele possui um perfil diferenciado e influencia na tomada de decisões. O contexto de um sistema ubíquo pode ser classificado em *Contexto Individual* e *Contexto Coletivo*.

Dispositivos conectados a uma rede tendem a possuir funções semelhantes ou que se sobrepõem. Vídeos em alta resolução podem ser visualizados tanto em um televisor quanto em um aparelho celular, por exemplo. A escolha de qual dispositivo deve assumir a reprodução do vídeo pode ser feita automaticamente levando em consideração a localização do usuário na casa ou ainda através de preferências pré-definidas por ele. Neste contexto, há somente um usuário que interage com os diversos aparelhos do ambiente. Este é um cenário típico de *Contexto Individual*.

O *Contexto Coletivo* é caracterizado por possuir dois ou mais usuários simultâneos. Por exemplo, imagine que há duas pessoas em uma sala que desejam assistir televisão. A televisão, por sua vez, é controlada por um sistema computadorizado que escolhe a programação automaticamente baseado em preferências dos usuários. Cada um dos usuários possui um perfil previamente registrado que contém os seus programas prediletos. Então o sistema irá cruzar os dados e decidir qual canal irá satisfazer melhor todos os usuários.

Sistemas ubíquos devem adaptar-se a mudanças no ambiente. Isto é, devem ser capazes de capturar um contexto, atribuir a ele um significado e, baseado nisso, alterar seu funcionamento. Descobrir se há uma rede com melhor disponibilidade ao seu alcance e identificar se há um dispositivo mais adequado para a atividade que o usuário deseja, são exemplos simples

dessas mudanças.

Mesmo com as tecnologias atuais, há tarefas que são preferenciais ou específicas de alguns dispositivos. O usuário pode preferir ler um $ebook$ em um $tablet$, pois possui uma tela maior que um $smartphone$. Enquanto inicia a leitura, a cafeteira envia para o seu $smartphone$ uma mensagem avisando que o café está pronto. Depois da leitura, o usuário se dirige ao seu local de trabalho. O sistema então detecta que não há ninguém em casa e automaticamente desliga os eletrônicos e luzes desnecessários. Neste contexto, os dispositivos formam um ambiente inteligente, um cenário de pesquisa atual dentro de grandes empresas e com enorme potencial para o futuro (BRUMITT et al., 2000; COOKA; AUGUSTOB; JAKKULAA, 2009).

2.2 Conflitos em computação ubíqua

Conflitos são decorrentes de uma incompatibilidade de ideias ou funções. Em computação ubíqua, eles se apresentam de maneiras diferenciadas. A sobreposição de funções dos dispositivos e as diferenças de perfis de usuários são as principais causas de conflitos.

Os conflitos provenientes de sistemas ubíquos com único usuário (*Contexto Individual*) são de mais simples resolução. Em uma casa inteligente, a escolha de qual dispositivo deve assumir a execução de um vídeo pode ser definida por informações armazenadas anteriormente ou simplesmente pela localização do usuário no ambiente.

Quando sistemas de computação ubíqua são compartilhados por mais de um usuário, novos conflitos aparecem. Estes conflitos são originados devido aos diferentes interesses dos usuários e/ou resultados da falta de recursos físicos do ambiente (SHIN; WOO, 2005). Dessa forma, o sistema não consegue adaptar-se para satisfazer todas as necessidades individuais e coletivas. Conflitos resultantes de sistemas multi-usuários são mais difíceis de serem resolvidos, pois envolvem uma análise dinâmica do contexto e das preferencias dos usuários. Estes conflitos originados de um *Contexto Coletivo* são nomeados *Conflitos Coletivos*.

Diferentes técnicas foram aplicadas dentro de sistemas de computação ubíqua para tratar dos conflitos resultantes da disponibilização dos recursos para múltiplos usuários. Algumas delas visam satisfazer necessidades individuais (tais como, FIFO, prioridade, votação, prioridades utilizando contexto histórico), enquanto outras visam a satisfação coletiva (CAPRA; EMMERICH; MASCOLO, 2002; PARK; LEE; HYUN, 2005).

No entanto, muitas das técnicas desenvolvidas para resolver conflitos são soluções específicas para determinados tipos de aplicações com limitado número de usuários. Não apresen-

tam um controle de entrada e saída de membros, portanto, não estão preparadas para lidar com uma variação na quantia de usuários e com os consequentes problemas que isto traz. Assim, a comunicação em grupo surge como um meio confiável para o gerenciamento das informações do contexto.

Outras técnicas consideram a escalabilidade do sistema, porém não abordam um problema comum quando trabalha-se com sistemas distribuídos: identificação e gerenciamento de falhas. Este tema será abordado na seção seguinte.

2.3 Comunicação em grupo como suporte para comunicação confiável

A comunicação de grupo é uma alternativa segura para comunicação entre dois ou mais computadores presentes em um sistema distribuído (TANENBAUM; STEEN, 2007). Usualmente ela é utilizada para providenciar transparência a réplicas de servidores, oferecendo um meio único para troca de mensagens entre os membros de um grupo específico. Dessa forma, permite ao sistema construir maneiras eficazes de proteger-se contra modificação, intervenção e interceptação de informações.

Uma importante característica de sistemas de comunicação de grupo é a confiabilidade (BIRMAN., 1997). Para um sistema de comunicação de grupo ser confiável ele deve certificar-se de que todas as mensagens enviadas a um grupo são entregues a todos os membros operacionais contidos nele.

A transmissão de informação dentro da comunicação em grupo pode ser implementada através de diferentes métodos de encaminhamento ($unicast$, $multicast$, $broadcast$, etc) e protocolos de transporte (UDP, TCP, JMS). Caso a comunicação de grupo esteja utilizando $broadcast$ ou $multicast$ e, por exemplo, a mensagem não chegar a um dos seus destinos, a conclusão do envio dos dados deve ser feita de forma individual, garantindo que todos os destinatários recebam a mensagem.

Outro aspecto muito importante para a confiabilidade do sistema é a consistência, o que exige que a troca de mensagens entre o grupo seja feita de forma síncrona e atômica. O sincronismo garante que os membros do grupo estejam recebendo as informações na ordem em que elas foram enviadas. Enquanto a atomicidade assegura que ou os dados foram entregues a todos os destinatários operacionais ou a nenhum. Além disso, o sistema deve funcionar de forma contínua apesar de falhas, mantendo a atomicidade e sincronia (BIRMAN; JOSEPH, 1987).

Estas características também são desejáveis dentro de um sistema ubíquo. Na literatura são encontradas poucas referências sobre resolução de conflitos utilizando comunicação de grupo dentro de sistemas ubíquos. Trabalhos recentes propõem diferentes esquemas para resolução de conflitos, porém apresentam caso de uso limitado, não foram planejados para um ambiente escalável e/ou não consideram a ocorrência de falhas.

$Park\ et\ al.$ (2005) propõe um esquema para resolução de conflitos baseado nas preferências de usuários, mas apresenta um caso de uso limitado e não considera a ocorrência de falhas.

$Lopez\ et\ al.$ (2009), por outro lado, implementa um $middleware$ diferenciado para comunicação de grupo em sistemas móveis, eliminando o $overhead$ causado pela troca de mensagens em sistemas tradicionais através do conhecimento da topologia da aplicação. Apresenta um modelo capaz de identificar e tratar as falhas, porém não aborda os conflitos que podem surgir devido aos diferentes perfis de usuário.

3 SUPORTE DE COMUNICAÇÃO DE GRUPO PARA RESOLUÇÃO DE CONFLITOS EM SISTEMAS UBÍQUOS

Este capitulo introduz características básicas utilizadas ao longo da implementação do trabalho. Na seção 3.1 são abordadas características do projeto de implementação do sistema de resolução de conflitos com comunicação de grupo. Em seguida, o item 3.2 introduz o conceito das diferentes técnicas de resoluções de conflitos implementadas, são elas: estratégia de votação, prioridades e leilão. Por fim, o item 3.3 discute as técnicas utilizadas neste trabalho em conjunto com um sistema de comunicação de grupo.

3.1 Projeto da implementação

Para implementar a proposta apresentada neste trabalho, primeiramente montou-se um esqueleto da estrutura do sistema ubíquo, considerando apenas a comunicação, identificação e solução de conflitos. Posteriormente, foi desenvolvido um esquema considerando um cenário de falhas onde a comunicação de grupos é utilizada como suporte.

O sistema ubíquo está envolto em um grande grupo, composto por diferentes dispositivos, que recebe e envia mensagens através de um canal único de comunicação. A atuação do sistema é acionada por sensores espalhados pelo ambiente (detectando novos dispositivos existentes na rede ou dispositivos falhos) ou, simplesmente, através da influência do usuário.

Cada um dos dispositivos presentes no grupo envia e recebe novas usando o suporte da comunicação de grupo, atualizando o contexto. A partir disso, o sistema procura por dados conflitantes e, caso necessário, aplica um dos algoritmos especificados para resolver o conflito. E por fim, o sistema adapta-se levando em consideração os resultados obtidos após a execução do algoritmo de resolução. A estrutura do sistema ubíquo apresentado até o momento é visualizada na figura 3.1.

A figura 3.1 organiza o sistema em três camadas. A primeira, nomeada *Sistema de Comunicação de Grupo*, é responsável por prover transparência e estabelecer a comunicação entre os diversos dispositivos do sistema (descrito na seção 2.3).

A segunda camada é composta pelos dispositivos do sistema ubíquo. É onde a resolução dos conflitos é executada, ou seja, em cada nodo individualmente. Cada dispositivo comunica-se com os demais membros do grupo através da primeira camada e, quando necessário, requisita informações através dela.

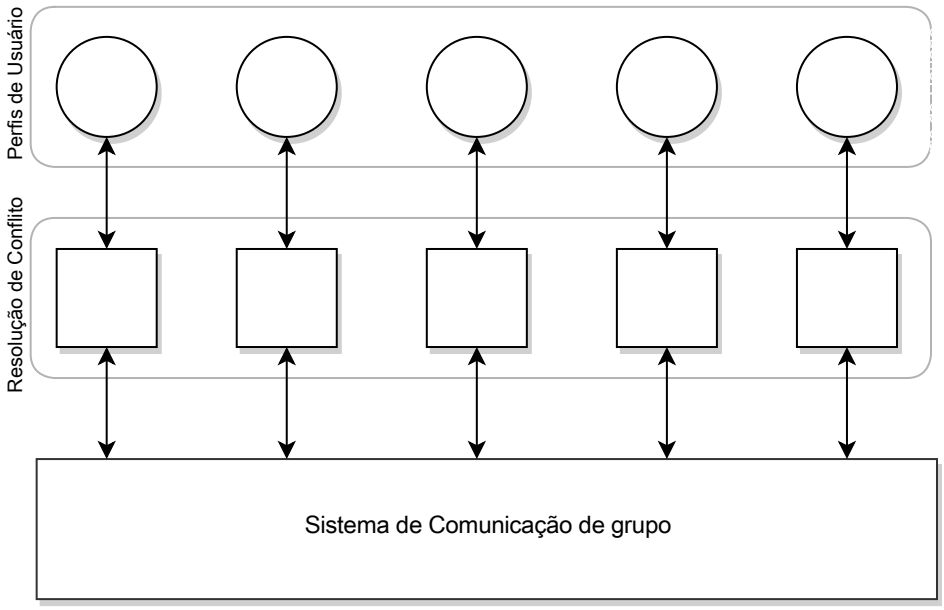

Figura 3.1 – Estrutura do sistema ubíquo com comunicação de grupo

A última camada é chamada de *Perfis de Usuário*. Esta camada é caracterizada pelas informações de perfil de cada dispositivo, disponibilizadas dependendo da necessidade do algoritmo de resolução de conflito (utilizadas no algoritmo de prioridade (seção 3.4) e leilão(seção 3.5). Assim, sempre que um dispositivo desejar solucionar um conflito, ele faz a requisição ao *Sistema de Comunicação de Grupo* que contacta os demais nodos. Cada um destes nodos consultará o seu *Perfil de Usuário* e, então, enviará as informações através do meio comum de comunicação.

Para um melhor entendimento do sistema, utilizaremos o ambiente da casa inteligente como exemplo. Para isto, retomamos a situação descrita na seção 2.1 onde uma televisão seleciona automaticamente o canal que melhor satisfaça os diferentes perfis de usuários. Neste ambiente, os dispositivos contidos no sistema de comunicação de grupo são a própria televisão, os usuários presentes no ambiente e os demais eletrônicos que nele influenciam.

A ação necessária para a atividade ser iniciada é definida como o ato de ligar a televisão. Logo em seguida, inicia-se o processo de comunicação entre os membros do grupo. Como resultado, os dispositivos já presentes no grupo tomam conhecimento que a televisão esta funcionando e, do mesmo modo, a televisão recebe as informações dos dispositivos conectados.

A partir deste momento, um ou mais dispositivos trocam as informações para obter a melhor escolha entre os canais habilitados. Se houver conflito de interesses entre as preferências dos usuários um algoritmo é aplicado. Após o algoritmo de resolução de conflito ser executado, todos os nodos presentes no grupo tomam conhecimento do resultado final.

Caso outra pessoa ingressar no grupo inicial o processo é reiniciado, portanto ela também é um agente inicializador do sistema. O mesmo acontece quando um dos membros (neste caso, uma das pessoas que estavam interessadas em assistir televisão) deixar o grupo, um novo contexto se forma e é necessário que todo o sistema esteja ciente dessas modificações. Ou seja, qualquer alteração que atue dentro de um grupo, sejam pessoas ou dispositivos, influencia diretamente na satisfação do sistema. Assim, será necessário formular uma arquitetura que suporte estas operações.

A comunicação de grupo também é responsável por identificar a disponibilidade dos nodos através de mensagens $multicast$. Quando um dispositivo deixa de responder ao grupo, o sistema o identifica como um nodo falho. Isto pode acontecer devido à perda de conexão, término de bateria, um fator de posição geográfico (por exemplo, um dispositivo móvel fica fora do alcance dos sensores), dentre outros.

Depois da detecção da falha, o sistema passa para a fase de tratamento de falhas. Este módulo compara o contexto do sistema antes e depois de ocorrer a falha. Através da identificação deste contexto ele toma medidas necessárias para uma melhor adaptação do sistema.

Por fim, as informações de cada um dos nodos é atualizada individualmente utilizando o sistema de comunicação de grupo. Caso seja necessário, as últimas atividades são refeitas. A figura 3.2 demonstra a estrutura com tolerância a falhas.

Mais precisamente, a figura 3.2 exibe quatro módulos diferenciados. A comunicação em grupo, a detecção e o tratamento de falha são mecanismos providos pela própria ferramenta de gerenciamento de grupos. Após o tratamento de uma falha, o contexto é atualizado dependendo das necessidades da aplicação. As consequências do tratamento de falhas na atualização do contexto serão abordadas no capítulo 4.

3.2 Técnicas de resolução de conflitos

Os algoritmos selecionados para implementação distribuída com suporte de grupo incluem técnicas simples como votação e prioridades, e uma estratégia mais sofisticada de leilão (CAPRA; EMMERICH; MASCOLO, 2002). Estas técnicas foram escolhidas pois consideram contextos diferenciados, possuem diferentes objetivos e, também, são aplicáveis a um mesmo cenário.

De forma geral, estratégias simples são mais fáceis de implementar, mas podem conduzir a resultados com baixa Qualidade de Serviço (QoS). Estratégias sofisticadas podem con-

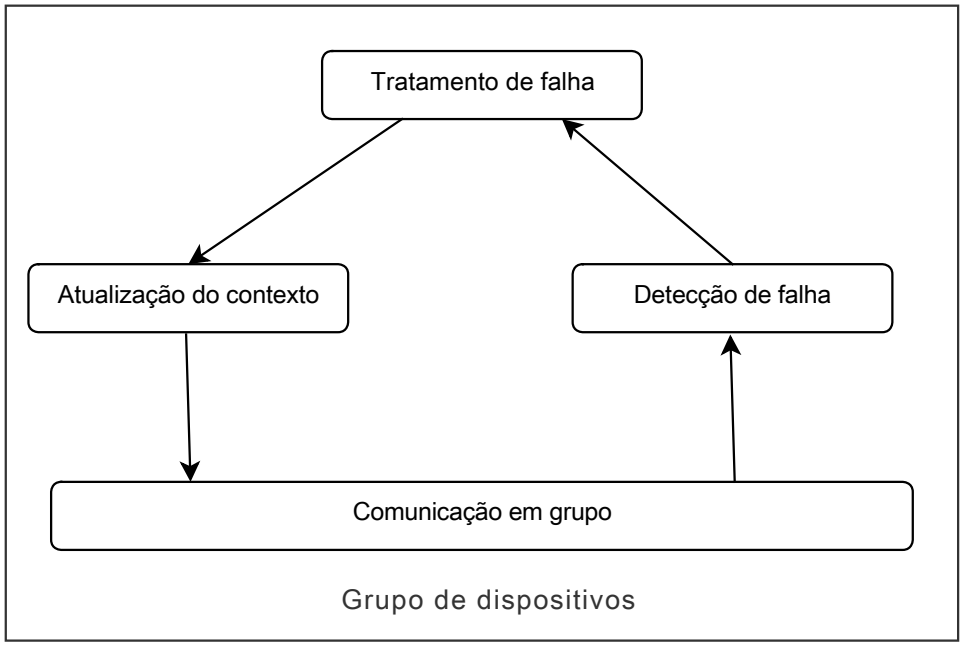

Figura 3.2 – Esboço da arquitetura do sistema de comunicação de grupos com tolerância a falhas

duzir a soluções com melhor QoS, mas incluem $overhead$ para cálculos, contabilização de informação de contexto e, possivelmente, manutenção de dados históricos.

Existem diferentes técnicas para resolver conflitos em sistemas distribuídos, porém nem todas são aplicáveis a sistemas móveis e pervasivos. Algumas destas técnicas já foram implementadas no contexto de sistemas ubíquos. A seguir, são descritas as técnicas relevantes para este trabalho.

3.2.1 Votação

O algoritmo de votação visa estabelecer um vencedor baseado em informações geradas dinamicamente pelos elementos do grupo. Este algoritmo possui objetivo individual, contemplando somente um dos nodos presentes no grupo. Como os votos são gerados dinamicamente, não é armazenada nenhuma informação do contexto. Caso dois ou mais dispositivos gerarem o maior voto (portanto, possuem o voto com o mesmo valor), uma nova votação é iniciada somente entre eles.

A técnica de votação foi implementada com suporte distribuído e operando em dois $rounds$. No primeiro $round$, um processo coordenador (aquele que detectou o conflito) inicia uma votação enviando a solicitação de votos aos demais participantes do grupo. Cada processo no grupo ao receber a mensagem do coordenador, gera o voto localmente e envia o voto a todos os processos não-falhos do grupo. No segundo $round$, cada processo ao receber a mensagem

com o voto de todos os processos não-falhos, compara os votos recebidos localmente, sem consenso distribuído, e identifica o maior voto. O maior voto será usado para a resolução de conflitos.

Algoritmo 1 Algoritmo de Votação

1: {Round 1}
Entrada: Conjunto de elementos do grupo $K = \{\sigma_1, \ldots, \sigma_n\}$ não vazio, n número de elementos do conjunto K
2: **para todo** $\sigma_i \in K$ tal que $0 < i \leq n$ **faça**
3: Envia voto gerado aleatoriamente para σ_i
4: **fim para**

5: {Round 2}
Entrada: Conjunto de elementos do grupo $K = \{\sigma_1, \ldots, \sigma_n\}$ não vazio, n número de elementos do conjunto K.
Saída: Método prioritário do elemento de K que possui o maior voto
6: $vencedor \leftarrow \sigma_1$
7: **para todo** $\sigma_i \in K$ tal que $1 < i \leq n$ **faça**
8: **se** voto de σ_i >voto do $vencedor$ **então**
9: $vencedor \leftarrow \sigma_i$
10: **fim se**
11: **fim para**
12: **retorna** $vencedor$

Figura 3.3 – Algoritmo 1. Resolução de conflitos por votação

3.2.2 Prioridades

O algoritmo de prioridade estabelece um vencedor baseado em um valor prioritário vinculado ao dispositivo. Para definir o vencedor é necessário que os dispositivos pertencentes ao grupo possuam atribuições de prioridades diferentes uns dos outros ou, caso houverem valores iguais, utilizar um outro algoritmo como critério de desempate.

Na estratégia de prioridades, um perfil local com informações sobre cada processo é previamente cadastrado. A informação deste perfil é usada para resolver o conflito. A maior prioridade vence. De forma similar a estratégia de votação, a estratégia de prioridades também opera em dois $rounds$, com coordenador detectando conflito e solicitando aos demais processos o envio da prioridade distribuída. A diferença, em relação à estrategia anterior, é que a prioridade é um valor fixo no perfil, pré-cadastrado. Em contraste, o voto é gerado dinamicamente. A consequência disso é que o resultado da técnica por prioridades é muito mais previsível (maior

prioridade sempre vence) do que o resultado obtido com a estratégia de votação.

Algoritmo 2 Algoritmo de Prioridade

1: {Round 1}
Entrada: Conjunto de elementos do grupo $K = \{\sigma_1, \ldots, \sigma_n\}$ não vazio, n número de elementos do conjunto K
2: **para todo** $\sigma_i \in K$ tal que $0 < i \leq n$ **faça**
3: Envia prioridade para σ_i
4: **fim para**

5: {Round 2}
Entrada: Conjunto de elementos do grupo $K = \{\sigma_1, \ldots, \sigma_n\}$ não vazio, n número de elementos do conjunto K
Saída: Método com maior frequência de prioridade nos elementos de K
6: $M = \{\ \emptyset\}$
7: **para todo** $\sigma_i \in K$ tal que $0 < i \leq n$ **faça**
8: Adiciona método prioritário de σ_i a M
9: **fim para**
10: Atribui a $vencedor$ o método que apareceu um maior número de vezes em M
11: **retorna** $vencedor$

Figura 3.4 – Algoritmo 2. Resolução de conflitos por prioridade

3.2.3 Leilão

A estratégia de leilão apresentada em (CAPRA; EMMERICH; MASCOLO, 2002) é mais sofisticada que as estratégias anteriores. Tal como na estratégia de prioridades, um perfil é pré-estabelecido para cada participante. Desta vez, os participantes precisam entrar em acordo sobre a escolha de um conjunto de regras (também pré-estabelecidas) para decidir sobre a ação que deverá ser tomada frente a um conflito, certificando-se de que fiquem satisfeitos o maior número de participantes possível.

A ação é decidida com base no perfil local de cada participante, com base no conjunto de regras disponíveis, e com base em um valor computado a cada rodada (isto é, a cada tomada de decisão), que descreve decisões tomadas anteriormente. Ou seja, o algoritmo de leilão considera tanto o contexto local, avaliando o perfil de cada participante, como o global, levando em conta resultados antecedentes.

Algoritmo 3 Algoritmo de Leilão

{Round 1}
Entrada: Conjunto de elementos do grupo $K = \{\sigma_1, \ldots, \sigma_n\}$ não vazio, n número de elementos do conjunto K
para todo $\sigma_i \in K$ tal que $0 < i \leq n$ **faça**
    Envia propriedades para σ_i
fim para

{Round 2}
Entrada: Conjunto de propriedades dos elementos do grupo $K = \{\sigma_1, \ldots, \sigma_n\}$ não vazio, n número de elementos do conjunto K
Saída: Se $M = \{\emptyset\}$ retorna $null$, senão retorna método do elemento de M vencedor do leilão

$M \leftarrow$ métodos de $\sigma_1 \cap \cdots \cap$ métodos de σ_n
se $M = \{\emptyset\}$ **então**
    retorna $null$
senão
    $maiorSoma \leftarrow 0$
    para todo $\gamma \in M$ **faça**
        para todo $\sigma_i \in K$ tal que $0 < i \leq n$ **faça**
            Calcula o valor da aposta do método γ em σ_i
        fim para
        Armazena em $somaAtual$ o valor da soma das apostas do método γ em todos os elementos de K
        se $somaAtual > maiorSoma$ **então**
            $vencedor \leftarrow \sigma_i$
            $maiorSoma \leftarrow somaAtual$
        fim se
    fim para
fim se
retorna método de $vencedor$

Figura 3.5 – Algoritmo 3. Resolução de conflitos por leilão

3.3 Análise das técnicas apresentadas

De forma geral, algoritmos de resolução de conflitos podem ser classificados segundo um conjunto de critérios a saber: (i) tipo de informação usada: uso de informação de contexto (local obtida por sensores, por exemplo, ou uso de perfil previamente armazenada) ou global, uso de informação histórica (obtida através de outras ações para resolver conflitos ou status anterior do sistema) ou previsão, (ii) tipo de objetivo: individual (de cada participante) ou coletivo (objetivo comum), e (iii) número de vencedores: único vencedor ou múltiplos vencedores.

Com o conhecimento prévio dessas características, foram selecionadas as três técnicas de resolução de conflitos descritas na seção anterior.

O algoritmo de votação é o mais simples entre os três apresentados. Ele não exige manutenção de contexto, pois os valores dos votos de cada membro do grupo são gerados de forma independentemente, no momento de execução. O algoritmo escolhe apenas um vencedor, isto é, o dispositivo que obtiver o maior voto ao final da execução.

Um problema da estratégia de votação é a potencialidade de geração votos idênticos, impossibilitando tomada de decisão consistente. Um gerador randômico gera votos em cada nodo. Se o gerador randômico considera um domínio pequeno de escolhas, pode acontecer de processos distintos gerarem votos idênticos (de alto valor). Neste caso, dois ou mais processos podem vencer uma mesma eleição. Para resolver este problema, precisa-se ou usar um gerador de votos centralizado ou incluir um novo $round$ para detectar a geração de votos idênticos.

No método de prioridades, cada um dos nodos conectados ao grupo possui um perfil específico. Este perfil é requisitado sempre que um conflito for identificado. Após a identificação da prioridade de cada perfil, o dispositivo com maior prioridade é escolhido.

Entretanto, da mesma forma que o algoritmo de votação, na estratégia de prioridade, se houverem dois ou mais nodos com o mesmo perfil o resultado pode tornar-se inconsistente. Esta questão pode ser resolvida de duas maneiras: obrigar cada processo a registrar previamente uma prioridade distinta ou resolver o conflito utilizando outro método de resolução de conflito (como por exemplo utilizar o próprio algoritmo de votação ou escolher o membro mais recente do grupo, especificamente para esses nodos).

Uma diferença importante entre a estratégia de leilão e as estratégias de votação e de prioridades é que as regras na estratégia de leilão são usadas para descrever restrições de redes móveis (limitação do uso de bateria, limitação de informação trocada no canal, etc.). As estratégias anteriores são bem mais simples e não consideram essas restrições.

Os algoritmos apresentados também diferem em seu objetivo final. Enquanto as técnicas de votação e prioridade visam objetivos individuais, a de leilão busca um acordo entre os diversos nodos do sistema baseado em informações locais (de cada um dos dispositivos) e globais (informações de resultados anteriores).

4 IMPLEMENTAÇÃO

Este capítulo descreve detalhes de implementação deste trabalho. Na seção 4.1 faz uma breve introdução sistema implementado. A seção 4.2 detalha as ferramentas e versões utilizadas para o desenvolvimento deste trabalho. Os itens 4.3 e 4.4 descrevem os arquivos de configurações utilizados para o desenvolvimento do $software$. Finalmente, a seção 4.5 descreve detalhadamente o funcionamento de um dispositivo dentro do grupo, demonstrando como os temas apresentados anteriormente interagem com o programa.

4.1 Descrição do sistema implementado

Para implementar as técnicas de resolução de conflito em um sistema ubíquo utilizando a comunicação de grupos foi utilizado como base o cenário demonstrado em *Capra et al.* (2002). As técnicas descritas na seção 3.2 foram moldadas para este cenário.

No ambiente de experimentação é considerado que cada nodo (ou $peer$) representa um dispositivo móvel. Estes nodos estão conectados através da rede e juntos formam um sistema de comunicação ubíqua. Cada dispositivo possui seu perfil especifico, cada um é responsável pela própria resolução de conflitos e todos estão interligados através do sistema de comunicação de grupos (ver figura 3.1 na seção 3.1).

Estes nodos comunicam-se com o objetivo de escolher um entre vários métodos de envio de mensagem. Estes métodos são apenas fictícios,representando um conflito de ideias e interesses entre os nodos do grupo. As ideias precisam ser adaptadas para atingir um objetivo comum: a escolha do método.

Cada um dos algoritmos apresentados necessita de diferentes propriedades para determinar o resultado da aplicação. Por isso, a cada nodo é atribuída uma lista de propriedades: largura de banda, disponibilidade, quantidade de bateria, privacidade e prioridade. Os atributos serão utilizados para definir um método de envio de mensagem vencedor.

Qualquer um dos nodos pertencentes ao grupo esta suscetível a falhas. As falhas são ocasionadas por problemas de conexão, término de bateria, $crash$ do sistema, entre outros. Para esta aplicação é simulada uma falha. A partir do momento da falha, um nodo não volta a ser operacional. Os nodos que não falharam durante a execução do programa são chamados de nodos ativos.

Assim, o resultado final produzido pelo programa é o método escolhido pelo algoritmo

selecionado considerando as propriedades dos nodos ativos do grupo.

4.2 Ferramentas utilizadas

O programa foi desenvolvido na linguagem Java, com a Java Development Kit 6. O suporte para esta linguagem está presente em variados dispositivos móveis e embutidos, facilitando a implementação em um ambiente real no futuro. Uma das ferramentas para comunicação de grupo implementadas para esta linguagem que se destaca é o JGroups (BAN, 2012).

O JGroups é uma biblioteca $opensource$ implementada em Java que oferece uma gama de serviços e protocolos que podem ser configurados conforme necessário pela aplicação. Além de implementar a comunicação em grupo, esta ferramenta identifica a junção e remoção de nodos no sistema através de uma comunicação $multicast$ confiável.

Trabalhos anteriores exploram a variedade de opções que esta $toolkit$ para comunicação de grupo oferece. Em *Lopez et al* (2009), por exemplo, os autores modificam a estrutura do JGroups visando maior eficiência em redes móveis, possibilitando novas abordagem para o desenvolvimento deste trabalho no futuro. A familiaridade do autor com as ferramentas, utilizada no desenvolvimento de um programa para gerenciamento autonômico de servidores (PASIN; BAZZAN; PERINI, 2011), também contribuiu para estas escolhas.

Para elaboração de arquivos de configuração e definição de propriedades dos nodos conectados, foi escolhida a linguagem de marcação XML ($eXtensible\ Markup\ Language$). Este formato proporciona uma maior facilidade para leitura visual e edição. Nele, cada propriedade é definida por uma marcação (tag) específica e pode ser composta por um ou mais atributos. A formatação dos arquivos e as marcações utilizadas neste trabalho são explicados e exemplificados nas seções 4.3 e 4.4.

4.3 Configurações de um grupo

Os nodos pertencentes a um grupo compartilham um arquivo de configuração único. Este arquivo possui informações do algoritmo que será utilizado na resolução de conflitos, o nome do canal em que os dispositivos estão conectados e os métodos disponíveis para troca de mensagem, bem como os seus requisitos mínimos. Cada uma dessas propriedades recebe nomes e valores específicos que serão utilizados posteriormente para comunicação e resolução de conflitos dentre todos os nodos do grupo.

O arquivo XML que define características do funcionamento do sistema deve ser iniciado com a marcação $config$, e a partir dela são nomeadas as demais propriedades. Na sequencia, é definido o algoritmo para resolução de conflitos, através da marcação $algorithm$. Esta propriedade pode assumir um dos seguintes valores: votação, prioridade ou leilão (representando cada um dos algoritmos implementados).

Em seguida, é determinado o nome do canal em que os dispositivos serão conectados através da marcação $channel$. A propriedade da marcação $channel$ pode ser especificada por qualquer palavra ou número suportado pelo formato XML. O nome do canal identifica a que grupo pertencem os dispositivos. Esta é uma opção útil quando existem dois ou mais arquivos de configuração. Supondo um cenário real, dispositivos presentes na cozinha compartilham um arquivo de configuração onde $channel$ é definido como $kitchen$, eles possuem características e necessidades diferentes dos dispositivos presentes na sala de estar, onde o canal é identificado por $livingroom$. Dessa forma, cada canal é responsável pelos seus próprios conflitos e pela resolução dos mesmos de acordo com as configurações pré-estabelecidas.

Então são estabelecidos os métodos comuns para troca de mensagens entre os dispositivos. Estas informações são fictícias, utilizadas apenas para simular o cálculo de um método que considera dados locais comuns quando lidamos com dispositivos móveis. Cada método possui pesos e medidas que servirão para estabelecer o vencedor em um conflito no algoritmo de Leilão (CAPRA; EMMERICH; MASCOLO, 2002).

A abordagem de $Capra\ et\ al.$ (2002) considera que cada $peer$ possui perfis diferentes, onde cada um deles tem disponível para utilização pelo menos um dos quatro métodos de troca de mensagem. Neste trabalho, os $peers$ presentes no grupo trocam mensagens através de todos os métodos que estiverem declarados no arquivo de configuração.

Dentro do arquivo de configuração, a marcação que referencia um método de troca de mensagens é chamada de $task$. Esta marcação possui um atributo e quatro propriedades distintas. O único atributo pertencente a esta marcação corresponde ao nome do método, referenciado por $name$, sua função é diferenciar visualmente um método dos demais.

As propriedades dos métodos assumem valores inteiros entre 0 e 10, dentre estas propriedades estão a largura de banda, a disponibilidade, a quantidade de bateira e a privacidade. São identificadas, respectivamente, pelas marcações $bandwidth$, $availability$, $battery$ e $privacy$. Caso um método não especificar uma das propriedades, o valor zero é atribuído a ele. Isto significa que esta propriedade não é necessária para a tarefa em questão e não terá influência nos

cálculos do algoritmo para escolha do método vencedor.

A ordem das propriedades e marcações presentes no arquivo de configuração não são substanciais, pois não alteram a execução e funcionamento do sistema. Servindo somente para melhor organização e leitura do arquivo. Um exemplo de arquivo de configuração utilizado na execução do programa desenvolvido é representado na figura 4.1.

A linha 4 da figura 4.1 define o algoritmo de prioridade para ser executado. O nome do canal em que os dispositivos estarão conectados foi chamado de $livingroom$ (linha 7). A linha 9 define o primeiro método, identificado pelo nome $sendEncryptedLine$. Em seguida, as linhas 10, 11, 12, 13, definem, respectivamente, os valores da largura de banda, da disponibilidade, da quantidade de bateria e da privacidade, necessários para este método.

Por fim, o arquivo xml ainda define os métodos $sendChar$, na linha 15, e $sendLine$, na linha 21. As definições das propriedades necessárias para execução do método seguem o padrão anterior, assim, das linhas 16 até 19 são definidas as propriedades do método $sendChar$ e das linhas 22 até 24 são definidos os métodos para $sendLine$. Nas definições das propriedades do método $sendLine$ há uma peculiaridade, como pode ser observado na figura 4.1, ele não possui a propriedade $privacy$, sendo assim, ela assumirá o valor zero durante a execução do programa.

4.4 Configurações dos nodos

Dispositivos móveis possuem caraterísticas peculiares, são geralmente dependentes de bateria e influenciados por fatores ambientais. Para simular algumas dessas propriedades dentro do grupo de comunicação, um arquivo XML foi elaborado para descrever as características de um nodo e, também, demonstrar suas preferências em relação aos métodos disponíveis. Cada um dos nodos presentes no grupo possui seu próprio perfil, estabelecido após a leitura do arquivo que é especificado através do comando de execução do programa.

O arquivo XML que define as características de um nodo inicia com a marcação $peer$. Em sequencia, são definidos os métodos de troca de mensagens prioritários para este $peer$, os métodos são descritos dentro da marcação $priority$. Estas opções devem ser condizentes com as descritas no arquivo de configuração (seção 4.3). Cada um dos métodos é demarcado por $task$.

A ordem da descrição dos métodos no arquivo influencia diretamente nas preferências do nodo. O primeiro método descrito logo em seguida a marcação $priority$ possui maior prioridade em relação aos demais. Caso o resultado das ações para solucionar um conflito não forem

```
1  <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2  <config>
3        <algorithm>
4              prioridade
5        </algorithm>
6        <channel>
7              livingroom
8        </channel>
9        <task name="sendEncryptedLine">
10             <bandwidth>4</bandwidth>
11             <availability>9</availability>
12             <battery>4</battery>
13             <privacy>2</privacy>
14       </task>
15       <task name="sendChar">
16             <bandwidth>5</bandwidth>
17             <availability>5</availability>
18             <battery>8</battery>
19             <privacy>0</privacy>
20       </task>
21       <task name="sendLine">
22             <bandwidth>7</bandwidth>
23             <availability>4</availability>
24             <battery>5</battery>
25       </task>
26 </config>
```

Figura 4.1 – Exemplo do arquivo XML de configuração compartilhado

conclusivos, a descrição de outros métodos prioritários pode ajudar a resolver o problema.

Por exemplo, três $peers$ estão em um grupo, cada qual com características distintas, e eles procuram escolher o melhor método para estabelecer comunicação. O primeiro $peer$ tem preferência apenas pelo método $sendChar$, o segundo somente pelo método $sendLine$. Já o terceiro $peer$ possui maior preferência pelo método $sendEncryptedLine$, porém, possui a opção $sendLine$ como secundária. Se para a resolução de conflitos o arquivo de configuração estiver utilizando o algoritmo de prioridade (seção 3.2.2) e levando em consideração somente o método de maior prioridade em cada um dos $peers$, o resultado será inconsistente ou levará a uma baixa qualidade de serviço. Visando satisfazer da melhor forma possível todos os nodos conectados ao grupo, as opções secundárias são levadas em consideração. Neste cenário, é fácil perceber que a escolha que melhor atende ao grupo é a do método $sendLine$.

Ainda, neste mesmo arquivo XML são definidas as propriedades do $peer$, assim como

em $Capra\ et\ al$ (2002). Estas configurações são utilizadas para escolha do método no algoritmo de Leilão (seção 3.2.3). Dessa forma, elas representam características do nodo em um determinado momento. Para este trabalho, foram consideradas as seguintes propriedades: largura de banda, disponibilidade, quantidade de bateira e privacidade. As marcações para estas propriedades devem ser equivalentes as utilizadas para os métodos de troca de mensagens, presentes no arquivo de configuração descrito na seção 4.3.

A figura 4.2 exemplifica um arquivo de informações de um nodo. As linhas 4 e 5 ilustram o exemplo mencionado anteriormente, onde o método $sendLine$ (linha 4) possui maior preferência em relação ao $sendChar$ (linha 5). Além disso, nas linhas 8, 9, 10 e 11 indicam, respectivamente, a definição dos valores de largura de banda, disponibilidade, quantidade de bateria e privacidade de um nodo durante a execução do programa.

```
1  <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2   <peer>
3        <priority>
4              <task>sendLine</task>
5              <value>11</value>
6        </priority>
7        <properties>
8              <bandwidth>7</bandwidth>
9              <availability>7</availability>
10             <battery>7</battery>
11             <privacy>1</privacy>
12       </properties>
13 </peer>
```

Figura 4.2 – Exemplo de arquivo XML contendo informações de um peer

4.5 Descrição detalhada do funcionamento do sistema

Cada $peer$ é iniciado através da linha de comando e dentre as opções de parâmetros está o arquivo XML que carregará as suas propriedades. Se este arquivo for encontrado, as informações contidas nele serão lidas e gravadas posteriormente na instância da classe $PeerInfo$. Porém, antes disso, o programa carrega os dados contidos no arquivo de configuração, especificado pelo nome $config.xml$.

O arquivo de configuração é carregado em primeira instância pois é necessário verificar se os dados do $peer$ são consistentes em relação aos dados do arquivo de configuração. Os

métodos prioritários declarados no arquivo do $peer$ devem obrigatoriamente possuir referência dentro do arquivo $config.xml$. Caso contrário o programa finaliza a sua execução. Os dados do arquivo $config.xml$ são armazenados em um objeto da classe $GlobalInfo$.

Depois de carregar as informações contidas no arquivo de configuração, um canal de comunicação é criado com o nome do canal especificado (ver linha 7 da figura 4.1). Para que os $peers$ estejam conectados ao mesmo grupo, o nome do canal deve ser idêntico em todos os nodos. Após o inicio do sistema de comunicação de grupos, as informações individuais do $peer$ são carregas.

O objeto da classe $PeerInfo$ é o responsável por armazenar os valores de prioridade, bateria, disponibilidade, largura de banda e privacidade contidos no arquivo, como visto na seção 3.2.2. Quando o arquivo especificado na linha de comando não for encontrado, um novo arquivo é criado com a opção repassada pelo parâmetro. As propriedades são geradas automaticamente, conforme os valores mínimo e máximo de cada uma delas. Além disso, a instância armazena o endereço do $peer$, esta informação é gerada pelo JGroups quando o sistema de comunicação de grupos inicia. O endereço é um valor único e é o atributo de identificação do $peer$, diferenciando-o dos demais.

Após a inicialização dos grupos e o carregamento das informações do $peer$, é criado um objeto responsável pelo controle de envio e recebimento das mensagens. Este objeto é uma derivação do JGroups e apresenta, além do controle de mensagens, funções fundamentais para a identificação de nodos falhos. Um método é automaticamente executado a cada período de tempo (por padrão, três mil milissegundos), verificando se há novos nodos conectados ou nodos suspeitos de falha. Dessa forma, ele é capaz de fornecer ao dispositivo uma visão dos nodos que estão ativos e pertencem ao grupo.

Quando uma mensagem é recebida, a função $receive$ implementada pela ferramenta é executada de forma automática. Neste método, são repassados o objeto da mensagem. Por sua vez, este objeto apresenta, entre outras informações, o endereço de identificação do $peer$ que enviou a mensagem, o tipo da mensagem e o conteúdo da mesma, representada obrigatoriamente por um objeto serializável. Quando uma mensagem é recebida, é responsabilidade do próprio $peer$ decidir o que deve fazer com ela.

Outro método presente na estrutura principal da classe é o responsável por enviar mensagens para um ou mais nodos. Para ele ser executado, duas informações devem ser repassadas: o endereço de destino da mensagem (ou um valor nulo, caso a mensagem deve ser enviada a

todos os nodos do grupo) e o seu conteúdo. A declaração da função responsável pelo envio da mensagem é verificada na linha 11 da figura 4.3. Os métodos responsáveis pelo controle de nodos ativos e envio de mensagens estão representados, respectivamente, nas linhas 3 e 7 nesta mesma figura.

```
1  public class ReceiverChannel extends ReceiverAdapter {
2
3     public void viewAccepted(View new_view){
4        [...]
5     }
6
7     public void receive(Message msg) {
8        [...]
9     }
10
11    public void sendMessage(Address d, MessageContent msg) {
12       [...]
13    }
14
15    [...]
16 }
```

Figura 4.3 – Estrutura principal da classe responsável pela troca de mensagens entre os nodos e pela identificação do grupo

Na linha 3 o método $viewAccepted$ recebe apenas um parâmetro da classe $View$, ambos implementados pela ferramenta de comunicação de grupos JGroups. Este parâmetro contêm todos os nodos ativos do grupo no momento em que foi executado. Na linha 7, o método $receive$ recebe um objeto da classe $Message$ (também derivado do JGroups) que contêm, entre outras propriedades, um objeto serializável com o conteúdo da mensagem e o endereço identificando o emissor.

O último método da figura 4.3, representado na linha 11, efetua o envio de mensagens. O método nomeado $sendMessage$ foi implementado devido às necessidades do programa, os dois parâmetros da função são pertencentes a classe $Address$ e $MessageContent$.

A classe $Address$ é responsável por identificar um nodo pertencente ao grupo, sendo assim, o primeiro atributo estabelece o destinatário: se o método receber um valor nulo, a mensagem é enviada para todos o grupo; caso contrário a mensagem é enviada apenas para o nodo identificado pelo parâmetro.

O segundo parâmetro do método $sendMessage$ é um objeto da classe $MessageContent$.

Esta classe é obrigatoriamente serializável possibilitando a transmissão de informações entre os nodos do grupo. O conteúdo deste objeto é definido pelas necessidades do método de resolução de conflito determinado no arquivo de configuração, descrito na seção 4.3.

Quando o algoritmo definido para resolução de conflito é o de votação, o conteúdo enviado é o voto do $peer$. No algoritmo de prioridade, é enviado somente o método prioritário, especificado localmente. Já o algoritmo de leilão envia um maior número de informações do dispositivo, são elas: a largura de banda, a disponibilidade, a privacidade e a quantidade de bateria. Estas informações são atribuídas ao nodo após a leitura do arquivo XML, descrito na seção 4.4.

Para que a resolução de conflitos seja executada, antes é necessário identificar o conflito e então repassar as informações necessárias para resolve-lo. No entanto, a identificação de conflitos não é uma tarefa trivial e foge do escopo deste trabalho. Em virtude disso, considera-se que o conflito exista desde o princípio e a resolução somente é iniciada a partir do momento que um número determinado de nodos estiverem conectados ao grupo.

Ao atingir o número limite de $peers$ conectados ao grupo, inicia-se a troca de mensagens. Cada um dos nodos envia uma mensagem $multicast$ para o grupo. O conteúdo desta mensagem varia de acordo com as necessidades do algoritmo definido, como descrito anteriormente.

O sistema de comunicação de grupos verifica, de forma autônoma, se todas as mensagens chegaram aos seus destinatários. Se houverem pacotes perdidos, o próprio JGroups é responsável por reenvia-los através de um método $unicast$. Por padrão, se a mensagem não chegar ao seu destino depois de três tentativas (uma $multicast$ e duas $unicast$), o nodo é considerado falho. Logo, na próxima chamada do método $viewAccepted$ este nodo será excluído do grupo. Este processo se repete até todos os nodos do grupo receberem todas as mensagens.

Quando todos os nodos ativos receberem as informações necessárias para o algoritmo executar, inicia-se a resolução do conflito. Se algum dos nodos falhar durante o processo de resolução do conflito ele é removido do grupo, a visão do grupo é atualizada e uma nova seção de troca de mensagens é iniciada. A execução é iniciada novamente somente se todos os nodos ativos pertencentes ao grupo receberem as mensagens (o escopo deste funcionamento foi ilustrado na figura 3.2 na seção 3.1).

Se nenhum dos nodos falhar durante o processo de adaptação o programa é concluído e todos os nodos do grupo ficam cientes da resolução. Atualmente, todos os nodos ativos

executam o algoritmo e todos devem encontrar o mesmo resultado. Este processo poderia ser executado somente em um $peer$ que, através da comunicação de grupos, notificaria os demais a solução encontrada. Porém, esta conduta adiciona um novo processo ao sistema onde, também, podem ocorrer falhas.

O diagrama na figura 4.4 ilustra o comportamento do sistema através de um fluxograma. Basicamente, durante a execução do algoritmo para resolução, se um nodo não recebeu a mensagem e esta ativo então ele é removido da lista, isto é, do grupo. Este nodo não irá participar da resolução do conflito.

O sistema desenvolvido neste trabalho considera apenas falhas do tipo $fail - stop$, ou seja, quando um nodo falha, ele pára de funcionar e não retorna ao grupo. No entanto, métodos disponibilizadas pelo JGroups, além de identificarem novos nodos do sistema, também oferecem opções para o tratamento de outros tipos de falhas, conforme as necessidades do desenvolvedor.

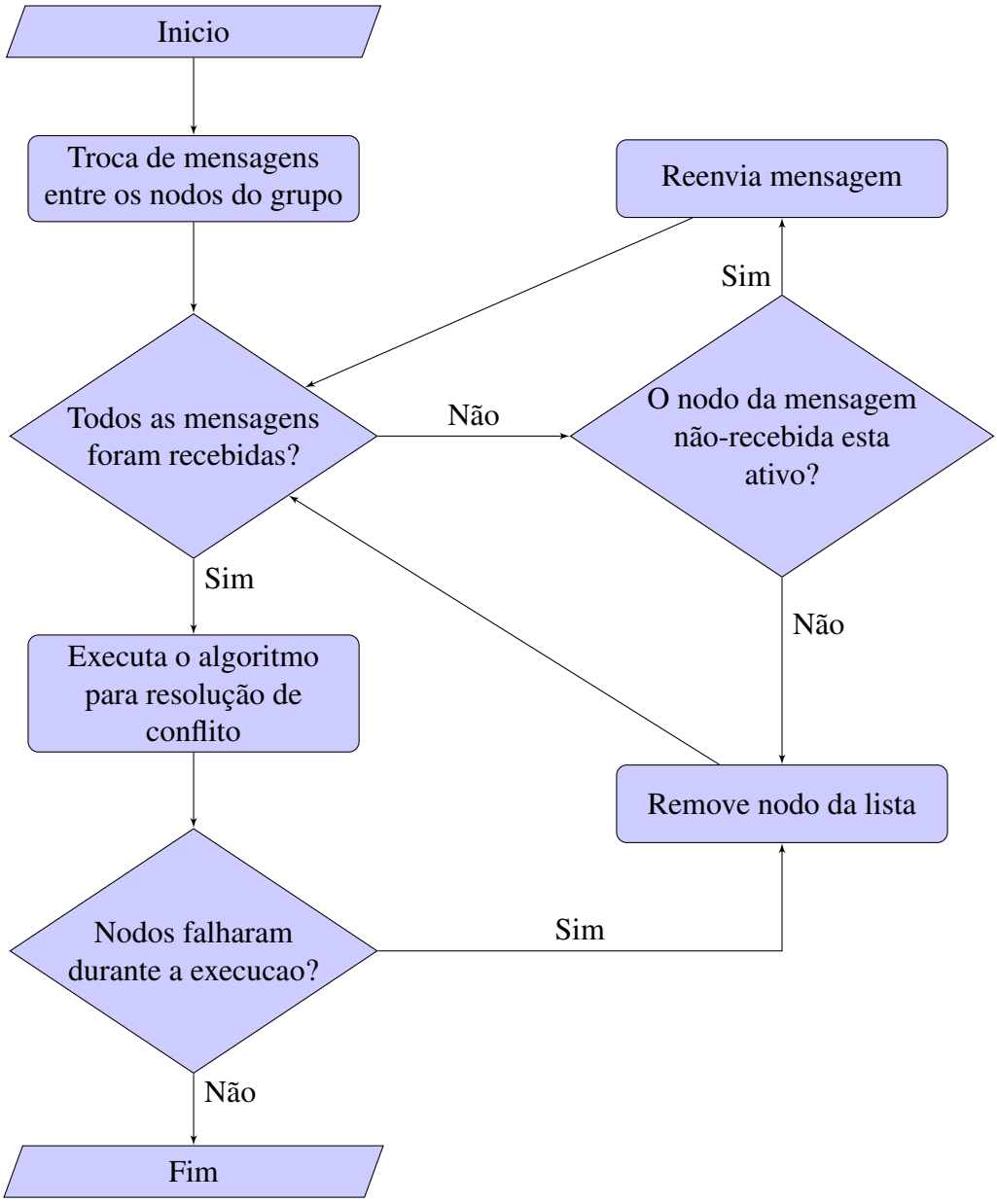

Figura 4.4 – Fluxograma que descreve o algoritmo para resolução de conflitos com suporte de comunicação de grupo

5 AVALIAÇÃO EXPERIMENTAL

Este capitulo apresenta informações sobre os resultados da execução do programa desenvolvido para este trabalho. A seção 5.1 descreve os cenários de testes. A seção 5.2 reproduz os resultados obtidos através da execução do programa nos cenários estabelecidos anteriormente. No final, a seção 5.3 discute problemas encontrados durante os testes e suas soluções.

5.1 Cenários de testes

Para avaliar os resultados do $software$ desenvolvido, dois cenários foram estabelecidos. O primeiro, visa analisar o comportamento da comunicação em grupo utilizando o $JGroups$ em um ambiente sem falhas. O segundo explora um ambiente com nodos falhos, avaliando o comportamento do sistema em relação ao número de falhas ocorridas.

Após um número de nodos n pré-estabelecidos conectarem-se ao grupo, inicia-se um cronômetro. Quando todos os nodos ativos finalizarem o algoritmo de resolução de conflito, a contagem é terminada e as informações (tempo cronometrado, método escolhido após a execução do algoritmo, número total de nodos antes da execução e número de nodos falhos) são salvas em um arquivo. Assim, o tempo durante o processo de troca de mensagens, mais a identificação das falhas e consequente reenvio das mensagens é contabilizado.

Para um número k, maior que um e menor que n, de nodos conectados, são efetuados dez testes. O maior e menor tempo final de execução são eliminados e uma média é feita entre os três restantes. Esta média serve como meio de análise para diferentes cenários apresentados. O processo é repetido para cada um dos algoritmos implementados em todos os cenários.

A identificação do conflito não é abordada neste trabalho, considera-se que há uma situação conflitante em princípio. Portanto, não há medida de tempo envolvida nesta ação.

5.1.1 Cenário 1: Execução dos algoritmos em um ambiente sem falhas

No primeiro cenário os nodos efetuam a resolução de conflito considerando um ambiente ausente de falhas. Não é necessário o reenvio de mensagens para a atualização do sistema. O cenário é avaliado executando um número de nodos conectados n.

Uma experimentação consiste na troca de mensagens necessárias e na resolução do conflito utilizando o algoritmo especificado no arquivo de configuração. Para cada n número de

nodos conectados são feitos cinco testes para cada um dos algoritmos implementados. Assim, os algoritmos são executados em um nova instancia dos nodos e o teste é repetido para verificar a consistência dos resultados.

5.1.2 Cenário 2: Execução dos algoritmos em um ambiente com falhas

Neste cenário considera-se que um número k de nodos, presentes em um grupo de n elementos que deseja resolver um conflito, falha. Dessa forma, é necessário o reenvio das informações de todos os $peers$ ativos do grupo e então, caso não forem detectadas novas falha, é efetuada a resolução do conflito. Essas informações devem ser reenviadas e os dados utilizados para solução do conflito são atualizados. Em um cenário real, as propriedades de um $peer$ podem sofrer alterações e, consequentemente, alterar o resultado final produzido pelo algoritmo.

Para testar este cenário foram executados n nodos simultaneamente, definido em trinta. A partir disso, a resolução do conflito inicia-se. Durante o período de troca de mensagens uma quantia predefinida k de nodos falha. Esta quantia é variável a fim de avaliar o desempenho do $software$ a medida que o número de nodos falhos aumentam.

Como resultado, o tempo final de uma execução do sistema considera o tempo necessário para troca de mensagens, o tempo para identificação da falha e, também, o tempo para reenvio das mensagens aos nodos ainda ativos no sistema.

5.2 Avaliação dos resultados

5.2.1 Avaliação do cenário 1

Para a avaliação dos algoritmos no cenário 1, um experimento foi executado com n nodos ativos. O valor de $n = \{5, 10, 15, 20, 25, 30\}$. Para cada algoritmo foram executados testes com o mesmo n número de nodos conectados e medido o tempo médio desde o inicio da troca de mensagens até o resultado final da resolução do conflito.

Os resultados obtidos para a execução do algoritmo de votação estão ilustrados no gráfico da figura 5.1. Os resultados demonstram que a cada cinco novos nodos conectados ao grupo são adicionados, aproximadamente, 0,07 nanosegundos ao tempo de finalização programa.

A figura 5.2 constrói o gráfico de resultados obtidos executando o algoritmo de prioridade. As informações são bastante próximas das encontradas com o algoritmo de votação, pois assemelham-se na sua implementação e na quantidade de informações transmitidas.

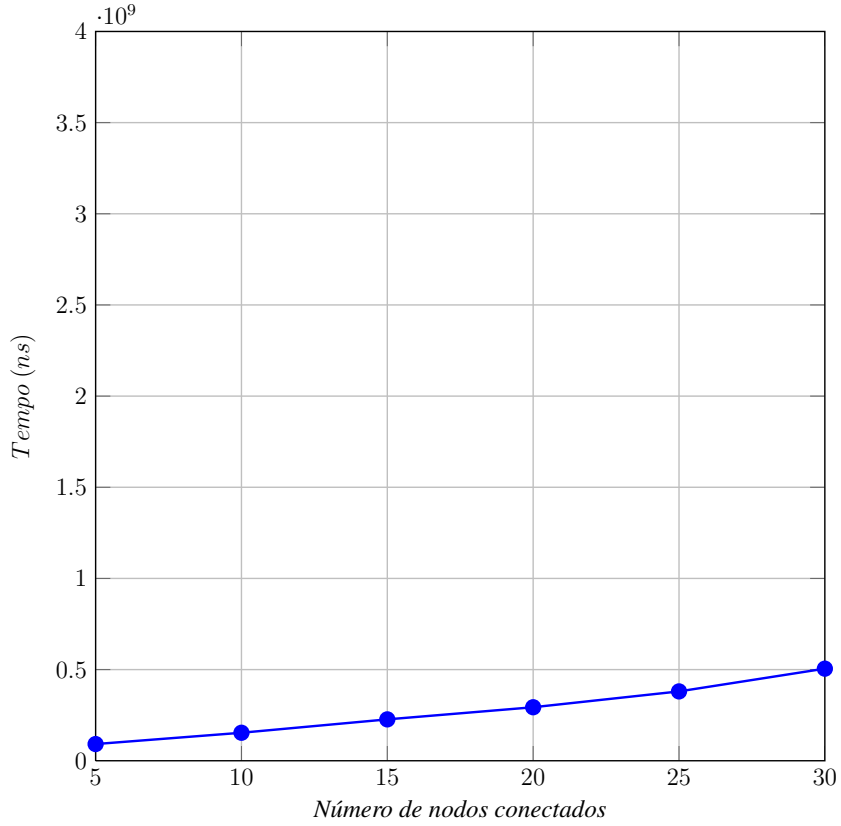

Figura 5.1 – Resultado da execução dos testes do algoritmo de votação no cenário 1

A figura 5.3 apresenta resultados obtidos utilizando o algoritmo de leilão para solucionar conflitos. Este algoritmo transmite um maior número de informações entre os nodos e efetua um maior número de cálculos e comparações que os demais visando a satisfação coletiva. Apesar disso, o $overhead$ causado pelas operações é mínimo. O tempo final aumenta em 0,08 nanosegundos para cada cinco novos nodos adicionados ao grupo.

5.2.2 Avaliação do cenário 2

No cenário 2 é avaliado o impacto da falha dos nodos no tempo final de resolução de conflito em um sistema com comunicação de grupo provida pelo JGroups. Em todos os testes foram iniciados com $n = 30$ nodos simultaneamente e ao decorrer do processo k nodos falham interruptamente. Os valores atribuídos a $k = \{5, 10, 15, 20, 25\}$.

Considerando este cenário de falhas, a figura 5.4 representa os resultados obtidos durante a execução do algoritmo de votação, a 5.5 o algoritmo de prioridade e a 5.6 o algoritmo de leilão. Estes gráficos são bastante semelhantes em seu comportamento.

Quando não possui falhas, a resolução de conflitos é finalizada em aproximadamente 0,5 nanosegundos. A partir do momento em que cinco nodos falham há uma grande discrepância

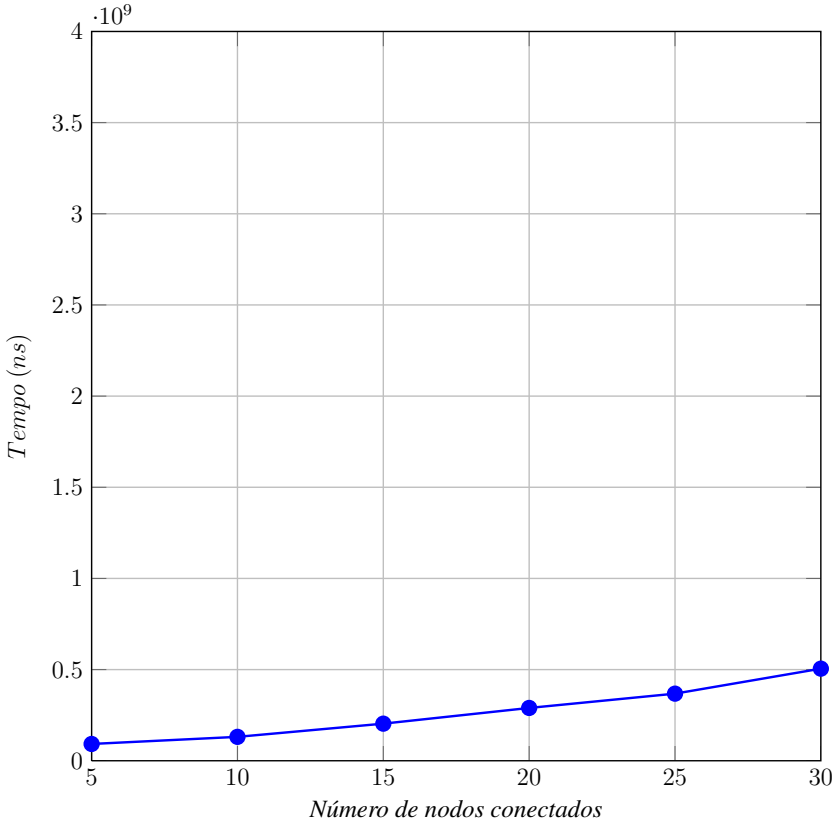

Figura 5.2 – Resultado da execução dos testes do algoritmo de prioridade no cenário 1

entre os tempos de execução com falhas e sem falhas.

Porém, aumentando o número de nodos falhos para dez, quinze, vinte ou vinte e cinco, o tempo final não aumenta drasticamente. Estes resultados são consequência das configurações padrões do mecanismo de identificação de falhas fornecido pelo JGroups.

Para identificar se um nodo está ativo ou não, o JGroups coordena mensagens entre os nodos do grupo. Por padrão, o sistema de comunicação de grupo verifica se os nodos estão ativos a cada mil e quinhentos(1500) milissegundos. Também por padrão, são feitas três tentativas de contactar o nodo suspeito, caso ele não responder é considerado falho.

Nos testes executados, as falhas ocorrem em um intervalo muito pequeno. Em virtude da configuração da comunicação de grupos, o tempo resultante para resolução do conflito é muito parecido independente do número de falhas. Caso o tempo de identificação de um nodo falho for reduzido, provavelmente o tempo médio de execução também seria reduzido.

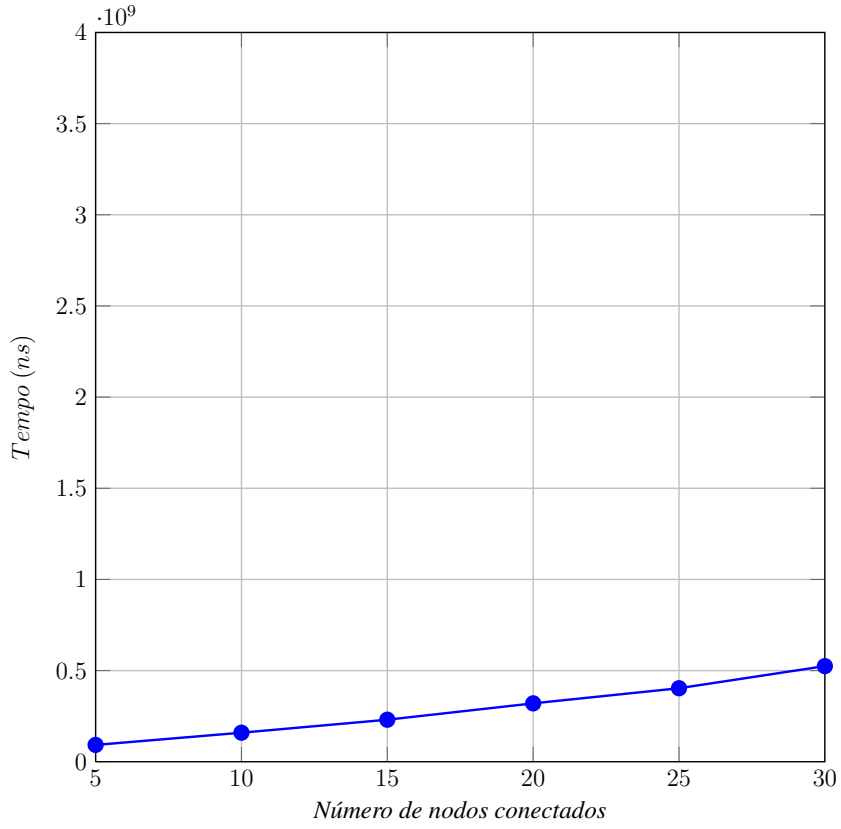

Figura 5.3 – Resultado da execução dos testes do algoritmo de leilão no cenário 1

5.3 Problemas encontrados

Na execução dos testes, observou-se que alguns nodos falhavam ao receber/enviar mensagens de forma inesperada. Esta situação acontece em ambientes reais, porém alguns deles não estavam programados para isto no cenário de testes. Também, verificou-se que esta situação só ocorria a partir de um certo número de nodos executados simultaneamente. Como consequência, duas tarefas foram planejadas baseados nesta ocorrência: revisão da sincronização das mensagens e análise de variáveis do sistema.

Durante a revisão da sincronização do $software$ notou-se que uma variável responsável pelo armazenamento das informações vindas de outros $peers$ poderia estar ocasionando o problema. Para resolve-lo, bastou transformar o objeto desta variável em um objeto com acesso sincronizado. No entanto, ao efetuar novos testes o problema persistia.

Através da análise das variáveis do sistema, foi possível identificar alguns fatores que poderiam gerar falhas inesperadas. Configurações de tamanho do $buffer$ para mensagens TCP, por exemplo, estavam setadas com valores baixos. Isto não se torna um problema quando há poucos nodos em cada computador. Entretanto, vários nodos são executados simultanea-

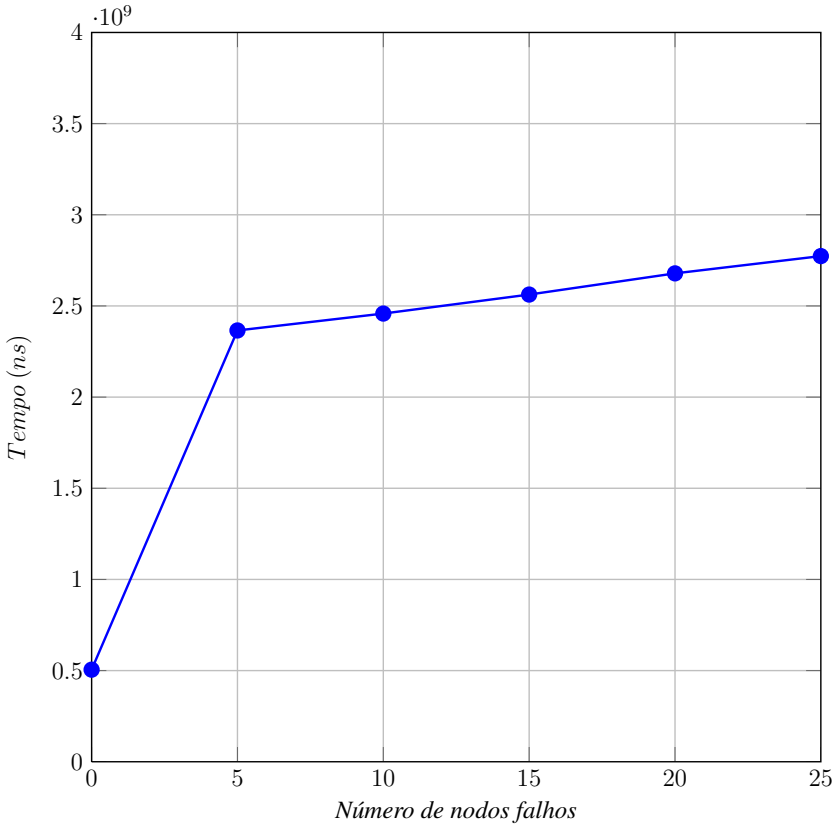

Figura 5.4 – Resultado da execução dos testes do algoritmo de votação dentro do cenário 2

mente em uma máquina no cenário de testes, sobrecarregando o sistema operacional.

Ao finalizar a configuração do tamanho do $buffer$, novos testes foram efetuados. Apesar disso, o problema persistia quando executados muitos nodos simultâneos. Esta situação não ocorre quando há um menor número de nodos e, mesmo quando executado um grande número de nodos, o resultado final de execução não é influenciado (visto que o tempo para a atualização dos dados pelo sistema é, principalmente, consequência da configuração do JGroups). Porém, em consequência disto, o número de nodos para os testes foi limitado em trinta.

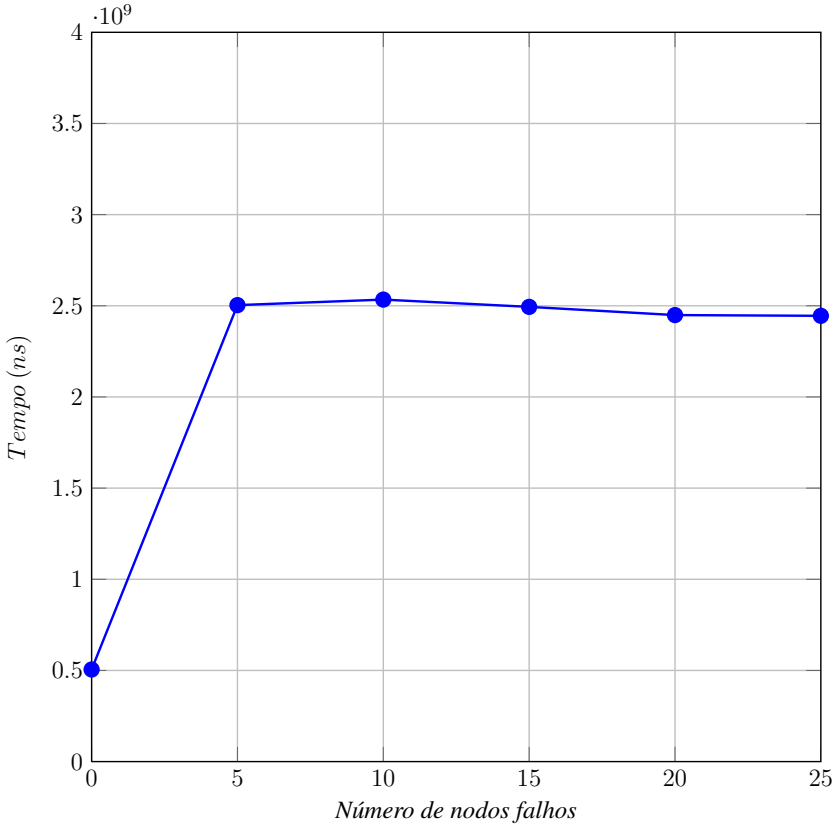

Figura 5.5 – Resultado da execução dos testes do algoritmo de prioridade dentro do cenário 2

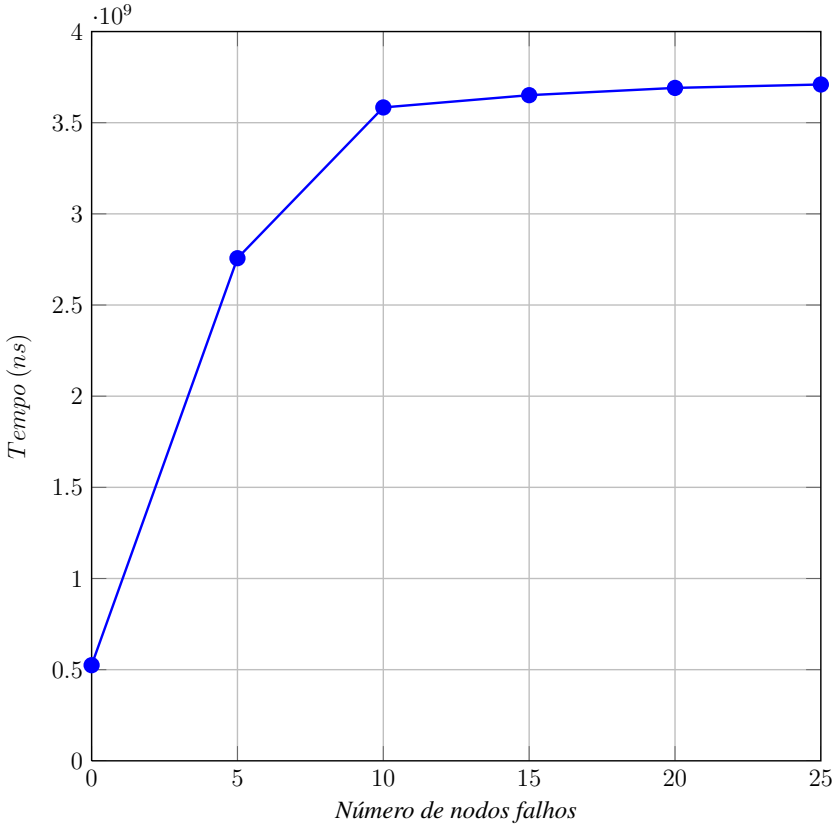

Figura 5.6 – Resultado da execução dos testes do algoritmo de leilão dentro do cenário 2

6 CONCLUSÃO E TRABALHOS FUTUROS

A implementação de tolerância a falhas traz novos desafios à viabilidade do sistema, tais como a postergação indefinida e a visibilidade do tempo de tomada de decisão pelo usuário. Quando há grande alternância nos nodos funcionais de um grupo em um curto período de tempo, o usuário torna-se ciente do processo e por, consequência, a transparência do sistema ubíquo não é atingida. Um nodo que alterna entre funcional e possível de falha no grupo e a entrada e saída de novos membros durante um período indefinido de tempo, exigem novas soluções para o funcionamento do sistema.

Uma alternativa para sanar os problemas gerados em consequência da entrada e saída de novos membros é impor um limite de tempo de execução do sistema. Isto é, a partir da identificação da necessidade de adaptação, inicia-se um cronômetro e quando ele ultrapassar um limite pré-estabelecido, o algoritmo para solução de conflito é iniciado pela última vez (até que o usuário solicite uma nova requisição). No entanto, apesar de manter a transparência, a execução do sistema com membros indesejáveis no grupo pode levar a um resultado insatisfatório para o usuário.

A alteração das configurações das propriedades do sistema de comunicação de grupos do JGroups provaram-se fundamentais para um melhor desempenho do sistema. Contudo, os valores devem ser alterados dependendo da disponibilidade e qualidade da rede. O ambiente em que foram executados os testes difere de um ambiente real, principalmente em relação a rede de comunicação. No ambiente de testes os computadores estão distribuídos em uma rede local cabeada fisicamente próximos. Já em um ambiente real existe heterogeneidade dos meios de comunicação, impondo novos limites de tempo e velocidade. Assim, a configuração adequada das variáveis existentes no sistema de comunicação de grupos para um ambiente real ainda é uma questão a ser avaliada.

É notória a sobrecarga do sistema quando utilizado uma ferramenta de comunicação de grupo para identificar nodos possíveis de falha. Não é uma tarefa trivial encontrar a configuração ideal para o sistema de comunicação de grupos mesmo no ambiente de testes utilizado, quanto mais em um ambiente caracterizado pela variedade de meios de comunicação. As ideias presentes no trabalho de $Lopez\ et\ al.$ (2009) surgem como alternativa futura para uma comunicação mais eficiente. Assim, as avaliações efetuadas neste trabalho servirão como ponto de partida para comparação entre os diferentes modelos.

Outras sugestões para implementações futuras incluem a ampliação do número de algoritmos apresentados, em destaque os de inteligência artificial. Modificação do cenário de testes, incluindo um modelo mais próximo do real e possibilitando a implementação de algoritmos mais eficientes. A seleção de um dos algoritmos dependendo do tipo de conflito ocorrido durante a execução, como apresentado por *Moura Braga Silva* (2010).

A implementação do tipo de sistema desenvolvido neste trabalho em um ambiente real ainda é um grande desafio nos dias de hoje. Mesmo com a popularidade dos dispositivos móveis, com suporte a variadas linguagens de programação, há um grande caminho a ser percorrido nas demais áreas. Na computação embarcada, por exemplo, os computadores são comumente caracterizados por executar apenas as funções necessárias do dispositivo ao qual foi embutido. A falta de padronização de hardware e software impede que a computação ubíqua expanda-se naturalmente.

REFERÊNCIAS

ARAUJO, R. B. de. Computação Ubíqua: princípios, tecnologias e desafios. **XXI Simpósio Brasileiro de Redes de Computadores**, [S.l.], 2003.

BAN, B. **JGroups - A Toolkit for Reliable Multicast Communication**. acessado em dezembro de 2012.

BIRMAN., K. P. Building Secure and Reliable Network Applications. **Worldwide Computing and Its Applications**, [S.l.], 1997.

BIRMAN, K. P.; JOSEPH, T. A. Exploiting virtual synchrony in distributed systems. **ACM**, [S.l.], p.123–138, 1987.

BRUMITT, B. et al. EasyLiving: technologies for intelligent environments. **Second International Symposium, HUC 2000 Bristol, UK**, [S.l.], p.12–29, 2000.

CAPRA, L.; EMMERICH, W.; MASCOLO, C. A Micro-economic approach to conflict resolution in mobile computing. **Proceedings of the 10th International Symposium on the Foundations of Software Engineering**, [S.l.], p.31–40, 2002.

CHAE, H. et al. Conflict resolution model based on weight in situation aware collaboration system. **Future Trends of Distributed Computing Systems**, [S.l.], p.99–106, Março 2007.

COOKA, D. J.; AUGUSTOB, J. C.; JAKKULAA, V. R. Ambient Intelligence: technologies, applications, and opportunities. **Pervasive and Mobile Computing**, [S.l.], v.5, p.277–298, 2009.

EFSTRATIOU, C. et al. Utilising the event calculus for policy driven adaptation in mobile systems. **Proceedings of POLICY 2002, IEEE Computer Society**, [S.l.], p.13–24, Junho 2002.

HASAN, M. K. et al. Conflict resolution and preference learning in ubiquitous environment. **International Conference on Intelligent Computing (ICIC)**, [S.l.], p.355–366, 2006.

HUEBSCHER, M. C.; MCCANN, J. A.; HOSKINS, A. Context as autonomic intelligence in a ubiquitous computing environment. **International Journal of Internet Protocol Technology**, [S.l.], v.Volume 2, Number 1/2007, p.30–39, 2006.

KRUMM, J. **Ubiquitous Computing Fundamentals**. [S.l.]: Chapman & Hall/CRC, 2009.

LOPEZ, P. G. et al. Topology-Aware Group Communication Middleware for MANETs. **ICST COMSWARE. Dublin, Ireland.**, [S.l.], 2009.

MOURA BRAGA SILVA, T. R. de. **Tratamento de Conflitos Coletivos em Sistemas Ubíquos Cientes de Contexto**. 2010. Tese (Doutorado em Ciência da Computação) — Universidade Federal de Minas Gerais.

PARK, I.; LEE, D.; HYUN, S. J. A dynamic context-conflict management scheme for group-aware ubiquitous computing environments. **Computer Software and Applications Conference**, [S.l.], p.359–364, 2005.

PASIN, M.; BAZZAN, A. L. C.; PERINI, F. S. Towards a self-managed distributed infrastructure to an advanced traveler information system. **Anais do CBSoft/AutoSoft**, [S.l.], 2011.

SHIN, C.; WOO, W. Conflict resolution method utilizing context history for context-aware applications. **Proceedings of the 1st International Workshop on Exploiting Context Histories in Smart Environments (ECHISE'05), Munich, Germany.**, [S.l.], 2005.

SILVA, T. R. M. B.; RUIZ, R. B.; LOUREIRO, A. A. F. Towards a conflict resolution approach for colective ubiquitous context-aware systems. **Proc. WAS 2010**, [S.l.], p.435–442, 2010.

TANENBAUM, A. S.; STEEN, M. van. **Distributed system**: principles and paradigms. [S.l.]: Prentice Hall, 2007.

TIGLI, J.-Y. et al. Context-aware authorisation in highly dynamic environments. **International Journal of Computer Science Issues (IJCSI)**, [S.l.], v.4, p.24–35, Setembro 2009.

WEISER, M. Some computer science issues in ubiquitous computing. **Communications of ACM**, [S.l.], v.36, p.75–84, Julho 1993.