UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO
UNIVERSIDADE DO ESTADO DO RIO GRANDE DO NORTE
CURSO DE MESTRADO EM CIÊNCIA DA COMPUTAÇÃO

LUANA DANTAS CHAGAS

PROCESSAMENTO DE CONSULTAS EM NÓS SENSORES CONSIDERANDO RESTRIÇÕES LÓGICAS E TEMPORAIS

MOSSORÓ - RN
2012

LUANA DANTAS CHAGAS

PROCESSAMENTO DE CONSULTAS EM NÓS SENSORES CONSIDERANDO RESTRIÇÕES LÓGICAS E TEMPORAIS

Dissertação apresentada ao Mestrado em Ciência da Computação – associação ampla entre a Universidade do Estado do Rio Grande do Norte e a Universidade Federal Rural do Semi-Árido, para a obtenção do título de Mestre em Ciência da Computação.

Orientador: Pedro Fernandes Ribeiro Neto - UERN

MOSSORÓ - RN
2012

LUANA DANTAS CHAGAS

PROCESSAMENTO DE CONSULTAS EM NÓS SENSORES CONSIDERANDO RESTRIÇÕES LÓGICAS E TEMPORAIS

Dissertação apresentada ao Mestrado em Ciência da Computação para a obtenção do título de Mestre em Ciência da Computação.

Aprovado em 30/03/2012

BANCA EXAMINADORA

Prof. D.Sc Pedro Fernandes Ribeiro Neto (Orientador)
Universidade do Estado do Rio Grande do Norte - UERN

Profa. D.Sc Cláudia Maria Fernandes Araújo Ribeiro
Universidade do Estado do Rio Grande do Norte - UERN

Profa. D.Sc Rossana Maria de Castro Andrade
Universidade Federal do Ceará - UFC

Aos meu pais, Liana e Chagas.

Agradecimentos

Muitas pessoas, direta ou indiretamente, contribuíram e fizeram parte desta etapa da minha vida. Tenho certeza de que se não fosse por elas, não teria chegado até aqui. Espero que elas se sintam agradecidas e saibam que esta conquista não é só minha, mas de todos nós.

Agradeço à minha família pelo apoio incondicional, pela torcida, pela ajuda, pela compreensão, pelos conselhos e pela paciência. Agradeço em especial a minha mãe por participar tão intensamente não somente das conquistas como também dos momentos difíceis!

Agradeço ao meu namorado, Mário, pelo apoio a todo momento, pela compreensão sempre, pelo estímulo (mesmo que muitas vezes à distância!), pela força, pelo carinho e pelos belos momentos já compartilhados.

Agradeço à Vazeli pelos tantos anos de amor, apoio, dedicação e cuidado comigo, sendo como uma mãe para mim e estando presente sempre nos mais importantes momentos, seja compartilhando alegrias ou dificuldades.

Agradeço aos meus amigos, Gabriel Aderbal e Sandja Sarandy, que embora à distância sempre se dispuseram a me ouvir, animar e a me ajudar. Agradeço também a família deles por serem sempre tão simpáticos e carinhosos comigo, me acolhendo em suas casas sempre que precisei.

Agradeço ao meu orientador, Pedro Fernandes Ribeiro Neto, por esses já (praticamente) quatro anos de orientação! Obrigada pela paciência, pela atenção e carinho, pela confiança e, principalmente, pelas lições acadêmicas!!

Agradeço aos membros do GES, inclusive aqueles que não são mais membros oficialmente, mas que não cortaram os laços!! Agradeço em especial a Jocileide Marinho, Márcia Aratusa, Lenardo Chaves, Cleone Silva, Gracon Huttenberg, Jomar Ferreira, Ronnison Reges e Antonio Denilson, que sempre estiveram dispostos a me ajudar, ouvindo minhas dúvidas, discutindo soluções, ajudando nas apresentações e oferecendo sempre apoio.

Agradeço aos colegas do mestrado, que muito contribuiram para a minha formação! Obrigada pelos grupos de estudos (inclusive nas madrugadas!) e, principalmente, pelas conversas produtivas de corredor ou de laboratório. Agradeço em especial a Adeline Marinho, Cláubio Landney, Dayanne Kelly, Selma Márcia, Abrahão Cristophe, Francisco Miguel, Marcos Tullyo e Luiz Cláudio pelas parcerias de sucesso nos trabalhos e constribuições decisivas nas tarefas acadêmicas.

Agradeço aos professores, do corpo docente do mestrado, do curso de Ciência da Computação da UERN ou membros do GES, que muito me ensinaram, e por vezes me inspiraram, a crescer como pesquisadora.

Agradeço à Rosita, secretária do Mestrado na UERN, por sempre estar disposta a me ajudar (desde o tempo da graduação!), pelas conversas, conselhos e, principalmente, pela torcida!

Agradeço à CAPES por todo o apoio financeiro concedido para o desenvolvimento da pesquisa.

Agradeço a Deus por trilhar meus caminhos e permitir que eu possa agradecer a tantas pessoas!

Resumo

Redes de Sensores sem Fio (RSSF) consistem em um conjunto de nós autônomos, espacialmente distribuídos, que trabalham de forma cooperativa para monitorar uma região obtendo dados acerca da mesma. Esses nós possuem capacidade de sensoriar parâmetros físicos, armazenar e processar, e realizar comunicação entre si. A coleta e o envio de dados pelos nós podem ser modificados, após implantação da rede, por meio de linguagens declarativas, definindo quando, onde e quão frequentemente os dados serão coletados e enviados. Para tal, os nós sensores devem ser capazes de processar consultas através de regras gramaticais previamente definidas. A modificação da coleta e envio de dados permite adaptar melhor a rede às necessidades das aplicações. As RSSF têm grande potencial para diversas aplicações como rastreamento de alvos militares, sistemas de vigilância, controle de processos em fábricas e monitoramento de desastres naturais. Dentre as aplicações, algumas caracterizam-se por, além de necessitar manter a lógica inerente em qualquer aplicação, haver a necessidade de obter dados mais atuais do ambiente dentro de prazos específicos. Nessas aplicações, o não cumprimento dos prazos na realização das tarefas pode acarretar nas mais diversas consequências, podendo trazer grandes prejuízos. A necessidade de tratar restrições de tempo impõe aos nós a adoção de estratégias para gerenciar atividades de coleta e envio de dados que visem o cumprimento de prazos dos dados e das transações que operam sobre esses. Manter a consistência temporal, no entanto, pode acarretar em impasses entre manter a consistência lógica e temporal dos dados em detrimento da consistência lógica e temporal das transações, e vice-versa. Neste trabalho, tem-se como objetivo realizar o processamento de consultas em nós sensores, considerando que esses atendem aplicações com restrições de tempo e, portanto, dados e transações possuem restrições temporais. Assim, é adotado nos nós sensores funções para o processamento de uma linguagem de consulta, possibilitando a comunicação com os nós, e uma estratégia para gerenciar dados e transações, de modo que restrições de tempo possam ser atendidas.

Palavras-chave: Redes de Sensores sem Fio, consulta aos dados, dados e transações de tempo real.

Abstract

Wireless Sensor Networks (WSN) consist of a set of autonomous nodes, spatially distributed, working cooperatively to monitor a region obtaining data about it. These nodes are capable of sensing physical parameters, store and process, and communicate with each other. The collection and submission of data by sensor nodes may be modified after deployment of the network using declarative languages, defining when, where and how often data will be collected and sent. To do so, the sensors must be able to process queries using grammar rules defined in advance. This modification allows the network to better adapt to the needs of applications. WSN have great potential for various applications such as military target tracking, surveillance systems, process control in factories and monitoring of natural disasters. Among the applications, some are characterized by, besides the need to maintain the logic inherent in any application, there is the need to obtain the most current data environment within specific deadlines. In these applications, the failure to meet deadlines in the tasks can result in several consequences, which may cause great damage. The need to address time constraints impose on sensor nodes the adoption of strategies to manage activities of collecting and sending data aimed at meeting deadlines of data and transactions that operate on that. Maintain the temporal consistency, however, can lead to deadlocks between maintain logical and temporal consistency of data over logical and temporal consistency of transactions, and vice versa. This work has as objective the query processing in sensor nodes, whereas those serving applications with time constraints and, therefore, data and transactions have time constraints. Thus, it is adopted in the sensor nodes functions for processing a query language, allowing communication with nodes, and a strategy to manage the data and transactions, so that time constraints can be met.

Keywords: Wireless Sensor Networks, data querying, real-time data and transactions.

Lista de Siglas

API Application Programming Interface

ARP Address Resolution Protocol

BD Banco de Dados

BDTR Banco de Dados de Tempo Real

CAD Conversor Analógico Digital

DDL Data Definition Language

DML Data Manipulation Language

IDE Integrated Development Environment

MAC Medium Access Control

Mixim Mixed Simulator

NED Network Description

OMNeT++ Objective Modular Network Test-bed in C++

PDCRS Processamento e Distribuição de Consultas em Redes de Sensores

QLSND Query Language for Sensor Network Databases

RSSF Redes de Sensores sem Fio

SNQL Sensor Network Query Language

SQL Structured Query Language

STR Sistemas de Tempo Real

XML Extensible Markup Language

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Sumário

1 Introdução

As Redes de Sensores sem Fio (RSSF) são compostas por nós sensores, posicionados em uma região geográfica, com o objetivo de coletar dados do meio ambiente relativos à detecção ou medição de algum fenômeno físico (MATOS *et al.*, 2008). A comunicação entre os nós sensores é feita sem a utilização de fios. Tal fato auxilia a abrangência de grandes dimensões de áreas, bem como áreas de difícil acesso. Além disso, as RSSF devem ser autônomas: uma vez implantadas, essas devem ser capazes de operar sem a intervenção de humanos. Tais características fazem das RSSF uma ferramenta capaz de trazer soluções para o monitoramento das mais diversas áreas. De acordo com Klingbeil e Wark (2008), as RSSF são uma ferramenta para medir características espaciais e temporais de quase todos os fenômenos. Exemplos de aplicações dessas redes são pesquisas científicas sobre monitoramento ambiental como em Szewczyk *et al.* (2004) e localização de danos estruturais em prédios como em Yan *et al.* (2009).

Segundo Ribeiro Neto (2006b), inicialmente os nós sensores tinham uma capacidade limitada de apenas detectar um evento e enviar um sinal, fazendo com que as RSSF pudessem apenas ser coletoras de dados. Dessa forma, os nós sensores tinham todo o seu comportamento previamente programado e, uma vez a rede implantada e em execução, o usuário interessado no fenômeno monitorado pela rede podia apenas trabalhar com os dados enviados dos dispositivos, sem interferir no modo de obtenção desses. Um exemplo de tais redes são alguns sistemas de vigilância com nós sensores de presença. De forma geral, nesses sistemas os nós apenas detectam presença, caso aconteça o estímulo físico de um objeto passando por eles, e enviam sinais para centrais de alarmes.

De acordo com Pagano *et al.* (2010), RSSF têm a vantagem de contar com nós sensores com capacidades cada vez maiores, aptos a realizar processamentos e armazenamento de dados, e que suportam aplicações mais complexas, que vão além de somente detectar um dado e enviar um sinal em resposta. A capacidade maior de processamento e de comunicação dos nós sensores possibilita o desenvolvimento de aplicações cujos usuários podem estabelecer parâmetros para o comportamento da rede. Através de consultas com gramáticas previamente conhecidas, pelos usuários e pelos nós sensores, é possível permitir que os usuários determinem exatamente quando, onde e quão frequentemente gostariam que os dados fossem coletados e enviados. Em Matos *et al.* (2008), um exemplo de tal situação é ilustrado, uma rede na qual o usuário pode determinar em quais momentos os nós sensores devem enviar seus dados interagindo com esses por meio de uma linguagem de consulta.

Os nós sensores têm como uma das suas características a capacidade limitada de recursos como, por exemplo, o armazenamento de energia. Dessa forma, quando as RSSF são implantadas é necessário adotar estratégias para economizar os recursos dos nós. A possibilidade de o usuário alterar o comportamento dos nós da rede, mesmo com a rede já implantada, permite

modificar estratégias adotadas para economizar energia. Através do processamento de consultas, por exemplo, pode-se limitar o número de nós coletando dados ou enviando mensagens ou, ainda, determinar que dados sejam agregados, resultando em economia de recursos energéticos.

Além de poder criar uma interface de comunicação com a RSSF para interagir com os nós, outro fator também se faz presente nas aplicações atuais, as restrições de tempo. Segundo Pagano *et al.* (2010), trata-se de uma segunda geração de RSSF, que contempla aplicações com restrições temporais como automação industrial, cujos nós precisam realizar suas tarefas dentro de prazos determinados, para que os sistemas funcionem corretamente. O não cumprimento dos prazos na realização das tarefas pode acarretar nas mais diversas consequências, trazendo ou não grandes prejuízos para as aplicações. Assim, é importante implementar mecanismos nos nós sensores para que esses sejam capazes de lidar corretamente com as restrições temporais das aplicações.

1.1 Motivação

Na literatura científica estão disponíveis diversos trabalhos cujo objetivo é realizar o processamento de consultas em nós sensores, preparando-os para processar primitivas de comunicação e configuração pré-estabelecidas entre os nós e os usuários.

Em Madden *et al.* (2005) é proposto um sistema processador de consultas, denominado TinyDB, para RSSF cujos nós utilizam o sistema operacional TinyOS (TINYOS, 2011). Esse sistema considera a rede como uma tabela de dados, na qual cada coluna é um parâmetro sensoriado pelos nós e os dados coletados pelos sensores correspondem às tuplas da tabela. Os nós sensores estão organizados em uma estrutura hierárquica em que um nó é a raíz da rede. Esse nó é pai de alguns outros nós que, por sua vez, também são pais de outros nós e assim por diante. A comunicação entre os nós acontece de nó pai para filho e vice-versa. Por meio de um computador conectado à rede, o usuário submete as consultas utilizando a linguagem declarativa TinySQL. A consulta é disseminada seguindo a hierarquia da rede, sendo enviada de nó pai para nó filho. As respostas à consulta são roteadas de nó filho para nó pai até que são colocadas em uma tabela virtual presente no computador e exibidas ao usuário. Nesse caso, tabela virtual significa que todas as linhas e colunas não estão de fato previamente geradas, sendo criadas à medida em que as respostas vão chegando.

No trabalho de Curino *et al.* (2005) é proposto o processador de consultas TinyLime, desenvolvido para nós sensores do tipo Crossbow Motes (CROSSBOW, 2011). Esse processador considera que os nós são parte de um modelo de memória compartilhada. Juntamente com os nós sensores, estão presentes na rede diversas estações base, que gerenciam as solicitações de consultas. Quando um nó é detectado pela estação base, esse passa a representar uma tupla da memória compartilhada da estação. Para submeter uma consulta, os usuários devem escrever

um aplicativo, baseado em uma API (*Application Programming Interface*) disponibilizada pelo projeto. Quando uma consulta é submetida à rede, essa é interpretada e traduzida em pacotes que podem ser entendidos pelos nós sensores. Os nós sensores atuam reagindo aos pacotes que chegam com solicitações de dados, enviando respostas às estações base. As respostas que chegam nas estações base são colocadas na memória compartilhada e a tupla solicitada na consulta é roteada pela rede até ser exibida ao usuário.

Na pesquisa de Meira (2007) é desenvolvido um processador distribuído de consultas, o PDCRS (Processamento e Distribuição de Consultas em Redes de Sensores). Este processador atua através de cinco etapas: interpretação, decomposição, distribuição das subconsultas, processamento das subconsultas e pós-processamento. Por meio de uma interface, o usuário submete consultas utilizando a linguagem SNQL (*Sensor Network Query Language*). A consulta é tratada, dando início a etapa de interpretação, seguindo as regras gramaticais da linguagem de consulta. Em seguida, a consulta é decomposta (fase de decomposição) de acordo com a localização dos sensores e os tipos desses. Após a decomposição, a consulta é disseminada pela rede, considerando uma estrutura hierárquica de nós pais e filhos. Esse processo corresponde a fase de distribuição das subconsultas. Ao receberem a consulta, os nós sensores a processam, extraindo respostas dos seus dados coletados, caracterizando a fase de processamento das subconsultas. Por fim, as respostas são enviadas para o processador de consultas, que as prepara para serem exibidas ao usuário. Esse processo caracteriza a etapa de pós-processamento.

Em Watfa *et al.* (2009) é proposto o processador de consultas Corona. Este processador tem como principal objetivo reduzir o número de pacotes de dados enviados pelos nós sensores, de modo a conservar os recursos energéticos desses. O trabalho considera que para realizar a comunicação os nós seguem uma hierarquia, na qual existe um nó raíz. Este nó é pai de outros nós que, por sua vez, também são pais de outros nós e assim por diante. Quando consultas são submetidas na rede, essas passam de nó pai para nó filho. As respostas às consultas, no entanto, não são simplesmente roteadas pelo caminho inverso. Essas sofrem um processo de agregação, no qual cada nó pai agrega as respostas de seus nós filhos à sua própria resposta, diminuindo o número de pacotes enviados. O Corona possui ainda um gerenciamento de consultas em que quando se tem consultas similares, apenas uma consulta é enviada para a rede e a resposta é considerada válida para todas as outras consultas. A economia de energia é então proveniente da agregação de dados e do gerenciamento de consultas.

Outro processador é apresentado em Choi e Chung (2011). Denominado Request, este processador de consultas agrupa os nós tomando como base regiões de localização. Cada agrupamento de nós possui um nó líder. As consultas são submetidas na rede sendo roteadas de nó em nó até que toda a rede seja abrangida. Os nós processam as consultas e enviam a resposta para o nó líder de seu grupo. Cada nó líder, por sua vez, deve aguardar por um tempo determinado que as respostas de seus filhos cheguem. Após a espera, o nó realiza agregação de dados e envia um pacote de dados em resposta à consulta, que será roteado até ser exibido ao usuário.

Nos trabalhos citados, o principal objetivo é possibilitar que os nós sensores sejam preparados para processar consultas provenientes de usuários, configurando, mesmo depois da RSSF já implantada, o comportamento dos nós. Essas pesquisas, no entanto, não são adequadas para atender aplicações com restrições de tempo, pois não fornecem mecanismos para gerenciar as consultas de modo a buscar que prazos temporais sejam alcançados.

Aplicações que possuem restrições de tempo necessitam adquirir os dados oriundos da RSSF quando esses ainda refletem o estado atual do ambiente. Mensagens de nós sensores atrasadas podem não ser consideradas aceitáveis (PAGANO *et al.*, 2010). Um exemplo de tal situação são sensores utilizados para o monitoramento do trânsito. Quando os nós detectam que um veículo cometeu alguma infração do tipo avançar o sinal vermelho ou parar na faixa de pedestres, uma câmera deve ser acionada para que seja tirada uma foto da placa do veículo infrator. Os dados dos sensores devem, então, chegar até ao aplicativo que aciona a câmera antes do veículo deixar a área monitorada, para que se consiga obter uma foto da placa.

No intuito de garantir que os dados dos sensores estejam sempre disponíveis em tempo hábil para que as aplicações possam tomar decisões, algumas soluções vêm sendo propostas. Em Soyturk e Altilar (2008) é sugerido um protocolo baseado em construções de rotas por demanda e prioridades para suportar o tráfego em tempo real. Em Borges Neto (2009) é desenvolvido um protocolo sensível ao tempo com o intuito de realizar o roteamento dos dados garantindo as restrições temporais das aplicações. Em Phong *et al.* (2010) é proposto um protocolo que busca o caminho menos demorado para rotear os dados. Em Szalapski e Madria (2010) é apresentado um esquema de compressão de dados que considera restrições de tempo, de forma a não inserir atrasos na rede.

As soluções apresentadas, no entanto, buscam tratar a restrição de tempo considerando o percurso do dado de um nó sensor até o usuário. Considerando as possíveis aplicações para as RSSF, é preciso ponderar sobre a demanda de consultas que um nó pode necessitar atender. Em Lu *et al.* (2010), por exemplo, sugere-se utilizar RSSF para prover serviços de múltiplas aplicações compartilhadas executando nos mesmos instantes de tempo. Dessa forma, tratar atrasos somente na rede como um todo pode não atender restrições de tempo das aplicações. Uma vez que não se sabe a demanda de consultas aos nós, é possível que, quando esses consigam enviar pacotes de resposta a uma solicitação, os dados não sejam mais válidos dada a restrição temporal da aplicação. Tal fato implica na necessidade de também tratar restrições de tempo considerando nós individualmente, na qual o nó deve buscar garantir restrições de tempo, tolerando sobrecargas de consultas. Essa necessidade é ilustrada em Conti *et al.* (2011), na qual a contenção de dados nos nós é apontado como um fator problema para se conseguir atender as restrições de tempo.

Visando os nós individualmente, é proposto em Chipara *et al.* (2007) o uso de prioridade de pacotes para escalonar as transmissões dos nós sensores de modo a possibilitar o atendi-

mento de restrições de tempo. Para tal, usuários submetem solicitações à rede para que os nós retornem dados periodicamente. Essas consultas determinam a prioridade da solicitação, o tempo de início da coleta e o instante que a solicitação deve ser encerrada. A pesquisa investiga o uso de três abordagens de escalonamento, com preempção, sem preempção e uma abordagem que mistura preempção e não-preempção. Na abordagem com preempção, os nós que receberem solicitações com prioridades maiores que a da solicitação que está sendo executada, deve interromper a execução e atender a solicitação de maior prioridade. Na abordagem sem preempção, uma solicitação com maior prioridade do que a solicitação que está sendo executada deve esperar o término da execução para ser atendida. E na abordagem que mescla preempção e não-preempção, solicitações de dados com prioridades maiores que a solicitação que está sendo executada devem aguardar o término da execução, a menos que seu prazo esteja para esgotar.

O trabalho proposto em Chipara *et al.* (2007) é a pesquisa mais próxima que trata de forma conjunta o processamento de consultas e o tratamento de restrições temporais nos nós sensores. No entanto, nesse estudo as consultas não são realizadas através de linguagens de consultas, nem possibilitam modificar quando, como ou quão frequentemente os nós devem coletar e enviar dados. A rede, no caso, atua somente como uma coletora de dados, em que todos os nós devem reportar dados e o usuário pode somente determinar quando deseja que isso aconteça. Tal abordagem é de grande utilidade quando se deseja obter dados de todos os nós, bem como quando pretende-se utilizar os dados para realizar estudos estatísticos e analisar tendências.

Nesta dissertação, é considerado o uso de linguagem de consultas para determinar o comportamento dos nós quanto a coleta e envio de dados, considerando ainda que as transmissões dos pacotes devem buscar que as restrições de tempo das aplicações sejam atendidas.

1.2 Objetivo

1.2.1 Objetivo Geral

Define-se como objetivo deste trabalho o processamento de consultas em nós sensores, considerando que as aplicações atendidas possuem restrições de tempo. Assim, além da necessidade de manter a consistência lógica inerente de qualquer aplicação, considera-se que dados coletados devem sempre representar o estado atual do ambiente, possuindo validade temporal, e as transações que operam sobre esses dados devem ser escalonadas objetivando atender restrições de tempo. Para tal, visa-se utilizar uma linguagem de consulta de modo a estabelecer uma interface de comunicação com os nós sensores e, para assegurar que as demandas de consultas sejam atendidas, visa-se utilizar funções de gerenciamento de dados e transações de tempo real, que tenham por objetivo garantir as restrições lógicas e temporais.

1.2.2 Objetivos Específicos

Define-se como objetivos específicos deste trabalho:

- Seleção de uma linguagem de consulta para estabelecer uma interface de comunicação com os nós sensores;
- Seleção de uma estratégia de gerenciamento de dados e transações considerando restrições de tempo;
- Implementação do processamento de consultas em nós sensores considerando restrições de tempo.

1.3 Metodologia

A metodologia utilizada para o desenvolvimento desta pesquisa consiste nas seguintes etapas:

- Revisão bibliográfica:
  Nessa etapa foi realizada uma revisão bibliográfica acerca das RSSF, no que se refere ao armazenamento e consulta de dados na rede. Foi também realizado um estudo sobre linguagens de consulta para aplicações que utilizam RSSF, buscando trabalhos já desenvolvidos. Ainda nesse momento, foi realizado um estudo sobre estratégias de gerenciamento de dados e transações de tempo real.
- Seleção da linguagem de consulta:
  Nesse período foi definida a linguagem de consulta a ser utilizada na pesquisa. Tal escolha baseou-se na adaptação da linguagem para as mais diversas aplicações, permitindo modificar comportamentos de coleta e envio de dados nos nós sensores.
- Seleção da estratégia de gerenciamento de dados e transações:
  Nessa fase foi definida a estratégia para gerenciamento de dados e transações com restrições de tempo. Tal escolha baseou-se na adequação dessa para aplicações que envolvem ambientes dinâmicos como as RSSF.
- Implementação do processamento de consultas:
  Nessa etapa foi realizado um estudo acerca de ferramentas de simulação para RSSF. Foram ainda implementados algoritmos para a criação de uma rede, na ferramenta escolhida, cujos nós possuissem mecanismos para responder consultas, bem como gerenciá-las objetivando que os dados retornados sejam considerados temporalmente válidos e as restrições de tempo das aplicações possam ser atendidas.

- Validação do processamento de consultas:
  Nessa fase foram realizadas simulações do processamento de consultas nos nós sensores, utilizando a ferramenta de simulação definida na etapa anterior, no intuito de avaliar o processamento de consultas.

1.4 Organização da Dissertação

Esta dissertação está estruturada conforme os seguintes Capítulos:

- Capítulo 2:
  No Capítulo 2 as RSSF são definidas e são apresentadas características dessas redes. Também são mostradas as abordagens presentes na literatura no tocante ao armazenamento e consulta dos dados nos nós sensores.

- Capítulo 3:
  No Capítulo 3 são conceituadas as linguagens de consulta de dados, sendo expostas características dessas. Além disso, são apresentados exemplos de linguagens de consulta desenvolvidas para atender aplicações que fazem uso das RSSF.

- Capítulo 4:
  No Capítulo 4 são exibidas as características dos dados e transações de aplicações que envolvem restrições de tempo, bem como são mostradas técnicas presentes na literatura para o gerenciamento desses.

- Capítulo 5:
  No Capítulo 5 é apresentada a implementação do processamento de consultas nos nós sensores. Tal implementação tem como principal característica permitir o acesso aos dados coletados pelos nós, gerenciando acessos para atender as necessidades de tempo das aplicações. Para tal, são definidas uma linguagem de consulta e uma estratégia de gerenciamento de dados e transações de tempo real a serem utilizadas. Nesse Capítulo são ainda realizadas simulações para avaliar o processamento de consultas.

- Capítulo 6:
  No Capítulo 6 são apresentadas as conclusões do trabalho e possíveis trabalhos futuros.

2 Redes de Sensores sem Fio

Redes de Sensores sem Fio consistem em um conjunto de nós autônomos, espacialmente distribuídos, que trabalham de forma cooperativa para monitorar uma região obtendo dados acerca dessa (YICK *et al.*, 2008). De acordo com Domingues *et al.* (2011), essas redes podem ser facilmente implantadas em diversas áreas para monitorar objetos alvo e coletar informações sobre ambientes, sendo possível embutir nós sensores até em objetos de uso diário como cadeiras e relógios. Na Figura 1 são apresentados diversos domínios em que as RSSF podem ser utilizadas com sucesso. Dentre os mais diversos domínios, aponta-se a defesa militar, indústria e agricultura, monitoramento ambiental e o campo do auxílio a desastres como os meios em que RSSF são amplamente utilizadas (WAN *et al.*, 2010).

Figura 1: Áreas que utilizam Redes de Sensores sem Fio

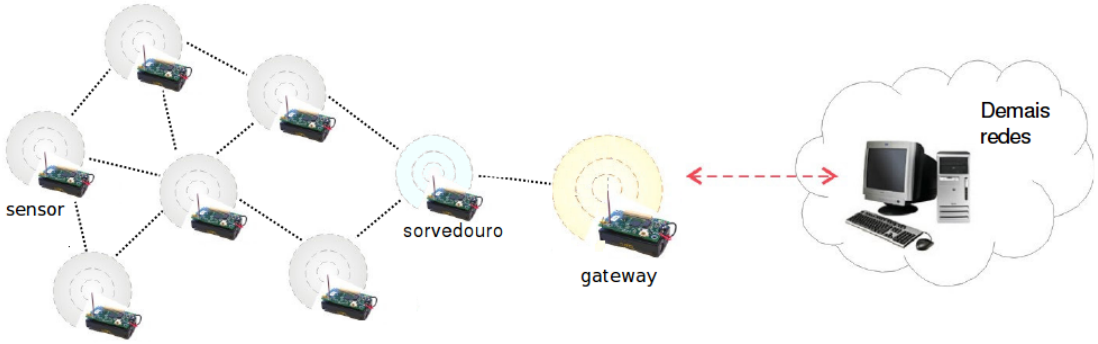
Adaptado de Brito (2009).

Os nós podem ser classificados de distintas maneiras de acordo com a função desses na rede, sendo típicos nós sensores, sorvedouros e *gateway*. Os típicos nós sensores atuam coletando dados sobre características físicas do ambiente. Assim, esses devem registrar dados, enviar seus próprios dados para nós vizinhos e repassar adiante pacotes de dados provenientes de nós da sua vizinhança. Os nós sorvedouros caracterizam-se por receber dados provenientes de outros nós sensores, atuando como um ponto de coleta. É comum que nesses nós seja realizada agregação de dados. Os nós do tipo *gateway* são aqueles que atuam como ponto de ligação entre a RSSF e outras redes (ou outros equipamentos), possibilitando os dados coletados chegarem a algum usuário final. Um mesmo nó pode atuar como sorvedouro e *gateway* ao mesmo tempo.

Na Figura 2, uma RSSF típica é ilustrada. Os nós comunicam-se entre si utilizando enlaces sem fio e os dados são roteados até o nó sorvedouro, que realiza a agregação. O nó sorvedouro

envia, então, o dado para o nó *gateway*. Esse nó estabelece a comunicação entre a RSSF e demais redes, de modo que os dados possam ser enviados a algum usuário.

Figura 2: Uma Rede de Sensores sem Fio genérica

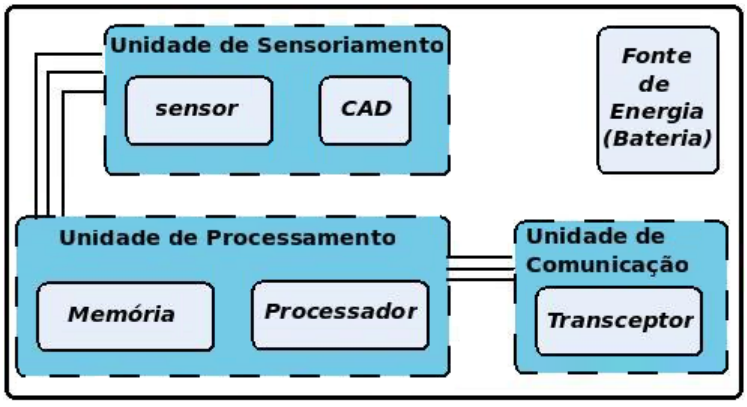


Adaptado de Lima (2008).

2.1 Nó Sensor

O nó sensor é um dispositivo autônomo, com capacidade de sensoriar propriedades físicas do ambiente, armazenar, processar e realizar comunicação sem fio com outros dispositivos. Devido a capacidade de processamento, os nós sensores são comumente chamados de nós sensores inteligentes. De forma geral, a arquitetura de um nó sensor é constituída por quatro unidades básicas: sensoriamento, processamento, comunicação e energia (RIBEIRO NETO *et al.*, 2008; VERDONE *et al.*, 2008, p. 5). Essa arquitetura é ilustrada na Figura 3.

Figura 3: Arquitetura básica de um nó sensor



Adaptado de Ribeiro Neto *et al.* (2008).

A unidade de sensoriamento é constituída por um ou mais tipos de sensores, tais como sensor de pressão, luminosidade, temperatura e presença. Os sensores são dispositivos que respondem a mudanças físicas do ambiente, produzindo um sinal analógico como resposta. Além dos sensores, a unidade também possui um conversor analógico digital (CAD), que converte os

sinais analógicos provenientes da resposta do sensor a estímulos físicos para sinais digitais que poderão ser posteriormente processados.

A unidade de processamento é formada, principalmente, por uma memória e um processador. Na memória ficam armazenados dados coletados e programas que determinam o comportamento do dispositivo. Dentre esses programas, dependendo do dispositivo a ser utilizado, pode até ser incluído um sistema operacional. Já o processador tem como função a execução de tarefas. Segundo Delicato (2005), essa unidade é responsável pela execução de protocolos de comunicação, funções de controle dos nós sensores e gerência dos procedimentos necessários para que os nós possam atuar de forma colaborativa.

A unidade de comunicação é responsável pela transmissão de dados entre os nós, sendo composta por um dispositivo transceptor. Tal dispositivo combina um transmissor e um receptor, possibilitando que os nós sensores enviem e recebam pacotes dos nós vizinhos. A vizinhança de um nó é conhecida por meio dos sinais que esse consegue captar proveniente dos outros nós localizados nas proximidades. A comunicação entre nós pode acontecer utilizando sinais ópticos, infravermelhos, radiofrequência, entre outros.

A unidade de energia é composta por uma bateria que alimenta o nó sensor.

2.2 Características das Redes de Sensores sem Fio

Uma RSSF representa o desafio da criação de um sistema distribuído de grande escala, fortemente inserido no mundo físico, com severas restrições de recursos e que seja capaz de operar por longos períodos mantendo um desempenho (TSIATSIS *et al.*, 2005). Tal desafio resulta em diversas características peculiares das RSSF no que se refere a tolerância a falhas, eficiência energética, topologia, escalabilidade, autonomia, latência e precisão, tarefas colaborativas e dependência da aplicação.

As RSSF podem ser implantadas nos mais diversos ambientes, o que inclui áreas em que os nós podem facilmente se tornar inoperantes devido a danos físicos, sofrer constantes interferências na comunicação ou, ainda, serem furtados. Tais falhas não devem afetar a tarefa global de monitoramento da rede e, portanto, protocolos e algoritmos devem ser implementados para alcançar tolerância a falhas de acordo com as características das aplicações (RIBEIRO, 2010). Em Cassimiro Neto (2010), essa dependência da aplicação, no tocante a tolerância a falhas, é ilustrada com a comparação entre uma RSSF implantada em um campo de batalha e uma residência, no qual a tolerância deve ser maior no cenário do campo de batalha, em que a probabilidade de falhas é maior, do que no de automação residencial.

Os nós sensores são alimentados através de uma bateria com capacidade restrita e finita, assim, uma vez a energia do nó esgotada, esse deixa de compor a rede. Quando a RSSF é implantada em um área de difícil acesso ou possui muitos nós a formando, pode não ser viável

que os nós que deixam de compor a rede sejam substituídos. As RSSF devem, portanto, operar otimizando o uso dos recursos energéticos, aumentando o tempo de vida útil dos dispositivos. De acordo com Lopes (2007), o projeto de uma RSSF deve ser conduzido considerando severas restrições de energia, na qual recomenda-se que as tarefas ou dispositivos mais dispendiosos energeticamente devem ser considerados apenas em momentos oportunos.

A topologia das RSSF caracteriza-se pela dinamicidade, pois mesmo que a rede seja implantada seguindo uma determinada topologia, a perda e/ou locomoção dos nós faz com que essa possa ser modificada constantemente. Outro fator que também influencia na topologia são os nós que, durante períodos de baixa atividade, ficam inoperantes temporariamente (RIBEIRO, 2007). Para lidar com a dinamicidade da topologia, a RSSF deve ser capaz de organizar a si mesma, adaptando-se às mudanças estruturais sem nenhuma intervenção humana.

A característica da escalabilidade em uma RSSF consiste na capacidade de se manter operando, com um determinado desempenho, mesmo com o acréscimo de novos nós. Segundo Ribeiro (2010), de acordo com a aplicação, o número de nós que compõem a rede pode chegar a ordem de milhares ou milhões e os protocolos e soluções implementados devem suportar esse número elevado de nós.

Ao se implantar uma RSSF, os nós devem ser programados de modo que sejam capazes de atuar sem a necessidade de operadores humanos, tornando a rede autônoma e capaz de, além de realizar as tarefas de monitoramento, lidar com falhas, mudanças de topologia, entre outros. Os usuários, por sua vez, após a implantação da rede precisam somente transformar os dados em informação útil, embora possam modificar comportamentos dos nós através de consultas (LOPES, 2007).

A característica de latência refere-se ao tempo percorrido entre o envio do pacote até que esse chegue ao destino final, ou seja, o tempo necessário para a rede notificar aos usuários a ocorrência de determinado fenômeno. Já a precisão determina o quão fiel à realidade é o dado coletado. Assim, os protocolos necessitam fazer com que o menor número possível de pacotes de dados enviados seja perdido, aumentando a precisão dos dados, e esses pacotes cheguem ao destino dentro de um intervalo de tempo em que o usuário final ainda mantém interesse em ser notificado acerca de determinado fenômeno.

O monitoramento de uma área é realizado por meio do trabalho em conjunto dos nós sensores, no qual cada um deles deve ser programado para realizar tarefas colaborativas. Assim, cada nó monitora uma determinada área e compartilha sua coleta com os outros nós que, por sua vez, realizam o mesmo comportamento, possibilitanto o monitoramento de toda extensão da área que a RSSF deve contemplar. Alguns nós podem monitorar a mesma área, nesse caso, tem-se redundância de dados.

Os nós de uma RSSF têm as funcionalidades implementadas de acordo com uma aplicação específica, ou seja, uma rede não necessariamente é capaz de atender uma outra aplicação para

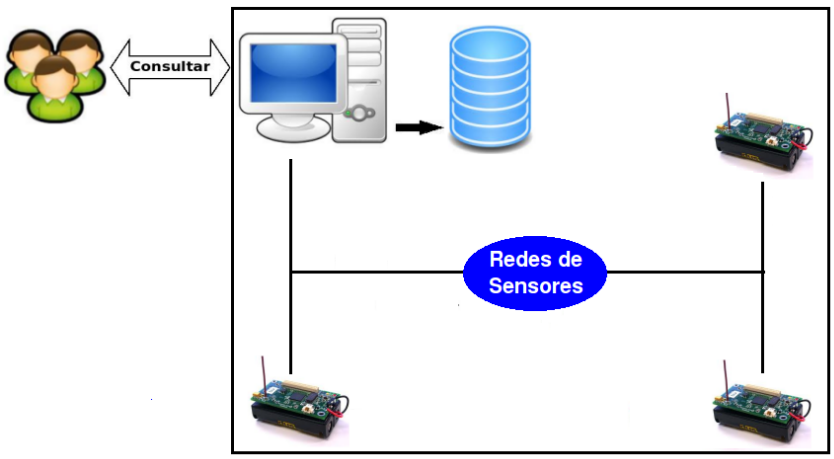
a qual ela não foi estruturada. As características do domínio da aplicação são os fatores determinantes para os requisitos que os nós devem atender. Por exemplo, uma aplicação de monitoramento do trânsito pode necessitar que os nós enviem dados em um curto espaço de tempo, com precisão e sem grandes preocupações com o gasto de energia. Já uma aplicação de monitoramento florestal pode requisitar que a economia de energia seja uma prioridade e os nós realizem o máximo de agregação de dados possível. Para Lopes (2007), a aplicação influencia nas funções exercidas pelos nós sensores, na quantidade de nós que devem compor a rede, na disposição dos nós no ambiente, nos protocolos de comunicação que serão utilizados, no tipo de dados a ser tratado e, até mesmo, no tempo de vida útil da rede.

2.3 Abordagens de Armazenamento e Consulta dos Dados

Os nós sensores trabalham de forma colaborativa para obter dados acerca de um ambiente. A obtenção desses dados pode ser realizada de distintas maneiras, com cada um dos nós da rede sendo pré-programado para enviar coletas de dados e repassar pacotes de nós vizinhos até que esses cheguem ao *gateway* ou por meio de consultas, em que é possível determinar que somente alguns nós retornem pacotes de dados e como e quando fazê-los. Segundo Chagas *et al.* (2010), essas duas abordagens são denominadas *warehousing* e distribuída, respectivamente.

A abordagem *warehousing* considera que as RSSF atuam apenas como coletoras de dados. Nesse caso, cada um dos nós ativos na rede enviam seus dados para que sejam roteados e cheguem até um servidor de banco de dados externo à rede. As consultas aos dados são realizadas no servidor. Essa abordagem é ilustrada na Figura 4.

Figura 4: Abordagem *warehousing*



Adaptado de Ribeiro Neto (2006a).

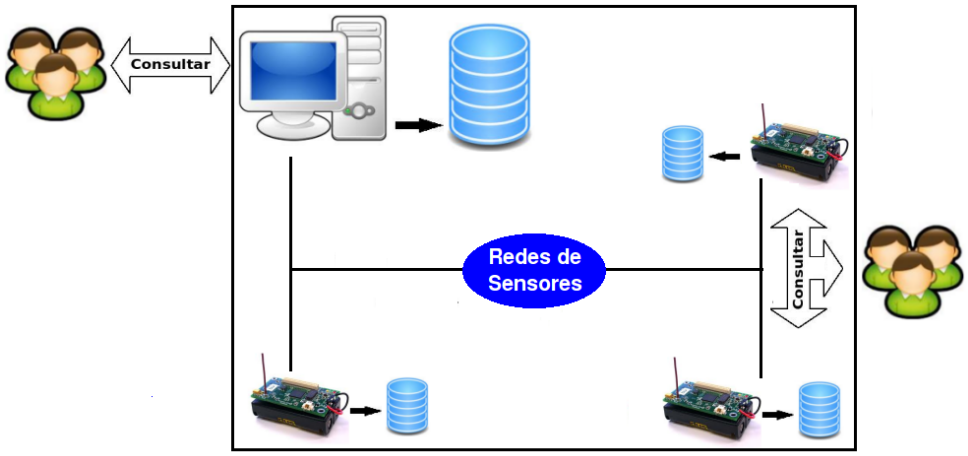
Os nós podem ser previamente programados para, periodicamente, enviar coletas de dados, como também podem ser programados para responder a requisições de dados provenientes de usuários. Uma vez que todos os nós ativos enviam suas coletas, essa estratégia pode levar

ao desperdício de recursos dos nós por transferir grandes quantidades de dados que nem sempre são relevantes, fazendo com que os nós realizem mais processamentos e comunicações e, consequentemente, gastem mais energia.

O problema do desperdício de recursos pode ser tratado adotando mecanismos como em Domingues *et al.* (2011). Nessa pesquisa, os nós sensores enviam pacotes de dados toda vez que um usuário requisita. No entanto, quando mais de um usuário solicita dados, em curtos intervalos de tempo entre solicitações cujos parâmetros físicos não apresentam grande variação na escala de minutos ou segundos, somente uma solicitação é enviada à rede. A resposta a essa solicitação, por sua vez, é também enviada às solicitações equivalentes. No intuito de economizar ainda mais recursos energéticos, funções de agregação são implementadas em todos os nós, reduzindo a quantidade de pacotes enviada na rede.

A abordagem distribuída considera a RSSF sobre dois enfoques: como uma coletora de dados, que possui um banco de dados externo à rede, e com os nós sensores atuando como BD locais. Nesse caso, é possível realizar consultas tanto no servidor de banco de dados como também nos próprios nós que compõem a rede. Essa abordagem possibilita submeter consultas nas RSSF de modo que somente um determinado grupo de nós retornem os dados coletados. Por exemplo, em uma RSSF implantada em uma floresta cujos nós detectam a temperatura local e o próprio posicionamento, é possível solicitar que somente os nós que em seu último sensoriamento verificaram uma temperatura acima de determinada medida retornem a consulta, identificando áreas com maiores chances de incêndio. A abordagem distribuída é ilustrada na Figura 5.

Figura 5: Abordagem distribuída



Adaptado de Ribeiro Neto (2006a).

É possível verificar na Figura 5 que os usuários podem direcionar as consultas aos dados tanto para o servidor de banco de dados quanto para os nós da rede. Para que os nós sensores sejam capazes de responder às consultas, esses são pré-programados com funções para interpretar cláusulas de uma linguagem de consulta previamente determinada.

A abordagem distribuída explora a capacidade de processamento dos nós sensores e permite a interação entre o usuário final e a RSSF. Essa interação pode ser bastante vantajosa na otimização da energia dos nós, uma vez que é possível limitar a quantidade de nós enviando pacotes. Além disso, determinar as consultas na rede pode atender melhor as necessidades das aplicações, já que essas são elaboradas de acordo com os dados que se deseja adquirir.

Além das características citadas, outro fator importante é que a abordagem distribuída permite obter dados levando em consideração períodos de tempo e eventos. No caso, pode-se solicitar que os nós sensores coletem dados somente durante determinado espaço de tempo ou, ainda, que comecem a coletar dados somente a partir de um determinado instante de tempo definido. É possível também determinar que os nós modifiquem períodos de coleta, quantidade de dados enviados, dentre outros, na ocorrência de algum determinado evento.

2.4 Considerações Finais

As RSSF são compostas por nós que realizam o monitoramento de determinada área respondendo a estímulos físicos do ambiente, coletando dados e enviando-os para outros nós através de enlaces sem fio. Os pacotes de dados são roteados pela rede até chegar a um usuário final interessado no fenômeno que está sendo monitorado. Os nós são programados para atuar de forma autônoma, dispensando operadores humanos e possibilitando um monitoramento remoto.

O projeto de uma RSSF deve ser conduzido considerando diversas características. De forma geral, a rede deve ser capaz de lidar com o acréscimo ou perda de nós e mudanças de topologia mantendo um certo desempenho e os nós devem ser programados para operar de forma colaborativa com outros nós, além de buscar sempre manter uma eficiência energética em suas atividades. Como serão programadas as funções exercidas pelos nós e quais protocolos de comunicação serão utilizados é determinado pelas características da aplicação para a qual a rede foi implantada.

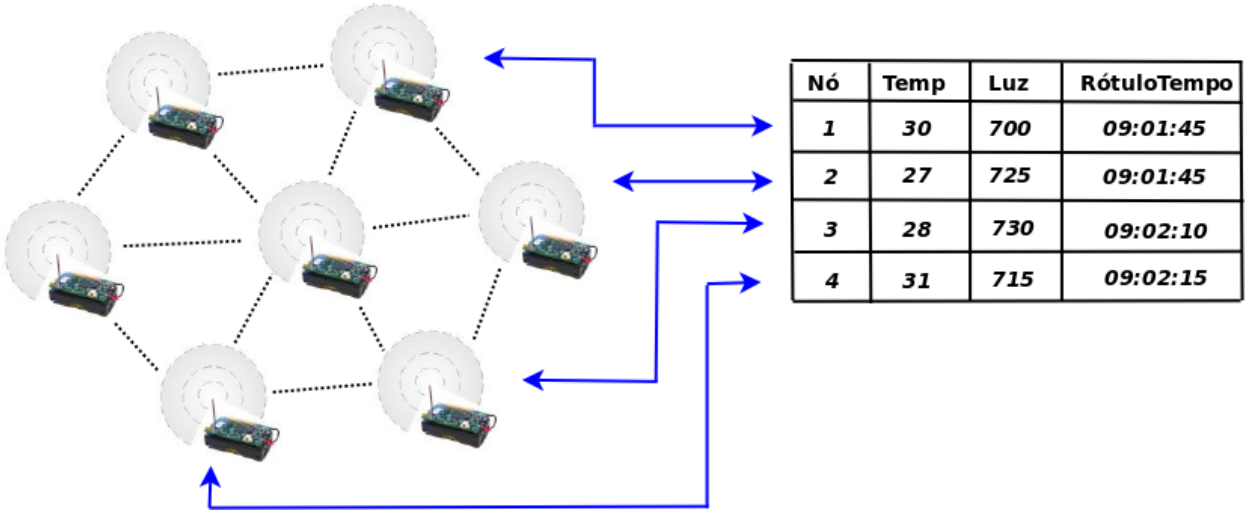
Além das características citadas, é preciso ainda considerar como os nós devem atuar no tocante à coleta e envio dos dados. Essas atividades podem ser realizadas por meio de duas abordagens. A primeira considera a RSSF como uma coletora de dados, na qual cada um dos nós deve enviar pacotes de dados, que serão armazenados e disponibilizados para consulta em um servidor de dados externo à rede. A segunda considera que os nós também podem atuar como banco de dados. Nesse caso, esses são preparados para processar consultas e os usuários podem consultar dados nos próprios nós ou, ainda, pode-se utilizar um servidor de dados.

Uma vez que é possível programar os nós considerando diferentes necessidades das aplicações e que esses nós podem sensoriar diferentes parâmetros físicos do ambiente, as RSSF se apresentam como uma importante ferramenta de monitoramento, que pode ser utilizada nas mais diversas aplicações.

3 Linguagem de Consulta

Segundo Nakamura (2007), as RSSF podem ser modeladas como Banco de Dados (BD), na qual cada nó sensor possui uma parte da informação, e o acesso aos dados pode ser realizado através de consultas declarativas. Um exemplo de tal situação é ilustrado na Figura 6. No caso, a RSSF é vista como uma tabela de dados em que cada tipo de dado possível de ser gerado representa as colunas dessa tabela, e os dados que são coletados por cada nó compõem as tuplas. A realização de consultas se faz através de uma linguagem de consulta. Por meio dessa, é possível estabelecer uma comunicação com o BD, na qual os usuários interessados nos dados são capazes de determinar quais são os objetos armazenados de interesse. As solicitações dos usuários, por sua vez, podem ser compreendidas já que são baseadas em regras gramaticais previamente estabelecidas.

Figura 6: RSSF como um banco de dados



A partir das linguagens de consulta é possível determinar, de forma geral, duas abordagens de consultas em RSSF, consultas para obter dados históricos e consultas para obter dados mais atuais. As consultas escritas para a recuperação de dados históricos remetem a dados cujo instante de aquisição é superior a um determinado intervalo de tempo e, portanto, não representam o estado mais imediato do ambiente. Os parâmetros para definir se os dados já podem ser considerados históricos ou não depende das características de cada aplicação. Em uma RSSF cujos nós coletam dados acerca do tráfego de determinada via pode-se considerar que os dados se tornam mais rapidamente desatualizados do que em uma RSSF cujos nós coletam dados para um sistema de monitoramento de danos em estruturas. Para realizar consultas visando dados históricos, pode-se considerar que os dados são armazenados em um banco de dados externo à rede ou ainda que os nós sensores armazenam uma determinada quantidade de dados, englobando os dados mais recentes e históricos. Esse tipo de consulta é importante para aplicações que necessitam gerar dados estatísticos, investigar padrões incomuns ou analisar tendências

históricas. Segundo Ai (2010), são exemplos de consulta de dados históricos a análise da temperatura máxima, mínima e média de dois meses atrás ou ainda os horários de maior movimento de uma rodovia em determinadas épocas do ano.

As consultas para a recuperação de dados mais atuais consideram os dados que do momento da aquisição até o momento em que a consulta está sendo realizada há somente um breve espaço de tempo. Tais consultas, portanto, devem ser processadas nos próprios nós sensores. Exemplos de aplicações em que os dados mais atuais necessitam ser consultados são sistemas de controle de processos de produção. Nessas aplicações, decisões são tomadas a partir dos dados coletados e, portanto, esses devem representar o estado mais atual do ambiente.

De acordo com Nam e Li (2010), é possível categorizar as consultas no contexto das RSSF em quatro tipos, históricas, instantâneas, longas e disparadas por eventos.

As consultas históricas recuperam dados que não são mais considerados atuais. Em geral, esses dados estão armazenados em bancos de dados externos à rede. Um exemplo de tal consulta é recuperar a média de precipitação chuvosa do mês anterior.

As consultas longas são aquelas processadas no intuito de obter continuamente os dados coletados durante um determinado intervalo de tempo. Um exemplo de tal consulta é a coleta e envio de dados da temperatura a cada hora durante uma semana.

As consultas instantâneas são processadas no intuito de obter o valor do dado mais atual armazenado no nó sensor naquele instante de tempo. Um exemplo de tal consulta seria a recuperação de dados para obter o número de veículos que já passaram em uma rodovia até determinado instante de tempo.

As consultas disparadas por eventos determinam que os dados devem ser recuperados na ocorrência de determinado evento. Um exemplo de tal consulta seria o envio periódico de dados da temperatura na detecção de um princípio de incêndio.

As linguagens de consulta suportam definição, manipulação e controle de dados, para tal, são divididas em duas partes: Linguagem de Definição de Dados (DDL, do inglês *Data Definition Language*) e Linguagem de Manipulação de Dados (DML, do inglês *Data Manipulation Language*).

3.1 Linguagem de Definição de Dados

A DDL consiste na parte da linguagem de consulta responsável por definir o esquema do BD, ou seja, quais estruturas de dados irão compor o repositório. De acordo com Silberschatz *et al.* (2006), a DDL é definida como o conjunto de instruções que determinam a estrutura de armazenamento do BD e as operações de acesso a essas estruturas. Através da DDL é possível também determinar restrições de integridade e domínio de valores dos dados, de modo que

quando um novo valor é inserido no BD, ele deve contemplar essas restrições. Um exemplo de restrição seria determinar o conjunto de valores que um dado pode assumir como sempre um valor maior ou igual a zero.

Em Leite (2005) são apontados como exemplos de comandos da DDL operações de criação e remoção de tabelas ou campos de dados, como também comandos para alterar as estruturas das tabelas já existentes no BD. A utilização da DDL ocorre com o recebimento de algumas instruções como entrada e a geração de alguma saída com base nessas instruções, por exemplo, a exclusão de uma tabela do BD (SILBERSCHATZ *et al.*, 2006).

No que se refere a RSSF, a DDL possibilita modelar a rede como um banco de dados, definindo como será a função dos nós sensores no repositório de dados. Por exemplo, pode-se considerar que as coletas dos nós sensores correspondem a tuplas de uma tabela e os parâmetros sensoriados correspondem as colunas. Pode-se ainda considerar que a RSSF seja vista como um conjunto de tabelas, uma para cada parâmetro sensoriado, e cada um dos nós corresponda a colunas.

3.2 Linguagem de Manipulação de Dados

A DML refere-se à parte da linguagem de consulta que atua na manipulação dos dados armazenados no BD, ou seja, através da DML o usuário pode acessar os dados ou realizar modificações nesses. Enquanto que a DDL permite manipular a estrutura do BD, a DML permite manipular os dados em si. Segundo Silberschatz *et al.* (2006), são exemplos de comandos da DML a recuperação, exclusão e modificação de dados armazenados no BD, bem como a inserção de novos dados no BD.

Segundo Silberschatz *et al.* (2006), as DML podem ser divididas em duas categorias: procedurais e declarativas. As procedurais requerem que o usuário determine quais dados deseja acessar e como obtê-los. Já as declarativas requerem apenas que o usuário indique quais dados deseja acessar, dispensando a forma de obtê-los.

No tocante a RSSF, a DML irá definir cláusulas de acesso a dados para serem utilizadas na comunicação entre as aplicações e os nós sensores.

3.3 Exemplos de Linguagens de Consulta para Redes de Sensores sem Fio

É possível encontrar na literatura exemplos de linguagens de consulta desenvolvidas para aplicações que fazem uso das RSSF. Em Madden *et al.* (2005) foi proposta a linguagem TinySQL. Essa linguagem foi desenvolvida no intuito de processar consultas nos nós sensores no contexto do projeto TinyDB. Baseando-se na SQL (*Structured Query Language*) (SQL, 2012), a TinySQL consiste em um conjunto de expressões para agregar dados em resposta a

uma consulta, predicados para filtrar dados e um conjunto de atributos. A DDL dessa linguagem considera que os dados podem assumir somente quatro tipos: luz, temperatura, umidade e nid. Este último determina um identificador único para o nó sensor.

No contexto da DML, a TinySQL considera que todas as consultas aos dados seguem o formato:

Select ... From ... Where ... Group By

A cláusula *Select* especifica quais atributos deverão ser retornados. É possível também utilizar a cláusula com uma expressão de agregação, em vez de um atributo. A TinySQL suporta as funções de agregação *count*, que computa o número de linhas retornadas; *sum*, que indica o somatório dos valores de determinado atributo; *max*, que fornece o maior valor de determinado atributo; *min*, que indica o menor valor de determinado atributo; e *avg*, que fornece a média de determinado atributo.

A cláusula *From* determina de qual estrutura de armazenamento os dados serão extraídos. No caso do projeto TinyDB, considera-se a RSSF como uma grande tabela denominada Sensors, sendo essa utilizada como parâmetro da cláusula *From*. É possível também considerar os dados somente de uma coluna da tabela Sensors. Nesse caso, essa coluna é indicada na cláusula *From*.

A cláusula *Where* faz o papel de filtrar dados, estabelecendo uma condição que os dados retornados devem obedecer.

Já *Group By* é uma cláusula opcional e agrupa as tuplas com base nos valores de determinado atributo.

Considerando uma RSSF implantada em um prédio, em que nós sensores estão espalhados pelos diversos andares, um exemplo de consulta utilizando a TinySQL seria:

```
SELECT AVG(temperatura), quarto FROM Sensors
WHERE andar = 6
GROUP BY quarto
```

nesse caso, de todos os nós da rede somente os nós presentes no sexto andar devem retornar a média da temperatura e o quarto em que se encontram. Os dados provenientes da resposta dessa consulta serão agrupados pelo atributo quarto.

A TinySQL diponibiliza ainda outras cláusulas para a DML, são elas *Sample Period*, *For*, *Stop Query*, *Create Storage Point ... Size ... As*, *On Event*, *Stop On Event*, *Query Lifetime* e *Output-Action*.

A cláusula *Sample Period* determina a periodicidade em que os nós devem retornar leituras e pode ser utilizada em conjunto com a cláusula *For*. Esta, por sua vez, limita o tempo de

execução da consulta. Por exemplo, "*Sample Period* 1s *For* 10s" significa que cada nó deve retornar leituras a cada segundo durante 10 segundos.

A clásula *Stop Query* finaliza uma consulta em execução. Quando consultas são enviadas aos nós sensores, essas possuem um identificador único. Dessa forma, a cláusula *Stop Query* recebe como parâmetro o identificador da consulta, determinando a finalização dessa. Por exemplo, "*Stop Query* id" irá finalizar a consulta com o identificador id.

A cláusula *Create Storage Point ... Size ... As* cria um ponto de materialização de dados dentro da RSSF. Por exemplo,

```
CREATE STORAGE POINT luzesMaisAtuais SIZE 8 AS(
  SELECT idNo, luz FROM Sensors
  SAMPLE PERIOD 10s)
```

cria o ponto de materialização "luzesMaisAtuais" de tamanho 8 que conterá os atributos identificador do nó e a leitura sobre a quantidade de iluminação a cada 10 segundos. O tamanho oito significa que só serão armazenados até oito registros.

A cláusula *On Event* detecta a ocorrência de algum evento. Tal evento deve estar previamente programado em cada nó sensor. Por exemplo,

```
ON EVENT deteccao-passaro(loc):
SELECT AVG(luz), AVG(temp)
FROM Sensors
SAMPLE PERIOD 2s FOR 30s
```

nesse caso, na ocorrência do evento "detetcção de um pássaro", na localização "loc", o nó em que o evento foi disparado deverá retornar a média da luz e temperatura a cada dois segundos durante trinta segundos.

A cláusula *Stop On Event* determina que uma consulta seja finalizada na detecção de determinado evento. Por exemplo,

```
STOP ON EVENT id
WHERE temp>25
```

irá finalizar a consulta com identificador id caso o nó detecte que o valor da temperatura está acima de 25 graus.

A cláusula *Query Lifetime* determina o tempo de atividade de uma consulta. Esse tempo pode ser expresso em dias, semanas ou meses. Por exemplo,

```
SELECT temp
FROM Sensors
QUERY LIFETIME 30 dias
```

indica que a consulta deve ficar ativa na rede durante trinta dias.

A cláusula *Output-Action* determina a realização de alguma ação em função de algum acontecimento. Por exemplo,

```
SELECT idNo, temp
FROM Sensors
WHERE temp>25
OUTPUT ACTION power-off(idNo)
```

nesse caso, os nós que detectarem uma temperatura maior que 25 graus devem realizar a ação de ficar no modo desligado.

Em Meira (2007) foi proposta a linguagem SNQL. Esta trata-se de uma extensão da linguagem SQL, que adiciona novas cláusulas no intuito de atender o processamento de consultas nas RSSF no contexto do projeto PDCRS. No caso da DDL, as cláusulas da SQL foram adaptadas, sendo geradas as cláusulas *Create Schema*, *Use*, *Drop Schema*, *Create Sensors* e *Drop Sensors*.

A cláusula *Create Schema* determina qual estrutura do banco será utilizada, considerando que a RSSF é modelada como um BD. Essa estrutura é determinada em arquivos, no formato de um arquivo XML (*Extensible Markup Language*), que disponibilizam um catálogo global da rede com todas as categorias de sensores que devem ser consideradas.

A cláusula *Use* indica qual esquema para indicar a estrutura do banco será utilizado. Ao se escolher um esquema, os atributos dos nós sensores ficam restritos às categorias definidas no catálogo.

A cláusula *Drop Schema* exclui um arquivo de esquema existente.

A cláusula *Create Sensors* cria uma categoria de sensores, na qual os nós da rede farão parte. Essa determina os atributos presentes no banco e os tipos de dados que esses podem assumir. Por exemplo,

```
CREATE SENSORS temperatura (
   localizacao integer,
   valor float,
   tempoColeta time);
```

No caso dessa consulta, é criada uma categoria de sensores denominada temperatura, que irá conter os parâmetros localização do nó, que só pode assumir valores inteiros; valor coletado, que pode assumir qualquer valor real; e o instante de tempo em que a coleta foi realizada, que só pode assumir valores que indiquem tempo.

A cláusula *Drop Sensors* exclui uma categoria criada.

No contexto da DML, as consultas da SNQL seguem o padrão SQL com as cláusulas do tipo

Select ... From ... Where ... Group By

Assim como a TinySQL, a cláusula *Select* pode ser utilizada com as funções de agregação *Count*, *Sum*, *Max*, *Min* e *Avg*. Além disso, foram adicionadas à DML as cláusulas *Time Window*, *Send Interval*, *Sense Interval*, *Schedule* e *Data Window*.

A cláusula *Time Window* determina quanto tempo uma consulta deve permanecer ativa na rede. A resposta a uma consulta só deve ser exibida ao usuário após todos os nós enviarem pacotes de dados. Como o processador de consultas PDCRS não considera sincronismo entre os nós, o tempo que a consulta deve estar ativa pode variar de um nó para o outro, já que o nó começa a contabilizar o tempo após receber a solicitação e cada nó a recebe em momentos distintos. Assim, a cláusula *Time Window* só é considerada pela dispositivo que estabelece a interface entre a rede e o usuário, que após o tempo determinado pela cláusula, encerra a consulta, mesmo que nem todos os nós tenham enviado respostas a mesma.

A *Send Interval* estabelece um intervalo de tempo que os nós devem aguardar entre envios dos dados coletados.

As cláusulas *Sense Interval* e *Schedule* determinam o intervalo de tempo que os nós devem considerar entre coletas de dados e a periodicidade em que uma consulta deve ser submetida à rede, respectivamente.

As cláusulas *Time Window*, *Send Interval*, *Sense Interval* e *Schedule* podem ser expressas em milisegundos, segundos, minutos, horas, dias, meses e anos.

A *Data Window* indica o máximo de coletas de dados que deve ser realizada pelos nós sensores.

Um exemplo de consulta utilizando a SQNL é ilustrado a seguir:

SELECT localizacao, valor
FROM temperatura
WHERE valor>30
SENSE INTERVAL 1 *min*
SEND INTERVAL 5 *min*
TIME WINDOW 1 *dia*
SCHEDULE 10

nessa consulta, os nós devem retornar sua localização e o valor do atributo “temperatura”, quando o valor detectado for acima de 30 graus. A coleta de dados deve ser realizada a cada um minuto e dados devem ser enviados a cada cinco minutos. Essa consulta deve ser encerrada após passado um dia de submissão e deve ser submetida à rede dez vezes.

Em Changbai *et al.* (2008) foi proposta a QLSND (*Query Language for Sensor Network Databases*). Esta linguagem foi criada no intuito de permitir eficiência na coleta de dados cujos

atributos assumem valores diferentes com bastante frequência e economizar energia em RSSF maciçamente grandes. No contexto da DML, as consultas consideram o formato:

Select ... From ... Where

Além dessas cláusulas, a QLSND também disponibiliza as cláusulas *On Event*, *Sample Period*, *For*, *Within* e *Case*.

A cláusula *On Event* determina que a consulta será disparada com o acontecimento de determinado evento, previamente conhecido pelo nó sensor.

A cláusula *Sample Period* define o período de coleta de dados dos nós sensores. Essa é utilizada juntamente com a cláusula *For*, que limita o tempo em que os nós devem coletar dados.

A cláusula *Within* determina o número de nós que irão participar da resposta à consulta. Essa cláusula controla a qualidade dos dados e a eficiência energética. Os nós processam a cláusula *Within* para determinar se irão participar da consulta. Essa determinação é baseada em um número randômico e uma função de utilidade. O nó gera um valor randômico entre 0 e 1 e, se esse valor for maior que o valor da função de utilidade, o nó responderá a consulta. A função de utilidade é definida por

$$f_{(x)} = \frac{p}{1 - p \times (1 - p) \times r}$$

onde p é definido por $\frac{pnr}{100}$, sendo pnr a porcentagem de nós que irão participar da consulta, e r é o número de vezes em que o nó não participou consecutivamente de consultas. Um exemplo de consulta com a cláusula *Within* é ilustrado a seguir

```
SELECT AVG(temp)
FROM Sensors
WHERE temp>25
WITHIN 50
```

no caso, o valor cinquenta determina a quantidade de nós, em porcentagem, que deve participar da consulta.

A cláusula *Case* permite a consulta adaptar-se à mudança do ambiente, a qual pode ter comportamentos alternativos variando período de amostragem, precisão dos dados, entre outros. Por exemplo,

```
SELECT temperatura
FROM Sensores
SAMPLE PERIOD 3h
CASE
   WHEN deteccaoFogo(loc) THEN
      SAMPLE PERIOD 6m FOR 1h
   WHEN temperatura > 40 THEN
      SAMPLE PERIOD 2h
END
```

nessa consulta, os nós inicialmente irão coletar dados da temperatura durante o período de três horas. Caso um nó detecte fogo em sua localização, o período de amostragem deve ser modificado para coletas a cada seis minutos durante uma hora. Já se um nó detectar o valor da temperatura acima de 40 graus, o período de coleta de dados deve ser modificado para duas horas.

No contexto da DDL, a linguagem não disponibiliza nenhuma cláusula.

As cláusulas disponibilizadas por cada linguagem de consulta são apresentadas na Tabela 1.

Linguagem de Consulta	**DDL**	**DML**
TinySQL	Somente define os tipos de dados: luz, temperatura, umidade e identificador único	Select - From - Where - Group By Sample Period For Stop Query Create Storage Point ... Size ... As On Event Stop On Event Query Lifetime Output-Action
Sensor Network Query Language	Create Schema Drop Schema Create Sensors Drop Sensors Use	Select - From - Where - Group By Time Window Data Window Send Interval Sense Interval Schedule
Query Language for Sensor Network Databases	Não disponibiliza cláusulas	Select - From - Where On Event Sample Period Within Case

Tabela 1: Cláusulas das linguagens de consulta

3.4 Considerações Finais

As linguagens de consulta permitem estabelecer uma comunicação com os nós das RSSF, possibilitando os usuários submeterem consultas aos próprios nós da rede. Para tal, os nós são preparados para processar consultas, que possuem regras gramaticais determinadas a priori.

Para ter regras gramaticais bem definidas, as linguagens de consulta se dividem em duas partes, Linguagem de Definição de Dados e Linguagem de Manipulação de Dados. A primeira determina a estrutura do banco de dados, definindo os atributos que irão compor o banco e restrições sobre esses. A segunda determina as operações de manipulação de dados.

Há na literatura exemplos de linguagens de consultas criadas para aplicações que fazem uso de RSSF. Para o uso dessas linguagens, considera-se a rede como um banco de dados. Assim, as linguagens permitem definir como a rede deve ser organizada quanto aos dados, quais atributos devem ser coletados, bem como as operações disponíveis para acessá-los.

As linguagens de consultas concedem ao usuário a flexibilidade de modificar o comportamento dos nós em relação a coleta e envio de dados, vinculando essas atividades de acordo com intervalos de tempo, restrições de valores de dados ou o acontecimento de eventos.

4 Gerenciamento de Dados e Transações de Tempo Real

À medida que sistemas computacionais proliferaram na sociedade, aplicações com comportamentos definidos segundo restrições de tempo tornaram-se cada vez mais comuns (FARINES *et al.*, 2000). De acordo com Mohamad *et al.* (2011), essas aplicações estão presentes em diversas atividades da sociedade, variando de processos industriais à automação residencial. Tais aplicações fazem uso de dados que devem refletir o estado atual de determinado ambiente. Dada a dinamicidade do mundo real, no entanto, não é possível garantir que, após certo tempo depois de serem adquiridos, os dados ainda representem as características atuais do ambiente. Uma vez que os dados são coletados em momentos discretos de tempo, à medida que o tempo avança continuamente, os dados vão se tornando cada vez menos precisos, até que não correspondam mais aos valores dos dados respectivos no meio em que foram coletados. São exemplos dessas aplicações sistemas de controle aéreo, redes de sensores, comércio eletrônico e sistemas de telefonia (IDOUDI *et al.*, 2009; CHAGAS; RIBEIRO NETO, 2010).

Nas aplicações com restrições de tempo, uma ação realizada muito tarde (ou muito cedo) ou um cálculo que utiliza dados temporalmente inválidos pode ser inútil e por vezes prejudiciais, mesmo se tal ação ou computação estiverem funcionalmente corretas (SHANKER *et al.*, 2008). Tais aplicações são denominadas Sistemas de Tempo Real (STR) e caracterizam-se por ter como principal requisito a necessidade de responder a eventos dentro de prazos pré-determinados (GUMZEJ; HALANG, 2010). Para tal, é necessário adotar estratégias que visem gerenciar o armazenamento e acesso dos dados, resultando no atendimento dos prazos temporais.

Segundo Krol e Pokorny (2006), o gerenciamento dos dados nos STR, tradicionalmente, era realizado utilizando bases individuais para cada tarefa da aplicação. Ainda de acordo com Krol e Pokorny (2006), o crescimento da complexidade dos STR criou a necessidade de gerenciar volumes de dados cada vez maiores, o que resultou no interesse de unir tais aplicações com a tecnologia de BD. Essa tecnologia permite a criação de um repositório de dados e, por meio de um sistema de gerenciamento, é realizada a inserção e recuperação desses. Para que sejam retornados dados corretos, que vão possibilitar gerar informações corretas para o usuário, é necessário preservar a consistência lógica dos dados. Ou seja, dados que atendem a certas regras pré-determinadas, por exemplo, um dado que não pode assumir um valor negativo, unicidade de registros, cópias de um mesmo dado com valores idênticos, entre outros.

No caso das aplicações com restrições de tempo, diferentemente dos BD tradicionais, os dados armazenados passam a ter a validade temporal como uma de suas propriedades. Dessa forma, é preciso manter os dados logicamente consistentes, bem como temporalmente consistentes. Para possibilitar que os dados possuam uma validade temporal, esses são associados a atributos de tempo. Uma vez que as aplicações necessitam ter acesso aos dados quando esses ainda são válidos temporalmente, as transações também possuem características temporais.

4.1 Características dos Dados

Devido à necessidade de refletir o estado atual do ambiente da aplicação, a estrutura dos dados indica que os valores gravados são válidos apenas por um determinado intervalo de tempo (LEITE, 2005). Quando o dado ainda reflete o estado atual do ambiente, diz-se que esse é temporalmente consistente (IDOUDI *et al.*, 2009). Para verificar se os dados são temporalmente consistentes, em cada um desses são definidos dois atributos: rótulo de tempo e intervalo de validade absoluta ou relativa. O rótulo de tempo consiste no instante em que o dado foi gravado no Banco de Dados de Tempo Real (BDTR). O intervalo de validade absoluta ou relativa indica o intervalo de tempo em que o dado ainda pode ser considerado temporalmente válido. Esse valor é determinado pela semântica dos dados no mundo real. Por exemplo, em um ambiente cujos parâmetros mudam muito constantemente, o intervalo de validade dos dados será menor do que em um ambiente cujos parâmetros não mudam de valor tão rapidamente.

De acordo com Ribeiro Neto (2006b), com base nos atributos de rótulo de tempo e intervalo de validade, a consistência temporal dos dados pode ser medida de duas formas:

- Consistência absoluta: é medida entre o valor real de um item de dado no ambiente da aplicação e o valor armazenado no BDTR que corresponde a esse. Considerando um item de dado x, tem-se que esse é temporalmente consistente se:

  $$(\text{tempo\_corrente} - \text{rótulo\_de\_tempo}_{(x)}) \leq \text{AVI}_{(x)}$$

  na qual tempo_corrente refere-se ao tempo atual do sistema, $\text{rótulo\_de\_tempo}_{(x)}$ refere-se ao tempo em que o dado x foi gravado no BDTR e $\text{AVI}_{(x)}$ refere-se ao intervalo de validade absoluta do item de dado x.

  Essa medida é proveniente da necessidade de sempre manter o BDTR consistente com o estado real do ambiente a cada momento.

- Consistência relativa: alguns dados podem ter seu valor derivado de outros itens de dados. Stankovic *et al.* (1999) ilustra como exemplo uma aplicação que faz uso dos valores de temperatura e pressão para tomar decisões sobre um processo químico. Nesse caso, a consistência é medida entre os dados bases que esses dados derivam. Por exemplo, um item de dado x, que é derivado de um conjunto de dados C=$\{$a, b$\}$, será temporalmente consistente se:

  $$(\text{rótulo\_de\_tempo}_{(a)} - \text{rótulo\_de\_tempo}_{(b)}) \leq \text{RVI}_{(x)}$$

  na qual $\text{rótulo\_de\_tempo}_{(a)}$ é o rótulo de tempo do item de dado a, $\text{rótulo\_de\_tempo}_{(b)}$ é o rótulo de tempo do dado b e $\text{RVI}_{(x)}$ é o intervalo de validade relativa do item de dado x.

  Essa medida surge da necessidade de assegurar que dados que serão usados na computação de outros dados representem aproximadamente o mesmo instante de tempo do ambiente. O valor aproximado é devido ao fato de que, não necessariamente, os dados possuirão o mesmo rótulo de tempo.

4.2 Características das Transações

As transações permitem a execução de ações sobre os repositórios de dados. No contexto de aplicações com restrições de tempo, as transações possuem atributos temporais associados a cada uma delas. Em Leite (2005), esses atributos são apresentados e consistem em tempo de liberação, tempo computacional, prazo máximo e periodicidade. O tempo de liberação trata-se do instante de tempo em que a transação se encontra pronta para executar. O tempo computacional indica o tempo necessário para a execução completa da transação. O prazo máximo (comumente chamado de *deadline*) determina o limite de tempo possível para que a transação seja executada. A periodicidade representa os períodos de tempo em que a transação é executada.

Os atributos das transações definem o comportamento temporal dessas e possibilitam tomar decisões acerca do gerenciamento das execuções das mesmas. Por exemplo, é possível definir que devem ser primeiramente executadas sempre as transações em que o prazo máximo está mais próximo do tempo corrente, mesmo que isso leve a interrupção da execução de outras transações.

Segundo Ribeiro Neto (2006b), as transações de tempo real podem ser classificadas quanto às restrições de tempo, aos padrões de chegada e ao tipo de acesso aos dados. Ainda de acordo com Ribeiro Neto (2006b), tais classificações são importantes para o gerenciamento dos dados e transações, pois podem definir prioridades, periodicidades e situações de conflito.

No que se refere às restrições de tempo, as transações são classificadas, considerando o prazo máximo de execução, em:

- Estritas: quando a transação necessita cumprir seu prazo máximo de execução, pois caso contrário, a perda do prazo é considerado uma falha fatal e pode provocar consequências catastróficas como a morte de um ser humano ou grandes perdas de dinheiro.
- Suaves: quando o não cumprimento do prazo máximo de execução da transação não leva a ocorrência de problemas sérios como as transações estritas, no entanto, diminui o desempenho do sistema.
- Firmes: quando o não cumprimento do prazo máximo de execução da transação não gera nenhum efeito. Nesse caso, quando perdem o prazo, as transações são abortadas, uma vez que manter a execução dessa não trará nenhum benefício ao sistema.

No tocante aos padrões de chegada, são considerados principalmente o tempo de liberação e a periodicidade, sendo as transações classificadas em:

- Periódicas: transações nas quais entre as execuções dessas existe um intervalo de tempo

regular. Assim, as transações são iniciadas em instantes de tempo específicos e devem ser executadas dentro de intervalos de tempo específicos.

- Aperiódicas: transações que não possuem intervalos regulares entre as execuções das mesmas. Assim, o tempo de início das transações é imprevisível. Devido a não garantia de previsibilidade, esse padrão de chegada não caracteriza transações do tipo estritas, pois não garante que sempre será realizado o cumprimento das restrições de tempo.

- Esporádicas: assim como as aperiódicas, as transações esporádicas também possuem tempo de início aleatórios. Nesse padrão de chegada, no entanto, tem-se como restrição a existência de um intervalo mínimo de tempo entre as execuções de uma mesma transação. Esse intervalo mínimo de tempo é necessário para garantir que as restrições de tempo sempre serão atendidas, possibilitando a existência de transações do tipo estritas.

Em relação ao tipo de acesso aos dados, as transações são classificadas, de acordo com as operações que realizam, em dois tipos:

- Escrita: transações que obtêm valores de algum atributo do ambiente e os escreve no BDTR. Essas transações são tipicamente periódicas. Em uma RSSF, por exemplo, pode-se determinar que os nós da rede obtenham e armazenem valores a cada determinado intervalo de tempo. No entanto, essas transações também podem ser esporádicas, como no caso em que se determine que um nó sensor obtenha e escreva o valor de um dado somente quando esse atender uma determinada restrição.

- Leitura: transações que lêem valores armazenados no BDTR. Essas transações são tipicamente aperiódicas, pois são, de maneira geral, provenientes dos usuários. No entanto, essas também podem ser periódicas, como no caso de uma RSSF cujos nós devem ler um determinado valor armazenado no banco e retorná-lo a cada determinado intervalo de tempo.

4.3 Estratégias para o Gerenciamento de Dados e Transações de Tempo Real

Aplicações que fazem uso de dados e transações com restrições de tempo precisam gerenciá-los de modo a permitir que diferentes transações executem simultaneamente e todas tenham as restrições atendidas. Tais restrições englobam propriedades temporais e lógicas dos dados e transações. De acordo com Kan e Ali (2009), há quatro restrições que devem ser consideradas, são elas consistência temporal das transações, consistência lógica das transações, consistência temporal dos dados e consistência lógica dos dados.

A consistência temporal das transações determina que essas tenham as propriedades temporais, como tempo de liberação e prazo máximo, sempre tratadas como restrições que não podem ser violadas, pois resultaria em tornar o BDTR inconsistente. A consistência temporal da transação é utilizada para determinar quando cada transação deve executar, de modo que a ordem de execução sempre obedeça as restrições temporais.

A consistência lógica das transações determina a ordem em que essas devem ser executadas levando em consideração o acesso aos dados. Os valores de dados resultantes da execução concorrente das transações devem ser sempre equivalentes aos dados que resultariam se cada transação tivesse sido executada individualmente.

A consistência temporal dos dados restringe a quantidade máxima de tempo que pode passar a partir do registro de um dado para esse ainda ser considerado como um valor que representa o estado atual do ambiente.

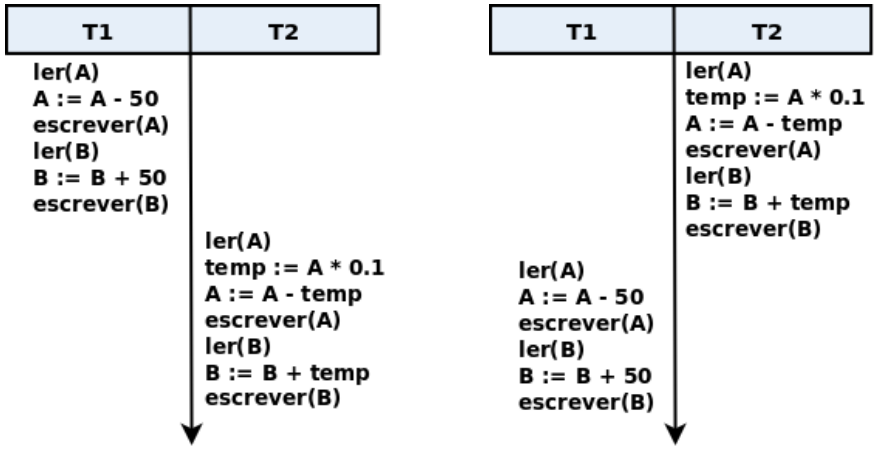
A consistência lógica dos dados determina as regras que devem ser consideradas para que a integridade dos dados sejam sempre mantidas, não permitindo contradições. Tais regras são determinadas de acordo com a aplicação para qual o BDTR foi desenvolvido. Em um sistema de compras, por exemplo, pode-se determinar que o dado que representa o valor total da compra nunca deve assumir um valor negativo.

Realizar o gerenciamento dos dados e transações considerando as restrições lógica e temporal desses pode levar a um impasse entre manter as restrições temporais e manter as restrições lógicas, na qual uma transação pode forçar uma outra transação a atrasar a execução. Por exemplo, para que um dado tenha a consistência temporal sempre mantida, é necessário atualizá-lo periodicamente. Isso significa que pode ser necessário interromper uma transação que esteja utilizando esse dado em favor de executar a transação de atualização. No entanto, interromper a transação pode acarretar na violação da consistência lógica do dado e da transação interrompida. Por outro lado, não interromper a transação pode levar à violação da consistência temporal do dado que não foi atualizado.

Além de considerar a disputa entre consistência lógica e temporal, o gerenciamento dos dados e transações deve ser realizado de forma que o resultado da execução simultânea das transações seja o mesmo resultado que seria obtido se essas tivessem sido executadas uma de cada vez. Para tal, o gerenciamento busca detectar situações de conflito entre as transações e adotar estratégias para gerenciá-las. De acordo com García-Molina *et al.* (2001), entende-se por conflito um par de ações consecutivas tal que, se a ordem da execução dessas ações for trocada, o comportamento de pelo menos uma das transações envolvidas poderá mudar, gerando valores inconsistentes. Por exemplo, considerando uma operação de leitura e escrita, e considerando o valor a ser escrito diferente do valor já gravado no banco, caso a operação de leitura seja executada antes da de escrita, o valor que essa lerá será diferente do valor que teria lido se a operação de escrita tivesse sido executada primeiro. Um exemplo de situação de conflito pode

ser visto na Figura 7.

Figura 7: Detecção de conflito entre transações



(a) Execução de T2 após T1 (b) Execução de T1 após T2

Adaptado de Silberschatz *et al.* (2006).

No caso da Figuras 7 (a) e 7 (b) há duas transações, T1 e T2. A transação T1 deve transferir R$ 50 de uma conta bancária A para uma conta bancária B. Já a transação T2 deve transferir 10% do valor presente na conta bancária A para a conta bancária B. Supondo que os valores nas contas A e B sejam R$ 1000 e R$ 2000, respectivamente, caso a transação T1 seja executada antes da transação T2, como ilustrado na Figura 7 (a), o valor final na conta A será R$ 855 e na conta B será R$ 2145. Caso seja executada a transação T2 antes da transação T1, como ilustrado na Figura 7 (b), o valor final na conta A será R$ 850 e o valor final na conta B será R$ 2150. As transações T1 e T2 são conflitantes, já que modificar a ordem de execução dessas resulta em modificar o valor final dos dados.

Quando duas transações podem executar concorrentemente, sem haver conflitos, diz-se que essas são serializáveis. Serialização é a noção padrão de corretude no processamento de transações (JUNG *et al.*, 2011). Algumas aplicações podem, ainda, fazer uso da exclusão mútua, na qual uma transação executa do início ao fim, não sendo possível ser interrompida por outras transações. Nesse caso, a concorrência não é permitida.

De forma geral, a detecção de conflitos entre as transações acontece de duas formas: pessimista ou otimista (ALI, 2006). A detecção de forma pessimista considera que sempre podem haver conflitos entre as transações e, portanto, quando uma transação requisita acesso a algum item de dado, primeiro é verificado se não haverão conflitos com nenhuma outra transação que esteja executando. Caso algum conflito seja detectado, a transação deve aguardar a execução das outras transações terminar ou ser abortada.

Um exemplo da detecção pessimista é apresentado em (NYSTROM, 2004 apud RIBEIRO NETO, 2006b, p. 6), denominada bloqueio em duas fases. Nessa técnica, como o próprio nome

já sugere, as execuções das transações são realizadas em duas fases. Na primeira, as transações que irão executar obtêm bloqueios sobre os itens de dados em que irão operar, evitando que outras transações operem sobre os mesmos itens. Nessa etapa, nenhum bloqueio adquirido pode ser liberado. O bloqueio pode ser realizado por meio de uma variável associada ao item de dado, cujo valor indica as possíveis operações que podem ser realizadas sobre o dado, por exemplo, uma transação de leitura que obtém um bloqueio sobre um dado pode permitir que outras transações de leitura também operem sobre o mesmo item de dado no mesmo instante de tempo. Quando uma transação não consegue obter bloqueios, essa deve aguardar até a liberação dos dados. Na segunda fase, a partir do momento em que as transações vão executando, os bloqueios vão sendo liberados. Enquanto as liberações acontecem, nenhum bloqueio pode ser adquirido pelas transações que estavam, ou ainda estão, executando. A técnica de bloqueio em duas fases garante a consistência lógica, porém diminui a concorrência entre as transações, limitando as chances de garantir a consistência temporal.

É possível encontrar na literatura algumas técnicas variantes do bloqueio em duas fases. Nas pesquisas de Kot (2008) e Zheng e Bi (2010), por exemplo, considera-se que as transações possuem prioridades e, após a fase de obtenção de bloqueios, caso uma transação requisite um bloqueio que foi adquirido por uma transação de menor prioridade, o bloqueio é concedido a essa transação, enquanto a transação de menor prioridade é reiniciada.

A detecção de forma otimista considera que conflitos são eventos raros e, portanto, a verificação se a execução concorrente das transações pode ser realizada é adiada até o momento final das execuções das transações. Dessa forma, todas as operações das transações vão sendo executadas. No caso de ter uma operação de escrita, essa não é realizada sobre o item de dado do banco imediatamente. Antes das transações encerrarem a execução, essas devem ser validadas. Nesse momento é que são verificados os conflitos e, caso não se tenha nenhum, as transações terminam a execução. No caso de haver algum conflito, é avaliado quais transações devem ser abortadas e quais podem finalizar a execução.

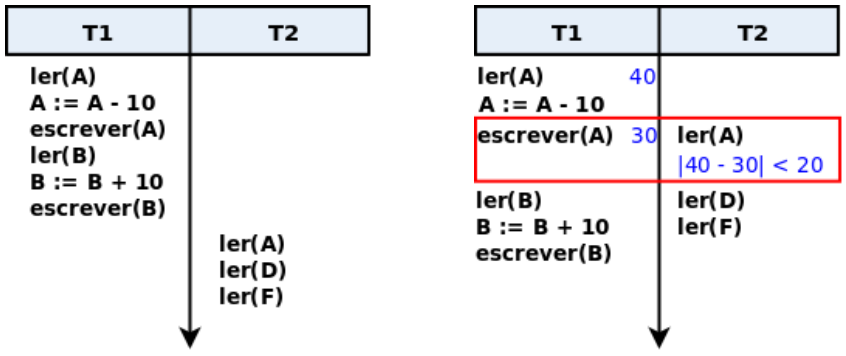
Um exemplo dessa abordagem é mostrada em Ali (2006), em que o gerenciamento das transações baseia-se em três fases: leitura, validação e escrita. Um gerenciador controla a execução das transações e estas possuem prioridades de acordo com o prazo máximo para execução. Na primeira fase, as transações começam a executar e, embora acessem os itens de dados do banco, as operações são realizadas em variáveis locais. Na segunda fase, as transações requisitam terminar a execução e é verificado se as operações não geraram inconsistências com outras transações. Na terceira fase, caso a verificação retorne verdadeiro, os valores são gravados no banco. Caso as operações gerem inconsistências, são abortadas as transações de menor prioridade em favor das transações de maior prioridade.

Outra abordagem para o gerenciamento das transações de tempo real baseia-se na negociação entre a consistência lógica e temporal dos dados e transações. Essa técnica foi in-

cialmente proposta por Dipippo (1995), denominada técnica de bloqueio semântico, e consiste no desenvolvimento de funções de compatibilidade que determinam quando é possível negociar consistência lógica em virtude de garantir a consistência temporal (e vice-versa) entre duas transações. Quando essa negociação é realizada, uma imprecisão é gerada no item de dado. De acordo com Fernandes (2005), o valor de um item de dado resultante da execução concorrente de transações é considerado impreciso se ele é diferente do valor correspondente resultante da não execução concorrente das transações.

A técnica de bloqueio semântico surgiu do fato de que nem todas as aplicações necessitam seguir rigorosamente o critério de corretude serialização, permitindo alguma imprecisão nos itens de dados. Segundo Keleher e Çetintemel (2004), a imprecisão é interessante quando os sistemas são de grande escala, possuem alta taxa de atualização e restrições de comunicação, pois nesses tipos de aplicações manter uma consistência de dados muito estreita não é vantajoso. Quando uma transação deseja operar sobre um item de dado, essa adquire um bloqueio semântico sobre esse. Caso uma outra transação deseje operar sobre o item de dado bloqueado, a função de compatibilidade é executada. Tal função determina se as transações podem executar concorrentemente ou se alguma delas deve ser abortada ou reiniciada. Um exemplo de tal situação pode ser visto na Figura 8.

Figura 8: Concorrência entre transações com negociação



(a) Execução sem concorrência (b) Execução com concorrência

No exemplo, duas transações desejam executar T1 e T2. A transação T1 tem como meta transferir R$ 10 da conta A para a conta B. Já a transação T2 tem como meta ler o valor armazenado nas contas A, D e F. Essas transações, em geral, não deveriam executar de forma concorrente, forçando uma transação a esperar pelo término da execução da outra transação, como na Figura 8 (a). No entanto, é possível considerar que com a transação T2 o aplicativo deseje apenas fazer uma estimativa do quanto está armazenado nas contas, aceitando uma imprecisão de R$ 20. Nesse caso, as transações podem executar concorrentemente, como mostra a Figura 8 (b). Quando as duas transações operam sobre o mesmo item de dado, é verificado que a imprecisão não ultrapassa o limite estabelecido, possibilitando as duas transações terminarem as execuções.

O bloqueio semântico é uma combinação das técnicas pessimista e otimista. Pode-se considerá-lo pessimista por fazer uso de bloqueios para resolver conflitos entre as transações. E pode-se considerá-lo otimista por possibilitar a execução de transações, a priori incompatíveis, simultaneamente. O critério de corretude para a execução concorrente das transações deixa de ser a serialização e passa a ser a serialização *epsilon*.

A serialização *epsilon* limita a possível divergência entre os dados que estão armazenados no banco de dados e o correspondente desses no ambiente. Essa limitação é estabelecida a partir do conhecimento que se tem acerca do domínio da aplicação para qual o BDTR foi desenvolvido. Com a permissão de haver imprecisão controlada no BDTR, mais concorrência entre as transações acontece, ampliando também a vazão dos dados e aumentando as chances de atender as restrições temporais (SOUSA *et al.*, 2004). Essa técnica é interessante para aplicações em que resultados aproximados obtidos logo são mais úteis do que resultados exatos obtidos posteriormente.

De acordo com Ribeiro Neto (2006b), tais técnicas, no entanto, não são totalmente adequadas para atender restrições de aplicações que executam em ambientes imprevisíveis como, por exemplo, aplicações que envolvem RSSF, pois não são capazes de atender várias aplicações com necessidades distintas. Dessa forma, é definido um modelo para auxiliar o desenvolvimento de aplicações que executam em ambientes imprevisíveis e necessitam gerenciar dados e transações com restrições de tempo. Para tal, são determinados um conjunto de funções e métricas que possibilitam atender diferentes necessidades das aplicações.

O modelo baseia-se na idéia de imprecisão, considerando que os valores dos itens de dados do mundo real não necessariamente precisam ser idênticos aos itens de dados correspondentes a esses armazenados no BDTR, sendo mais interessante obter um valor aproximado dentro do prazo de tempo do que o valor exato tempos depois. Em uma RSSF, por exemplo, que coleta dados acerca de um incêndio, é mais desejável ter uma idéia do estado do ambiente antes do que saber o estado exato depois. Assim, para o gerenciamento das transações são propostas Funções de Negociação para negociar entre a consistência lógica e temporal. Essas funções diferem das Funções de Compatibilidade por possibilitarem avaliar mais de duas transações concorrentes, além de possibilitar definir o limite de imprecisão de uma transação por meio de uma métrica.

A Função de Negociação visa permitir a execução de transações conflitantes concorrentemente, se a imprecisão dos dados estiver dentro de um limite aceitável, para que essas executem dentro de seus prazos. Dessa forma, a função define expressões que avaliam parâmetros de tempo dos dados e das transações e limites de imprecisão para determinar se é possível a execução concorrente de transações de leitura quando uma transação de escrita já estiver executando sobre o mesmo item de dado, transações de escrita quando uma transação de leitura já estiver executando sobre o mesmo item de dado e transações de escrita quando outra transação de escrita já estiver executando sobre o mesmo item de dado.

Além da Função de Negociação, são propostas também as Funções de Especificação, Mapeamento, Monitoração e Re-negociação. A Função de Especificação tem como papel determinar os parâmetros lógicos e de tempo dos dados e transações, tais como periodicidade de uma transação e imprecisão admitida em um item de dado. Alguns parâmetros são especificados a priori e outros em tempo de execução. A Função de Mapeamento mapeia os parâmetros definidos na Função de Especificação durante a execução das transações. A Função de Monitoração examina se os prazos das transações estão sendo atendidos e se as imprecisões admitidas nos dados não estão sendo excedidas. A Função de Re-negociação atua caso uma transação tenha seu prazo não atendido, verificando se essa pode ser reiniciada.

No tocante às métricas, duas são definidas: Número de transações que perderam o prazo (Pt) e Imprecisão máxima do dado (Impr). A primeira estabelece uma taxa de perda de prazos das transações que pode ser tolerada durante um determinado intervalo de tempo e define-se por:

$$\text{Pt} = 100 \text{ x } \frac{\text{Transações que perderam prazo}}{\text{Transações concluídas}}$$

Enquanto a segunda determina a imprecisão máxima que é possível admitir em um item de dado para que esse ainda posso ser considerado válido logicamente. Esta métrica é definida por:

$$\text{Impr} = \text{Valor Corrente x } \frac{\text{Imp}}{100}$$

onde Valor Corrente é o valor do item de dado armazenado no BDTR e Imp é a imprecisão admitida associada a esse item de dado.

O gerenciamento dos dados e transações em aplicações de tempo real são de grande importância, pois sem um plano estabelecido de como realizar a execução das transações e o acesso e escrita dos dados no BDTR, não é possível manter as restrições lógicas e temporais que essas aplicações necessitam.

4.4 Considerações Finais

Aplicações de tempo real caracterizam-se por, além da necessidade de se considerar resultados lógicos, serem considerados também resultados temporais. Nessas aplicações, as tarefas devem ser realizadas dentro de prazos específicos. Exemplos de aplicações que envolvem tempo são o controle de processos em fábricas, em que pode-se utilizar as RSSF como fonte coletora de dados.

Tais aplicações fazem uso de dados que devem refletir o estado mais atual do ambiente para o qual a aplicação foi desenvolvida. Para tal, os dados possuem características temporais, que

determinam quando esses foram coletados e até quando esses podem ser considerados válidos para representar o estado atual do ambiente. Para manter a consistência temporal dos dados, as transações também possuem propriedades temporais, que as caracterizam quanto a necessidade de atender as restrições de tempo, aos padrões de chegada e ao tipo de acesso aos dados.

Nas aplicações de tempo real é necessário gerenciar a execução das transações, determinando a ordem de execução dessas e se é possível realizar concorrência entre as execuções. Existe três abordagens principais para realizar tal gerenciamento, pessimista, otimista e bloqueio semântico. O uso de cada uma delas está vinculado às características das aplicações.

Através do gerenciamento de dados e transações, é possível adotar estratégias para realizar a execução das transações de forma a manter consistências lógicas dos dados e transações, bem como possibilitar manter as consistências temporais.

5 Processamento de Consultas em RSSF

O processamento de consultas em nós sensores consiste em preparar os nós para que atendam solicitações provenientes dos usuários. Para tal, é necessário a utilização de uma linguagem de consulta, de modo que os usuários elaborem consultas através dessa linguagem e os nós sensores tenham mecanismos implementados para tratar as cláusulas da linguagem, sendo capaz de responder a consulta proveniente do usuário. Uma vez que os nós podem receber mais de uma consulta ao mesmo tempo, esses devem possuir mecanismos para tratar a demanda de consultas, mantendo a consistência lógica.

Quando as aplicações atendidas pela RSSF caracterizam-se pela necessidade de realizar tarefas dentro de prazos específicos, os nós sensores devem ter mecanismos implementados para permitir a concorrência de consultas, aumentando a vazão dos dados e atendendo prazos. Tais mecanismos devem, além de manter a consistência lógica, manter também consistências temporais.

Para realizar a implementação do processamento de consultas em nós sensores foi utilizado o simulador OMNeT++ 4.1 (*Objective Modular Network Test-bed in C++*) (OMNET, 2011). Esse simulador, juntamente com o *Network Simulator*, são as ferramentas mais utilizadas para a simulação de redes em pesquisas acadêmicas (ALBERT, 2011). A escolha pelo OMNeT++ se deve ao fato de que esse é de fácil instalação, disponibiliza uma vasta documentação, além de um IDE (*Integrated Development Environment*) para o desenvolvimento da simulação. Além dessas considerações, o OMNeT++ possui mais recursos do que os demais simuladores (ALBERT, 2011).

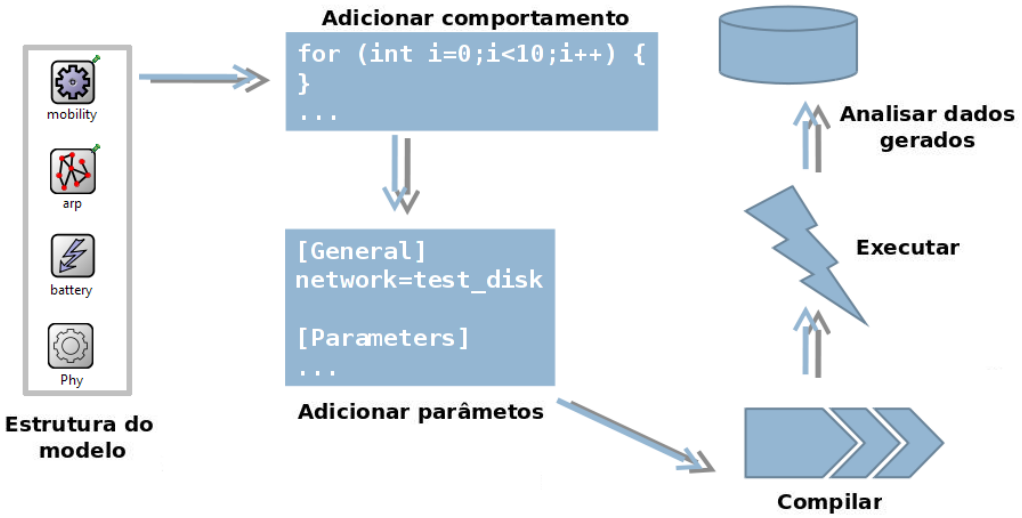
5.1 Simulador Omnet++

OMNeT++ é um simulador modular de eventos discretos, que possibilita a construção de *frameworks* sobre seu ambiente de simulação (OMNET, 2011). Os módulos determinam o comportamento de componentes do sistema simulado e são desenvolvidos utilizando a linguagem de programação C++. Um modelo de simulação no OMNeT++ é a junção de todos os módulos, que se unem de forma hierárquica, de modo que um módulo pode ser composto por outros módulos que, por sua vez, também podem ser compostos por outros módulos e assim por diante. A estrutura do modelo de simulação deve ser descrita usando a linguagem de descrição NED (NEtwork Description), disponibilizada pelo próprio simulador OMNeT++. Essa linguagem descreve quais módulos irão formar a rede, determinando a topologia dessa, e a estrutura dos próprios módulos, se esses são compostos por outros módulos ou não.

A construção do modelo de simulação ocorre com a definição da estrutura da rede e dos módulos, utilizando a linguagem NED. A esses módulos são adicionados comportamento

com códigos escritos com a linguagem C++. Em seguida, são adicionados parâmetros de configuração aos módulos como, por exemplo, a dimensão da área em que a rede será implantada ou o tempo total de simulação. A partir desse momento, o projeto de simulação é compilado e, se não houver erros, esse é executado. Dessa execução podem ser gerados dados para análise posterior. Essa dinâmica da construção do projeto de simulação é ilustrada na Figura 9.

Figura 9: Construção de um modelo de simulação no OMNeT++



Adaptado de Kang (2010).

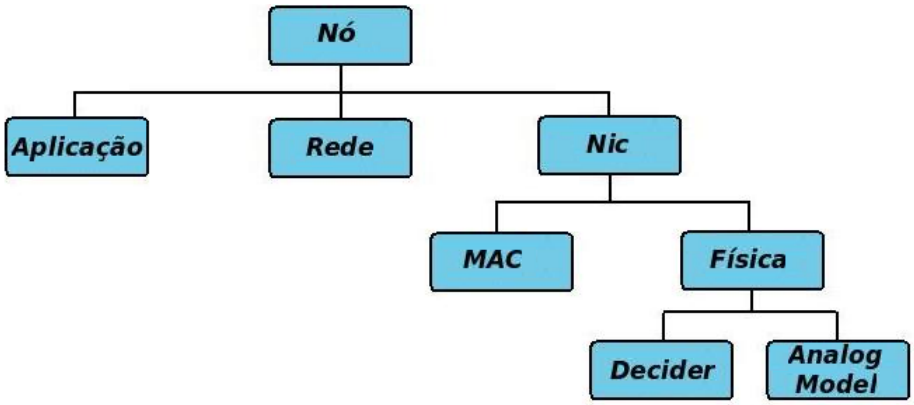
O OMNeT++ é bastante flexível para a construção de ambientes de simulação, não sendo direcionado a uma área específica do conhecimento, no entanto, esse é utilizado principalmente para a construção de *frameworks* de simulação de redes. O simulador já disponibiliza modelos de simulação para diversos tipos de redes. Dentre eles tem-se o Mixim (*Mixed Simulator*), que trata-se de um conjunto de *frameworks* desenvolvidos para a simulação de redes sem fio móveis ou estáticas (MIXIM, 2011).

O Mixim combina *frameworks* para prover suporte a mobilidade dos nós, gerenciamento de conexões entre nós vizinhos, modelos de propagação de rádio, além de uma biblioteca de protocolos de acesso ao meio (KÖPKE *et al.*, 2008). Nesse *framework*, o desenvolvimento das simulações segue um modelo em camadas, as quais possuem uma hierarquia, como ilustrado na Figura 10.

A camada de aplicação é responsável pelos programas que irão executar no nó, ou seja, quais tarefas esse nó necessita desempenhar para realizar o monitoramento de uma área e atender as necessidades de uma determinada aplicação.

A camada de rede é responsável pela definição de rotas para os pacotes que serão enviados a outros nós, ou seja, determina qual o nó que envia o pacote e qual nó será o destino do pacote. Entre esses dois nós, podem existir diversos nós intermediários.

Figura 10: Camadas de um nó sensor no Mixim



Adaptado de Kang (2010).

A camada Nic (*Network Interface Card*, em português, Cartão de Interface de Rede, ou ainda, Placa de Rede) é modelada com base em algum padrão de rede (*Bluetooth*, GSM, IEEE 802.11, entre outros) e é composta por duas outras camadas, a MAC e a Física.

A camada MAC (*Medium Access Control*, em português, Controle de Acesso ao Meio) determina qual e quando uma mensagem tem acesso ao meio para poder ser enviada, no intuito de evitar conflitos e colisões, além de determinar quando o rádio do sensor deve ficar ligado ou desligado, de modo a economizar energia do dispositivo.

A camada Física é responsável pela modelagem do meio físico, sendo dividida em duas outras camadas: *AnalogueModels* e *Decider*. A primeira tem como função calcular a atenuação de uma mensagem recebida, como sombreamento e enfraquecimento. Já a segunda é responsável por determinar se um sinal trata-se de uma mensagem ou de um ruído, realizar a demodulação de uma mensagem e ainda notitificar sobre o estado do meio físico para que a camada Física possa informar à camada MAC sobre a ocupação ou liberação desse.

As camadas são implementadas por meio de módulos e a comunicação entre esses é realizada por meio de mensagens. Essa comunicação é unidirecional e, portanto, cada módulo possui interfaces de comunicação de entrada e saída (denominadas *gates*), para o envio e recebimento das mensagens. A direção das mensagens é sempre para camadas logo acima ou abaixo da camada que está enviando.

Além dos módulos referentes às camadas citadas, o Mixim ainda provê os módulos *Mobility*, *Battery*, *Arp* e *Utility* para os nós sensores e os módulos *World Utility*, *Objects*, *ObjetsManager* e *ConnectionManager* para a rede como um todo. As funcionalidades de cada um desses módulos são descritas na Tabela 2.

Módulo	Funcionalidade
Mobility	Responsável pelo deslocamento dos nós
Battery	Modela a energia dos nós relacionando o processamento, deslocamento e comunicação com o consumo de energia
Arp	Modela o protocolo ARP (*Address Resolution Protocol*, em português, Protocolo de Resolução de Endereços)
Utility	Responsável por coletar dados estatísticos dos nós como, por exemplo, o nível de energia
World Utility	Modela a área em que a rede abrange
Objects	Modela o ambiente, criando objetos, em que a rede será implantanda
ObjectsManager	Gerencia os objetos do ambiente em que a rede está implantada
ConnectionManager	Gerencia dinamicamente as conexões entre os nós, criando e desfazendo conexões de acordo com a distância entre os nós ou o aparecimento de objetos entre eles

Tabela 2: Módulos do Mixim

5.2 Criação da Rede

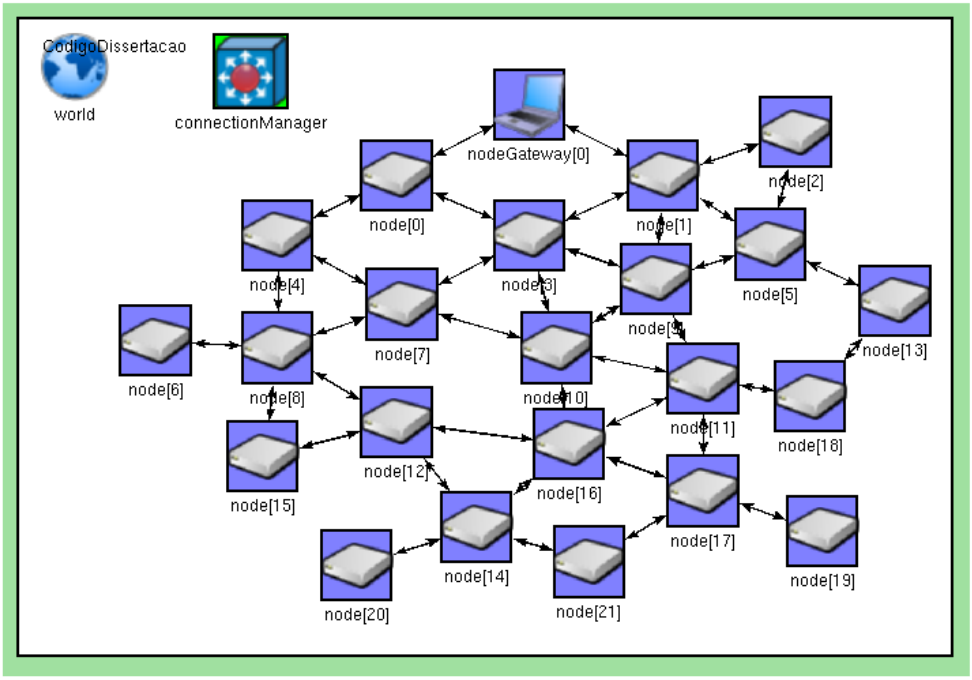
A RSSF definida nessa pesquisa para a implementação do processamento de consultas considera dois tipos de nós sensores, *gateway* e típico nó sensor (a partir de agora, esse será chamado apenas de nó sensor). Esses nós seguem a abordagem de armazenamento e consulta de dados distribuída, em que usuários podem submeter consultas aos nós da rede. Assim, o nó *gateway* é o responsável por permitir a comunicação entre os usuários e a rede, submetendo consultas para serem processadas pelos nós e recebendo as respostas desses. Os nós sensores são responsáveis pela coleta de dados, realizando o monitoramento de determinada área, e são preparados para processar as consultas enviadas pelo nó *gateway*.

As consultas são submetidas na rede para adquirir os dados mais atuais do ambiente e, portanto, não foi considerado um servidor de dados externo à rede. Também não foram considerados nós atuando somente como sorvedouros, pois embora possam ser realizadas agregações de dados na rede, quando essas acontecem, diversos nós distintos a realizam. A rede implementada é ilustrada na Figura 11. Nota-se a presença do nó *gateway*, bem como dos nós sensores.

Na Figura 11, verifica-se dois símbolos que indicam a presença dos módulos *World Utility* e *ConnectionManager* atuando no ambiente de implantação da rede. O primeiro foi utilizado para modelar a área que a rede abrange, enquanto o segundo é responsável por estabelecer as conexões entre os nós com base nas localizações desses. As conexões entre os nós são representadas por setas bidirecionais.

Para compor os nós sensores foram utilizados módulos para as camadas de aplicação, rede e Nic, além dos módulos *Arp* e *Battery*. A distinção entre os nós sensores e nó *gateway* se dá em nível de camada de aplicação. Uma vez que o nó *gateway* estabelece a interface entre o usuário e

Figura 11: RSSF no OMNet++



a rede, ele possui funções para atender a submissão de consultas aos nós, processar as respostas parciais oriundas dos nós e enviar respostas de uma consulta ao usuário. Já os nós sensores possuem funções para coletar dados acerca do ambiente, processar parâmetros de pacotes de consultas, enviar pacotes para responder consultas ou estabelecer rotas de comunicação com outros nós.

O Mixim não possui módulos específicos para modelar os dispositivos sensores dentro dos nós. Para suprir a necessidade de coleta de dados, foi desenvolvida uma função para gerar números aleatórios, dentro de intervalos específicos, representando os valores sensoriados pelos nós. Cada nó sensor, no modelo desenvolvido nesta pesquisa, possui a capacidade de sensoriar dois parâmetros, temperatura e luz. Assim, cada nó possui duas funções para gerar dados, uma para cada tipo de sensor definido.

5.3 Seleção da Linguagem de Consulta

A linguagem de consulta permite usuários estabelecerem uma comunicação com os nós da rede possibilitando até mesmo, caso seja de interesse, modificar o comportamento desses. Embora as linguagens de consultas pesquisadas na literatura científica tenham sido desenvolvidas no contexto de algum projeto específico de sistema processador de consulta em nós sensores como, por exemplo, o projeto PDCRS, é possível fazer uso dessas linguagens, considerando algumas ressalvas, em qualquer pesquisa.

A DDL, por ser uma linguagem de definição, possui cláusulas mais específicas, definindo

como deve ser estruturada a RSSF para que consultas possam ser realizadas. Assim, a rede a ser implementada deve ser semelhante às redes desenvolvidas nos projetos. Por exemplo, no caso do projeto PDCRS, seria necessário estabelecer e implementar categorias de sensores dentro da rede. Já a DML possui cláusulas mais gerais, e as particularidades de cada rede como, por exemplo, os tipos de dados que podem ser coletados, são passados por parâmetros.

Nesta dissertação foi utilizada a linguagem TinySQL. Tal escolha se deve ao fato dessa linguagem não necessitar de uma DDL que determine como deve ser estruturada a RSSF e por fornecer cláusulas na DML que atendem a necessidade de elaborar consultas que permitam determinar o comportamento do nó quanto a coleta e envio de dados, de acordo com as exigências das aplicações.

A linguagem de consulta permite atender diferentes aplicações, com diferentes necessidades de monitoramento. Essas necessidades são customizadas por meio de parâmetros referentes às cláusulas da DML, que devem ser previamente conhecidas por todos os nós. Os parâmetros são enviados para os nós na forma de pacotes de dados, assim como as respostas às consultas. Para tal, foram definidos dois tipos de pacotes a serem trafegados na rede. O primeiro é enviado apenas por nós do tipo *gateway* e possui cláusulas da linguagem TinySQL. Dentre todas as cláusulas disponíveis pela TinySQL, foram selecionadas algumas para compor as consultas. A estrutura do pacote é descrita no Algoritmo 1.

Algoritmo 1 Estrutura do pacote enviado pelo nó *gateway*

```
packet GatewayPacket {
   int msgId;
   string param;
   string restr;
   simtime_t samplePeriod;
   simtime_t forClause;
   string agreg;
   bool stopQuery;
}
```

Nesse pacote são determinados:

a) um identificador único para a consulta, uma vez que é possível ter mais de uma consulta executando na rede ao mesmo tempo. Esse parâmetro é identificado no pacote por *msgID*;

b) qual propriedade física do ambiente se deseja obter dados, no caso, temperatura ou luz. A propriedade é identificada por *param*;

c) qual a restrição sobre os dados, na qual é possível determinar que um dado deve ser maior, menor ou igual a um determinado valor. Essa restrição é determinada no pacote por *restr*;

d) o período de coleta de dados, ou seja, de quanto em quanto tempo um nó deve coletar dados. Esse parâmetro é identificado por *samplePeriod*;

e) o tempo total em que coletas de dados devem ser realizadas para a consulta, indentificado por *forClause*;

f) qual função de agregação deve ser utilizada, identificado por *agreg*;

g) se determinada consulta já ativa na rede deve ter a execução encerrada. Esse parâmetro é identificado por *stopQuery*.

A partir dessa estrutura é possível criar diversas consultas que, não necessariamente, devem necessitar utilizar todos os parâmetros. Por exemplo, se um usuário deseja submeter uma consulta na rede apenas para interromper determinada consulta em execução, somente é necessário preencher os parâmetros que indicam qual a consulta e que essa deve ser interrompida. Na Figura 12, um exemplo de tal situação é ilustrado.

Figura 12: Identificação de parâmetros de uma consulta de dados

(a) Consulta de dados (b) Identificação de parâmetros

Na Figura 12 (a), uma consulta proveniente de um usuário é exibida. Na Figura 12 (b), são identificados os parâmetros que irão compor o pacote de consultas. Nesse caso, os nós devem coletar valores da temperatura a cada 10 segundos, durante 60 segundos.

Os tipos de consultas consideradas nesta pesquisa contemplam consultas longas e consultas instantâneas.

O segundo tipo de pacote é enviado pelos nós sensores em respostas às consultas. A estrutura desse pacote é descrita no Algoritmo 2.

Algoritmo 2 Estrutura do pacote enviado pelos nós sensores

```
packet NodePacket {
    int msgId;
    int valueTemp;
    simtime_t timeTemp;
    int valueLight;
    simtime_t timeLight;
}
```

Nesses pacotes é necessário determinar o identificador da consulta que está sendo respondida, o valor dos parâmetros sensoriados e o rótulo de tempo desses. É possível obter outros

dados através desses pacotes como, por exemplo, o tempo de atraso do pacote. Tais dados são obtidos desempacotando cabeçalhos de pacote provenientes das outras camadas do nó.

Ao receber pacotes de respostas às consultas, o nó *gateway* os transmite imediatamente para serem exibidos aos usuários. Tal estratégia faz com que os dados de um nó não precisem esperar por outros dados de nós mais distantes ou que deixaram de compor a rede.

Para submeter consultas na rede, o nó *gateway* possui uma função, denominada *sendQuery*. Tal função é responsável por enviar os pacotes de dados com consultas para os nós vizinhos ao nó *gateway* que, posteriormente, serão enviados para todos os nós da rede. Um trecho dessa função é descrito no Algoritmo 3. Um pacote de dados do tipo "GatewayPacket" é criado (linha 2) para compor a consulta. Os parâmetros da consulta são adicionados ao pacote (linhas 3, 4, 5 e 6). Após todos os parâmetros da consulta serem adicionados ao pacote, solicita-se que esse seja enviado (linha 7).

Algoritmo 3 Algoritmo para o envio de consultas

```
1: void MyGatewaySensorApplLayer :: sendQuery(int identificador) {
2:     GatewayPacket *pkt = new GatewayPacket("Query", QUERY_MESSAGE);
3:     pkt->setMsgId(identificador);
4:     pkt->setForClause(forClause_var);
5:     pkt->setSamplePeriod(samplePeriod_var);
6:     //Outros parâmetros são adicionados ao pacote
7:     sendDown(pkt);
8: }
```

Para tratar os pacotes de consultas que vão sendo recebidos, os nós sensores possuem uma função, denominada *handleQuery*, para identificar quais dados estão sendo requisitados e as características das consultas como, por exemplo, tempo em que a consulta deve ficar ativa na rede. Essa função é descrita no Algoritmo 4. Inicialmente o pacote proveninente do nó *gateway* é desempacotado (linha 2) e os parâmetros que compõem a consulta são verificados (linha 3 e 4). Após obter todas as informações do pacote, os nós verificam se a consulta trata-se de uma requisição para encerrar uma consulta já ativa na rede (linha 5). Caso sim, a consulta é retirada da lista de consultas ativas do nó (linha 6). Caso contrário, uma função para tratar novas consultas é chamada a executar (linha 8).

A função para lidar com novas consultas que são recebidas pelos nós sensores é denominada *newQuery* e um trecho dessa é descrito no Algoritmo 5.

Nessa função, primeiramente é verificada a cláusula que indica o tempo de vida da consulta (linha 2). Se a cláusula *for* é nula, significa que trata-se de uma consulta instantânea e, portanto, a consulta pode ser logo executada sem necessitar armazenar parâmetros (linha 3). Caso contrário, trata-se de uma consulta do tipo longa e essa deve ser armazenada na lista de consultas ativas (linha 5). Antes de ser chamada a executar, a consulta deve ter sua próxima

Algoritmo 4 Algoritmo para o recebimento de consultas no nó sensor

```
1: void MySensorApplLayer :: handleQuery(cMessage * msg) {
2:    pkt = static_cast<MyGatewayPacket *> (msg);
3:    qAux.msgId_var = pkt->getMsgId();
4:    //Os outros parâmetros são verificados
5:    if(qAux.stopQuery_var == true)
6:       queries.remove(qAux);
7:    else
8:       newQuery(qAux);
9: }
```

Algoritmo 5 Algoritmo para armazenar uma nova consulta

```
1: void MySensorApplLayer :: newQuery(query q) {
2:    if (q.forClause_var==0)
3:       executeQuery(q);
4:    else
5:       queries.push_back(q);
6:       queryTimer = new cMessage(q.msgId_var, TIME_TO_QUERY);
7:       scheduleAt(simTime() + q.samplePeriod_var, queryTimer);
8:       executeQuery(q);
9: }
```

execução programada. Para tal, é necessário criar uma mensagem que atua como um temporizador para o próprio nó (linha 6). Essa mensagem é escalonada para ser enviada no momento da próxima execução da consulta, indicado pelo parâmetro "samplePeriod" (linha 7). Após o escalonamento da mensagem, a consulta pode ser executada (linha 8).

5.3.1 Economia de Energia

Uma estratégia para diminuir o gasto de energia dos nós no processamento de consultas é fazer uso de agregação de dados. São disponibilizadas pela linguagem TinySQL as funções de agregação *Count*, *Sum*, *Max*, *Min* e *Avg*, na qual:

Count - Número de ocorrências dos valores de entrada não-nulos.
Sum - Somatório de todos os valores de entrada.
Max - Maior valor entre todos os valores de entrada.
Min - Menor valor entre todos os valores de entrada.
Avg - Média aritmética de todos os valores de entrada.

A realização de uma função de agregação deve ser determinada na consulta. Quando é determinado que alguma função deve ser utilizada, os nós sensores interpretam essa cláusula e associam ao identificador único da consulta. Ao receber pacotes de dados de nós sensores vizinhos, o nó sensor identifica a consulta e, ao verificar que essa requisitou o uso de agregação,

antes de repassar os pacotes de dados de nós vizinhos, agregam o valor do próprio dado coletado ao dado presente no pacote. Para tal, os nós necessitam esperar, pelo menos durante um determinado intervalo de tempo, pelos pacotes de dados dos outros nós.

As funções de agregação permitem a economia de energia por diminuir a quantidade de pacotes enviados na rede (ROY *et al.*, 2012). No entanto, obter essa economia vai contra a tentativa de atender restrições de tempo, pois a espera dos nós pelos pacotes de dados de outros nós acaba por introduzir atrasos. Idealmente, a opção de agregar dados somente deve ser solicitada em consultas que não possuam prazos muito restritos de tempo e em situações em que economizar recursos se torna maior prioridade do que a questão temporal.

A economia de energia também ocorre nas situações em que se determinam restrições para filtrar dados como, por exemplo, em uma cláusula do tipo "where temp < 30". Uma vez que nem sempre todos os nós vão atender a restrição imposta ao dado, o número de nós respondendo a consulta é reduzido e, consequentemente, também é reduzida a quantidade de pacotes trafegando na rede.

5.4 Comunicação entre os nós

Para realizar o monitoramento de toda uma área que a RSSF deve abranger, os nós sensores necessitam trabalhar de forma colaborativa, enviando pacotes uns aos outros. A comunicação entre os nós é relizada por meio de um protocolo de comunicação já implementado no *framework* Mixim do OMNeT++, que estabelece a construção de uma árvore de roteamento a partir de um ponto inicial. O processo de descoberta de rotas ocorre durante a inicialização da rede, antes de qualquer outra atividade.

A árvore de roteamento estabelece uma hierarquia entre os nós, que serão classificados em nós do tipo pai ou do tipo filho. Mensagens que necessitam ser difundidas na rede, como pacotes de consultas, fluem de nó pai para nó filho. Já mensagens que necessitam chegar ao nó *gateway*, como pacotes de respostas às consultas, fluem de nó filho para nó pai. Para saber quem é o nó pai e quem são os nós filhos, cada nó possui uma tabela de roteamento. O funcionamento do protocolo é apresentado nas etapas a seguir:

a) O nó que desempenha o papel de ponto inicial envia uma mensagem de roteamento para os nós vizinhos. Esse nó sempre será o nó *gateway*.

b) Os nós vizinhos ao nó inicial, ao receberem a mensagem de roteamento, escolhem o nó inicial como nó pai.

c) Os nós que já receberam a mensagem de roteamento a encaminha para seus nós vizinhos.

d) Cada nó que recebe a mensagem de roteamento deve selecionar o primeiro nó de quem recebeu a mensagem como nó pai. Quando o nó recebe mais de uma mensagem de rote-

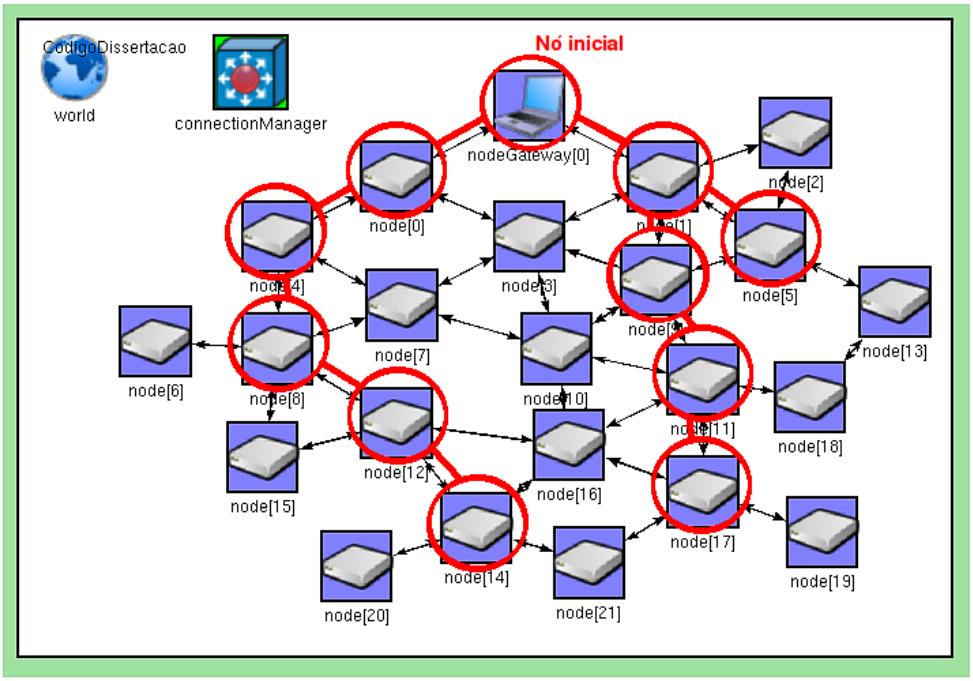
amento no mesmo instante, opta-se por ser nó pai aquele em que o *link* de comunicação entre os nós esteja acima de um limiar determinado a priori.

e) Quando um nó já possui um nó pai e recebe uma mensagem de roteamento, esse apenas encaminha a mensagem para os nós vizinhos.

f) O processo continua até que todos os nós possuam um nó pai.

Um exemplo de árvore de roteamento estabelecida é ilustrado na Figura 13. Os nós pais estão em destaque com círculos em vermelho.

Figura 13: Árvore de roteamento



Uma vez que a rede está sujeita a mudanças de topologia, a árvore pode ser reconstruída periodicamente. O tempo de espera até uma nova construção da árvore é determinado pelo nó inicial, que deve reiniciar o processo de construção.

5.5 Seleção da Estratégia de Gerenciamento dos Dados e Transações

O gerenciamento dos dados e transações de tempo real é realizado no intuito de determinar como as transações devem executar, a fim de manter as consistências lógicas e temporais dos dados e transações. Para tal, faz-se necessário identificar conflitos entre as transações e adotar estratégias para resolvê-los. Nesta pesquisa, foi utilizada a estratégia apresentada em Ribeiro Neto (2006b). Nessa estratégia, são utilizadas funções de negociação para tratar conflitos entre as transações, em que se verifica quando é possível relaxar a necessidade de consistência lógica para atender aos requisitos de consistência temporal e vice-versa.

Os conflitos acontecem em duas situações, quando uma operação de leitura de dados está sendo executada e uma transação de escrita do nó sensor deseja executar sobre o mesmo item de dado ou quando uma operação de escrita do nó sensor está executando e uma transação de leitura deseja executar sobre o mesmo item de dado. Seria possível existir um conflito entre uma escrita proveniente de um usuário e uma escrita do dispositivo sensor. Por exemplo, em uma rede cujos nós possuem dados para identificar posicionamento, pode-se considerar a atualização desses dados sendo oriunda tanto do dispositivo como do usuário, levando a um conflito. Nesta pesquisa, no entanto, considera-se que o usuário pode apenas realizar leituras de dados.

Nesta dissertação, o gerenciamento é utilizado para gerir transações do tipo suave, ou seja, quando o não cumprimento de prazos não acarreta em consequências catastróficas, como a morte de um ser humano. Assim, é realizado um esforço para preservar a consistência temporal, no entanto, não é possível oferecer a priori a garantia que as restrições de tempo serão completamente atendidas. As aplicações visadas, por sua vez, são aquelas em que é permitido manipular dados imprecisos e/ou ter descumprimento de prazos, nas quais é melhor ter resultados recebidos em tempo, embora esses resultados não estejam completos. Por exemplo, em uma aplicação de monitoramento de animais em uma floresta, pode ser melhor ter uma noção do posicionamento desses antes, do que ter a noção exata depois, quando esses já podem ter se deslocado.

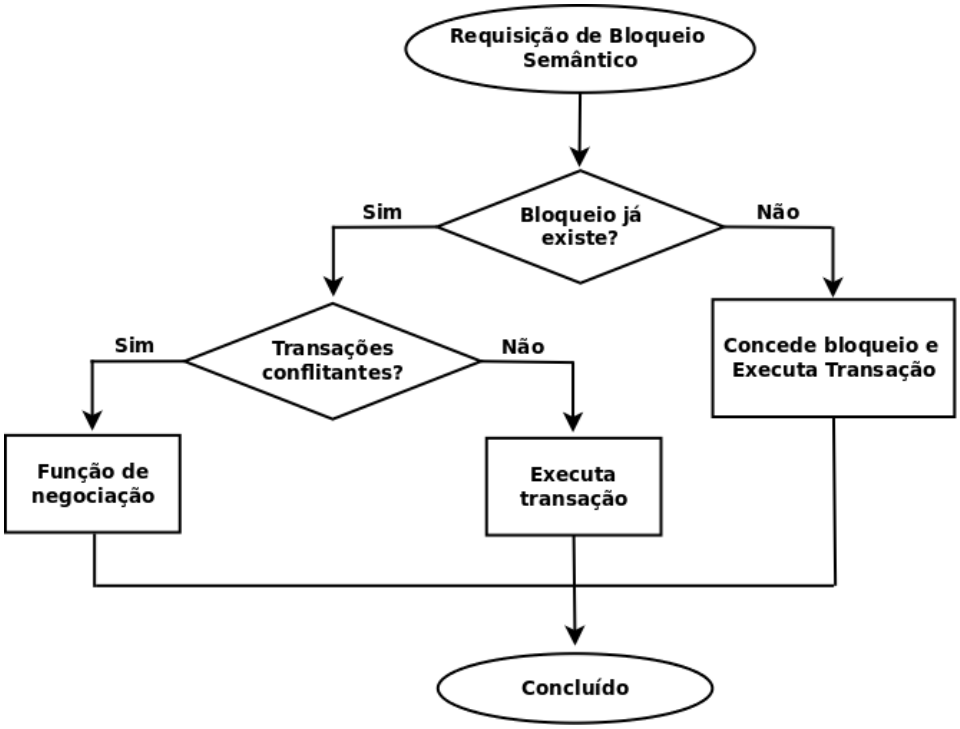
O gerenciamento é distribuído nos nós sensores individualmente, na qual cada um controla o acesso concorrente aos seus dados baseando-se em uma função de negociação. Para identificar que uma transação de escrita ou leitura está executando sobre um item de dado, é utilizada uma variável de controle para indicar um bloqueio semântico (de leitura ou escrita) sobre o dado. Quando uma outra transação deseja executar, é verificado o estado da variável de controle. Se o item de dado já tem um bloqueio semântico sobre ele, é verificado se as transações são conflitantes. Caso seja identificado algum conflito, a função de negociação é executada. Esse fluxo é ilustrado na Figura 14. Após as execuções, as transações liberam os bloqueios semânticos adquiridos.

A função de negociação determina se é possível permitir a execução das transações simultaneamente, fazendo com que nenhuma delas tenha de esperar para ter sua execução realizada, aumentando a concorrência e a vazão dos dados. Para tal, a função verifica a consistência lógica e temporal por meio de expressões booleanas, sendo definida por:

$$\begin{aligned} FN(TL,\ TE) \ &= \ (agora - x.tempo \leq x.avi) \ \wedge \\ &\quad\ (|x.val - x_{novo}.val| \leq \ (Lim\_Imp - x.quant\_imp) \end{aligned}$$

onde *agora* determina o instante de tempo atual, *x.tempo* determina o rótulo de tempo do dado x, *x.avi* determina o intervalo de validade absoluta do dado x, *x.val* refere-se ao valor do item de dado, x_{novo}*.val* trata-se do valor que vai ser escrito, *Lim_Imp* determina a quantidade de

Figura 14: Fluxograma do gerenciamento das transações concorrentes



imprecisao que pode ser aceita no dado e *x.quant_imp* é a imprecisão já existente no dado.

A negociação foi implementada através de uma função, denominada *negotiation*, cuja implementação é descrita no Algoritmo 6. Essa função recebe a consulta que deseja ser executada como parâmetro (linha 1) e verifica se o dado em que se está requisitando o acesso é temporalmente válido (linha 2). Uma vez sendo válido, é verificado se a diferença entre o valor atual e o novo valor do dado é menor que o limite de imprecisão aceitável para aquele dado somada a imprecisão que o dado já possa conter (linha 3). Caso sim, a imprecisão do dado é atualizada (linha 4) e a função retorna verdadeiro (linha 5), permitindo a que transações executem simultaneamente. Caso contrário, a função retorna falso (linha 7) e a transação que solicitou a execução posteriormente deve aguardar.

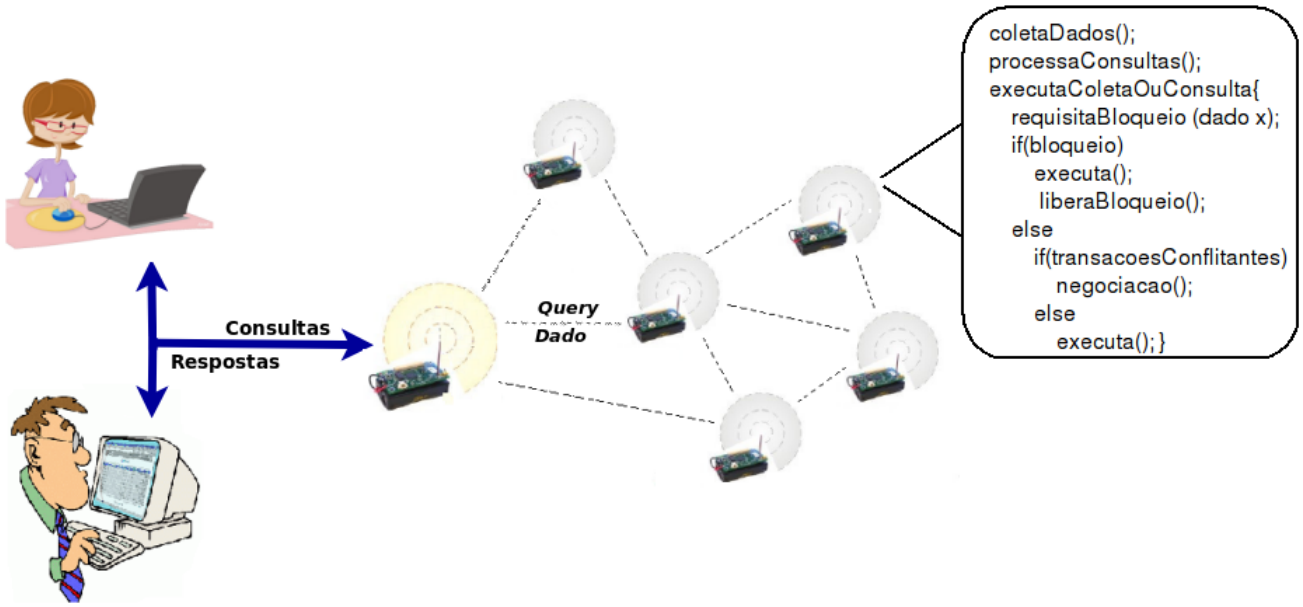
Algoritmo 6 Algoritmo da Função de Negociação

```
1: bool MySensorApplLayer :: negotiation(query &q){
2:     if((simTime() - valueTemp.time) < valueTemp.avi){
3:         if(abs(valueTemp.value - novaTemp) <= abs(imp_max - valueTemp.imprecisao)){
4:             valueTemp.imprecisao = valueTemp.value - novaTemp;
5:             return true;
6:     else
7:         return false;
8: }
```

5.6 Simulações e Resultados

No intuito de avaliar a execução do processamento de consultas, foram realizadas simulações considerando um cenário em que usuários submetem consultas à rede, com o objetivo de obter valores de temperaturas e antecipar possíveis focos de incêndio, preservando uma área florestal. Os usuários podem realizar consultas longas ou instantâneas e essas são distribuídas na rede a partir do *gateway*. Os nós são preparados para processar consultas e gerenciar dados e transações, preservando as consistências lógica e temporal. Tal cenário é ilustrado na Figura 15. Para fins de implementação, foi considerada uma rede composta por 22 nós sensores e um nó *gateway*, distribuídos em uma área de 100m x 100m.

Figura 15: Cenário da avaliação do processamento de consultas



Os nós sensores precisam gerenciar dois tipos de transações, de leitura (provenientes dos usuários) e de escrita (oriundas dos próprios nós sensores). A primeira tem como característica ser aperiódica, pois não é possível definir a priori quando os usuários irão submeter consultas e, portanto, essas transações não têm um tempo de início definido. Já a segunda caracteriza-se por ser periódica, sendo estabelecido que os nós devem atualizar seus dados a cada determinado intervalo de tempo.

Quando um nó está atualizando os dados e uma consulta proveniente dos usuários tenta acessá-los ou o contrário, tem-se um conflito entre as transações. Nessas situações, os nós executam a função de negociação no intuito de permitir que as transações executem concorrentemente, aumentando a vazão dos dados. Para tal, considerou-se uma imprecisão de dados, utilizando a métrica de imprecisão máxima, definida por

$$\text{Impr} = \text{Valor Corrente x } \frac{\text{Imp}}{100}.$$

Nessa métrica, determina-se uma imprecisão em porcentagem que é possível admitir sobre um dado. Por exemplo, considerando que o valor da temperatura detectado é de 30 graus Celsius e a imprecisão admitida (Imp) é de até 10%, é possível aceitar um valor que esteja dentro do intervalo de 27 a 33 graus Celsius. Quando a avaliação da execução concorrente das transações resulta em uma imprecisão maior do que a admitida, as transações não conseguem executar simultaneamente e a transação que teve o bloqueio semântico negado deve ser sempre reiniciada.

O simulador OMNeT++, quando utilizado com o *framework* Mixim, não permite a execução de um programa externo enquanto a rede está ativa. Assim, não é possível haver uma interface em que os usuários possam submeter consultas aos nós enquanto esses estão executando. Para suprir essa necessidade, foi acrescentado ao nó gateway um conjunto de funcionalidades e de consultas distintas para que esse atue também como usuário. Assim, esporadicamente, uma consulta presente no nó *gateway* é interpretada e um pacote de consulta correspondente a essa é gerado e submetido na rede. Um exemplo de pacote de consulta é descrito no Algoritmo 7.

Algoritmo 7 Exemplo de pacote de consulta

```
1: void MyGatewaySensorApplLayer::generateQuery() {
2:     query[0].msgId = 0;
3:     query[0].param = “TEMP”;
4:     query[0].restr = “no”;
5:     query[0].samplePeriod = 1;
6:     query[0].forClause = 0;
7:     query[0].agreg = “no”;
8:     query[0].stopQuery = false;
9: }
```

No caso desse pacote, trata-se de uma consulta instantânea, pois a cláusula *for*, que indica o tempo que a consulta deve ficar ativa na rede, é zero (linha 6). Nessa consulta, verifica-se que é indicado um identificador único para a consulta (linha 2), que o parâmetro em que dados devem ser obtidos é a temperatura (linha 3), que não há restrições quanto aos valores dos dados (linha 4), que os dados devem ser coletados no próximo segundo após a chegada da consulta no nó sensor (linha 5), que o usuário não deseja que sejam realizadas operações de agregação (linha 7) e que nesta consulta não está sendo solicitado que uma consulta já ativa na rede tenha a execução encerrada (linha 8).

Quando um pacote de dados em resposta a uma consulta é recebido pelo nó *gateway*, esse deve ser imediatamente exibido ao usuário. Como não é possível exibir os dados em uma interface enquanto a rede está ativa, em substituição a tela de exibição, os dados que chegam são armazenados em um arquivo externo à rede. Um exemplo de tal arquivo é ilustrado na Figura 16.

Figura 16: Arquivo de exibição dos dados

Hora Atual	Consulta	Nó	Temperatura	Luz
21:34:01	0	4	26	-
21:34:03	0	13	28	-
21:34:04	0	17	35	-
21:34:07	0	14	32	-
21:34:08	0	1	27	-
21:35:00	1	15	31	-
21:35:02	1	20	25	-
21:35:07	1	12	29	-
21:35:08	1	8	27	-
21:35:09	1	11	31	-
21:35:30	2	3	26	-
21:35:32	2	19	28	-
21:35:35	2	18	28	-

As consultas começam a ser submetidas somente depois de ser executado o protocolo de comunicação, que determina quais nós devem ser pais e quem são seus nós filhos. Com a simulação, a proposta é verificar a quantidade de dados que têm uma liberação imediata com a execução concorrente das transações, evitando que esses fiquem retidos nos nós, por mais algum tempo, devido a conflitos entre transações. A liberação imediata de dados proveniente do gerenciamento de dados e transações diminui o atraso dos pacotes e aumenta a chance desses cumprirem os prazos temporais das aplicações.

Foram realizadas quatro simulações, cada uma com duração de 1500 segundos, distinguindo entre elas os valores da imprecisão máxima admitida e o intervalo de tempo entre coletas de dados dos nós. Assim,

a) Na simulação 1 considerou-se que os nós coletam dados periodicamente, a cada 5 segundos. Já as consultas provenientes dos usuários são aperiódicas. Quando acontecem conflitos entre transações, a função de negociação considera uma imprecisão máxima de 10% do valor do dado.

b) Na simulação 2 é considerado o mesmo intervalo de 5 segundos definido na simulação 1 entre coletas de dados dos nós sensores, no entanto, a imprecisão máxima admitida sobre os dados é restringida a 5%.

c) Na simulação 3 é determinado que os nós devem coletar dados periodicamente, a cada 3 segundos. Quando conflitos entre as transações ocorrem, é admitida uma imprecisão máxima de 10% sobre os dados.

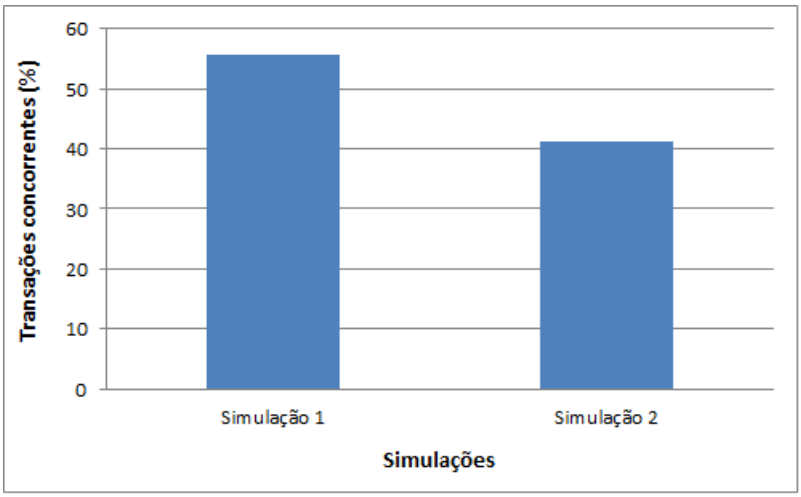
d) Na simulação 4 os nós devem coletar dados, assim como na simulação 3, no intervalo de 3 segundos. No entanto, a imprecisão máxima admitida nos dados é de apenas 5%.

Ao executar as simulações 1 e 2, verificou-se que dentre as transações que os nós precisaram executar para responder as consultas, cerca de 3 a 5.8% dessas estiveram envolvidas em conflitos com outras transações. Sem realizar nenhum gerenciamento de dados e transações, essas transações necessitam aguardar o término da execução das outras transações conflitantes

para poder executar. Aplicando o gerenciamento de dados e transações, verificou-se que na simulação 1, considerando a média entre os nós, 55.7% das transações envolvidas em conflitos puderam executar concorrentemente e, por consequência, liberar imediatamente o dado em resposta a consulta para ser roteado até o nó *gateway*.

Na simulação 2, em que houve uma redução da imprecisão máxima admitida para 5%, uma taxa menor de transações puderam executar concorrentemente. No caso, tal taxa correspondeu a 41.2% das transações. Essa diminuição é esperada, uma vez que restringindo o intervalo de dados que é possível admitir como imprecisão, um número menor de dados irá conseguir atendê-lo. Uma comparação entre a taxa de transações envolvidas em conflitos que executaram concorrentemente nas simulações 1 e 2 é ilustrada na Figura 17.

Figura 17: Comparação entre as simulações 1 e 2 quanto a execução concorrente de transações

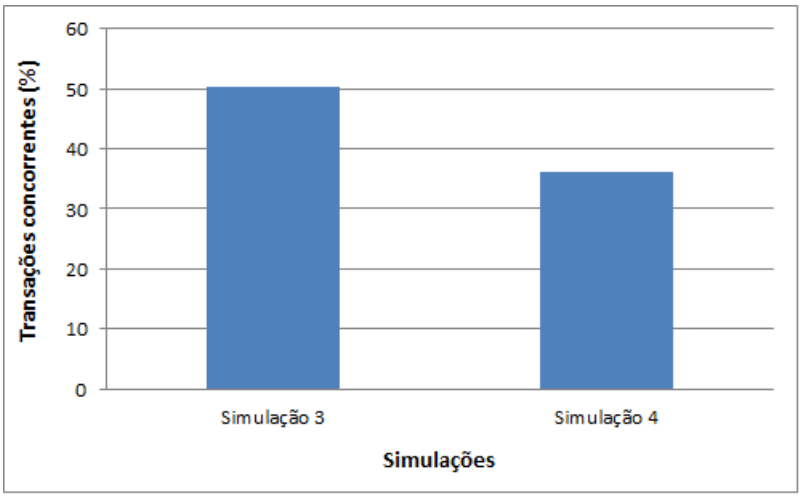


Nas simulações 3 e 4, o intervalo entre coleta de dados dos nós foi reduzido a 3 segundos. Tal diminuição provoca que um maior número de transações de escrita para atualizar dados seja executado. Por consequência, amplia-se também a concorrência entre as transações e a possibilidade de haver conflitos. Verificou-se nessas simulações que a taxa de transações envolvidas em conflitos foi cerca de 5.1 a 10% do total de transações.

Na simulação 3, realizando o gerenciamento de dados e transações, a taxa de transações envolvidas em conflitos que executaram concorrentemente nos nós foi de 50.2%. Já na simulação 4, embora se tenha uma diminuição da imprecisão máxima admitida, ainda foi possível permitir que 36% das transações conflitantes executassem de forma concorrente. Uma comparação entre a taxa de transações que executaram concorrentemente nas simulações 3 e 4 é ilustrada na Figura 18.

Com as simulações, é possível verificar que adotando a estratégia de gerenciamento de dados e transações, o processamento de consultas pode ser realizado permitindo manter a consistência lógica de dados e transações e, ao mesmo tempo, que um maior número de transações

Figura 18: Comparação entre as simulações 3 e 4 quanto a execução concorrente de transações



não necessitem aguardar o término de outras para executar, aumentando a liberação de dados para as aplicações. Evitar que os dados fiquem contidos nos nós é de grande importância, pois aumenta a chance de que as restrições de tempo das aplicações sejam atendidas.

5.7 Considerações Finais

O processamento de consultas em nós sensores tem por objetivo permitir que usuários acessem os dados armazenados nos próprios nós sensores, além de poder modificar padrões de envio e coleta de dados. Nesta dissertação, considerou-se que as aplicações a serem atendidas com o processamento de consultas tinham como característica possuir restrições de tempo. Assim, dados devem sempre representar o estado mais atual do ambiente e as transações que acessam tais dados devem ter suas execuções escalonadas visando atender restrições de tempo.

Para realizar o processamento de consultas, os nós foram programados para processar cláusulas de uma linguagem de consulta previamente definida. Tal linguagem pode definir quais dados devem ser retornados como resposta a uma consulta e quando esses dados devem ser enviados.

Para gerenciar a execução das transações, os nós foram programados para realizar o gerenciamento de dados e transações de modo que na ocorrência de transações conflitantes, uma função de negociação é executada a fim de negociar entre a consistência lógica e temporal, permitindo execuções concorrentes e evitando que dados fiquem contidos nos nós.

Através de simulações, é possível verificar que com gerenciamento de dados e transações grande parte das transações conflitantes conseguem executar concorrentemente, reduzindo a contenção de dados nos nós.

6 Conclusões e Trabalhos Futuros

6.1 Conclusões

As RSSF são uma ferramenta de grande potencial no que se refere ao desenvolvimento de aplicações de monitoramento das mais diversas áreas. Estas redes são formadas por nós sensores autônomos, que possuem a capacidade de sensoriar parâmetros físicos, além de realizar processamento, armazenamento e comunicação sem fio, possibilitando monitoramento remoto. O comportamento dos nós sensores pode ser previamente determinado antes da implantação da rede, como também pode ser modificado através de linguagens declarativas quando os nós sensores já estão em atividade. A última abordagem traz a vantagem de poder atender melhor as aplicações, customizando o comportamento dos nós de acordo com as necessidades dessas.

Dentre as aplicações que utilizam RSSF, algumas caracterizam-se pela necessidade de adquirir dados dentro de determinados prazos. O não cumprimento destes prazos pode acarretar em prejuízos. Para atender tais aplicações, os nós sensores precisam ser preparados para gerenciar as atividades considerando tempo, de forma que suas atividades sejam guiadas visando o atendimento das restrições temporais.

O principal objetivo desta dissertação foi realizar o processamento de consultas em nós sensores, levando em consideração que, além da necessidade de manter a consistência lógica dos dados, esses são sensíveis ao tempo, possuindo uma validade temporal. Tal validade se deve ao fato que, devido a dinamicidade dos ambientes, não é possível garantir que após certo tempo um dado ainda representa o estado atual do meio. Para tratar a questão do tempo nos dados, as transações também possuem restrições temporais.

Na busca de alcançar tal objetivo, foram realizados estudos acerca das linguagens de consulta para aplicações com RSSF. As linguagens de consulta permitem que os usuários solicitem acesso aos dados coletados e, ainda, determinem como gostariam de modificar o comportamento dos nós quanto ao envio e coleta de dados. Os nós, por sua vez, são programados para processar cláusulas das linguagens e gerar novos padrões de coleta e envio de dados a partir dessas. Foram realizados também estudos acerca de estratégias de gerenciamento de dados e transações de tempo real. Esse gerenciamento se faz necessário para que restrições lógicas e temporais possam ser atendidas.

O atendimento de restrições temporais é bastante desafiador, gerando a necessidade de processar transações em tempo usando dados atuais. Essa necessidade leva a um grande impasse, pois é possível ser necessário interromper transações de acesso a dados, para atualizar o valor desses, mantendo-os o mais atual possível, como também pode ser necessário deixar o dado desatualizado, para que transações possam executar em tempo. Tais interrupções podem acarretar em ferir a consistência lógica e temporal dos dados e das transações. Em algumas aplicações,

no entanto, é possível relaxar a consistência temporal e lógica, realizando negociações entre elas e permitindo transações conflitantes executarem simultaneamente.

Nesta dissertação, o grupo de aplicações que se buscou atender são aquelas classificadas como não críticas, em que atender os prazos temporais é bastante desejável, porém não é obrigatório, e uma negociação entre as consistências lógicas e temporais é possível. Assim, é permitido manipular dados imprecisos e/ou ter descumprimento de prazos, não obrigando os nós sensores a ter que, estritamente, garantir restrições, mas realizando um esforço para tal.

A partir das escolhas da linguagem de consulta e do gerenciamento de dados e transações, foi implementado o processamento de consulta em nós sensores, na qual os nós são capazes de processar primitivas de uma linguagem de consulta, adequando-se as necessidades das aplicações, bem como definem a execução de coletas e envio de dados com base na necessidade de atender restrições de tempo das aplicações.

Como principal contribuição desse trabalho tem-se a realização do processamento de consultas, em nós sensores, considerando restrições lógicas e temporais. As contribuições alcançadas com o processamento de consultas consistem em:

a) Seleção e implementação de uma linguagem de consulta, com regras gramaticais pré-estabelecidas, voltada para aplicações de RSSF. Tal linguagem permite a comunicação com os nós sensores, modificando o comportamento desses quando desejável.

b) Seleção e implementação de uma estratégia de gerenciamento de dados e transações de tempo real, que visa conseguir atender a maior quantidade de requisições de consultas, realizando um esforço para possibilitar também o atendimento das restrições temporais das aplicações.

c) Implementação do processamento de consultas em nós sensores, que permite usuários submeterem consultas aos nós da RSSF, enquanto esses, além de atenderem as requisições de dados, gerenciam dados e transações visando manter a consistência lógica e temporal desses.

6.2 Trabalhos Futuros

A partir deste trabalho, é possível apontar alguns desafios que ainda precisam ser aprofundados ou desenvolvidos. Assim, perspectivas de trabalhos futuros são destacadas:

- Gerenciar o envio de consultas à rede para evitar que consultas equivalentes sejam emitidas ao mesmo tempo, diminuindo o número de pacotes enviados na rede, bem como o processamento nos nós.

- Implementar um protocolo de comunicação que tenha como objetivo atender aplicações com restrições de tempo real e, dessa forma, tratar restrições de tempo não somente no envio e coleta de dados, como também no percurso do dado dos nós até o usuário.
- Adotar mais estratégias para o tratamento de energia, como oscilar momentos de atividade e momentos em que os nós devem estar desligados.
- Implementar e analisar o processamento de consulta considerando restrições temporais e lógicas em nós sensores reais.
- Verificar a influência da realização de operações de agregação de dados sobre o uso de estratégias para atender restrições de tempo das aplicações.

Referências

AI, C. *Energy-Efficient Data Management in Wireless Sensor Networks*. Tese (Doutorado) — Georgia State University, Atlanta, Georgia, 2010.

ALBERT, M. *Analysis of the IEEE 802.15.4A Ultra Wideband Physical Layer through Wireless Sensor Network Simulations in OMNeT++*. Tese (Doutorado) — Universidade de Pretoria, Pretoria, África do Sul, Fevereiro 2011.

ALI, A. A. *On Optimistic Concurrency Control for Real-Time Database Systems*. American Journal of Applied Sciences, v. 3, n. 2, p. 1706–1710, 2006.

BORGES NETO, J. B. *PROST: Um Protocolo de Roteamento Sensível ao Tempo para Redes de Sensores sem Fio*. Dissertação (Mestrado) — Universidade Federal do Ceará, Fortaleza, 2009.

BRITO, L. F. M. N. de. *Análise Comparativa da Vazão e do Consumo de Energia em Redes de Sensores Sem Fio com Topologia Estrela Utilizando o Padrão IEEE 802.15.4 no Modo Beacon-enabled e Nonbeacon*. Monografia (Graduação) - Universidade de Brasília, 2009.

CASSIMIRO NETO, F. *SEMSUS: Middleware para Redes de Sensores sem Fio Baseado em Serviços Semânticos*. Dissertação (Mestrado) — Universidade do Estado do Rio Grande do Norte e Universidade Federal Rural do Semi-Árido, Programa de Pós-Graduação em Ciência da Computação, 2010.

CHAGAS, L. D.; LIMA, E. P.; RIBEIRO NETO, P. F. *Real-Time Databases Techniques in Wireless Sensors Networks*. 6th International Conference on Networking and Services (ICNS), p. 182–187, 2010.

CHAGAS, L. D.; RIBEIRO NETO, P. F. *Sistema de Gerenciamento de Banco de Dados em Tempo-Real para Aplicações em Redes de Sensores*. Revista Ciência Sempre, n. 18, p. 12–19, 2010. ISSN 1808-8007.

CHANGBAI, C.; JAEHYOUNG, L.; JUYEON, H.; INSUNG, J.; MINSOO, K.; HYUN, S. J. *SNQL: A Query Language for Sensor Network Databases*. Proceedings of the International Conference on Telecommunications and Informatics, p. 114–119, 2008.

CHIPARA, O.; LU, C.; ROMAN, G.-C. *Real-time Query Scheduling for Wireless Sensor Networks*. 28th IEEE International Real-Time Systems Symposium, p. 389–399, 2007.

CHOI, D.-W.; CHUNG, C.-W. *REQUEST: Region-based Query Processing in Sensor Networks*. 16th International Conference on Database Systems for Advanced Applications, p. 266–279, 2011.

CONTI, M.; ANASTASI, G.; FRANCESCO, M. D. *A Comprehensive Analysis of the MAC Unreliability Problem in IEEE 802.15.4 Wireless Sensor Networks*. IEEE Transactions on Industrial Informatics, v. 7, p. 52–65, 2011.

CROSSBOW. *Crossbow*. 2011. Disponível em: <http://www.xbow.com/>. Acesso em: 16 Dezembro 2011.

CURINO, C.; GIANI, M.; GIOGETTA, N.; GIUSTI, A.; MURPHY, A. L.; PICCO, G. P. *Mobile Data Collection in Sensor Networks: The TinyLime Middleware*. Journal of Pervasive and Mobile Computing, v. 4, n. 1, p. 446–469, 2005.

DELICATO, F. C. *Middleware Baseado em Serviços para Redes de Sensores sem Fio*. Tese (Doutorado) — Universidade Federal do Rio de Janeiro, 2005.

DIPIPPO, L. B. *Object-Based Semantic Real-Time Concurrency Control*. Tese (Doutorado) — Universiy of Rhode Island, 1995.

DOMINGUES, J.; DAMASO, A.; NASCIMENTO, R.; ROSA, N. *An Energy-Aware Middleware for Integrating Wireless Sensor Networks and the Internet*. International Journal of Distributed Sensors Networks, p. 1–19, 2011. ISSN 1550-1477.

FARINES, J.-M.; FRAGA, J. da S.; OLIVEIRA, R. S. de. *Sistemas de Tempo Real*. São Paulo: IME-USP, 2000.

FERNANDES, Y. Y. M. P. *Técnica de Controle de Concorrência Semântico para Sistemas de Gerenciamento de Bancos de Dados em Tempo-Real*. Dissertação (Mestrado) — Universidade Federal de Campina Grande, Campina Grande, Paraíba, Agosto 2005.

GARCÍA-MOLINA, H.; ULLMAN, J.; WIDOM, J. *Implementação de Sistemas de Banco de Dados*. Rio de Janeiro: Campus, 2001.

GUMZEJ, R.; HALANG, W. A. *Real-Time System's Quality of Service*. New York: Springer, 2010.

IDOUDI, N.; DUVALLET, C.; SADEG, B. *How to model a real-time database?* IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing, p. 321–325, 2009. ISSN 1555-0885.

JUNG, H.; HAN, H.; FEKETE, A.; RöHM, U. *Serializable Snapshot Isolation for Replicated Databases in High-Update Scenarios*. International Conference on Very Large Data Bases, v. 4, n. 11, p. 783–794, 2011.

KAN, S. S.; ALI, M. S. *Prototype Architecture for Real-Time Object Oriented Distributed Database*. International Journal of Engineering and Technology, v. 2, n. 1, p. 13–20, 2009.

KANG, K.-D. *OMNeT++ and the Mixim Framework*. 2010. 63 slides: color.

KELEHER, P.; ÇETINTEMEL, U. *Efficient distributed precision control in symmetric replication environments*. 21st IEEE Symposium on Reliable Distributed Systems, p. 119–128, 2004.

KLINGBEIL, L.; WARK, T. *A Wireless Sensor Network for Real-Time Indoor Localisation and Motion Monitoring*. Proceedings of the 7th International Conference on Information Processing in Sensor Networks, 2008.

KÖPKE, A.; SWIGULSKI, M.; WESSEL, K.; WILLKOMM, D.; HANEVELD, P. T. K.; PARKER, T. E. V.; VISSER, O. W.; LICHTE, H. S.; VALENTIN, S. *Simulating wireless and mobile networks in OMNeT++ the MiXiM vision*. 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops, Bruxelas, Bélgica, p. 71:1–71:8, 2008.

KOT, M. *Modeling Real-Time Databases Concurrency Control Protocol Two-Phase-Locking in Uppaal*. International Multiconference on Computer Science and Information Techonology, p. 673–678, 2008.

KROL, V.; POKORNY, J. *Design of V4DB - Experimental Real-Time Database*. Proceedings of the 32nd Annual Conference of the IEEE Industrial Eletronics Society, 2006.

LEITE, C. R. M. *Linguagem de Consulta para Aplicações em Tempo-Real*. Dissertação (Mestrado) — Universidade Federal de Campina Grande - UFCG, Campina Grande, Paraíba, Setembro 2005.

LIMA, R. de C. A. *Middleware para Rede de Sensores sem Fio Baseado em Espaço de Tuplas*. Dissertação (Mestrado) — Universidade Federal de Pernambuco, Programa de Pós-Graduação em Ciência da Computação, Recife, PE, 2008.

LOPES, C. E. R. *Uma Rede de Sensores sem Fio multicamada, multimodal para monitoração ambiental*. Dissertação (Mestrado) — Universidade Federal de Minas Gerais, Programa de Pós-Graduação em Engenharia Elétrica, Belo Horizonte, MG, 2007.

LU, C.; BHATTACHARYA, S.; SAIFULLAH, A.; ROMAN, G.-C. *Multi-Application Deployment in Shared Sensor Networks Based on Quality of Monitoring*. The 16th IEEE Real-Time and Embedded Technology and Applications Symposium, p. 259–268, 2010.

MADDEN, S. R.; FRANKLIN, M. J.; HELLERSTEIN, J. M.; HONG, W. *TinyDB: an acquisitional query processing system for sensor networks*. ACM Transactions on Database Systems Journal, v. 30, 2005.

MATOS, R. T. B. R.; BRAYNER, A.; MAIA, J. E. B. *Agregação e Predição de Dados em Rede com Precisão Ajustável no Processamento de Consultas em Redes de Sensores Sem Fio*. XXIII Simpósio Brasileiro de Banco de Dados, Campinas, São Paulo, p. 150–164, 2008.

MEIRA, D. M. *Um Mecanismo de Processamento de Consultas Distribuído em Redes de Sensores sem Fio*. Dissertação (Mestrado) — Universidade de Fortaleza, Fortaleza, CE, 2007.

MIXIM. *Mixim Project*. 2011. Disponível em: <http://mixim.sourceforge.net/>. Acesso em: 02 Dezembro 2011.

MOHAMAD, R.; AZIZ, M.; JAWAWI, D.; GHAZALI, M.; ARBAIE, M.; IBRAHIM, N. *Service identification guideline for developing distributed embedded real-time systems*. IET Software, v. 6, p. 74–82, 2011. ISSN 1751-8806.

NAKAMURA, E. F. *Fusão de dados em Redes de Sensores sem Fio*. Tese (Doutorado) — Universidade Federal de Minas Gerais, 2007.

NAM, G.; LI, M. *Energy-Efficient Query Management Scheme for a Wireless Sensor Database System*. Journal on Wireless Communications and Networking, p. 1–18, 2010. ISSN 1687-1472.

OMNET. *OMNet++ Network Simulator Framework*. 2011. Disponível em: <http://www.omnetpp.org>. Acesso em: 29 Novembro 2011.

PAGANO, P.; CHITNIS, M.; LIPARI, G.; NASTASI, C.; LIANG, Y. *Simulating Real-Time Aspects of Wireless Sensor Networks*. Wireless Communications and Networking, 2010.

PHONG, N. H.; YUNSUNG-UHM; KIM, M.-K. *Real-Time Message Transmission Over Wireless Sensor and Actor Networks*. The 2nd International Conference on Computer and Automation Engineering, p. 649–653, 2010.

RIBEIRO, A. de R. L. *SensorBus: Um Middleware Baseado em Políticas para Redes de Sensores sem Fio*. Dissertação (Mestrado) — Universidade Federal do Pará, Programa de Pós-Graduação em Engenharia Elétrica, Belém, Pará, 2007.

RIBEIRO, J. E. *Middleware de Serviços Multi-Camadas para Rede de Sensores sem Fio*. Dissertação (Mestrado) — Universidade Federal de São Carlos, Programa de Pós-Graduação em Ciência da Computação, São Carlos, SP, 2010.

RIBEIRO NETO, P. F. *Mecanismos de Qualidade de Serviços para Gerenciamento de Dados e Transações em Tempo-Real*. 2006. 71 slides: color.

RIBEIRO NETO, P. F. *Mecanismos de Qualidade de Serviços para o Gerenciamento de Dados e Transações em Tempo-Real*. Tese (Doutorado) — Universidade Federal de Campina Grande, Campina Grande, Paraíba, 2006.

RIBEIRO NETO, P. F.; FERNANDES, Y. Y. M. P.; LEITE, C. R. M.; COSTA, C. M.; Mendes Neto, F. M. *Redes de Sensores: O Estado da Arte*. EPOCA - Escola Potiguar de Computação e Suas Aplicações, p. 1–16, 2008.

ROY, S.; CONTI, M.; SETIA, S.; JAJODIA, S. *Secure Data Aggregation in Wireless Sensor Networks*. IEEE Transactions on Information Forensics and Security, p. 1556–6013, 2012.

SHANKER, U.; MISRA, M.; SARJE, A. K. *Distributed Real Time Database Systems: background and literature review*. Distributed and Parallel Databases, v. 23, n. 2, p. 127–149, 2008.

SILBERSCHATZ, A.; KORTH, H. F.; SUDARSHAN, S. *Sistemas de Banco de Dados*. Rio de Janeiro: Campus, 2006.

SOUSA, A.; CORREIA JÚNIOR, A.; SOARES, L.; MOURA, F.; OLIVEIRA, R. *Revisiting the Epsilon Serializability to improve the Datase State Machine*. Workshop on Dependable Distributed Data Management (SRDS), 2004.

SOYTURK, M.; ALTILAR, D. T. *Reliable Real-Time data Acquisition for Rapidly Deployable Mission-Critical Wireless Sensor Networks*. IEEE International Conference on Computer Communications Workshops, p. 1–6, 2008.

SQL. *SQL - Structured Query Language*. 2012. Disponível em: <http://www.sql.org/>. Acesso em: 03 Março 2012.

STANKOVIC, J. A.; SON, S. H.; HANSSON, J. *Misconceptions about Real-Time Databases*. IEEE Computer, v. 32, n. 6, p. 29–36, 1999.

SZALAPSKI, T.; MADRIA, S. *Real-Time Data Compression in Wireless Sensor Networks*. The 11th International Conference on Mobile Data Management, p. 307–308, 2010.

SZEWCZYK, R.; OSTERWEIL, E.; POLASTRE, J.; HAMILTON, M.; MAINWARING, A.; ESTRIN, D. *Habitat Monitoring with sensor networks*. Communications of ACM - Wireless Sensor Networks, v. 47, 2004.

TINYOS. *TinyOS*. 2011. Disponível em: <http://www.tinyos.net/>. Acesso em: 16 Dezembro 2011.

TSIATSIS, V.; KUMAR, R.; SRIVASTAVA, M. B. *Computation Hierarchy for in-network processing*. ACM Mobile Networks and Applications, p. 505–518, 2005.

VERDONE, R.; DARDARI, D.; MAZZINI, G.; CONTI, A. Wireless Sensor and Actuator Networks. [S.l.]: ACADEMIC PRESS, 2008. ISBN 0123725399.

WAN, J.; WANG, D.; XU, X. *Design of Wireless Sensor Network Management System based on ZigBee Technology*. 2nd International Workshop on Intelligent Systems and Applications (ISA), p. 1–4, 2010.

WATFA, M.; DAHER, W.; AZAR, H. A. *An Energy Aware Sensor Network Query Processing System*. INFOCOMP Journal of Computer Science, v. 8, n. 1, p. 37–44, 2009.

YAN, G.; DYKE, S. J.; SONG, W.; HACKMANN, G.; LU, C. *Structural damage localization with tolerance to large time synchronization errors in WSN*. Proceedings of the 2009 conference on American Control Conference, 2009.

YICK, J.; MUKHERJEE, B.; GHOSAL, D. *Wireless sensor networks survey*. Computers Networks, v. 52, n. 12, p. 2292–2330, 2008. ISSN 1389-1286.

ZHENG, Q.; BI, X. *An Improved Concurrency Control Algorithm for Distributed Real-Time Database*. International Conference on Advanced Management Science, p. 364–367, 2010.