KATRINY ZAMPRONI

DESENVOLVIMENTO DE CÓDIGO DE SIMULAÇÃO BASEADO EM COMPONENTES REUTILIZÁVEIS PARA REDES DE SENSORES SEM FIO

Monografia apresentada à disciplina de Trabalho de Graduação em Banco de Dados como requisito à conclusão de curso de Bacharelado em Ciência da Computação, Setor de Ciências Exatas, Universidade Federal do Paraná.

Orientadora: Profª. Drª. Carmem Satie Hara

CURITIBA

2017

Resumo

Redes de sensores sem fio (RSSFs) estão presentes em vários locais do cotidiano e esse tipo de dispositivo tem sido utilizado para diversos fins, como saúde, segurança e monitoramento de ambientes. As RSSFs auxiliam em aplicações que lidam com grandes quantidades de informações, as quais em geral devem ser obtidas de forma autônoma.

Dentre as dificuldades estruturais de uma rede de sensores estão a baixa capacidade de armazenamento dos dispositivos, além da limitação dos recursos energéticos. Existem diversas propostas de modelos de armazenamento em redes de sensores sem fio que buscam contornar esses problemas. Essas soluções possuem diversos elementos em comum, que poderiam ser explorados por meio de um *framework* de desenvolvimento genérico, facilitando a reutilização de código e também a implementação desses modelos nas ferramentas de simulação existentes.

Tendo isso em vista, foi proposto o modelo *Reusable Component-based Model for WSN Storage Simulation* (RCBM) (Carrero et al., 2017), que define alguns conjuntos de funcionalidades e elementos que são comuns a vários tipos de modelos de armazenamento em RSSFs. O objetivo deste trabalho é estudar o modelo RCBM por meio da implementação do sistema *Distributed Clustering Scheme based on Spatial Correlation in WSNs* (DCSSC) (Le et al., 2008).

Dentre os principais resultados, podem ser citadas a obtenção de um percentual de reuso de código de 71,6% e a validação do sistema implementado por meio da comparação aos resultados obtidos pelo trabalho original. Como contribuição ao modelo RCBM, foi implementado um novo componente de entrada que poder ser agregado à biblioteca do *framework*.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Códigos

Capítulo

1

Introdução

As redes de sensores sem fio (RSSFs) têm sido amplamente utilizadas no cotidiano para os mais diversos fins. Utilizando essa tecnologia, é possível monitorar a localização de animais selvagens, detectar invasões em ambientes controlados, receber informações constantes de condições climáticas em determinado local, além de aplicações em áreas como saúde e militar (Akyildiz et al., 2002).

No contexto de Internet das Coisas (*Internet of Things* - IoT), que consiste em conexões de dispositivos capazes de armazenar dados e comunicar-se por meio de sensoriamento inteligente, as RSSFs auxiliam no desenvolvimento de aplicações que lidam com grande quantidade de dados e que necessitam de um meio autônomo de obtenção de informações que devem ser repassadas utilizando uma conexão de rede (Patel et al., 2011).

Redes de sensores sem fio são conjuntos de nós sensores, constituídos de mecanismos para sensoriamento, processamento de dados e comunicação, além de uma fonte de energia. Esses nós são dispositivos pequenos, de baixo custo e capacidade computacional reduzida, capazes de coletar informações e detectar eventos transmitindo dados para a própria rede ou para uma estação central. Em geral, eles são densamente dispostos ao redor de um objeto de interesse a ser monitorado (Loureiro et al., 2003). Os sensores comunicam-se por meio de sinais de rádio. Um salto é uma medida da distância entre um par de sensores. Dois dispositivos são ditos vizinhos quando estão no raio de comunicação do outro, e a distância entre eles é de um salto.

As principais dificuldades estruturais de uma rede de sensores são a baixa capacidade de armazenamento de informações nos sensores, que lidam com um grande volume de dados sensoriados que estão em constante variação, além da limitação dos recursos energéticos e do próprio ambiente, que podem dificultar a comunicação. Contudo, existem propostas para contornar os problemas citados, de forma a manter a RSSF autônoma e garantindo escalabilidade. Essas soluções, que em geral possuem vários elementos em comum, procuram utilizar algoritmos de agrupamento da rede e outras estruturas de dados associadas. Os agrupamentos são conjuntos de sensores, que são reunidos de acordo com algum critério que possuam em comum. Em geral, são eleitos representantes para cada agrupamento. Para obter as leituras dos sensores, ou enviar mensagens para cada dispositivo individualmente, é possível reduzir a comunicação à troca de mensagens com os representantes, diminuindo os custos com comunicação e consequentemente auxiliando na economia da bateria dos dispositivos e aumentando o desempenho do monitoramento.

Para validação desses algoritmos, além do estudo de novas abordagens relacionadas a redes de sensores sem fio, foram desenvolvidas diversas ferramentas de simulação e modelagem, que facilitam o estudo desse tipo de rede e a validação dos modelos sem que seja necessária uma implantação muitas vezes inviável no mundo real. Ao contrário de redes tradicionais, muitos aspectos de RSSFs ainda não foram padronizados, como tipos de arquitetura, protocolos e hardware, uma vez que existe uma ampla gama de aplicações com diversas finalidades, que requerem funções específicas (Minakov et al., 2016). Sendo assim, a falta de um *framework* de desenvolvimento genérico dificulta a implementação das simulações e praticamente impossibilita a reutilização de código.

Modelos de software com componentes possibilitam que o desenvolvedor reutilize códigos previamente definidos, facilitando o trabalho de codificação. Tendo isso em vista, foi proposto o modelo *Reusable Component-based Model for WSN Storage Simulation* (RCBM) (Carrero et al., 2017), que define alguns conjuntos de funcionalidades e elementos que são comuns a vários tipos de modelos de armazenamento em sensores sem fio. O objetivo deste trabalho é estudar o modelo RCBM e implementar um estudo de caso, a fim de entendê-lo e compará-lo a abordagens já existentes.

O restante desse trabalho está estruturado da seguinte forma: o Capítulo 2 descreve os trabalhos relacionados ao desenvolvimento baseado em componentes para RSSFs e apresenta a proposta do RCBM. O Capítulo 3 descreve o modelo de armazenamento que será implementado nesse estudo de caso, e o Capítulo 4 descreve os detalhes da implementação desse modelo, além do resultado dos experimentos realizados. O Capítulo 5 apresenta a conclusão do estudo, além da validação da implementação feita e descrição de trabalhos futuros.

Capítulo

2

Trabalhos relacionados

Este capítulo apresenta os conceitos gerais relacionados aos modelos de armazenamento em redes de sensores sem fio, além de abordar o desenvolvimento de software baseado em componentes e citar trabalhos relacionados a essa proposta. Além disso, é apresentado um modelo baseado em componentes reutilizáveis para simulação de RSSFs.

2.1 Modelos de Armazenamento em RSSFs

Os modelos de armazenamento em redes de sensores sem fio são classificados como de armazenamento externo (centralizado) e em rede (distribuído), como mostra a Figura 2.1

Nos modelos de armazenamento externo, os dados sensoriados são enviados para uma entidade externa à rede, chamada de estação-base, que possui maior capacidade de processamento e armazenamento, e nenhum dado fica armazenado no próprio sensor.

Os modelos de armazenamento em rede podem ser subdivididos entre de armazenamento local, em que os dados ficam nos próprios sensores, ou em repositórios, situação na qual os sensores formam repositórios de dados distribuídos na própria rede. Os repositórios são sensores específicos responsáveis por armazenar dados de um conjunto de sensores, e por isso são denominados modelos hierárquicos. Esse tipo de modelo tem sido muito utilizado nas aplicações para redes de sensores, pois promove maior escalabilidade

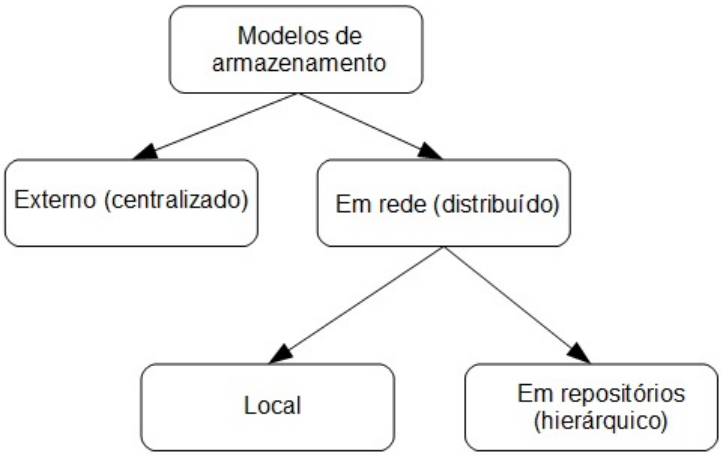

Figura 2.1: Classificação dos modelos de armazenamento

e aumenta o desempenho do processamento de consultas devido ao fato de dispensar a necessidade de consultar pontualmente cada nó sensor da rede por meio de inundações (ou *broadcast*), podendo obter informações apenas do repositório de dados. Por isso, ele foi escolhido como foco principal desse estudo, embora o modelo de componentes possa ser aplicado a qualquer tipo de modelo citado (Carrero et al., 2017).

Os modelos de armazenamento hierárquicos são compostos por agrupamentos de sensores, ou *clusters*, que são formados de acordo com critérios específicos determinados pelo algoritmo de formação. Dentro desses agrupamentos, pode existir a eleição de um líder ou *cluster-head* (CH), que em geral atua como unidade local de processamento de consultas relacionadas ao seu *cluster*. Aos sensores que fazem parte desses agrupamentos são atribuídos os papéis de membros, ou *cluster member* (CM). A partir do estudo de vários modelos de armazenamento de dados é possível associar algumas entidades e funcionalidades que são comuns a vários sistemas, de acordo com a Tabela 2.1.

Por meio dessa identificação de padrões comuns, é possível determinar componentes, que são definidos como trechos de código que se comunicam com outros componentes e oferecem um serviço determinado, utilizando uma interface bem definida. Além disso, componentes devem ser substitutíveis, independentes de contexto, encapsulados e devem constituir uma unidade independente para implantação e versionamento (Niekamp, 2005).

Na Seção 2.2, é discutida a utilização de componentes de software, com exemplos de trabalhos que utilizam o conceito de modelagem baseada em componentes e suas

Conceito	Descrição	Funcionalidades
Estação-base (EB)	Dispositivo conectado à rede que possui grande poder computacional	Armazenar dados, monitoramento e processamento de consultas
Sensor	Dispositivo com baixa capacidade de processamento, armazenamento e comunicação	Coleta e envio de dados para a EB, processamento de consultas
Cluster-head (CH)	Sensor líder de um grupo	Seleção do líder do grupo, rotação do líder, processamento de consultas
Cluster member (CM)	Sensor membro de um grupo	Coleta e envio de dados para o CH, associação ao CH
Repositório	Local de armazenamento de dados na rede	Seleção de repositório, processamento de consultas

Tabela 2.1: Conceitos e Funcionalidades dos Modelos de Armazenamento. Fonte: (Carrero, 2016)

vantagens. Na Seção 2.3, é abordado o uso de componentes aplicados ao escopo de redes de sensores sem fio.

2.2 Componentes de software

A utilização de componentes de software que buscam facilitar a implementação de sistemas que possuam diversos elementos em comum é algo recorrente em Ciência da Computação. Dentro da área de Sistemas Distribuídos, existem diversas propostas para a utilização do modelo baseado em componentes, os quais geralmente estão associados à existência de uma biblioteca que possui *templates*. Os *templates* implementam parcialmente determinados códigos associados a elementos ou funcionalidades (Carrero et al., 2017).

Um exemplo de *framework* baseado em componentes é o Exhibit (Huynh et al., 2007), uma ferramenta para publicação de dados estruturados desenvolvida com a proposta de facilitar o trabalho de criação de páginas *Web*, permitindo a criação de páginas interativas tendo apenas conhecimento básico em HTML. Uma página *Web* que utiliza o Exhibit tem duas interfaces. Uma delas é a que fica visível a todos os usuários que a acessam, e se parece com uma página *Web* comum. A segunda é a interface do autor. Nela, é preciso fazer a criação dos dados a serem utilizados pela página, e também a apresentação do conteúdo.

Os dados são definidos em arquivos e são compostos por objetos que contêm pares chave-valor. Para a apresentação, é necessário criar uma página HTML simples, que inicializa as funcionalidades do Exhibit. Os códigos são pré-definidos e existem vários trechos de código já implementados que servem de base para o desenvolvimento da aplicação. É possível inserir trechos de *Cascading Style Sheets* (CSS), linguagem que descreve o estilo de um documento HTML, para sobrescrever o *default* da aplicação. O modelo de dados do Exhibit é composto por itens, que são classificados em tipos e possuem certas propriedades. A partir dessa definição, é possível estabelecer um paralelo dessa ferramenta para facilitar o desenvolvimento de aplicações *Web* que apresentam conteúdo de uma ou mais fontes de forma única (também chamadas de *Web mashups*) com o modelo de componentes, que utiliza a ideia de padronização de objetos com funcionalidades comuns.

2.3 Componentes para RSSFs

As redes de sensores sem fio têm atraído muito interesse na área de tecnologia recentemente, demonstrado pela grande quantidade de trabalhos que propõem modelos aplicados a sensores. Entretanto, uma quantidade limitada de implementações está disponível para a comunidade. Além disso, a maior parte desses algoritmos está diretamente relacionada a um sistema específico de RSSF, ou então está integrada a outra aplicação, dificultando o reuso por futuros desenvolvedores (Amaxilatis et al., 2011).

Alguns estudos anteriores já identificaram a similaridade dos modelos de armazenamento para redes de sensores sem fio, e propuseram uma abordagem baseada em componentes. O trabalho feito por Amaxilatis *et al.* (Amaxilatis et al., 2011) faz um levantamento de alguns algoritmos existentes, analisando especificamente o aspecto de formação de *clusters* utilizada em cada um deles. Por meio desse estudo, os autores definiram três componentes: *Cluster-head decision* (responsável por definir o algoritmo de eleição de líder de acordo com os critérios utilizados pelo modelo), *Join decision* (relacionado à metodologia com a qual os nodos definem a qual *cluster* irão se associar) e *Iterator* (organiza quais nodos já se associaram a algum *cluster* e coleta informações sobre agrupamentos já formados). O trabalho apresenta uma abordagem limitada, uma vez que forma compo-

nentes baseando-se apenas em algoritmos que utilizem formação de agrupamentos, além de determinarem um fluxo de execução pré-definido, dificultando a adaptação de modelos que não sigam o padrão previsto (Carrero et al., 2017).

Outro trabalho relacionado foi proposto por Cheong *et al.* (Cheong et al., 2003), que define o *TinyGALS*, um modelo de programação para sistemas embarcados em rede. O programa é um sistema composto por módulos, os quais por sua vez são constituídos por componentes. Um componente tem váriáveis internas, externas e métodos que lidam com essas variáveis, semelhante a um objeto de uma linguagem de programação orientada a objetos. Além disso, um componente possui uma interface e sua implementação. Por ser implementado utilizado o sistema *TinyOS*, o modelo mencionado acaba sendo limitado a redes de sensores em pequena escala.

O OASiS (Kushwaha et al., 2007) é outra proposta para RSSFs, que define um *framework* para redes de sensores sem fio orientadas a serviço. Nesse tipo de rede, cada atividade dos sensores é definida como um serviço, de forma similar aos serviços *Web*. Serviços são modulares, autônomos e têm interfaces pré-definidas, que permitem que eles sejam invocados na rede. Estes serviços são constituídos de componentes que fazem a comunicação e gerenciam os sensores e suas operações. Os serviços compostos resultam num grafo, que determina o fluxo de execução da aplicação. Os nodos fazem amostras periódicas do ambiente, criando objetos que traduzem logicamente um evento físico (por exemplo, para detectar a presença de fogo um sensor deve ter uma leitura superior a 100°C). Se em algum momento o protocolo de manutenção identificar mudança no status do objeto (de falso para verdadeiro ou vice-versa), é feita a transição para um novo modo, e um novo grafo de serviços é definido de acordo com a sequência de ações a serem tomadas. O OASiS também possui a limitação de tamanho de rede, devido à utilização do sistema *TinyOS*. Além disso, ele enfrenta desafios em aspectos relacionados à dinamicidade da rede.

A proposta do RCBM (*Reusable component-based model for WSN storage simulation*) (Carrero et al., 2017), que será abordada na Seção 2.4, provê uma alternativa aos trabalhos mencionados, pois é aplicável a redes de sensores em larga escala, por meio de simulações

com a ferramenta *Network Simulator 2* (NS-2). Além disso, ele possui um fluxo de execução flexível, que permite que vários tipos de modelo sejam explorados por meio do *framework*.

2.4 A reusable component-based model for WSN storage simulation

Com base no problema de pesquisa das redes de sensores sem fio e a dificuldade na implementação das simulações dos modelos que utilizam como base esse tipo de arquitetura, surgiu a proposta do *Reusable component-based model for WSN storage simulation* (RCBM) (Carrero et al., 2017). Ele consiste em um metamodelo de armazenamento para RSSFs que utiliza entidades, propriedades e funções para descrever o que seriam as instâncias desse metamodelo, ou seja, as propostas de modelo de armazenamento. O RCBM torna possível a reusabilidade de código, facilitando o desenvolvimento de novos modelos.

No modelo RCBM, cada entidade é associada a um conjunto de componentes que implementam as funcionalidades comuns aos modelos. Dentre os exemplos de entidades estão os próprios sensores, *clusters*, *cluster-heads* e a estação-base.

Existem três tipos distintos de componentes no RCBM:

- Componentes de biblioteca: implementam funcionalidades úteis para os modelos, como funções de agregação e temporizadores;
- Componentes de aplicação: compõem o modelo de armazenamento e possuem um conjunto de funções que podem ser chamadas por outros componentes ou pelo coordenador, de acordo com o funcionamento do sistema proposto;
- Componente do coordenador: controla o fluxo de execução do sistema.

Cada componente é definido a partir de um *template* e de sua implementação. O *template* define os protótipos das funções, descrevendo seus parâmetros de entrada e os valores de saída.

O modelo de funcionamento do processo de desenvolvimento baseado em componentes para RSSFs está representado na Figura 2.2. A Figura 2.2 (a) apresenta a classificação dos modelos de armazenamento, que são subdivididos em locais, externos e de repositórios, conforme já descrito na Figura 2.1. Cada uma dessas classes do metamodelo está associada a duas instâncias de modelos (L1-2, E1-2, R1-2). A Figura 2.2 (b) ilustra os componentes de aplicação existentes, que podem ser associados e/ou reestruturados de forma a compor as instâncias da Figura 2.2 (a), resultando na sua implementação, como mostra a Figura 2.2 (c).

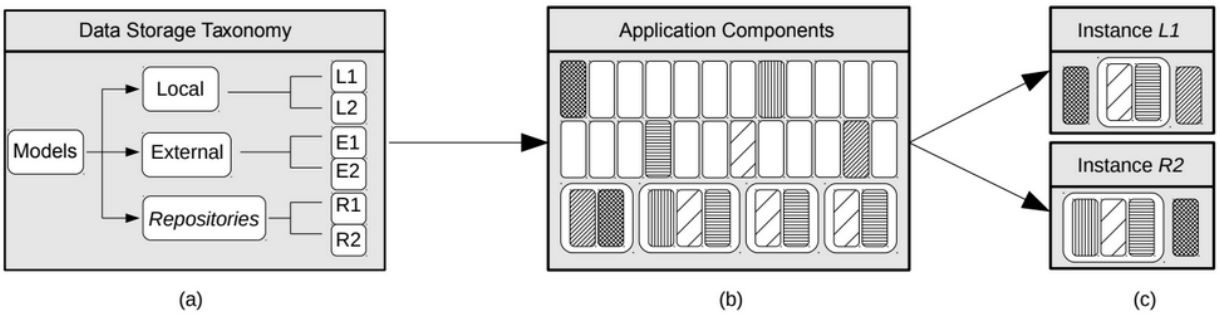

Figura 2.2: Desenvolvimento baseado em componentes para RSSFs. Fonte: (Carrero et al., 2017)

Na sequência é descrita a plataforma do RCBM, demonstrando os conceitos associados ao metamodelo de desenvolvimento para RSSFs.

2.4.1 Plataforma RCBM

A arquitetura do RCBM é composta por três camadas: especificação, implementação e comunicação, conforme especificado na Figura 2.3.

A camada de especificação é composta pelos componentes de biblioteca, *template* do coordenador e *templates* dos componentes de aplicação. Esses elementos são utilizados da forma como foram especificados, sem a necessidade de alterações realizadas pelo usuário.

A camada de implementação é a camada que de fato será construída pelo desenvolvedor, e contém o código que implementa os componentes de aplicação e o componente do coordenador, que estão definidos da camada de especificação.

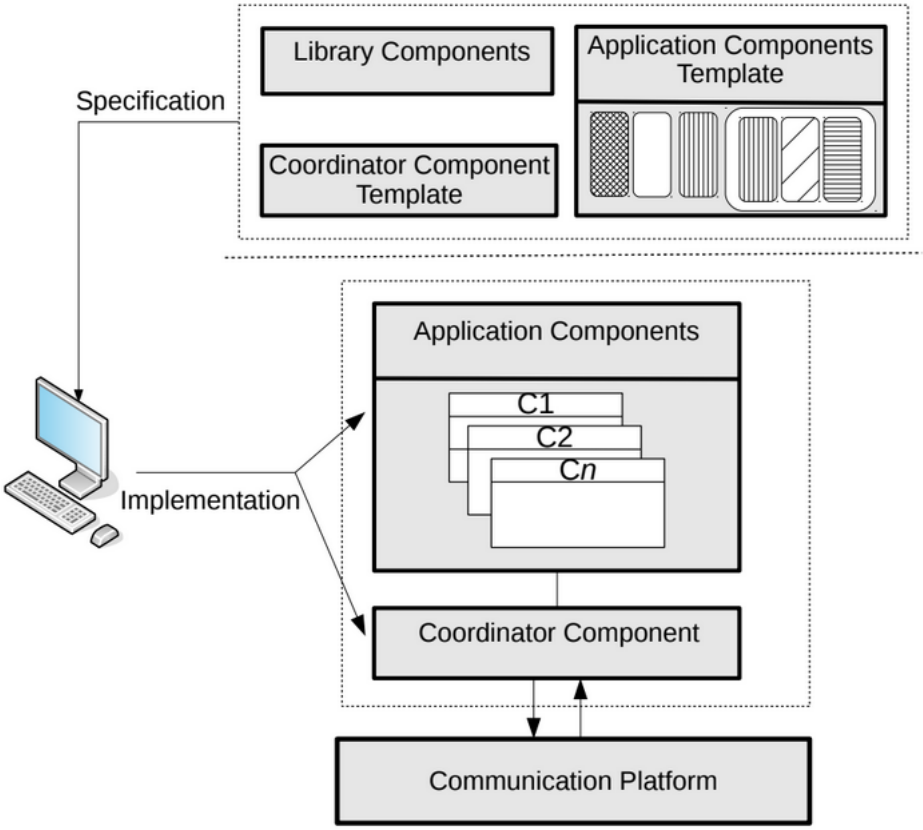

Figura 2.3: Arquitetura do RCBM. Fonte: (Carrero et al., 2017)

A camada de comunicação é a responsável por interligar as camadas de especificação e de implementação, fornecendo a infraestrutura de comunicação entre os componentes. No caso das redes de sensores sem fio, a camada de comunicação é a própria ferramenta de simulação.

2.5 Sumário

Neste capítulo foi apresentado um esquema de classificação para modelos de armazenamento em redes de sensores sem fio, além da identificação de funcionalidades e elementos em comum nesses modelos. Foi apresentado o conceito de desenvolvimento baseado em componentes de software e posteriormente sua aplicação no escopo de RSSFs. Além disso, foi detalhado o funcionamento do metamodelo de armazenamento RCBM.

No Capítulo 3, será discutido um modelo de armazenamento em redes de sensores sem fio baseado em correlação espacial, o qual será implementado para o estudo de caso do RCBM, apresentado neste capítulo.

Capítulo

3

Distributed Clustering Scheme based on Spatial Correlation in WSNs

A fim de estudar e entender o funcionamento do modelo RCBM, foi desenvolvido um estudo de caso a partir do modelo de armazenamento em redes de sensores sem fio chamado *Distributed Clustering Scheme based on Spatial Correlation in WSNs* (DCSSC) (Le et al., 2008). Nesse trabalho, os autores exploram a similaridade de dados em RSSFs de larga escala, aumentando a eficiência e prolongando a vida útil dos sensores da rede. O algoritmo proposto constrói e mantém os *clusters* de forma distribuída e dinâmica, e seu funcionamento será detalhado a seguir.

3.1 Funcionamento

Nesse modelo, cada sensor comunica-se apenas com os sensores que estão à distância de um salto e todos têm os dados relativos ao nível de energia de seus vizinhos. Esta informação é enviada por meio de mensagens do tipo HELLO trocadas periodicamente. O modelo de rede consiste em uma estação-base e N sensores. São atribuídos estados a todos os dispositivos, que determinam seu papel na rede. No início, os sensores estão no estado INI e ao final da fase de construção de agrupamentos eles estarão em um dos seguintes estados: CH (*cluster-head*), GW (*gateway*), EXT (*cluster-extend*) ou MEM (*member*). Os sensors

com os estados CH, GW e EXT ficam nesse estado até o fim da fase de construção e são chamados de nodos de *backbone*. Existem também estados temporários, como o INI, GWR (*gateway-ready*) e CHC (*cluster-head candidate*).

As mensagens de formação de agrupamento incluem sempre a média de valores das leituras aferidas no tempo que precede o envio da mensagem. Baseado no tipo da mensagem, o receptor irá mudar seu estado, criar uma nova mensagem e propagar para seu vizinhos. A nova mensagem também deve conter o identificador (ID) do emissor original da mensagem recebida.

A média de valores de leitura de um sensor s com n tipos de leitura é definida como o conjunto $s_1, s_2, ..., s_n$. A medida de dissimilaridade $d(s, v)$ entre dois sensores s e v é calculada da seguinte forma:

$$d(s, v) = \omega_1|s_1 - v_1| + ... + \omega_n|s_n - v_n| \tag{3.1}$$

Na Equação 3.1, o valor positivo da constante ω_i ($\sum_{i=1}^{n} \omega_i = 1$) indica o quanto o *i-ésimo* dado das leituras do sensor afeta o grau de dissimilaridade. Dois sensores s e v estão altamente correlacionados se a medida de dissimilaridade $d(s, v)$ definida pela equação é menor do que um limiar τ definido previamente pelo usuário como parâmetro, ou seja, $d(s, v) \leq \tau$.

No início da simulação, é iniciada a fase de construção de agrupamentos, determinada pelos passos que seguem:

- A estação-base inicia a formação de agrupamentos fazendo *broadcast* de uma mensagem do tipo CFRM (*Cluster Formation Message*). Cada sensor que recebe essa mensagem e está no estado INI muda seu estado para GWR, criando dois *timers*, t_{req} e t_{wait}, escolhidos aleatoriamente.

- Quando o tempo definido por t_{req} expira, sensores no estado GWR fazem o envio de uma mensagem CHREQ (*Cluster Head Request*) para seus vizinhos a fim de encontrar um *cluster-head*.

- Quando um sensor INI recebe a mensagem CHREQ, calcula a dissimilaridade (Equação 3.1) com o sensor GWR que enviou a mensagem. Os sensores que forem altamente correlacionados mudam seu estado para CHC (candidatos a *cluster-head*).

- Os nodos CHC criam um novo *timer* t_{adv}, baseado no seu nível relativo de energia *re(s)*, calculado de acordo com a equação a seguir:

$$re(s) = \frac{e(s) + \sum_{i \in nbr(s)} e(i)}{e(s) \times (|nbr(s)| + 1)} \times \alpha \tag{3.2}$$

  Na Equação 3.2, *e(s)* representa a energia disponível no sensor, *nbr(s)* a média do nível de energia dos seus vizinhos e α um número constante de unidades de tempo. O *timer* t_{adv} é escolhido aleatoriamente entre o intervalo de $[min_{adv}, re(s)]$, em que min_{adv} é uma constante pré-definida. Um sensor CHC se declara como CH fazendo *broadcast* da mensagem CHADV (*Cluster Head Adverstisement*) quando seu timer t_{adv} expirar. Dessa forma, sensores com maiores níveis de energia têm maior chance de serem selecionados como líder.

- Quando algum sensor em um estado temporário (INI, GWR, CHC) recebe uma mensagem CHADV, ele calcula a dissimilaridade com o sensor que enviou a mensagem (Equação 3.1). Se eles forem altamente correlacionados, o receptor se torna membro (estado MEM) do *cluster* formado por aquele CH. Senão, ele vai para o estado GWR, seguindo os mesmos passos definidos anteriormente para um sensor no estado GWR. Todo sensor altamente correlacionado compara o ID da fonte original da mensagem recebida com o seu próprio ID. Se forem iguais, ou seja, o CHADV recebido foi uma resposta ao seu próprio CHREQ, o sensor no estado GWR altera seu estado para GW.

- Quando o *timer* t_{wait} expira, o sensor no estado GWR também muda seu estado para CHC se não houver nenhuma mensagem CHADV proveniente dos seus vizinhos.

- Os sensores nos estados GW e MEM propagam a mensagem de formação criando e fazendo *broadcast* de mensagens do tipo CEXT (*Cluster Extend*), que incluem a média das leituras do CH que originou a mensagem, e não do sensor atual.

- Após receber uma mensagem CEXT, todo sensor em estado temporário (INI, GWR, CHC) calcula a sua dissimilaridade (Equação 3.1) com relação ao CH. Se forem altamente correlacionados, ele integra o cluster formado e se torna um membro (estado MEM), fazendo com que o transmissor da mensagem CEXT vá para o estado EXT. Caso não seja altamente correlacionado, ele vai para o estado GWR e repete os passos atribuídos a ele.

O modelo DCSSC prioriza como representantes de agrupamentos os sensores com maior nível relativo de energia dentre os que apresentam leituras similares, prolongando a vida útil da rede. Os nodos com os papéis de CH, GW e EXT (nodos de *backbone*) têm a função de coletar dados das leituras dos sensores, que enviam suas informações através do *backbone* a cada intervalo de tempo segundo o esquema de escalonamento Round-Robin (fila circular). As leituras recebidas são armazenadas no CH, e podem ser comprimidas a fim de reduzir o espaço necessário de armazenamento. Os dados agregados são transmitidos do CH para o GW do respectivo *cluster*, que encaminha para um sensor vizinho que pertença a outro *cluster*. Dessa forma os dados trafegam na rede através de nodos intermediários, até chegar à estação-base, com menor custo de comunicação. O caminho inverso também é válido, ou seja, o *backbone* é responsável por transmitir mensagens de controle da estação-base, propagando a mensagem quando é recebida pela primeira vez e evitando tráfego desnecessário de mensagens.

3.2 Classificação

Conforme proposto na Figura 2.1, o modelo DCSSC se enquadra na categoria de armazenamento na rede (ou distribuído), uma vez que os dados sensoriados ficam armazenados nos próprios sensores e não são transferidos para uma entidade externa. Além disso, o modelo é hierárquico, pois os nós que integram o *backbone* funcionam como repositórios,

agregagando os dados de modo que não é necessário consultar pontualmente cada sensor para obter informações.

No Capítulo 4 serão abordados os detalhes da implementação do modelo DCSSC, apresentando a divisão em componentes de acordo com os conceitos do *framework* RCBM.

Capítulo

4

Implementação

Para a implementação do modelo DCSSC utilizando o *framework* RCBM, além do desenvolvimento da camada de implementação foi necessária a criação de um componente de biblioteca, responsável por atribuir leituras aos sensores.

A Figura 4.1 representa uma instância da arquitetura do metamodelo RCBM, com base na implementação do modelo DCSSC. Os componentes em preto foram utilizados conforme fornecido, e os destacados na cor azul são aqueles que foram modificados e/ou criados para o desenvolvimento deste trabalho. A plataforma de comunicação escolhida foi a ferramenta de simulação *Network Simulator 2* (NS-2).

É possível perceber que a parte da especificação, que contém os *templates* e componentes de biblioteca, praticamente não foi alterada. A única modificação foi a inclusão de um novo componente de biblioteca, que inicializa as leituras do sensores e será descrito na Seção 4.1.

Já no que envolve a parte de implementação, os componentes de aplicação *Cluster member* (CM) e *Cluster-head* (CH) tiveram que ser adaptados ao modelo como será discutido na Seção 4.2. Além disso, a implementação do Coordenador também foi desenvolvida para adequar-se ao fluxo de execução do DCSSC, conforme será apresentado na Seção 4.3. A Seção 4.4 descreve os experimentos realizados durante o desenvolvimento deste estudo de caso.

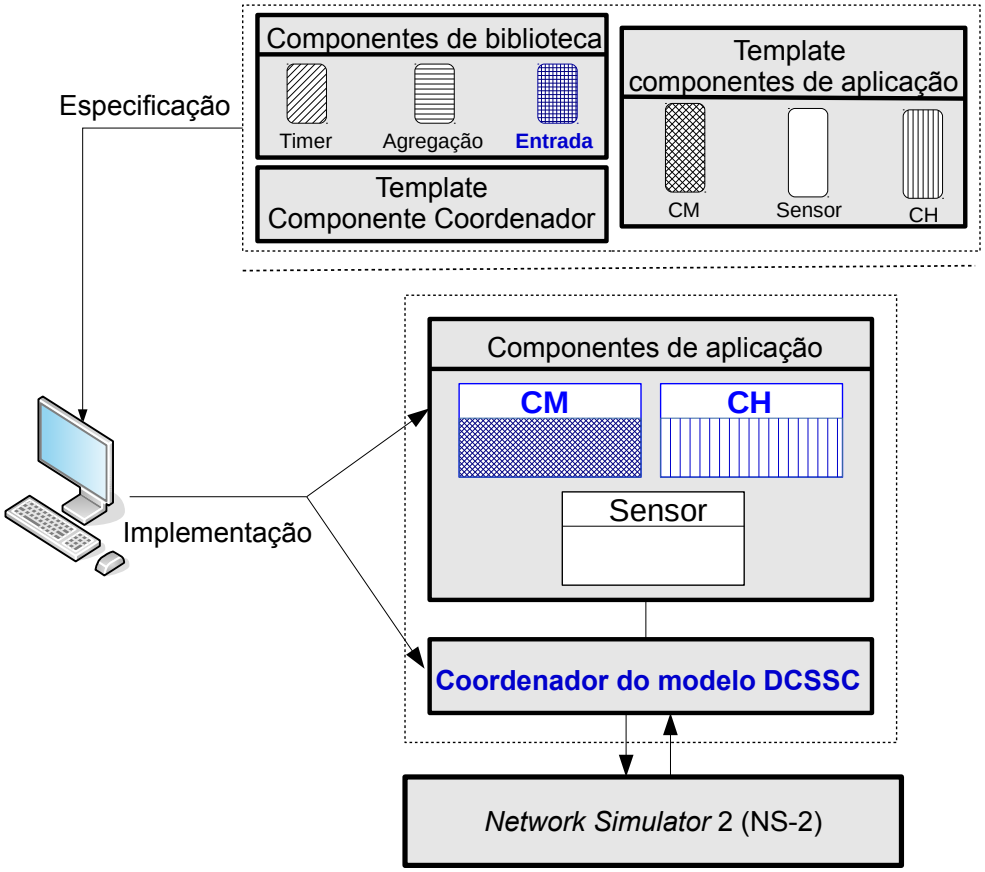

Figura 4.1: Instância da arquitetura RCBM aplicada ao modelo DCSSC

4.1 Componentes de biblioteca

Para a implementação do modelo DCSSC, foi desenvolvido um componente de entrada, que apresenta a função `getReadings`, cujos parâmetros são: o identificador do sensor cujas leituras devem ser obtidas, o descritor do arquivo onde estão as leituras e um ponteiro para um inteiro que identifica o número de leituras que foram obtidas. O valor de retorno é um ponteiro para um vetor de números reais que são as leituras do sensor (Código 4.1).

```
1 class Read {
2     public:
3         double * getReadings(int id, FILE *f, int *numReadings);
4 }
```

Código 4.1: Especificação do componente de leituras

O arquivo deve estar em um formato específico e seu nome é informado como parâmetro para a execução do programa. Cada linha inicia com um inteiro x, que determina quantas leituras deverão ser realizadas, seguido de x números reais que representam os valores das leituras. O índice da linha corresponde ao número do identificador do sensor, ou seja, a *i-ésima* linha contém os valores das leituras do sensor cujo identificador é i. É imediato que o número de linhas do arquivo de leituras deve ser igual ou superior ao número de sensores da rede. Um exemplo de arquivo de entrada está ilustrado na Figura 4.2.

readings.txt

```
3 25.107 2.007 6.052
3 24.966 1.970 6.039
3 25.085 2.014 5.955
3 25.002 1.971 6.100
3 24.073 1.981 3.976
3 24.026 1.963 3.930
3 24.025 1.997 3.971
3 23.905 1.966 3.936
3 24.008 1.958 4.051
3 23.940 2.006 3.914
3 24.070 2.050 4.002
3 23.944 1.975 4.089
3 23.971 2.033 4.075
3 24.056 1.965 3.917
3 24.911 1.987 6.028
```

Figura 4.2: Exemplo de arquivo de entrada com 3 leituras para rede com até 15 sensores

A função também atualiza o valor do parâmetro numReadings, passado por referência, que posteriormente deve ser armazenado em cada sensor para determinar quantas leituras ele está armazenando.

O componente de leituras é essencial para o desenvolvimento de modelos em RSSFs, já que a premissa básica desse tipo de rede é que os dispositivos estão constantemente capturando dados do ambiente em que estão localizados.

4.2 Componentes de aplicação

Os componentes de aplicação estão relacionados às entidades de uma rede de sensores sem fio. Foram utilizados para o desenvolvimento do modelo DCSSC os componentes: sensor, CH e CM. Todos os *templates* disponibilizados pelo RCBM foram reutilizados, e

apenas as especificações dos componentes CH e CM foram modificadas para se adaptar ao funcionamento desse modelo específico.

Como o modelo de armazenamento em questão é classificado como em repositórios, os sensores armazenam e gerenciam informações sobre seus vizinhos localmente. Dessa forma, tanto o *template* como a especificação do componente Sensor do RCBM, descritos no artigo do modelo (Carrero et al., 2017), foram utilizados sem modificações.

O componente CH contém a função `SelectCH`, que leva em consideração os requisitos de eleição de um representante de acordo com o modelo utilizado. No caso do DCSSC, essa função é executada por um sensor ao receber uma mensagem do tipo CHREQ. Com base na dissimilaridade entre o sensor emissor da mensagem e o receptor, o receptor irá declarar-se ou não como um CH daquele possível agrupamento. A implementação está detalhada no Código 4.2.

```
1  if (dissimilaridade <= agent->getCompSensor()->threshold) {
2      agent->getCompSensor()->role = CHC;
3      double energy = agent->getCompSensor()->energia;
4      double energyNBR = 0;
5      int nbr = 0;
6      for (for i in = agent->getCompSensor()->vecNeighbors) {
7          energyNBR += (*i).energia;
8          nbr++;
9      }
10     double res = (energy + energyNBR)/(energy * (nbr+1));
11     res = res * alfa;
12     agent->TimerTadv(minAdv, res);
13     agent->round = TADV;
14     agent->getCompSensor()->role = CH;
15 }
```

Código 4.2: Especificação do componente CH

A dissimilaridade é recebida como parâmetro da função `SelectCH`, e é comparada ao valor do limiar que é armazenado pelo sensor (linha 1). Se for menor ou igual, o sensor

adquire o papel de CHC (linha 2). O nível relativo de energia do sensor (Equação 3.2) é calculado (linhas 3-11), e é passado como parâmetro para setar o *timer* Tadv. Esse *timer* está implementado no componente de biblioteca Timer. Quando Tadv expirar, será feito o *broadcast* da mensagem CHADV, e o sensor recebe o papel de CH.

O outro componente utilizado no desenvolvimento foi o CM, ou *cluster member*. Nesse componente existe a função `JoinCluster`, na qual é feita a decisão se um sensor vai ou não se juntar a um *cluster*. No caso do DCSSC, a função é executada ao receber uma mensagem do tipo CHADV. Além de decidir se irá se juntar ao agrupamento, o sensor pode se tornar um GW ou GWR, baseado nas condições mencionadas. A implementação está detalhada no Código 4.3.

```
1  if (dissimilaridade <= agent->getCompSensor()->threshold) {
2      if (agent->getCompSensor()->role == GWR)
3      {
4          if (sp.pID == agent->getCompSensor()->getSensorId()
5              agent->getCompSensor()->role = GW;
6          else
7              agent->getCompSensor()->role = CM;
8      }
9      else
10         agent->getCompSensor()->role = CM;
11     agent->getCompSensor()->setCHReading(sp.reading);
12     agent->TimerText();
13     agent->round = TEXT;
14 } else {
15     agent->getCompSensor()->role = GWR;
16     agent->TimerTreq();
17     agent->round = TREQ;
18 }
```

Código 4.3: Especificação do componente CM

A dissimilaridade é recebida como parâmetro da função `JoinCluster`, e é comparada ao valor do limiar que é armazenado pelo sensor (linha 1). Se for menor ou igual, o sensor é altamente correlacionado, e com base em seu papel atual deve tomar ações específicas. Caso o sensor seja GWR, compara se o pID (ID do predecessor) da mensagem recebida é igual ao seu próprio ID (linha 4). Se forem iguais, é atribuído a ele o papel de GW (linha 5). Caso contrário, ele irá se tornar um CM (linha 7). Caso o sensor tenha um papel diferente de GWR, ele também se torna um CM (linha 10). Ambos os tipos de sensores armazenam a leitura do seu representante (linha 11). É setado o *timer* Text, implementado no componente de biblioteca Timer. Quando Text expirar é disparado o evento de *broadcast* da mensagem CEXT (linhas 12 e 13). Caso o sensor não seja altamente correlacionado ao emissor da mensagem CHADV recebida, ele se torna um GWR (linha 15) e executará o ciclo atribuído aos sensores com papel GWR quando o *timer* Treq expirar (linhas 16 e 17).

4.3 Componente do coordenador

De acordo com Carrero (Carrero et al., 2017), a principal função do coordenador é fazer a junção dos componentes, coordenando as interações entre eles. As tarefas do coordenador são tratar as mensagens recebidas e estabelecer o fluxo de execução do algoritmo.

O fluxo de execução do DCSSC segue uma máquina de estados como representado na Figura 4.3. Todos os nodos iniciam no estado INI, e modificam seu estado ao longo da execução.

Conforme descrito na notação, as mudanças de estado estão classificadas em três tipos:

- Mundaça de estado por evento: são aquelas que ocorrem quando um sensor recebe algum tipo de mensagem.
- Mudança de estado lógica: é executada com base no resultado de uma computação (como o cálculo de dissimilaridade) ou então após o tempo de expiração de um *timer*.

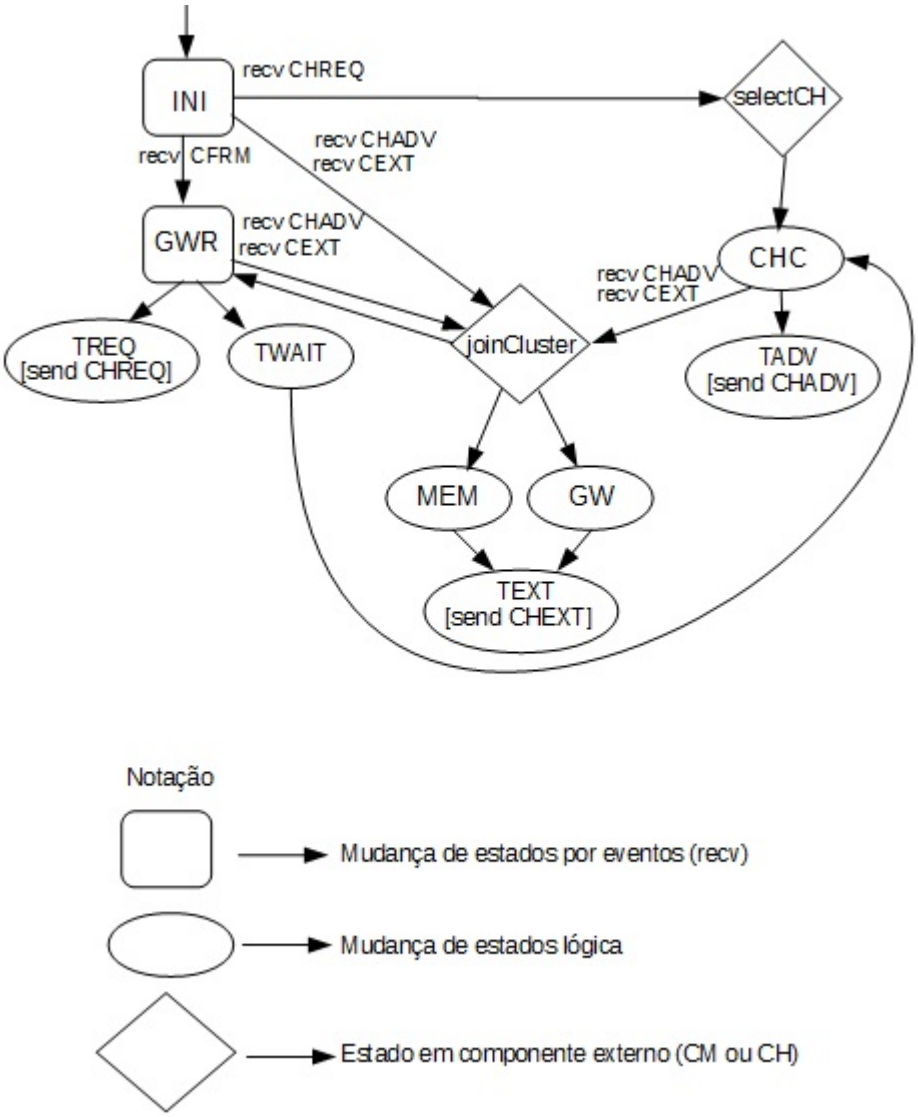

Figura 4.3: Máquina de estados de execução do coordenador

- Estados em componente externo ao coordenador: são executados em um componente externo ao coordenador, como por exemplo o CM ou CH.

O coordenador tem duas funções principais a serem implementadas pelo desenvolvedor: `recv` e `TimerHandle`. A função `recv` é responsável pelo recebimento de pacotes pelos sensores e trata essas mensagens. É a parte do código que faz as transições do estado do tipo "mudança por evento"e também faz a chamada para os componentes externos. O `recv` é composto por um comando *switch*, com vários *cases*, cada um equivalente a um estado que pode ser atingindo pelas mudanças de estado por evento. Cada mensagem possui um ID, e dentro dela existem alguns parâmetros como o ID do emissor, suas

coordenadas e leituras, além do ID do emissor da última mensagem recebida pelo sensor. Parte da implementação da função `recv` está contida no Código 4.4.

```
1  void WSN_ComponentsAgent::recv(Packet* pkt, Handler *) {
2      SensorDataParams sp;
3      WSN_Components_Message param = pkt;
4
5      switch(param.getMsgId()) {
6          case(CFRM): {
7              lastMsg = CFRM;
8              if ( compSensor->role == INI ) {
9                  compSensor->role = GWR;
10                 TimerTreq();
11                 round = TREQ;
12             }
13         }
14         break;
15         case(CHREQ): {
16             lastMsg = CHREQ;
17             if ( compSensor->role == INI ) {
18                 compSensor->pID = param.getHdrCmn()->prev_hop_;
19                 sp.reading = param.getHdrWsnComp()->
20                                 msgParams.msgParam[0].reading;
21                 double s1 = compSensor->getReadings();
22                 double dissimilaridade = fabs(s1-sp.reading);
23                 map<string, double> sim;
24                 sim.insert(pair<string,
25                                 double> ("K", dissimilaridade));
26                 compCH->SelectCH(sim);
27             }
28         }
29         break;
```

```
30            (...)
31        }
32    }
```

Código 4.4: Função `recv` do coordenador

Ao receber um pacote, é feita uma comparação do tipo da mensagem recebida (linha 5). Caso seja um CFRM (linha 6) e o papel do sensor for INI, o dispositivo muda para o papel GWR e a rodada a ser tratada pela função `TimerHandle` será TREQ (linhas 8 a 11). A variável lastMsg armazena o tipo da última mensagem recebida (linha 7 e linha 16). Se a mensagem recebida foi um CHREQ (linha 15) e o papel do sensor for INI, o sensor calcula a dissimilaridade com a leitura recebida como parâmetro da mensagem (linhas 19 a 22). Em seguida, a função `SelectCH` do componente CH é chamada (linhas 23 a 26). Trechos de código similares são executados para os outros estados.

A função `TimerHandle` é executada quando existe a transição do tipo "mudança de estado lógica"e é ativada quando um *timer* setado durante a execução da função `recv` expira e uma nova rodada (ou *round*) é iniciada. O `TimerHandle` cria pacotes e os envia conforme o tipo da rodada atual. Um trecho do código dessa função é apresentado no Código 4.5.

```
1  void WSN_ComponentsAgent::TimerHandle(RoundCmd roundCmd) {
2      SensorDataParams sp;
3      MsgParam newp;
4      Packet* pkt = getNewPkt();
5      WSN_Components_Message param(pkt);
6      param.setId(compSensor->getSensorId());
7
8   switch (roundCmd) {
9       case TREQ: {
10                 param.setMsgId(CHREQ);
11                 sp.id = param.getId();
12                 sp.coordX = compSensor->getX();
```

```
13              sp.coordY = compSensor->getY();
14              sp.coordZ = compSensor->getZ();
15              sp.reading = compSensor->getReadings();
16              newp.msgParam[0] = sp;
17              SendPkt(CHREQ, &param);
18          }
19          break;
20          case TADV: {
21              param.setMsgId(CHADV);
22              sp.id = param.getId();
23              sp.coordX = compSensor->getX();
24              sp.coordY = compSensor->getY();
25              sp.coordZ = compSensor->getZ();
26              sp.pID = compSensor->pID;
27              newp.msgParam[0] = sp;
28              SendPkt(CHADV, &param);
29          }
30          break;
31          (...)
32      }
33  }
```

Código 4.5: Função `TimerHandle` do coordenador

No início da função são inicializados os parâmetros necessários para o envio do pacote (linhas 2-6). Em seguida, é feito um *switch* da rodada atual (linha 8). Caso a rodada seja TREQ (linha 9) é criada uma mensagem do tipo CHREQ, com os parâmetros relacionados ao sensor (ID, coordenadas, leitura), e o pacote é enviado (linhas 10 a 17). O código para o caso TADV é similar, uma mensagem do tipo CHADV é criada com os mesmos parâmetros da mensagem CHREQ e é enviada (linhas 20 a 28). O tratamento dos outros tipos de rodada ocorre de forma análoga.

Os experimentos realizados a partir do sistema implementado são descritos na sequência.

4.4 Experimentos

Para a realização dos experimentos foram observados dois aspectos: a porcentagem do código que foi possível reutilizar do modelo RCBM (Subseção 4.4.1) e a corretude da implementação, comparando os resultados das execuções com as do artigo original do DCSSC (Subseção 4.4.2).

4.4.1 Reutilização de código

Para analisar a eficácia do modelo RCBM durante o desenvolvimento de código, foram contabilizadas as linhas de código necessárias para implementar o modelo. A Tabela 4.1 apresenta o número total das linhas de código, quantas delas foram criadas, quantas foram reutilizadas e a porcentagem equivalente à reutilização.

O número de linhas equivalentes à implementação do componente de entrada da biblioteca foi inserido na contagem A como linhas novas e na contagem B como linhas reutilizadas. Considerando que o componente não existia no modelo RCBM e foi criado para o desenvolvimento deste estudo, seu código pode ser classificado como novo. Entretanto, levando-se em consideração implementações futuras, esse componente pode ser considerado reutilizável, justificando as linhas de sua implementação inseridas como linhas reutilizadas.

Contagem	**Total de linhas**	**Linhas reutilizadas**	**Linhas novas**	**Porcentagem de reuso**
A	1333	905	428	67,9%
B	1333	954	379	71,6%

Tabela 4.1: Proporção de linhas reutilizadas do RCBM

4.4.2 Corretude da implementação

Conforme já mencionado, uma das dificuldades para a reutilização de código em modelos de armazenamento para RSSFs é a falta de códigos disponibilizados à comunidade. O modelo DCSSC é um dos que apresenta essa falha. Sua implementação não foi fornecida pelos autores do artigo, portanto a comparação do código original com o código gerado não foi possível. Para validar a implementação realizada, foi comparado o número de mensagens necessárias para a formação dos *clusters* demonstrado no artigo com o que foi observado durante o desenvolvimento deste estudo. O objetivo é verificar se o código implementado possui o mesmo comportamento do que o que foi reportado no artigo do modelo DCSSC (Le et al., 2008). Esse aspecto foi escolhido por ser o diferencial do DCSSC em relação a outros modelos similares que foram analisados no artigo original (Le et al., 2008). O DCSSC executa o algoritmo de formação de *clusters* com um número linear de mensagens, fato que deve também ser observado na implementação desse estudo de caso.

Para a execução dos experimentos, foram feitas execuções com 200, 400 e 600 nodos, considerando uma área monitorada de 1400m X 1000m, e o alcance dos sensores em 100m. Foi utilizado como parâmetro um limiar de 20%. O cenário foi adaptado, já que os autores do artigo original não disponibilizaram os arquivos de cenário utilizados. Os resultados originais estão representados na Figura 4.4, e os obtidos durante experimentos são retratados na Figura 4.5.

É possível notar pela comparação dos gráficos que a implementação obteve o resultado esperado, ou seja, é condizente com o encontrado em (Le et al., 2008). Os valores originais não foram apresentados no artigo, sendo reportados apenas em formato de gráfico, e por isso não foram citados. É possível observar o crescimento linear do número de mensagens conforme a densidade da rede aumenta em ambas as implementações.

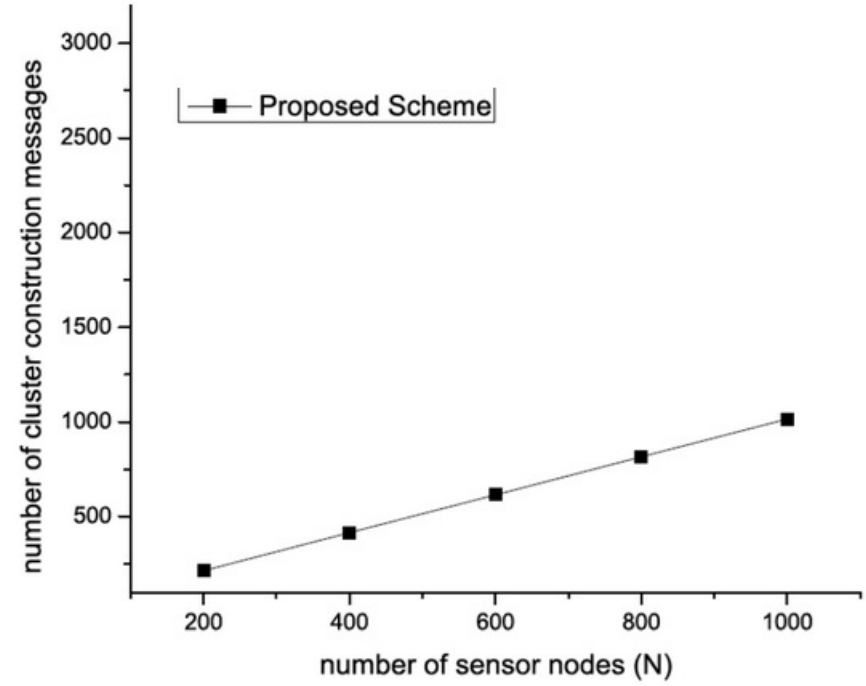

Figura 4.4: Influência do número de sensores na quantidade de mensagens de formação. Fonte: (Le et al., 2008)

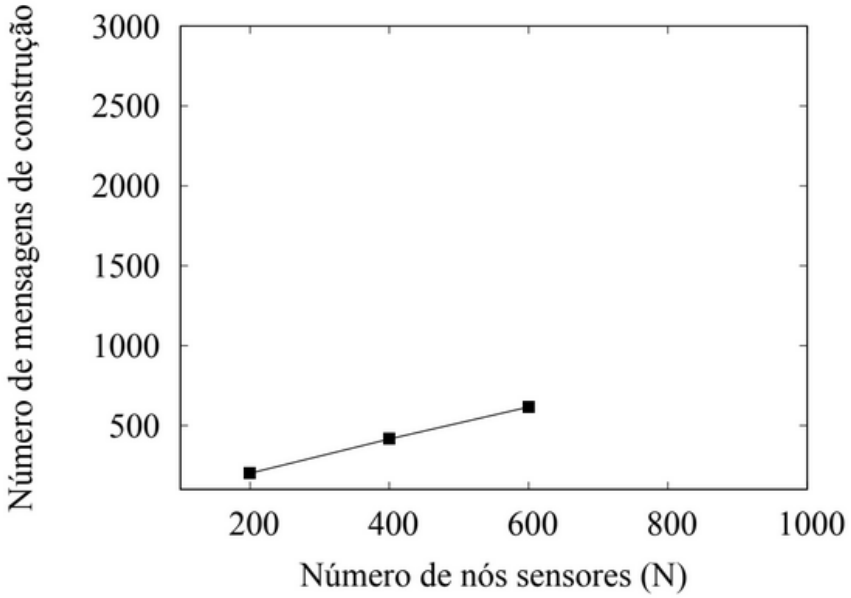

Figura 4.5: Resultados da influência do número de sensores na quantidade de mensagens de formação obtidos nos experimentos do código implementado

Capítulo

5

Conclusão

Neste trabalho foi realizado um estudo de caso do modelo RCBM por meio da implementação do modelo de armazenamento DCSSC. No Capítulo 1 foi abordado o problema de pesquisa que motivou a realização deste trabalho, incluindo definições sobre redes de sensores sem fio, modelos de armazenamento e como a abordagem baseada em componentes seria interessante dentro desse escopo. No Capítulo 2, houve a discussão de trabalhos relacionados ao desenvolvimento de software baseado em componentes, além da análise das propostas existentes no âmbito de RSSFs e a apresentação do modelo RCBM, um modelo reutilizável para desenvolvimento em redes de sensores sem fio. O modelo DCSSC, implementado como estudo de caso do *framework* RCBM, foi detalhado no Capítulo 3. Os detalhes da implementação, assim como os experimentos para estudo e validação, foram apresentados no Capítulo 4.

Os experimentos demonstram que o percentual de reutilização de código utilizando o RCBM para a implementação do DCSSC foi de 71,6%, considerando o componente de biblioteca como reutilizável, e de 67,9%, considerando o componente de biblioteca criado como linhas de código novas. Os valores são significativos considerando a complexidade dos códigos de simulação para redes de sensores sem fio. Nota-se que grande parte do código criado também foi similar, como o tratamento do recebimento das mensagens e o controle das ações a serem tomadas após o acontecimento de um evento.

Observou-se que os resultados obtidos pelo programa gerado utilizando-se a plataforma RCBM foram condizentes com o que foi apresentado no trabalho original (Le et al., 2008), o que corrobora a validade da solução implementada.

Além disso, baseado em experiências prévias com a implementação *from scratch* de modelos de armazenamento em redes de sensores sem fio, foi possível perceber aumento significativo da facilidade do processo de desenvolvimento. Com o RCBM, os componentes e suas funcionalidades estão bem definidos, o que torna o código mais modular e claro. Os componentes de biblioteca também são muito úteis, uma vez que fornecem funções que auxiliam o desenvolvimento genérico para redes de sensores sem fio, e não somente o desenvolvimento associado a uma linguagem de programação ou ferramenta específica.

Como trabalhos futuros, pode-se citar o desenvolvimento de uma linguagem baseada em máquina de estados para a geração automática do coordenador, a realização de estudos experimentais adicionais envolvendo outras métricas e utilização da abordagem de reutilização em outros simuladores, como por exemplo o *Network Simulator 3* (NS-3).

Referências Bibliográficas

Akyildiz, I. F.; Su, W.; Sankarasubramaniam, Y.; Cayirci, E. A survey on sensor networks. *IEEE Communications Magazine*, v. 40, n. 8, p. 102–114, 2002.

Amaxilatis, D.; Chatzigiannakis, I.; Koninis, C.; Pyrgelis, A. Component based clustering in wireless sensor networks. *CoRR*, v. abs/1105.3864, 2011.

Carrero, M. A. *Proposta de doutorado: Um metamodelo de armazenamento de dados para rssfs urbanas.* Dissertação de Mestrado, Universidade Federal do Parana, 2016.

Carrero, M. A.; Santos, A. L. d.; Musicante, M. A.; Hara, C. S. A reusable component-based model for wsn storage simulation. In: *Proceedings of the ACM Conference*, Conference'17, Washington, DC, USA, 2017 (*Conference'17*,).

Cheong, E.; Liebman, J.; Liu, J.; Zhao, F. Tinygals: A programming model for event-driven embedded systems. In: *Proceedings of the 2003 ACM Symposium on Applied Computing*, SAC '03, New York, NY, USA: ACM, 2003, p. 698–704 (*SAC '03*,).

Huynh, D. F.; Karger, D. R.; Miller, R. C. Exhibit: Lightweight structured data publishing. In: *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, New York, NY, USA: ACM, 2007, p. 737–746 (*WWW '07*,).

Kushwaha, M.; Amundson, I.; Koutsoukos, X.; Neema, S.; Sztipanovits, J. Oasis: A programming framework for service-oriented sensor networks. In: *2007 2nd International Conference on Communication Systems Software and Middleware*, 2007.

Le, T. D.; Pham, N. D.; Choo, H. Towards a distributed clustering scheme based on spatial correlation in wsns. In: *2008 International Wireless Communications and Mobile Computing Conference*, 2008, p. 529–534.

Loureiro, A. A.; Nogueira, J. M. S.; Ruiz, L. B.; Mini, R. A. d. F.; Nakamura, E. F.; Figueiredo, C. M. S. Redes de sensores sem fio. *XXI Simpósio Brasileiro de Redes de Computadores*, p. 179–226, 2003.

Minakov, I.; Passerone, R.; Rizzardi, A.; Sicari, S. A comparative study of recent wireless sensor network simulators. *ACM Trans. Sen. Netw.*, v. 12, n. 3, p. 20:1–20:39, 2016.

Niekamp, R. Software component architecture. *Gestión de Congresos-CIMNE/Institute for Scientic Computing*, 2005.

Patel, P.; Pathak, A.; Teixeira, T.; Issarny, V. Towards application development for the internet of things. In: *Proceedings of the 8th Middleware Doctoral Symposium*, MDS '11, New York, NY, USA: ACM, 2011, p. 5:1–5:6 (*MDS '11*,).