UNIVERSIDADE FEDERAL DO PARÁ
CENTRO TECNOLÓGICO
PROGRAMA DE PÓS – GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Admilson de Ribamar Lima Ribeiro

SensorBus: Um *Middleware* Baseado em Políticas para Redes de Sensores sem Fio

BELÉM – 2007

UNIVERSIDADE FEDERAL DE PARÁ
CENTRO TECNOLÓGICO
PROGRAMA DE PÓS – GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Admilson de Ribamar Lima Ribeiro

SensorBus: Um *Middleware* Baseado em Políticas para Redes de Sensores sem Fio

Tese submetida à Universidade Federal do Pará como parte dos requisitos para obtenção do grau de doutor em Engenharia Elétrica

BELÉM – 2007

SENSORBUS: UM *MIDDLEWARE* BASEADO EM POLÍTICAS PARA REDES DE SENSORES SEM FIO

Este trabalho foi julgado adequado em 09/02/2007 pelo colegiado do Programa de Pós-Graduação em Engenharia Elétrica do Centro Tecnológico da Universidade Federal do Pará, como parte dos requisitos para obtenção do título de Doutor em Engenharia Elétrica, e aprovado na sua forma final pela banca examinadora composta pelos seguintes membros:

.....
Prof. Dr. João Crisóstomo Weyl Albuquerque Costa (UFPA)-Orientador

.....
Prof. Dr. Carlos Renato Lisboa Francês (UFPA)-Co-Orientador

.....
Prof. Dr. Aldebaro Barreto da Rocha Klautau Júnior (UFPA)-Membro

.....
Prof. Dr. Antônio Jorge Gomes Abelém (UFPA) – Membro

.....
Prof. Dr. Antônio Alfredo Ferreira Loureiro (UFMG) - Membro externo

.....
Prof. Dr. Solon Venâncio de Carvalho (INPE) – Membro externo

VISTO:

.....
Prof. Dr. Evaldo Gonçalves Pelaes
Coordenador do PPGEE/CT/UFPA

Aos meus batalhadores pais, Nestor Ferreira Ribeiro e Emelina de Lima Ribeiro. A ele, por ter vendido muitos frangos para custear meus estudos. A ela, por ter dormido várias vezes nas portas das escolas públicas a fim de conseguir uma vaga para mim. Eles sabiam, e somente eles, que um dia eu escreveria esta dedicatória.

AGRADECIMENTOS

Ao Professor João Crisóstomo, por ter acreditado no trabalho, pelo apoio logístico e incondicional, e pela orientação valiosa no decorrer de toda a pesquisa.

À Lílian Coelho, pelo trabalho que teve na instalação, configuração e programação da rede de sensores sem fio de testes.

Ao Professor Renato Francês pelo apoio a idéia do trabalho e orientação importante na fase inicial da pesquisa.

Ao Professor Antônio Loureiro pela colaboração no direcionamento que a pesquisa deveria seguir, por ocasião de minha qualificação ao doutorado.

Ao amigo e cunhado Renoir, pela revisão de todo o texto da tese, mesmo sacrificando suas férias.

À Liane Márcia que viabilizou minha inscrição e as passagens aéreas para que eu pudesse participar de congressos e conferências, além das passagens aéreas para meu deslocamento Aracaju-Belém.

Às amigas Silvana, Belém e Cláudia, pela acolhida e hospedagem em Belém para que eu pudesse realizar uma parte do trabalho.

Aos amigos Vanilson Gomes e Adelson Medeiros pela convivência agradável e divertida em que passamos juntos nesses anos de doutorado.

Finalmente, a minha família (Kátia, Ad e Ian) que foram obrigados a suportar minha ausência para que eu pudesse terminar esta tese.

Da próxima vez que alguém lhe disser algo que parecer importante, pense: "Será que isso é o tipo de coisa que as pessoas sabem por causa de provas? Ou será o tipo de coisa que as pessoas acreditam só por causa de tradição, autoridade ou revelação?". E, da próxima vez que alguém lhe disser que uma coisa é verdade, por que não perguntar: "Que tipo de prova há para isso?". E se ela não puder lhe dar uma boa resposta, eu espero que você pense com muito cuidado antes de acreditar em qualquer palavra daquilo que foi dito.

Richard Dawkins

RESUMO

Construir aplicações distribuídas com *middleware* libera o programador das preocupações (comunicação e coordenação entre componentes de *software*) impostas pelo ambiente distribuído. Além dessas preocupações, em redes de sensores sem fio consideram-se também suas características específicas (endereçamento, mobilidade, quantidade e recursos limitados dos nós sensores). Esta tese descreve o projeto, a implementação e a avaliação de desempenho de SensorBus, um *middleware* adaptativo orientado à mensagem para redes de sensores sem fio, baseado no paradigma *publish-subscribe*, que utiliza políticas para atender às diversas características das redes de sensores sem fio. As políticas são implementadas em um perfil de aplicação, através de metadados, codificados em documentos XML. O SensorBus incorpora linguagens de restrição e de consulta a fim de facilitar a construção de aplicações *on-line* pelos usuários.

Numa avaliação qualitativa, verificou-se a usabilidade do SensorBus implementando uma aplicação de monitoramento ambiental. Essa avaliação mostrou que a construção de aplicações distribuídas é bastante direta, sendo suficiente para isso que o programador de aplicações decida quais políticas deseja incorporar e as implemente no perfil de aplicação.

Numa avaliação quantitativa, verificou-se o desempenho do SensorBus. Essa avaliação foi realizada através de seus principais componentes: serviço de aplicação, serviço de mensagem e serviço de contexto. Para avaliar o serviço de aplicação, o desempenho de duas consultas usando-se a linguagem de restrição do SensorBus foram comparadas através de simulação. Resultados mostraram que a consulta que utilizou um filtro de aplicação consumiu 1 Joule menos de energia do que a consulta que não utilizou filtro. Observou-se também a eficiência da utilização de filtros no tempo de resposta em relação à consulta que não utilizou filtros. Assim, a utilização de filtros restringiu a movimentação de dados na rede e possibilitou significante economia de energia nos nós sensores. Para avaliar o serviço de mensagem utilizou-se a técnica de medição para realizar um experimento com objetivo de analisar o impacto

de dois protocolos de roteamento *multi-hop*. Esse experimento mostrou que diferentes protocolos de roteamento influenciam em aspectos importantes como vazão, porcentagem de pacotes entregue e consumo de energia. Para avaliar o serviço de contexto foi implementado um *benchmark* que utiliza como medida de desempenho o tempo decorrido. Demonstrou-se que o emprego de metadados para incorporar políticas no SensorBus não implica em custo adicional significativo em comparação com um *middleware* básico que não incorpora customização de políticas.

ABSTRACT

The use of a middleware to develop distributed applications liberates the programmer of the concerns (communication and coordination among software components) imposed by the distributed network environment. Besides these concerns, the design of a wireless sensor network must consider its specific characteristics (address, mobility, amount and limited resources of the sensor nodes). This thesis describes the design, implementation and performance evaluation of SensorBus, a message-oriented adaptive middleware for wireless sensor networks that uses policies to assist in several characteristics of the wireless sensor networks. Policies are implemented in an application profile, through metadata, encoded in XML documents. SensorBus incorporates constraint and query languages which will aid the development of interactive applications.

In a qualitative evaluation, the usability of SensorBus was verified implementing an application of environmental monitoring. This evaluation showed that the construction of distributed applications is quite direct; being enough for the application programmer to decide which policies he/she wants to incorporate and implement those policies in the application profile.

In a quantitative evaluation, the performance of SensorBus was verified through their main services: application service, message service and context service. To evaluate the application service, two queries of constraint language of SensorBus were compared using simulations. Results showed that the query that used an application filter consumed 1 Joule less of energy than the query that did not use a filter. It was also observed the efficiency of using filters in the response time with respect to the query that did not use filters. Thus, the use of filters restricted the movement of data in the network and made possible significant economy of energy in the sensor nodes. To evaluate the message service the measurement technique it was performed an experiment with the objective of analyzing the impact of two multi-hop routing protocols. That experiment showed that different multihop routing protocols influence the network behaviour in important aspects such as throughput, package delivery fraction and energy

consumption. To evaluate the context service, a benchmark was implemented and provides as performance metric the elapsed time. It was demonstrated that using metadata to incorporate policies in SensorBus does not imply in significant additional cost in relation to a basic middleware that does not incorporate customization of policies.

SUMÁRIO

LISTA DE SIGLAS

AAP	*Association of American Publishers*
ACQP	*Acquisitional Query Processing*
AODV	*Ad Hoc Demand Distance Vector*
A/D	*Analog-to-Digital*
API	*Application Programming Interface*
ASCII	*American Standard Code for Information Interchange*
AUTOSEC	*Automatic Service Composition*
BER	*Bit Error Rate*
COBOL	*Common Business Oriented Language*
CORBA	*Common Object Request Broker Architecture*
COTS	*Commercial Off The-Shelf*
CPU	*Central Processing Unit*
CSMA	Carrier Sense Multiple Access
DCOM	*Distributed Component Object Model*
DLLs	*Dynamic Link Libraries*
DOM	*Document Object Model*
DSR	*Dynamic Source Routing*
DSDV	*Destination-Sequenced Distance-Vector*
DSWARE	*Data Service Middleware*
DTD	*Document Type Definition*
EAR	*Eavesdrop and Register*
EBCDIC	*Extended Binary Coded Decimal Interchange Code*
EWMA	*Exponentially Weighted Moving Average*
FORTRAN	*Formula Translation*
HTTP	*HyperText Transfer Protocol*

IEEE	*Institute of Electrical and Electronic Engineers*
IIOP	*Internet Inter-ORB Protocol*
IP	*Internet Protocol*
ISO	*International Standards Organization*
JVM	*Java Virtual Machine*
Kbps	*Kilobits per second*
KVM	*Kilobyte Virtual Machine*
LEACH	*Low Energy Adaptive Clustering Hierarchy*
LEPS	*Link Estimation and Parent Selection*
LEPSM	*Link Estimation and Parent Selection Method*
LWIN	*Low-power Wireless Integrated Microsensor*
MANET	*Mobile Ad hoc Network*
Mbps	*Megabits per second*
MEMS	*Micro Electro-Mechanical Systems*
MIG	*Message Interface Generator*
MHz	*Megahertz*
MOM	*Message-Oriented Middleware*
NES	*Networks Embedded Systems*
NPDUs	*Network Protocol Data Units*
NS-2	*Network Simulator 2*
ODP	*Open Distributed Processing*
OMG	*Object Management Group*
ORB	*Object Request Broker*
OSI	*Open Systems Interconnection Reference Model*
PDAs	*Personal Digital Assistants*
PDF	*Packet Delivery Fraction*
PCs	*Personal Computers*

PEGASIS	*Power-Efficient Gathering in Sensor Information Systems*
QUASAR	*Quality-Aware Sensing Architecture*
QoS	*Quality of Service*
RAM	*Random Access Memory*
RF	*Radio Frequency*
RMI	*Remote Method Invocation*
RSSFs	*Redes de Sensores Sem Fio*
RSN	*Reactive Sensor Network*
RUNES	*Reconfigurable Ubiquitous Networked Embedded*
SAR	*Sequential Assignmnent Routing*
SEE	*Sensor Execution Environment*
SGML	*Standard Generalized Markup Language*
SINA	*Sensor Information Networking Architecture*
SMACS	*Self-Organizing Medium Access Control for Sensor Networks*
SQL	*Structured Query Language*
SQTL	*Sensor Querying and Tasking Language*
TCL	*Tool Command Language*
TCP	*Transmission Control Protocol*
TDMA	*Time Division Multiple Access*
TEEN	*Threshold Sensitive Energy Efficient Sensor Network Protocol*
TinyOS	*Tiny Microthreading Operating System*
TORA	*Temporally Ordered Routing Algorithm*
UCLA	*University of California, Los Angeles*
UDP	*User Datagram Protocol*
UML	*Unified Modeling Language*

URLs	*Uniform Resource Locators*
WAP	*Wireless Application Protocol*
WINS	*Wireless Integrated Network Sensors*
WMEMA	*Window Mean with Exponentially Weighted Moving Average*
W3C	*World Wide Web Consortium*
XML	*eXtensible Markup Language*
XPath	*XML Path*

LISTA DE FIGURAS

1 INTRODUÇÃO

1.1 MOTIVAÇÃO

Na década passada, duas tecnologias, comunicações móveis e a Internet, cresceram de uma maneira significativa. O serviço móvel celular obteve um crescimento muito grande de usuários juntamente com a redução dos custos dos planos para os serviços de voz. Por outro lado, a Internet proporciona o acesso a informações para os usuários fixos e móveis. Assim, a combinação dessas duas tecnologias possibilita o acesso à informação e serviços, a qualquer hora e em qualquer lugar, com os usuários acessando a informação na Internet através de diversos dispositivos móveis, como telefones celulares, PDAs (*Personal Digital Assistants*) e *laptops*. Além desses recentes avanços em redes sem fio, está ocorrendo o desenvolvimento de projetos de circuitos digitais de baixa potência, de novos materiais de sensoriamento e de micro sistemas eletromecânicos (MEMS - *Micro Electro-Mechanical Systems*). A combinação dessas tecnologias permite construir pequenos nós sensores com capacidade de processamento, sensoriamento e comunicação sem fio. Um grande número de nós sensores quando espalhados em uma região formam uma rede de sensores sem fio (RSSF), cujo objetivo principal é coletar e analisar os dados do ambiente físico (RENTALA *et al*., 2002).

Redes de sensores sem fio (RSSFs) consistem em um conjunto de nós distribuídos espacialmente em uma área física. Cada nó possui um ou mais sensores, como termômetro, manômetro, detector de luz, microfone, ou sensores biológicos e químicos (RENTALA *et al*., 2002). Além dos sensores, cada nó tem uma CPU (*Central Processing Unit*), um rádio de faixa estreita, uma fonte de energia representada normalmente por uma

bateria de baixa autonomia, pequena quantidade de memória e um ou mais atuadores. Os nós se comunicam entre si colaborando na tarefa de obter dados do ambiente. As RSSFs são empregadas em aplicações específicas como monitoramento ambiental, rastreamento de veículos, monitoramento de vida selvagem, entre outras. Esses exemplos mostram que a utilidade de uma RSSF baseia-se na sua capacidade de fornecer informações de áreas extensas em resposta às questões colocadas pelo usuário.

Apesar da diversidade dos nós sensores e do grande tamanho dessas redes é importante que o próprio usuário seja capaz de programar e gerenciar aplicações distribuídas que realizam a coleta de informações. Os modelos de computação distribuída tradicionais não são adequados por causa de algumas características específicas das RSSFs, listadas a seguir.

- **Endereçamento de nós sensores**: dependendo do tipo de aplicação, talvez seja mais conveniente acessar unicamente um determinado nó ou não. Por exemplo, na aplicação de rastreamento de veículos, pode-se querer saber se determinado veículo é uma moto, um carro, um ônibus ou um caminhão. Nesse caso, é necessário endereçar unicamente os veículos. Por outro lado, na aplicação de monitoramento ambiental, os nós sensores não precisam ser identificados unicamente visto que nesse tipo de aplicação, o importante é saber o valor de determinada variável em uma dada região.
- **Mobilidade de nós sensores**: em algumas aplicações, a mobilidade dos nós sensores deve ser considerada. Por exemplo, nós sensores espalhados em uma floresta para coleta de dados sobre a umidade e temperatura são estáticos, enquanto que nós sensores colocados em uma superfície de um rio para coleta de dados sobre a poluição do mesmo são móveis.
- **Quantidade de nós sensores**: a faixa de sensoriamento de cobertura de um nó sensor é menor que sua faixa de alcance de rádio. Além disso, muitas vezes nós sensores operam em ambientes com bastante ruído. Para obter uma resolução de sensoriamento confiável é necessária uma alta densidade de nós

sensores. Em algumas aplicações, o tamanho da área de cobertura leva a uma grande quantidade de nós sensores. Uma simples aplicação na área ambiental, como monitoramento de oceanos e floresta, requer centenas a milhares de nós sensores. Em outras aplicações, como em rastreamento de veículos, a quantidade de nós sensores é limitada pelo tamanho da frota.

- **Recursos limitados dos nós sensores**: na grande maioria das aplicações que executam em RSSFs, os nós são espalhados em uma região hostil, onde não é possível acessá-los para manutenção. O tempo de vida de cada nó sensor fica dependendo exclusivamente da pequena energia disponível para o nó. Para que se possa conservar a energia, a velocidade da CPU e a largura de banda do canal de RF (*Radio Frequency*) devem ser limitadas. Em algumas aplicações, como aplicações militares, os nós sensores devem ser pequenos para dificultar a detecção deles pelo inimigo. Esse novo requisito impõe restrições adicionais na velocidade de CPU, tamanho da memória, largura de banda de RF e mesmo no tamanho da bateria. Em aplicações em que os nós sensores não estão espalhados em uma região hostil é possível acessá-los para manutenção e o tempo de vida de cada nó sensor não se torna um fator crítico.

Observa-se que as aplicações estão estreitamente ligadas ao projeto das RSSFs. Para cada tipo de aplicação há necessidade de se alterar o mecanismo de comunicação para atender determinadas características. Essas características são, na maioria das vezes, abordadas e resolvidas nos diversos protocolos de roteamento usados em RSSFs. Nem todos os protocolos satisfazem essas características. Por exemplo, alguns são eficientes do ponto de vista de economia de energia nos nós sensores, mas não permitem endereçamento único; outros permitem endereçamento único, mas não são eficientes em economia de energia (ver seção 2.7.2).

A construção de aplicações distribuídas pode ser feita através da utilização das primitivas do sistema operacional de rede. Esse procedimento acarreta grandes dificuldades para o programador de aplicação, pois além de trabalhar com primitivas de baixo nível ainda deve se preocupar com

questões de comunicação e coordenação entre os componentes de *software* distribuídos na rede. Uma outra abordagem é a construção de um *middleware* para fornecer primitivas de mais alto nível e esconder, do programador, as preocupações impostas pelo ambiente distribuído.

Um *middleware* é a camada de *software* situada entre o sistema operacional e os componentes de aplicação, e tem como finalidade facilitar a comunicação e a coordenação desses componentes que estão distribuídos nos diversos computadores de uma rede (EMMERICH, 2000b). O objetivo principal de um *middleware* é fornecer aos programadores primitivas de alto nível que simplificam a construção de aplicações. Assim, eles ficam liberados de preocupações com controle de concorrência, gerenciamento de transações e comunicação de rede, permitindo concentrar esforços nos requisitos funcionais das aplicações. Um *middleware* é aplicado em sistemas distribuídos para fornecer: serviços de sistemas padronizados para diversas aplicações, um ambiente de execução para coordenar múltiplas aplicações e mecanismos para obter utilização eficiente de recursos de sistema.

A utilização de um *middleware* convencional (para sistemas distribuídos fixos) em ambientes sem fio não é adequada. Produtos de *middleware* existentes tratam como erro e levantam exceções quando não encontram um componente distribuído, e essa situação é mais regra do que exceção em ambientes sem fio. A baixa largura de banda das redes sem fio exige a otimização dos dados transportados e isso não é considerado em um *middleware* convencional. As primitivas de coordenação desse tipo de *middleware* não consideram as freqüentes desconexões que ocorrem nas redes sem fio. Um outro problema é a carga computacional desse tipo de *middleware* ser demasiadamente grande para os poucos recursos dos dispositivos sem fio. Finalmente, a transparência apresentada não é adequada, pois as aplicações precisam possuir informações sobre o contexto de execução para melhor adaptabilidade do *middleware*.

Novos tipos de *middleware* foram desenvolvidos para atender os requisitos introduzidos pelo ambiente sem fio (ROMAN *et al*., 2000), os quais foram baseados em três novos conceitos: reflexão computacional, espaço de tuplas e ciência de contexto. Um *middleware* baseado em reflexão computacional é projetado para que apresente carga computacional

leve e seja facilmente configurável. Um *middleware* baseado em espaço de tuplas é projetado para resolver os constantes problemas de desconexões e utiliza comunicação assíncrona de forma mais natural. Um *middleware* com o conhecimento do contexto de execução é desenvolvido para fornecer informações de contexto para aplicações. A construção de *middlewares* utilizando esses novos conceitos resolveu de maneira satisfatória os problemas impostos pelo ambiente sem fio, mas não são adequados para suportar as aplicações específicas utilizadas em RSSFs, pois eles são projetados para suportar a generalidade das aplicações tradicionais de sistemas de informação.

As RSSFs têm muitas semelhanças com redes sem fio tradicionais, incluindo energia limitada disponível em cada nó e canais de comunicação sujeito a erros. Contudo, a comunicação em RSSFs difere da comunicação fim a fim encontrada em outros tipos de rede sem fio (RENTALA *et al*., 2002). Em outras palavras, a função da RSSF é reportar informação considerando o fenômeno para o usuário que não está necessariamente ciente da infra-estrutura da rede e dos nós sensores individuais como um ponto final de comunicação. Além disso, a energia é mais limitada em RSSFs do que em outros tipos de redes sem fio devido à natureza dos dispositivos sensores e à dificuldade em recarregar suas baterias em regiões inóspitas. Estudos mostraram que o custo de energia na execução de 3000 instruções é igual ao custo de energia no envio de 1 (um) bit a 100 metros via rádio (POTTIE & KAISER, 2000).

O desenvolvimento e implantação de aplicações em RSSFs é uma tarefa difícil e trabalhosa. Por exemplo, na representativa plataforma TinyOS/MICAz (Hill *et al*., 2000), o desenvolvimento de aplicações através de componentes fortemente ligados ao *hardware* limita o desenvolvimento de aplicações flexíveis. Desse modo, uma vez implantadas, aplicações TinyOS podem ser levemente atualizadas alterando parâmetros que foram definidos antes da implantação. Normalmente, uma grande quantidade de nós espalhados em uma região inóspita, impossibilita que eles sejam manualmente coletados, reprogramados e implantados novamente. Ainda mais, memória e outros recursos computacionais são escassos e a comunicação via rádio é não confiável.

A aplicação de *middleware* em RSSFs é adequada devido à quantidade de informação que precisa ser processada por dispositivos que apresentam restrição de desempenho devido aos seus limitados recursos (YU et al., 2004). Um *middleware* tenta solucionar o problema de adaptação e flexibilidade de aplicações para RSSFs. Diversos tipos de *middleware* (BOULIS & SRIVASTAVA, 2003)(LIU & MARTONOSI, 2003) foram desenvolvidos para essas redes. Essas soluções suportam reconfiguração de rede através de código móvel. Um outro modo de reconfiguração é através de reprogramação de memória. Ambos os tipos de reconfiguração apenas abordam a questão da flexibilidade de aplicações deixando a questão da adaptação em aberto em RSSFs.

1.2 OBJETIVOS

Para abordar a questão de adaptabilidade de aplicações e também apresentar uma outra abordagem de reconfiguração, desenvolveu-se o SensorBus, um *middleware* orientado à mensagem para RSSFs, que permite a customização de políticas para atender às principais características de RSSFs. As políticas são divididas em políticas de aplicação (e.g., período de amostragem), comunicação (e.g., roteamento central ou distribuído) e contexto (e.g., nível de bateria). O SensorBus se comporta como um provedor de serviços customizáveis, onde a customização ocorre por meio de metadados, que codificam o comportamento do *middleware* para responder a requisições de serviços das aplicações em vários contextos. Pretende-se fornecer uma plataforma que possibilite ao usuário, o desenvolvimento de aplicações que fazem uso eficiente de energia nos nós sensores através da utilização de filtros. O uso de filtros restringe a movimentação de dados na rede. O SensorBus incorpora linguagens de restrição e de consulta a fim de facilitar a construção de aplicações *on-line* pelos usuários.

1.3 DESENVOLVIMENTO DO TRABALHO

Para contemplar os objetivos descritos acima, o trabalho foi desenvolvido com os seguintes passos:

- Analisou-se vários tipos de ferramentas para desenvolvimento de *software* para RSSFs;
- Analisaram-se vários tipos de *middleware* para redes sem fio para permitir a escolha de um sistema mais adequado para os propósitos do trabalho;
- Especificou-se um modelo de *middleware* que atenda aos requisitos necessários para computação sem fio;
- Analisaram-se diversos padrões de projeto existentes que possam ser utilizados na definição do modelo de *middleware*;
- Verificou-se diversos padrões de projeto que possam ser usados no projeto de implementação do SensorBus;
- Implementou-se um protótipo do SensorBus;
- Realizou-se simulações híbridas com um simulador acrescido de um ambiente de execução, para testes do SensorBus;
- Instalou-se uma RSSF para testes e avaliação do SensorBus;
- Definiu-se e implementou-se as linguagens de restrição e de consulta a serem utilizadas no *middleware* SensorBus;
- Realizaram-se simulações híbridas com um simulador acrescido de um ambiente de execução, para testes das linguagens de restrição e de consulta;
- Desenvolveu-se um filtro específico para aplicações de monitoramento ambiental.

1.4 ORGANIZAÇÃO DA TESE

A tese está estruturada em nove capítulos que obedecem à seguinte organização:

- Capítulo 1 – **Introdução** – Apresentam-se os fatos que motivaram o presente trabalho e os objetivos que visa atingir.
- Capítulo 2 – **Redes de Sensores sem Fio** – descrevem-se os componentes de uma RSSF. Apresentam-se suas principais

aplicações, desafios e estratégias. Por fim, abordam-se seus principais protocolos de enlace e de roteamento.

- Capítulo 3 – ***Middleware*** – abordam-se os principais mecanismos usados para construção de *middleware* para computação móvel. Apresentam-se os requisitos necessários para o desenvolvimento de um *middleware* para redes móveis e mostra-se como se pode utilizar um *middleware* em RSSFs.
- Capítulo 4 – **Abstrações e Mecanismos** – apresenta-se a aplicação alvo para levantar os requisitos necessários para o desenvolvimento do SensorBus e abordam-se as abstrações e mecanismos necessários para considerar os requisitos levantados.
- Capítulo 5 – **Configuração de Políticas** – descrevem-se as políticas incorporadas no SensorBus e suposições sobre o tipo de RSSF considerada.
- Capítulo 6 – **Arquitetura SensorBus** – apresenta-se a arquitetura do SensorBus assim como a descrição dos principais componentes que o constituem, descrevendo-se os passos necessários para o desenvolvimento de uma aplicação que o utilize.
- Capítulo 7 – **Implementação e Avaliação** – procede-se a uma avaliação quantitativa do SensorBus, em termos de avaliação de desempenho, e também uma qualitativa, em termos de usabilidade.
- Capítulo 8 – **Trabalhos Relacionados** – apresentam-se os principais trabalhos relacionados com a pesquisa, a fim de que se possam estabelecer distinções ante a solução proposta.
- Capítulo 9 – **Conclusões e Trabalhos Futuros** – conclui-se a tese, apresentando as principais contribuições e apontando novas opções de pesquisa.

2 REDES DE SENSORES SEM FIO

Como já afirmado anteriormente, uma RSSF consiste em um conjunto de nós distribuídos espacialmente em uma área física, que usam enlaces sem fio para realizar tarefas de sensoriamento distribuído. Os nós se comunicam entre si colaborando na tarefa de obtenção de dados do ambiente. A forma de comunicação dessas redes utiliza um esquema de retransmissão *multi-hop*, como mostra a figura 2.1 (ULMER, 2002). A maneira como os nós se comunicam com outros fora de seu alcance de transmissão é através de formação de uma cadeia de encaminhamento na qual os nós fonte e destino estão nas extremidades. Os pacotes são enviados salto-a-salto (*hop by hop*). Esse esquema de roteamento é utilizado devido à limitação da faixa de transmissão de rádio dos nós sensores.

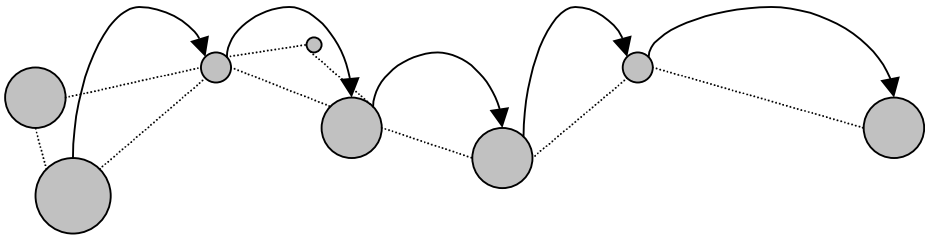


Figura 2.1: Transmissão *multi-hop* em uma RSSF.

Esse tipo de rede apresenta algumas características específicas, em que (RENTALA *et al.*, 2002):

- Os nós são geralmente estacionários após a implantação exceto alguns poucos nós móveis.
- A taxa de transmissão de dados é relativamente baixa na ordem de 1 kbps - 1 Mbps.

- A maior parte do fluxo de dados é unidirecional dos nós sensores para o usuário.
- Os nós sensores são dispositivos pequenos com baixo poder de processamento, capacidade de energia limitada e pouca memória.

2.1 REDES DE SENSORES SEM FIO VERSUS REDES SEM FIO TRADICIONAIS

As RSSFs diferem das outras redes sem fio (infra-estruturada ou *ad hoc*) em dois aspectos principais: implantação e operação. A implantação de RSSFs consiste em um processo contínuo e envolve atividades de espalhamento dos nós e formação da rede. A figura 2.2 (ULMER, 2002) mostra a implantação de uma RSSF.

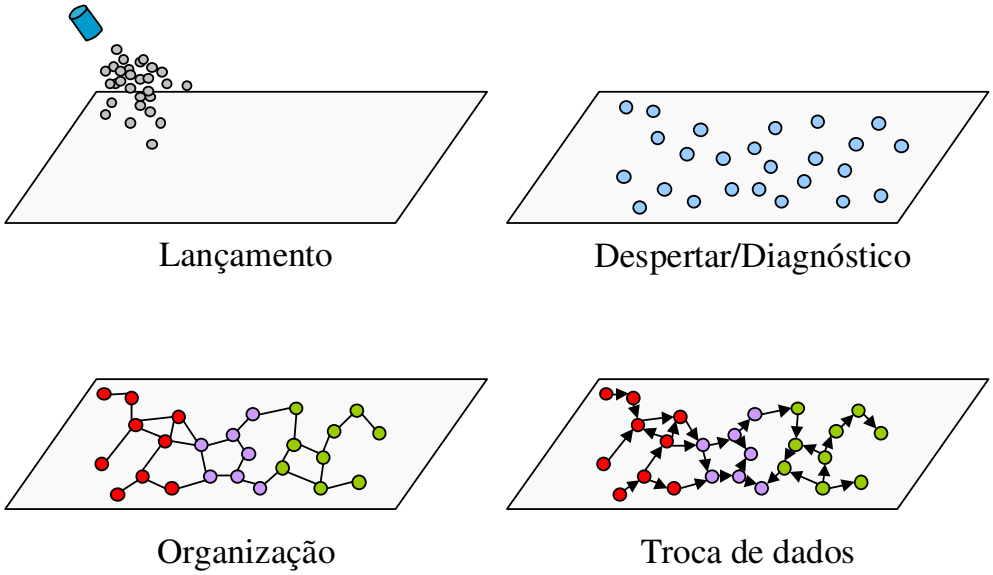


Figura 2.2: Implantação de uma RSSF.

Os nós sensores são algumas vezes lançados sobre a área a ser monitorada, caindo de forma aleatória e despertando para a formação da rede. Antes de começarem com as atividades de sensoriamento, os nós podem realizar tarefas de descoberta de localização e formação de agrupamentos (*clusters*). Os nós sensores colocados no meio físico são associados a objetos a serem monitorados, fato que aumenta a capacidade de processamento de informação dos objetos em estudo. Devido ao grande número de nós sensores necessários para monitoração de uma área, os nós

ficam operando de forma não-assistida após a implantação. Freqüentemente há necessidade de substituir nós inoperantes devido à influência ambiental ou perda de carga da bateria.

A operação de RSSFs é bastante diferente das redes sem fio tradicionais, estas sendo usadas basicamente para transferir dados entre nós. Em RSSFs, a grande quantidade de nós detecta vários aspectos do fenômeno em estudo, combinam suas informações e as transmitem para o usuário interessado na informação. Com isso, a detecção distribuída fornece mais robustez para obstáculos ambientais enquanto que o processamento distribuído internamente nos nós reduz o número de *bits* a ser transmitido, economizando largura de banda da rede (ESTRIN *et al*., 2001).

2.2 ÁREAS DE APLICAÇÃO DE REDES DE SENSORES SEM FIO

As RSSFs são vocacionadas para serem empregadas nas seguintes áreas:

- Controle – Para realizar tarefas de controle de processos em fábricas.
- Ambiente – Para monitorar variáveis ambientais em uma determinada região. O monitoramento pode ser em locais internos como prédios e residências ou locais externos como florestas, rios, lagos, desertos, etc.
- Segurança – Para fornecer segurança em centros comerciais.
- Medicina – Para controlar o estado clínico de um paciente pelo monitoramento de alguns órgãos vitais como o coração.
- Biologia - Para detectar a presença de substâncias nos organismos vivos que possam indicar a presença de algum problema biológico.
- Militar – Para detectar a presença de inimigos, explosões, materiais radioativos e gases venenosos.

Cotidianamente, já é possível relacionar alguns exemplos práticos de aplicações de RSSFs, como na (LOUREIRO *et al*., 2003):

- Produção industrial – Através do monitoramento em indústrias petroquímicas, fábricas, refinarias e siderúrgicas. As variáveis de interesse são pressão, temperatura, fluxo e nível. Os problemas de vazamento e aquecimento podem ser detectados com o auxílio desses parâmetros.
- Rede de distribuição de energia, gás ou água – Através do monitoramento em linhas de distribuição de energia e de sistemas de distribuição de gás ou água. As variáveis de interesse são fluxo, pressão, temperatura e nível.
- Área industrial – Mediante coleta e análise de dados em áreas de difícil acesso.
- Extração de petróleo e gás – Pelo monitoramento em plataformas em alto-mar.
- Indústria de aviação – Garantindo suporte a tecnologia *fly-by-wire*, onde sensores e atuadores sem fio começam a ser empregados para reduzir a quantidade de cabos necessários à implantação da rede.

De um modo geral, RSSFs podem ser usadas em monitoramento e segurança, controle e manutenção de sistemas complexos, e também em monitoramento de ambientes internos e externos. Essas áreas de aplicação são atendidas normalmente por sistemas de sensores centralizados, como imagem de satélite ou radares terrestres. As principais vantagens de RSSFs quando comparadas com sistemas de sensor centralizados são basicamente as seguintes (AGRE & CLARE, 2000):

- Permitir maior tolerância a falhas através de um nível alto de redundância.
- Fornecer cobertura de uma determinada área através da união de áreas cobertas por nós individuais.
- Estender a área de cobertura, reconfigurando o sistema quando um nó sensor falha.
- Melhorar o desempenho do sensoriamento com múltiplos tipos de sensores.

- Superar os efeitos ambientais colocando os nós sensores próximos aos objetos de interesse.
- Localizar um fenômeno discreto com um nó sensor de faixa limitada e combinar a informação com os outros nós.
- Monitorar regiões onde não existe infra-estrutura para reabastecimento de energia.

2.3 TAREFAS COMUNS EM REDES DE SENSORES SEM FIO

As aplicações utilizadas em RSSFs são muito diversificadas, como no caso de: monitoramento ambiental, manutenção baseada em condição, agricultura de precisão, transporte, instrumentação de fábrica e rastreamento de inventário, entre outras. Objetivando identificar os modos comuns de operação dessas aplicações, serão descritos na seqüência os cenários de duas aplicações representativas do domínio da pesquisa.

Em primeiro lugar a aplicação de rastreamento de veículos, que consiste em localizar um veículo ou objeto específico e monitorar seu movimento. Para detectar e identificar um objeto pode ser necessário analisar os dados obtidos por mais de um tipo de sensor, como por exemplo as imagens de uma câmera, a vibração de um sensor sísmico ou os ruídos de um sensor de áudio. Esses resultados são processados e comparados com a assinatura do objeto de interesse. A figura 2.3 mostra um cenário típico desse modo de operação.

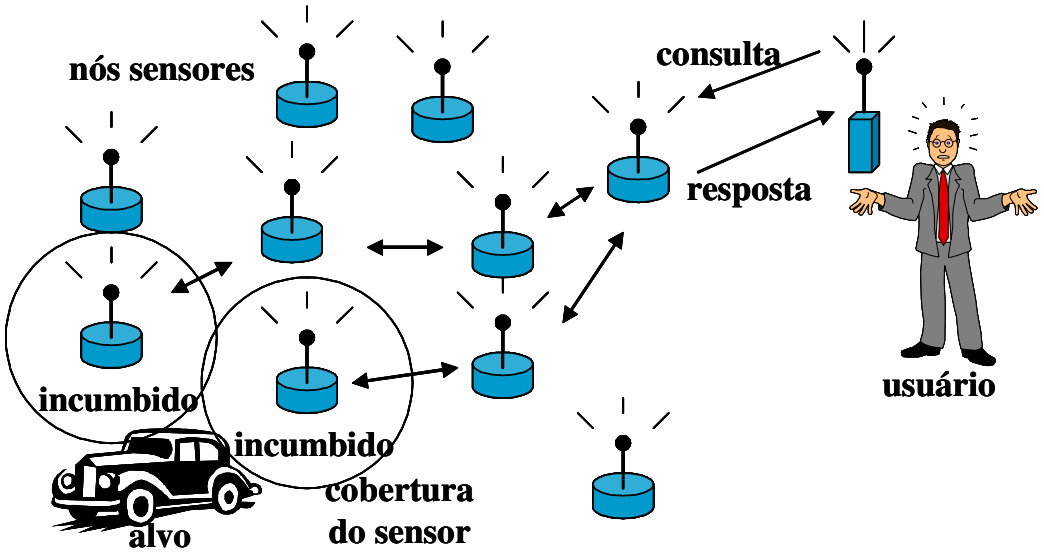


Figura 2.3: Cenário de operação de uma RSSF.

Nesse caso a consulta consiste em saber se existe algum veículo na região especificada pelo usuário. A consulta é encaminhada para os nós apropriados na rede, que são então incumbidos (*tasked*) de atender à consulta do usuário. Se um veículo é detectado, os nós indicam uma resposta positiva que é encaminhada de volta ao usuário.

Em segundo lugar no monitoramento ambiental, milhares de nós são espalhados em uma área em que se deseja inspecionar. Quando ocorrer algum evento programado, tal como a elevação da temperatura, o aumento do nível de CO_2 ou mudanças na taxa de umidade do ar, entre outros, dispara-se um módulo de gerenciamento de eventos que envia um sinal para o usuário. Nesse processo muitos nós sensores combinam suas detecções para fornecer a informação precisa. Usando-se RSSFs pode-se reduzir a quantidade de alarmes em um sistema de gerenciamento de fogo em florestas (ARRUE *et al*., 2000).

Os exemplos anteriores mostram que a utilidade de uma RSSF baseia-se na sua capacidade de fornecer informações de determinadas áreas em resposta às questões colocadas pelo usuário. Por exemplo, na aplicação de monitoramento ambiental pode-se desejar saber qual é o valor da temperatura, pressão atmosférica e umidade em diferentes locais. A abordagem de consulta (*query*) é o modo mais comum em que a rede é empregada. Em um outro modo de operação, nós sensores podem

permanecer em espera aguardando por alguma ocorrência específica. Por exemplo, na aplicação de rastreamento de veículos pode-se estar interessado em saber se há um determinado tipo de veículo trafegando em um cruzamento. A detecção de ocorrência de eventos de interesse é um outro modo de operação em que esse tipo de rede é empregada.

2.4 DESAFIOS E ESTRATÉGIAS EM REDES DE SENSORES SEM FIO

As características dos nós sensores (tipo de endereçamento, mobilidade, quantidade e recursos limitados) acarretam os seguintes desafios no projeto de RSSFs (ESTRIN *et al*., 2001):

- Topologia de rede dinâmica – Em outros tipos de rede sem fio como as redes celulares e *ad hoc* as mudanças na topologia são devidas à mobilidade dos nós. Em RSSFs a maioria dos nós é estacionária, mas mesmo assim a topologia da rede muda constantemente. Essa mudança ocorre devido à inoperância permanente ou temporária de nós da rede. Durante períodos de baixa atividade, nós podem ficar em estado de espera para conservar energia. Também, os nós podem ficar fora de serviço quando termina a energia fornecida pela bateria ou quando ocorre algum evento destrutivo que afeta o nó sensor. Falhas nos nós e obstruções ambientais, também contribuem para deixar nós sensores fora de serviço.
- Implantação *ad hoc* – A grande quantidade de nós sensores para cobrir uma área de difícil acesso ou inóspita exige que a rede se forme através da distribuição aleatória dos nós e forme conexões entre eles.
- Autonomia de energia e comunicação – A autonomia dos nós em relação à energia e comunicação combinada com a capacidade finita de energia armazenada pela bateria exige máximo foco em eficiência de energia.
- Operação não-assistida - A enorme quantidade de nós sensores exige que a rede opere de uma maneira não-assistida. Essa

operação requer que a configuração e a reconfiguração da rede seja automática.

Para enfrentar esses desafios, várias estratégias (mais comuns) são empregadas no projeto de RSSFs (ESTRIN *et al*., 2001):

- Processamento de sinal colaborativo – Esse tipo de processamento entre os nós melhora bastante a eficiência de energia. Uma vez que um estímulo comum tenha sido detectado ambos os processamentos (coerente e não-coerente) (ver seção 2.7.2) podem ser explorados.
- Exploração da redundância – É adequada quando o custo de implantação de um conjunto inicial de nós sensores é muito menor quando comparado com os custos de substituir nós danificados ou falhos ou de renovar recursos de nós. Essa redundância pode prolongar o tempo de vida útil do sistema. Quando os nós sensores não podem ser posicionados cuidadosamente, uma outra possibilidade para explorar redundância é pela extensão da cobertura usando um subconjunto de nós posicionados adequadamente.
- Processamento de sinal adaptativo – É usado para balancear a energia, precisão e rapidez dos resultados. A ocasião e fidelidade do processamento do sinal podem ser adaptadas observando os requisitos de recursos de energia e latência.
- Arquitetura hierárquica – Nós com maior energia ou capacidade podem ser usados para descarregar (*offload*) informação enquanto que nós com pequena capacidade podem ser usados mais próximos dos objetos de interesse. Em redes com nós homogêneos, criação de *clusters* e assinalamento de funções especiais para os nós principais contribuem para a escalabilidade, tempo de vida e eficiência do sistema como um todo.

2.5 TÉCNICAS USADAS EM REDES DE SENSORES SEM FIO

As RSSFs apresentam diversos requisitos operacionais e características que as distinguem de outras redes sem fio tradicionais, e

muito esforço de pesquisa está sendo feito atualmente para desenvolver soluções e projetos que melhor atentem para essas diferenças. O mais difícil desses requisitos é que todo *hardware*, *software* e protocolos para essas redes devem ser projetados para serem altamente eficientes, em termos de consumo de energia. Esse é o objetivo principal das técnicas – gerenciamento de energia, fusão de dados e auto-organização da rede - a serem apresentadas a seguir.

2.5.1 Gerenciamento de energia

Nós sensores, como visto anteriormente, são dispositivos com limitada capacidade de energia proveniente de uma bateria e, além disso, são geralmente espalhados para atuarem em uma região de difícil acesso, o que dificulta a possibilidade de reabastecimento de energia. Assim, uma RSSF deve operar com eficiência de energia para aumentar o tempo de vida dos nós sensores. Essa eficiência é alcançada quando a rede opera com uso mínimo de energia nos nós sensores. Portanto, é necessário que os componentes que constituem a rede (*hardware*, rede sem fio, protocolos e aplicações) sejam projetados considerando a conservação de energia como um dos seus aspectos mais importantes. Pode-se conservar energia em RSSFs de diversas maneiras: Projeto eficiente de *hardware* dos nós; Projeto eficiente de algoritmos e protocolos; agregação de dados; particionamento do sistema; topologias apropriadas.

Antes de decidir em que componentes da rede será aplicada a conservação de energia, padrões de consumo de energia de nós individuais e de toda a rede devem ser sistematicamente caracterizados e perfilados. Esse processo fornece informações onde aplicar compensações no projeto de algoritmos e *hardware* para os nós. As informações sobre o consumo de energia nos nós são apresentadas em modelos de energia enquanto que mapas de energia apresentam as informações da quantidade de energia disponível em cada parte da rede.

Um modelo de energia apresentado por Srivastava *et al*. (2000) introduz o conceito de consumidores e produtores de energia. O processador, rádio, e elementos de sensoriamento de um nó são os consumidores, enquanto que a fonte de bateria é o produtor. Cada entidade

consumidora notifica o provedor o seu consumo de energia que, por sua vez, informa a quantidade de energia disponível.

A partir do modelo de energia dos nós, é possível, através de um processo de obtenção de informações individuais, levantar o mapa de energia da rede. De posse desse mapa, pode-se tomar decisões mais adequadas do que deve ou pode ser feito na rede.

Zhao *et al*. (2002) apresenta a construção de um mapa de energia baseado em agregação. O procedimento de construção do mapa ocorre da seguinte maneira:

- Em cada nó, a energia residual local é medida periodicamente.
- A informação da energia residual local é disseminada através da rede.
- Ao longo do caminho para o nó destino, os nós que recebem duas ou mais informações de energia podem agregá-las de acordo com várias regras. O objetivo da agregação é reduzir o custo de se coletar a informação de energia através de pouca perda de informação.

2.5.2 Fusão de dados

Fusão de dados é o pré-processamento dos dados de forma distribuída aproveitando a capacidade de processamento dos nós sensores (LOUREIRO *et al*., 2003). É importante que os nós sensores compartilhem dados entre si para que a qualidade da saída final possa ser melhorada. Um simples nó pode reportar valores errados em relação à média de valores considerando diversos nós. Além disso, a cobertura de um simples nó pode não ser adequada para fornecer uma saída precisa, e é necessária a utilização de nós vizinhos para realizar a cobertura completa. Ambos os casos requerem uma forma de processamento colaborativo entre os nós para se obter os níveis de serviço e confiabilidade desejada pelo usuário. Outra motivação para fusão de dados está no fato que transmitir dados brutos sobre um enlace de rádio é mais dispendioso do que processar localmente nos nós e depois realizar a transmissão (POTTIE & KAISER, 2000).

Agregação de dados é um tipo especial de fusão de dados onde um conjunto de dados é condensado com o objetivo de reduzir seu volume. Para Cohen *et al*. (2001), a agregação de dados é um processo que inclui um conjunto de dados, a composição flexível e programada desses dados em um novo conjunto de dados refinados de menor volume e a entrega desse novo conjunto agregado aos seus consumidores. A vantagem desse método reside na redução do volume de dados que trafega pela rede e apresenta operações de agregação como *média*, *máximo*, *mínimo*, *contagem* e *supressão*, as quais são realizadas de forma distribuída pelos próprios nós sensores.

A utilização de fusão de dados traz como vantagem a economia de energia através da redução do volume de dados que trafega na rede. Contudo, é importante alertar para o fato de que o preço dessa economia é a demora na entrega dos dados das fontes ao destino. Além disso, ao se utilizar fusão de dados surgem novos problemas como sincronização, que consiste em se determinar quanto tempo e por quantos nós um nó sensor deve esperar antes de realizar a fusão.

2.5.3 Auto-organização da rede

Devido à implantação *ad hoc* e à topologia dinâmica, RSSFs devem possuir a capacidade de se ajustar a possíveis alterações sem interferência humana, o que é chamado de auto-organização (LOUREIRO *et al*., 2003).

A auto-organização de RSSFS consiste na capacidade de realizar mudanças estruturais sem intervenção humana, de forma a torná-las escaláveis e robustas face às características dinâmicas inerentes a esse tipo de rede. A auto-organização é uma atividade realizada pelos protocolos de comunicação sem fio que serão descritos na seção 2.7.

2.6 COMPONENTES DE *HARDWARE*

Os principais componentes de *hardware* de RSSFs são: nós sensores, interfaces de comunicação sem fio e nós para comunicação com outras redes (nós *gateway*).

Os nós sensores são dispositivos autônomos com capacidade de sensoriamento, processamento e comunicação. Cada nó possui um ou mais sensores tais como termômetro, manômetro, detector de luz, microfone, ou sensores biológicos e químicos. Além dos sensores, cada nó tem uma CPU (microprocessador ou um processador), rádio de faixa estreita, pequena quantidade de memória e, eventualmente, atuadores. A figura 2.4 (ULMER, 2002) mostra os componentes básicos de um nó sensor.

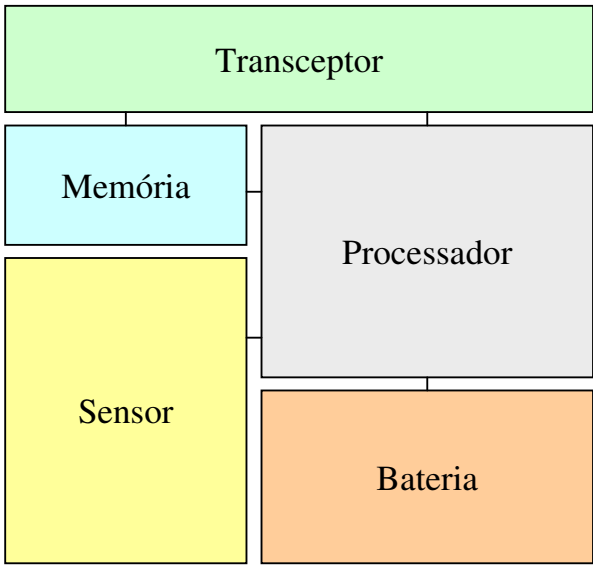
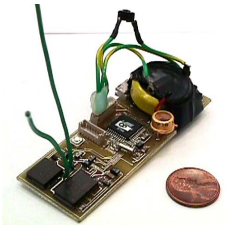


Figura 2.4: *Hardware* básico de um nó sensor.

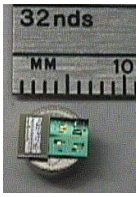
Os processadores são normalmente de 8 *bits* com freqüência de 10 MHz (*Megahertz)*, os transceptores possuem largura de banda na faixa de 1 kbps a 1 Mbps e a capacidade de memória fica compreendida entre 128 *kbytes* a 1 *Mbyte*. Na escolha da bateria deve-se considerar características como volume, condições de temperatura e capacidade inicial.

O projeto de um nó sensor é motivado pela necessidade de criar um dispositivo barato com um pequeno fator de forma e baixa dissipação de potência. A figura 2.5 (ULMER, 2002) apresenta diversos tipos de nós sensores sem fio resultantes de pesquisas em algumas instituições, como o Smart Dust (2002) da *University of California, Berkeley*, WINS (*Wireless Integrated Network Sensors*) (2002) da *University of California*, *Los Angeles* (UCLA) e JPL *Sensor Webs* (2002) do *Jet Propulsion Lab* da NASA.

UC Berkeley: COTS Dust



UC Berkeley: COTS Dust



UC Berkeley: Smart Dust

UCLA: WINS

Rockwell: WINS

JPL: Sensor Webs

Figura 2.5: Projetos acadêmicos de nós sensores.

O projeto Smart Dust usa nós sensores baseado em MEMS com tamanho de aproximadamente um milímetro cúbico que se comunicam com uma estação base via um enlace óptico. Esses nós não armazenam mais do que 1 Joule de energia e apresentam um consumo de potência na ordem de *microwatt*. Nós sensores individuais podem ser espalhados em uma região ou incorporados a objetos a serem monitorados. Um usuário pode então se comunicar com esses nós usando uma estação base móvel equipada com uma unidade transceptor.

No projeto WINS, em UCLA e Rockwell, foram desenvolvidos nós sensores que integram sensoriamento, processamento e comunicação em plataformas de micro-sensor. Esses nós sensores são fabricados usando tecnologia LWIM (*Low-power Wireless Integrated Microsensor*) e são capazes de formar redes *multi-hop* com mecanismos de autoconfiguração.

A comunicação entre uma RSSF e outras redes é realizada através de nós *gateways*. É tarefa dos nós *gateway* interceptar as mensagens que estão percorrendo a RSSF e encaminhá-las para uma outra rede, como a Internet, até um computador que está executando uma aplicação. A figura 2.6 mostra uma RSSF conectada a uma rede fixa através de um nó *gateway*.

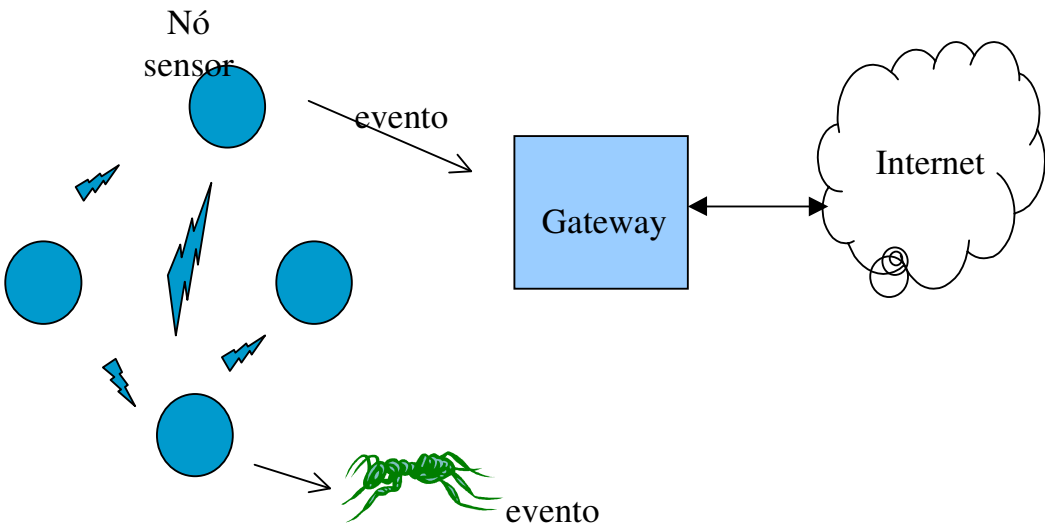


Figura 2.6: Uma RSSF com um *gateway*.

A grande variedade de aplicações em que se utilizam sensores e atuadores (transdutores) leva à fabricação de diferentes tipos de transdutores por diversos fabricantes, o que acarreta grandes problemas para interconexão desses dispositivos. Para resolver esse problema foi proposto o padrão IEEE 1451 (IEEE1451, 1998) que define uma interface de comunicação para transdutores. A figura 2.7 (IEEE1451, 1998) mostra a arquitetura desse padrão. Com uma interface que segue esse padrão conectam-se diferentes transdutores em uma rede, como mostra a figura 2.8.

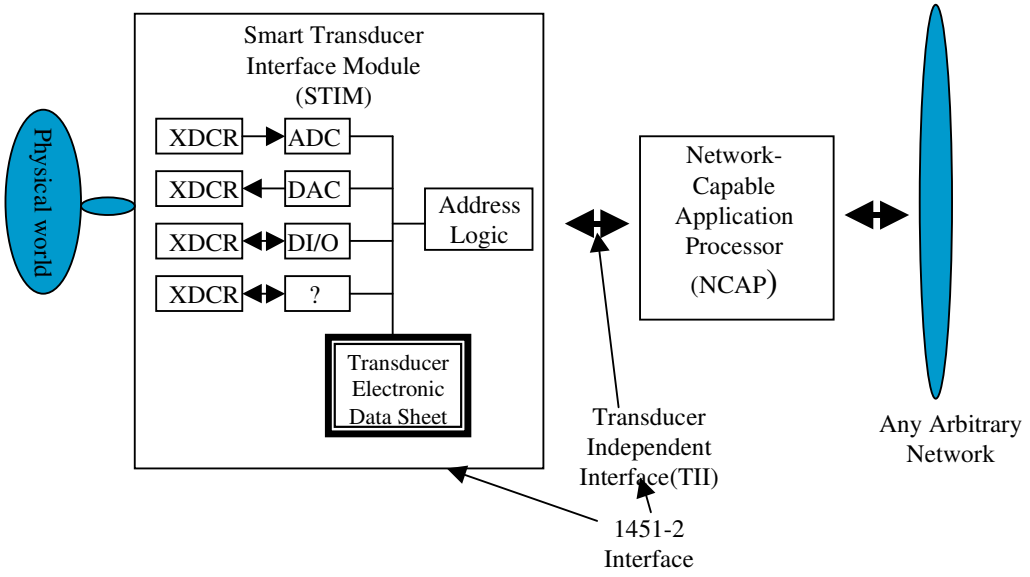


Figura 2.7: Arquitetura do padrão IEEE 1451.

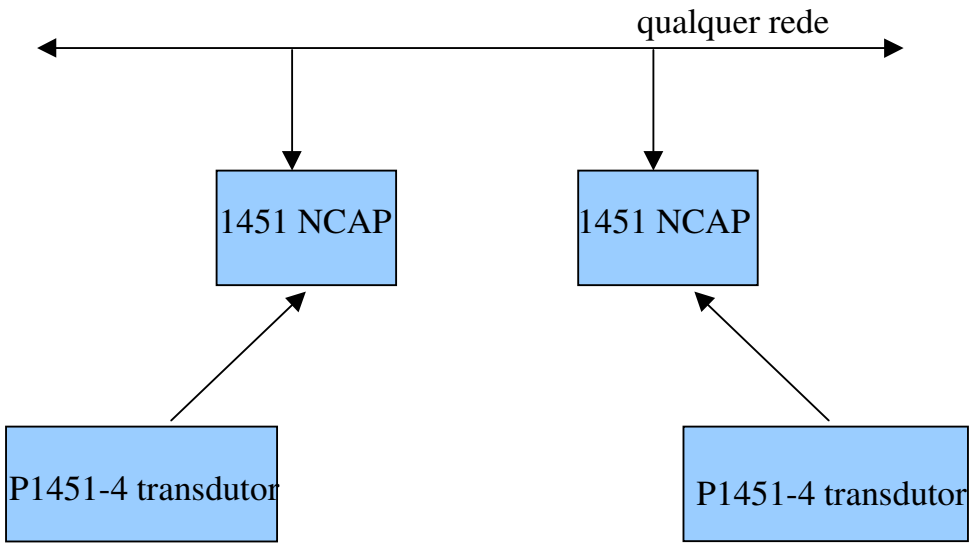


Figura 2.8: Transdutores usando o padrão IEEE 1451.

2.7 PROTOCOLOS DE COMUNICAÇÃO SEM FIO

2.7.1 Protocolos da camada de enlace

Os métodos de acesso a canal compartilhado em redes *ad hoc* podem ser divididos em duas categorias: métodos baseados em contenção e métodos organizados. Os esquemas baseados em contenção não são

adequados para RSSFs devido à atividade de continuamente sentir o canal de acesso e gastar recursos sempre que uma colisão ocorre.

Os métodos organizados de acesso ao canal tentam determinar a conectividade entre os nós em primeiro lugar e depois tratam do assinalamento de *slots* de canal de uma maneira hierárquica, formando *clusters* e assinalando nós principais para esses *clusters*. Essa solução deve ser distribuída porque a sincronização de uma RSSF grande seria um procedimento que gastaria muita energia. Outra razão pela utilização de algoritmos distribuídos é que eles se comportam bem com o aumento do tamanho da rede e são robustos para partições da rede e falhas nos nós.

2.7.2 Protocolos da camada de rede

Os protocolos de roteamento podem ser, de uma maneira geral, divididos em protocolos de roteamento plano e protocolos de roteamento hierárquico.

Os protocolos de roteamento plano podem ser, adicionalmente, subdivididos em roteamento de rede cooperativo e *multi-hop*. O roteamento *multi-hop* envolve roteamento entre uma fonte e um sorvedouro, normalmente um nó móvel. Os protocolos de roteamento *multi-hop* usados em redes *Ad hoc* como AODV (*Ad Hoc Demand Distance Vector*) (PERKINS & ROYER, 1999) e TORA (*Temporally Ordered Routing Algorithm*) (PARK & CORSON, 1997), normalmente não são adequados para RSSFs. Esses protocolos eliminam o alto custo na atualização de tabelas que ocorrem em cenários de alta mobilidade fornecendo um sistema sob demanda. Então é preferível um sistema baseado em tabela para RSSFs. Para roteamento *multi-hop*, a robustez é necessária então muita energia pode ser gasta para manutenção e configuração da rota.

O roteamento cooperativo envolve cooperação entre nós sensores que detectam um alvo comum e enviam dados para agregação a um nó central para adicional processamento. No roteamento cooperativo, quando o tráfego de dados é leve, a redução do custo com a configuração da rota assume importância significativa.

Os protocolos de roteamento plano *multi-hop* podem também ser subdivididos de acordo com técnicas de processamento de sinal, as quais pode se apresentar em dois tipos: não-coerente e coerente. No processamento não-coerente, os dados brutos são processados no próprio nó antes de serem enviados para um nó central. No processamento coerente, há um processamento mínimo no nó antes dos dados serem enviados para um nó central. Para algoritmos eficientes de consumo de energia, o processamento coerente é importante desde que o tráfego de dados seja baixo enquanto que um caminho otimizado é sempre importante para esse tipo de processamento.

Há diversos protocolos de roteamento para RSSFs tais como *Sequential Assignment Routing* (SAR) (SOHRABI *et al*., 2000), *Directed Diffusion* (ESTRIN *et al*., 1999), *Minimum Cost Forwarding Algoritm for Large Sensor Networks* (YE *et al*., 2001), *Sensor Protocols for Information via Negotiation* (SPIN) (HEINZELMAN *et al*., 1999), *Low Energy Adaptive Clustering Hierarchy* (LEACH) (HEINZELMAN *et al*., 2000), *Threshold sensitive Energy Efficient sensor Network protocol* (TEEN) (MANJESHWAR & AGRAWAL, 2001) e *Power-Efficient Gathering in Sensor Information Systems* (PEGASIS) (LINDSEY & RAGHAVENDRA, 2001).

2.8 CONSIDERAÇÕES

O objetivo principal das redes sem fio convencionais é fornecer alta vazão com baixo atraso e uma eficiente largura de banda em face da mobilidade dos nós. Em RSSFs, o interesse é o prolongamento do tempo de vida da rede, visto que os nós dependem de alimentação de pequenas baterias que, tipicamente, não são substituídas devido aos nós estarem operando em regiões hostis. Assim, o projeto de *software* para RSSFs deve considerar algoritmos que sejam eficientes do ponto de vista da economia de energia.

3 MIDDLEWARE

A possibilidade de acesso à informação e serviços a qualquer hora e em qualquer lugar está moldando uma nova sociedade de informação, com os usuários acessando a informação na Internet através de diversos dispositivos móveis, tais como telefones celulares, *personal digital assistants* (PDAs) e *laptops*. Entretanto, alguns problemas ainda precisam ser solucionados para que a computação distribuída móvel torne-se confiável e eficiente. Os principais fatores que limitam o desenvolvimento de sistemas de *software* para redes móveis são devidos à mobilidade dos usuários, à rede de comunicação sem fio e à portabilidade dos dispositivos.

Esses fatores acarretam muitos problemas no desenvolvimento de *software* para esse ambiente, tais como: algoritmos distribuídos precisam ser alterados para levar em conta a mobilidade dos usuários; a baixa confiabilidade do canal de comunicação sem fio acarreta freqüentes desconexões dos dispositivos; a localização dos dispositivos móveis deve ser feita freqüentemente, aumentando o custo de cada comunicação; esses dispositivos apresentam baixa autonomia de energia; entre outros (FORMAN & ZAHORJAN, 1994).

Apesar dessas limitações, vários acontecimentos estão tornando viável o desenvolvimento de sistemas de *software* para o ambiente móvel (EMMERICH, 2000b): organizações internacionais começaram um processo de definir novos padrões para serviços de dados sem fio para grandes áreas que suportará taxas de dados mais altas; recentes avanços tecnológicos têm aumentado os recursos (memória, processadores, etc) dos dispositivos sem fio; novas abordagens de desenvolvimento de *software* foram definidas, tais como adaptação e código móvel; novas arquiteturas de

software estão sendo pesquisadas para esse domínio, sendo uma delas a utilização de plataforma de sistemas distribuídos móveis também chamados de *middleware* móvel (CAPRA et al., 2001a).

Um *middleware* para redes móveis parece ser uma boa solução para os desafios impostos pelo ambiente sem fio, visto que a finalidade dele em sistemas distribuídos fixos é facilitar as interações entre os componentes que constituem a rede de computadores. Em redes sem fio, além das questões tratadas para a rede fixa, o *middleware* terá a função de atender os novos requisitos que surgem dos problemas inerentes à computação móvel.

Parece óbvia, então, a utilização em ambientes móveis de *middleware* para sistemas distribuídos fixos, mas isso pode não ser de todo adequado. Produtos de *middleware* existentes tratam como erro e levantam exceções quando um componente distribuído (cliente ou servidor) não é encontrado, e essa situação é mais regra do que exceção em ambientes sem fio. A baixa largura de banda das redes sem fio exige que haja otimização dos dados transportados e isso não é considerado em um *middleware* convencional. As primitivas de coordenação de um *middleware* convencional não consideram as freqüentes desconexões que ocorrem nas redes sem fio.

Assim, uma nova abordagem está sendo pesquisada nos dias atuais. Trata-se do desenvolvimento de *middleware* próprio para computação móvel. Esse tipo de *middleware* trata das soluções das questões levantadas acima, utilizando novas técnicas de reflexão computacional, metadados e espaço de tuplas.

3.1 *MIDDLEWARE* PARA SISTEMAS DISTRIBUÍDOS MÓVEIS

Sistemas distribuídos são aqueles que os componentes de *software* e de *hardware* localizados em uma rede de computadores (fixa ou sem fio) se comunicam apenas através de troca de mensagens devido à ausência de uma memória compartilhada (COULORIS *et al*., 2001). A figura 3.1 ilustra um exemplo de um sistema distribuído. Essa distribuição de componentes

acarreta diversos problemas (requisitos) que devem ser solucionados, através de decisões de projeto, pelos desenvolvedores de tais sistemas.

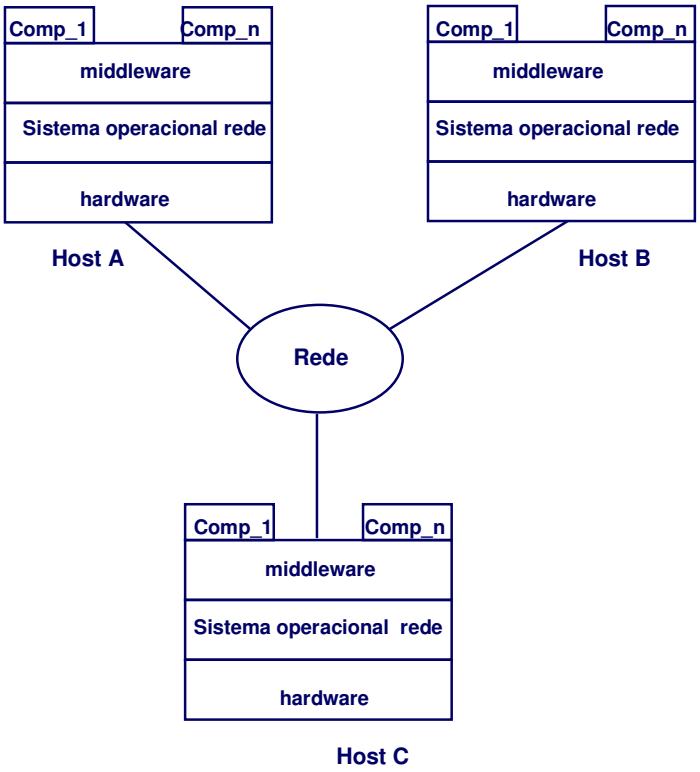


Figura 3.1: Exemplo de um sistema distribuído.

A utilização de *middleware* desenvolvido em cima de sistemas operacionais fornece aos programadores um maior nível de abstração, escondendo a complexidade introduzida pela distribuição. O objetivo do *middleware* é tornar transparente a distribuição para o programador. Um *middleware* implementa as camadas de sessão e apresentação do modelo de referência OSI (*Open Systems Interconnection*) (STALLINGS, 2001), como ilustra a figura 3.2. Seu principal objetivo é permitir a comunicação entre componentes distribuídos. Para fazer isso, ele fornece aos programadores de aplicação um nível de abstração mais alto construído através de primitivas do sistema operacional de rede.

Figura 3.2: Modelo de referência ISO/OSI.

Os requisitos que um *middleware* para sistemas distribuídos móveis deverão atender são aqueles já suportados pelos tipos de *middleware* convencional para redes fixas, tais como confiabilidade, escalabilidade, heterogeneidade, entre outros. Além disso, há a exigência dos novos requisitos impostos pela mobilidade do ambiente sem fio, como localização do elemento móvel, capacidade de migrar funcionalidade, uso eficiente dos recursos do dispositivo sem fio, entre outros. A seguir, apresentam-se os requisitos e mecanismos necessários para construção de *middleware* para sistemas distribuídos móveis.

3.1.1 Requisitos impostos pelo ambiente móvel

Software para computação distribuída móvel executa em pequenos dispositivos sem fio, tais como PDAs, telefones celulares, etc., que estão conectados em redes sem fio. Esse ambiente apresenta características próprias, a saber: limitação de recursos dos dispositivos móveis (pouca memória, bateria de vida curta, CPU de baixa velocidade); limitações apresentadas pela rede sem fio e mobilidade dos usuários.

Essas características introduzem novos requisitos que os sistemas de *software* devem satisfazer: capacidade de localizar e endereçar elementos móveis, capacidade de perceber mudanças no ambiente de execução, capacidade de se autoconfigurar, capacidade de migrar funcionalidade, uso eficiente dos recursos no elemento móvel e do canal de comunicação sem fio, alto grau de tolerância a falhas e mecanismos para autenticação e cifragem de dados (ENDLER & SILVA, 2001).

3.1.1.1 Ciência do contexto de execução (*Context-awareness*)

Devido às variações na disponibilidade dos recursos locais dos dispositivos sem fio e também variações nas características do canal de comunicação sem fio, faz-se necessário o monitoramento contínuo do contexto de execução, o qual se refere a tudo que pode afetar o comportamento de uma aplicação. Podem-se identificar dois níveis de ciência de contexto: ciência de dispositivo e ciência de ambiente. Ciência de dispositivo refere-se a tudo que reside no dispositivo físico onde a aplicação está executando, por exemplo, memória, tamanho da tela, energia da bateria, poder de processamento, etc. Ciência de ambiente refere-se a tudo que está fora do dispositivo físico, por exemplo, largura de banda, conectividade de rede, localização, outros computadores, etc.

Ciência do contexto de execução requer que o programador de aplicação saiba qualquer informação coletada do sistema operacional de rede, como por exemplo a localização do dispositivo. A capacidade de localização e endereçamento é necessária devido à mobilidade dos usuários e pode ser feita em diversos níveis do sistema. A própria infra-estrutura de uma rede sem fio pode fornecer esse serviço, como ocorre no padrão IS-41 dos sistemas celulares de segunda geração. Alguns protocolos de rede já fornecem essa facilidade como IP (*Internet Protocol*) móvel. As aplicações móveis precisam satisfazer esse requisito se não for suportada em outro nível do sistema.

Os mecanismos de monitoramento podem ser feitos através de *polling*, notificação de eventos ou de uma combinação das duas e pode ocorrer em qualquer nível do sistema (*hardware*, *firmware*, *software* de sistema ou aplicação). O monitoramento do ambiente de execução fornece

informações necessárias para adaptação dinâmica do sistema para um novo ambiente de execução. Assim, o sistema deve ser projetado para suportar mecanismos de configuração dinâmica.

3.1.1.2Carga computacional leve

A limitação dos recursos dos dispositivos móveis (pouca memória, bateria de vida curta, CPU de baixa velocidade) obriga que o sistema de *software* que executa nesses dispositivos tenha baixa carga computacional em detrimento de novos requisitos não funcionais que poderiam ser acrescentados. Há um compromisso entre a carga computacional que o dispositivo suporta e os requisitos não funcionais para possibilitar um mínimo de sobrecarga possível.

3.1.1.3Comunicação assíncrona

Em um ambiente de rede sem fio, os componentes distribuídos no dispositivo móvel conectam-se à infra-estrutura de rede para acessar dados e solicitar serviços. É comum que quando o cliente está conectado, o servidor que forneceria determinado serviço não está, isto é, o cliente e o servidor normalmente não executam ao mesmo tempo. Uma forma de comunicação assíncrona é necessária para fornecer interações entre componentes cliente e servidor que não executam ao mesmo tempo.

3.1.1.4Segurança

Os requisitos de segurança específicos desse ambiente são devidos ao fato de que o meio de comunicação sem fio é um meio acessível a todos e também à portabilidade dos dispositivos que podem ser perdidos ou furtados. Utiliza-se de criptografia para evitar que os dados sejam acessados por intrusos que monitoram o meio sem fio, ao passo que é exigida a autenticação do dono do dispositivo para evitar que intrusos acessem aos dados locais ou remotos. Essa autenticação pode ser feita por *smart cards* que são inseridos no dispositivo móvel.

3.1.1.5 Tolerância a falhas

Em uma rede móvel, os dispositivos podem falhar devido principalmente à falta de energia. Em casos em que o reabastecimento não seja possível, o sistema deve prover mecanismos de tolerâncias a falhas que possam garantir o funcionamento da rede.

3.1.2 Mecanismos para construção de *middlewares* móveis

Apesar de já ser muito utilizado em produtos comerciais, um *middleware* convencional ainda sofre de várias limitações que impossibilita seu uso em muitos domínios de aplicações. Eles não são ainda seguros, e são usados somente nos limites de uma rede local, não sendo adequados para redes sem fio. Assim, novos mecanismos de construção de *software* estão sendo pesquisados para suportar uma nova geração de *middleware*, alguns dos quais serão abordados nas próximas subseções.

3.1.2.1 Reflexão computacional

Reflexão computacional permite a um programa acessar, raciocinar sobre e alterar sua própria interpretação (ELIASSEN *et al.*, 1999). O mecanismo de reflexão tem sido usado principalmente em linguagens de programação como Java, onde o programa acessa sua própria implementação com a finalidade de prover dinâmica de execução. Em *middleware*, o mecanismo de reflexão permite que as aplicações forneçam informações do ambiente de execução para que o *middleware* se adapte adequadamente. O uso de reflexão objetiva criar, ler e modificar o perfil das aplicações, de modo que mudanças no contexto de execução possibilitem alterações no comportamento do *middleware* (CAPRA *et al*., 2001b).

3.1.2.2 Código móvel

Computação móvel, da perspectiva da engenharia de *software*, é o estudo de sistemas de *software* em que os componentes computacionais podem alterar suas localizações (ROMAN *et al*., 2000). A mobilidade desses componentes pode ser de dois tipos: física e lógica. A mobilidade física é o movimento de computadores em relação a outros computadores que

normalmente formam uma rede, sendo que esse movimento pode ser dentro de um determinado prédio ou mesmo em áreas dentro de cidades. A mobilidade lógica é movimento de código e dados entre computadores. A mobilidade física é vista como um requisito de projeto que deve ser considerado, ao passo que a mobilidade lógica é uma nova técnica de projeto para os desenvolvedores. Essa diferença influencia nas características do *middleware* a ser projetado. Um *middleware* que trata mobilidade lógica permite a transferência de código e dados, provendo assim uma maior adaptação da aplicação móvel. Um *middleware* para mobilidade física tenta minimizar o impacto da mobilidade dos computadores na aplicação, isto é, tenta fornecer uma transparência de mobilidade.

A possibilidade de ocorrência de desconexões entre o elemento móvel e a rede fixa, e também a necessidade de poupar recursos do dispositivo móvel, tornam necessária a existência de mecanismos que possibilitem a migração de tarefas de processamento dos elementos móveis para a rede fixa e vice-versa.

3.1.2.3Trading

Em ambientes distribuídos, antes de um componente cliente solicitar uma requisição, ele deve identificar unicamente um componente servidor, obtendo assim uma referência desse componente. Normalmente, a resolução de nomes é feita por um serviço de nomes. CORBA (*Common Object Request Broker Architecture*) (OMG, 1996) tem um serviço de nomes próprio; Java/RMI (*Remote Method Invocation*) (SUN, 2000) usa um RMIRegistry e assim por diante.

Trading surge como uma alternativa aos serviços de nomes e oferece maior flexibilidade. A idéia é semelhante às páginas amarelas das listas telefônicas, onde em vez de se procurar o componente pelo seu nome, o mesmo é localizado pelo tipo de serviço que oferece. Para que os componentes possam ser encontrados é preciso que eles registrem os serviços que oferecem a um servidor *Trader*. O *trader* faz um *match* com as requisições solicitadas com os serviços que ele oferece, e se o *match* for positivo o *trader* retorna a referência do componente servidor para o

componente cliente. Em seguida o servidor e o cliente ficam se comunicando sem envolvimento do *trader*.

3.1.2.4 Transporte ao nível da aplicação

A combinação de XML (*eXtensible Markup Language*) com *middleware* foi investigada por Emmerich (EMMERICH et al., 1999), onde foi sugerida a utilização de XML para o transporte de dados entre as aplicações usando *middleware*. Isso é possível devido à possibilidade de XML suportar traduções semânticas entre estruturas de dados e também devido à grande disponibilidade de linguagens de marcação, tais como FpML (FPML, 1999) e FIXML (INFINITY, 1999), adequadas para transportar mensagens entre aplicações.

É esperado que a interoperabilidade fornecida pelo *middleware* se dê do ponto de vista da aplicação, devido à iniciativa da OMG (*Object Management Group*) de começar um processo de adoção de interoperabilidade entre estruturas de dados CORBA e documentos estruturados XML (OMG, 1999).

3.1.2.5 Metadados

Metadados é um termo genérico aplicado a qualquer dado que descreva um outro. Pode ser utilizado para descrever o conteúdo de um documento, como seu título, autor, tamanho do arquivo, data da criação, etc. Metadados podem ser usados para gerência, pesquisa e filtragem de documentos.

Em *middleware*, metadados podem ser usados para codificar um perfil de aplicação informando como o *middleware* deve se comportar quando ocorrer um contexto de execução particular. Assim, é possível à aplicação fornecer diretivas que devem ser seguidas em certos contextos de execução (CAPRA *et al*., 2001b).

3.1.2.6 Espaço de tuplas

O ambiente sem fio requer que o paradigma de comunicação seja assíncrono de modo que o cliente e o servidor possam executar de modo

independente. Além dessa independência, seria melhor que os componentes pudessem também ser desacoplados, isto é, os componentes podem continuar a execução mesmo que ocorra uma desconexão na rede. Devido a isso um novo paradigma de comunicação está sendo usado e que se denomina de Espaço de Tuplas (*Tuple Spaces*) (CAPRA *et al*., 2001b).

Esse paradigma não é novo e nem foi projetado para esse propósito, pois sua origem foi para prover coordenação para a linguagem de programação concorrente Linda (GELERNTER, 1985). Nessa linguagem, um espaço de tuplas é um espaço de memória global e compartilhado, usado para os processos se comunicarem. O espaço de tuplas atua como um repositório de estruturas de dados chamados de tuplas que podem ser vistas como um vetor de valores tipados. As tuplas são os elementos básicos desse modelo, as quais são criadas pelos processos e colocadas no espaço de tuplas através da primitiva *write,* podendo ser acessadas concorrentemente pelos demais processos através das primitivas *read* e *take*. As tuplas não possuem nomes e são acessadas pelo mecanismo de *matching* de conteúdo. As comunicações são desacopladas no tempo e espaço e os componentes não precisam estar disponíveis ao mesmo tempo porque as tuplas têm seu próprio ciclo de vida, independente do processo que as criou.

Esse mecanismo é adequado para um *middleware* para computação móvel devido à grande ocorrência de desconexões e mudanças de localização dos componentes. Alguns *middlewares* baseados em espaços de tuplas têm sido desenvolvidos para aplicações de computação móvel, como Lime (MURPHY *et al.*, 2001), Tspaces (WYCKOFF *et al*., 1998) e L2imbo (DAVIES *et al*., 1998).

3.2 *MIDDLEWARE* EXISTENTES PARA REDES MÓVEIS

Alguns *middlewares* tradicionais foram aplicados em redes móveis. CORBA/IIOP (*Internet Inter-ORB Protocol)* foram adaptados com sucesso para redes sem fio e usados em um ORB (*Object Request Broker*) mínimo para dispositivos móveis (HAAHR *et al*., 1999). Uma adaptação de IIOP especificamente para redes móveis foi desenvolvida no projeto DOLMEN (REYNOLDS & BRANGEON, 1997). WAP (*Wireless Application Protocol)* é

utilizado conjuntamente com IIOP para permitir o uso de serviços CORBA de uma rede fixa através de dispositivos móveis conectados a um *gateway* WAP (REINSTORF *et al*., 2001).

A abordagem acima apresenta diversos problemas quando utilizada em um ambiente móvel. A comunicação síncrona não é adequada para ambientes com muitas desconexões. Um outro problema é a carga computacional desses *midllewares* ser demasiadamente grande para os poucos recursos dos dispositivos móveis. Finalmente, a transparência apresentada nem sempre é adequada para aplicações móveis que precisam possuir alguma informação sobre o contexto de execução para uma melhor adaptabilidade do *middleware*.

Assim, novos tipos de *middleware* têm sido desenvolvidos apropriadamente para o ambiente móvel com o intuito de resolver o problema da mobilidade (ROMAN *et al*., 2000), os quais podem ser agrupados em três grandes classes, cada classe correspondendo a um requisito importante imposto pela mobilidade que o *middleware* deve atender. Os principais requisitos impostos pela mobilidade são (CAPRA *et al*., 2000): carga computacional leve para que possa ser suportada pelos dispositivos móveis; comunicação assíncrona para contornar os problemas causados pelas freqüentes desconexões; e conhecimento do contexto de execução permitindo à aplicação realizar a adaptabilidade do *middleware*. Assim, caracteriza-se o sistema de *middleware* baseado em sua carga computacional, em seu paradigma de comunicação e no tipo de representação de contexto que ele fornece para aplicações, como mostra a figura 3.3.

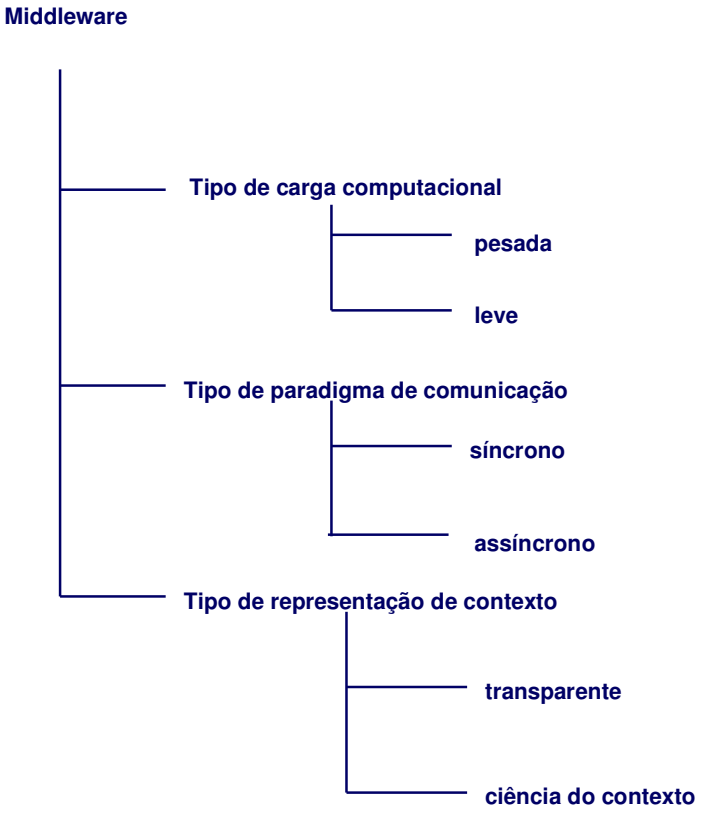


Figura 3.3: Caracterização de sistema de *middleware*.

Um *middleware* baseado em reflexão computacional é projetado para que apresente carga computacional leve e seja facilmente reconfigurável. Exemplos desse tipo de *middleware* são: OpenCORBA (LEDOUX, 1999), OpenORB (EXOLAB.ORG, 2001), dynamicTAO (KON *et al*., 2000) entre outros.

Um *middleware* baseado no espaço de tuplas é projetado para resolver os constantes problemas de desconexões e apresenta comunicação assíncrona de uma forma mais natural. Algumas questões para esse tipo de *middleware* permanecem abertas, tais como fazer a persistência e o compartilhamento do espaço de tuplas mais adequado para as aplicações. Algumas soluções já foram implementadas considerando essas questões, tais como Tspaces (WYCKOFF *et al*., 1998), JavaSpace (WALDO, 1998) e Lime (MURPHY *et al*., 2001).

Um *middleware* com o conhecimento do contexto de execução é desenvolvido para fornecer informações de contexto para aplicações. Essas informações incluem localização do usuário, características dos dispositivos

e informações sobre o ambiente físico. Atualmente, apenas informação de localização é fornecida pelos ambientes de desenvolvimento, tais como CyberGuide (LONG *et al*., 1996) e Shopping Assistant (ASTHANA *et al.*, 1994).

3.3 *MIDDLEWARE* EM REDES DE SENSORES SEM FIO

Como visto no capítulo anterior, RSSFs são, basicamente, utilizadas para monitorar e, algumas vezes, controlar o ambiente de determinada região. Devido à diversidade de ambientes, o projeto dessas redes é dependente da aplicação e dos parâmetros que se deseja monitorar ou controlar. Assim, grande parte das pesquisas em RSSFs trata de algoritmos e funções para um tipo particular de rede. Um *middleware* tradicional é projetado para servir a uma grande variedade de aplicações sem necessariamente precisar do conhecimento de tais aplicações. Diferentemente, em RSSFs as aplicações são construídas especificamente para um determinado tipo de rede. Assim, um *middleware* para RSSFs necessita de mecanismos para incorporar o conhecimento da aplicação em sua infra-estrutura. Dessa maneira pode-se classificar um *middleware* para RSSFs de acordo com sua principal finalidade.

3.3.1 *Middleware* para configuração dos elementos de rede

A constituição em módulos de um nó sensor permite que se possa configurar um dos módulos de maneira independente. Por exemplo, pode-se configurar a potência de saída do transceptor para economizar gasto de energia. A configuração diretamente sobre o *hardware* é uma tarefa complexa e difícil. Além disso, a configuração diretamente sobre o *hardware* não pode ser feita em tempo de execução. Um *middleware* pode ser projetado para configurar os módulos de nós sensores, em tempo de execução, considerando os requisitos da aplicação. Algumas tarefas de configuração que um *middleware* pode auxiliar são: configuração da potência de transmissão do transceptor, calibração do sensor, configuração do tipo de coleta entre outras.

3.3.2 *Middleware* para redes heterogêneas

As RSSFs são constituídas de diversos elementos que podem ser de diferentes tecnologias. Entre esses elementos podem-se citar, por exemplo, *hardware* do nó sensor, protocolos de rede, sistemas operacionais e mesmo linguagens de consulta. Um *middleware* para redes heterogêneas pode ser projetado para compatibilizar formato de dados, métodos de acesso, tipos de coleta entre outros.

3.3.3 *Middleware* para interoperabilidade

O objetivo principal de um *middleware* é fornecer interoperabilidade entre os elementos que compõem uma RSSF. Uma outra possibilidade é prover interoperabilidade entre diferentes RSSFs, entre os observadores e uma RSSF e entre uma aplicação de gerenciamento e uma RSSF. Nesses casos, o *middleware* pode ser disponibilizado no nó *gateway* ou implantado nos pontos de acesso para tornar transparente, ao observador, a configuração e manutenção da rede.

3.3.4 *Middleware* para manutenção

Um *middleware* projetado para as tarefas de manutenção é similar aos tipos de *middleware* para configuração, e deve possuir características de reflexão computacional. Com o conhecimento do estado da rede sob diferentes aspectos (energia, topologia, largura de banda), o *middleware* será capaz de alterar o comportamento da rede em tempo de execução. Esse tipo de *middleware* é capaz de fornecer suporte flexível às aplicações.

3.4 CONSIDERAÇÕES

Como visto anteriormente, as características próprias de RSSFs acarretam vários desafios que são enfrentados com algumas técnicas, tais como fusão de dados, auto-organização e conservação da energia. Pode-se projetar um *middleware* específico para abordar determinada técnica. Assim, pode-se projetar um *middleware* para fusão de dados. Uma outra possibilidade é especificar um *middleware* para uma aplicação específica, por exemplo, um *middleware* para consulta de informações de rastreamento

de veículos. De uma maneira mais geral, pode-se especificar um *middleware* que seja utilizado em um conjunto de aplicações voltadas para extração de informações pelos usuários.

Esta tese aborda especificamente esse tipo de *middleware*, isto é, *middleware* para aplicações específicas, o qual terá seu projeto geral apresentado nos próximos três capítulos.

4 ABSTRAÇÕES E MECANISMOS

Para suportar aplicações em RSSFs, um *middleware* deve ser construído com abstrações e mecanismos que permitam atender aos diversos requisitos que determinado tipo de aplicação requer. Com a finalidade de restringir o escopo do *middleware*, nesta tese levantam-se apenas os requisitos necessários para que o modelo de *middleware* proposto suporte aplicações de monitoramento ambiental. Esse tipo de aplicação foi escolhido devido a sua importância para região amazônica, local de desenvolvimento da pesquisa.

Neste capítulo, primeiro introduzem-se vários exemplos que ilustram a classe de aplicações que o modelo de *middleware* pretende suportar. Após, levantam-se os requisitos dessa classe de aplicação. Finalmente, apresentam-se os mecanismos e abstrações que um *middleware* deve possuir para permitir o atendimento dos diversos requisitos de aplicações de monitoramento ambiental.

4.1 Aplicações de Monitoramento Ambiental

Aplicações de monitoramento são utilizadas para coletar dados, analisar e acompanhar contínua e sistematicamente as variáveis ambientais, com objetivo de identificar e avaliar qualitativa e quantitativamente as condições dos recursos naturais em um determinado momento, assim como as tendências verificadas ao longo do tempo (IBAMA/GTZ, 2000). Monitoramento ambiental fornece informações sobre os fatores que influenciam no estado de conservação, preservação, degradação e recuperação ambiental, constituindo um instrumento de avaliação e controle.

Como já afirmado anteriormente, RSSFs são utilizadas para monitoramento de variáveis ambientais em locais internos como prédios e residências, ou locais externos como florestas, rios, lagos, desertos, etc. Monitoramento ambiental interno consiste em localizar um local específico (por exemplo, uma sala) e acompanhar seu comportamento. Podem ser necessários resultados integrados de mais de um tipo de sensor, como por exemplo, as imagens de uma câmera, a vibração de um sensor sísmico ou ruídos de um sensor de áudio. Esses resultados são processados e comparados com a assinatura do evento de interesse.

No monitoramento ambiental externo, milhares de nós são espalhados em uma área em que se deseja inspecionar. Quando ocorre alguma condição programada, tal como a elevação da temperatura, aumento do nível de CO_2, mudanças na taxa de umidade do ar, entre outras, é disparado um módulo de gerenciamento de eventos que manda um sinal para o usuário. Muitos nós sensores combinam suas detecções para fornecer a informação precisa.

Os exemplos anteriores mostram que a utilidade de uma RSSF baseia-se na sua capacidade de fornecer informações de áreas locais ou extensas em resposta às questões colocadas pelo usuário. A abordagem de consulta é o modo mais comum em que a rede é empregada. Em um outro modo de operação, nós sensores podem permanecer em espera aguardando por alguma ocorrência específica. Detectar a ocorrência de eventos de interesse é um outro modo de operação em que a rede é empregada. Assim, obtém-se o primeiro requisito (R1) do modelo de *middleware*: o sistema deve ser capaz de operar em dois modos de operação, a saber, por consulta e por ocorrência de eventos.

Dependendo do tipo de aplicação, talvez seja mais conveniente acessar unicamente um determinado nó ou não. Por exemplo, na aplicação de monitoramento ambiental interno, pode-se querer saber se em determinada sala o valor da temperatura já atingiu um ponto crítico. Nesse caso é necessário endereçar unicamente os nós sensores para saber a identificação do local. Por outro lado, na aplicação de monitoramento ambiental externo, os nós sensores não precisam ser identificados unicamente, pois nesse tipo de aplicação o importante é saber o valor de

determinada variável em uma dada região. Desse modo, tem-se o segundo requisito (R2) do modelo de *middleware*: o sistema deve ser capaz de endereçar os nós sensores de maneira única e também de modo não único, isto é, por atributo.

Em algumas aplicações, a mobilidade dos nós sensores deve ser considerada. Por exemplo, nós sensores espalhados em uma floresta para coleta de dados sobre a umidade e temperatura são estáticos, enquanto que nós sensores colocados em uma superfície de um rio para coleta de dados sobre a poluição do mesmo são móveis. Logo, o terceiro requisito (R3) é que a mobilidade dos nós sensores deve ser considerada no modelo de *middleware*.

O raio de sensoriamento de um nó sensor é menor que seu raio de comunicação. Além disso, nós sensores operam em ambientes com bastante ruído. Para obter uma resolução de sensoriamento confiável é necessária uma alta densidade de nós sensores. Em algumas aplicações o tamanho da área de cobertura demanda uma grande quantidade de nós sensores. Uma simples aplicação na área de ambiente como monitoramento de oceanos ou florestas requer centenas a milhares de nós sensores. Em outras aplicações como em monitoramento ambiental interno, a quantidade de nós sensores é limitada pelo tamanho do prédio. Portanto, o quarto requisito (R4) é que a variação do tamanho de uma RSSF deve ser levada em conta no modelo de *middleware*.

Em monitoramento ambiental externo, os nós sensores são espalhados em uma região hostil, onde não é possível alcançá-los para manutenção. O tempo de vida de cada nó sensor fica dependendo exclusivamente da pequena energia nele armazenada. Para que se possa conservar a energia, a velocidade da CPU e a largura de banda do canal de RF são limitadas. Em algumas aplicações tais como aplicações militares, os nós sensores devem ser pequenos para dificultar a sua detecção pelo inimigo. Esse novo requisito impõe restrições adicionais na velocidade de CPU, tamanho da memória, largura de banda de RF e mesmo no tamanho da bateria. Em aplicações em que os nós sensores não estão espalhados em uma região hostil é possível alcançá-los para manutenção e o tempo de vida de cada nó sensor não se torna um fator crítico. Enfim, o quinto requisito

(R5) é que os limitados recursos dos nós sensores devem ser considerados no modelo de *middleware*, principalmente a pequena energia disponível para o nó.

4.2 Paradigma *Publish-Subscribe*

Em aplicações interativas que usam o paradigma da orientação a objetos, as ações que o usuário executa em objetos são percebidas como eventos que causam mudanças em outros objetos que mantêm o estado da aplicação. As ações como manipulação de um botão do *mouse* ou entrada de texto via teclado geram eventos que são notificados para os objetos responsáveis por manter o estado atual da aplicação. Esse paradigma é conhecido como *modelo de eventos locais* e é usado na maioria dos sistemas de computadores pessoais atuais.

Sistemas baseados em eventos distribuídos estendem o modelo de eventos locais permitindo a múltiplos objetos localizados em diferentes lugares serem notificados de eventos que ocorrem em um outro objeto (COULORIS *et al.*, 2001). As duas principais características desses sistemas são heterogeneidade e comunicação assíncrona.

A figura 4.1 mostra um esquema com os componentes participantes de um sistema baseado em eventos distribuídos.

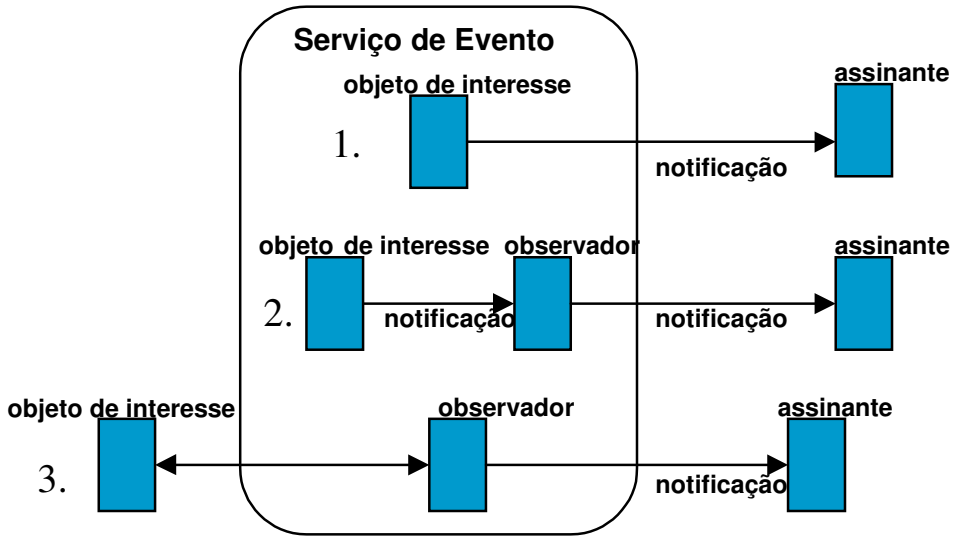


Figura 4.1: Sistema baseado em eventos distribuídos.

A função de cada componente que participa no sistema é como segue:

- Objeto de interesse – Esse objeto muda de estado quando suas operações são invocadas. A mudança pode interessar a outros objetos.
- Evento – Um evento ocorre em um objeto de interesse quando um método é executado.
- Notificação – Uma notificação é um objeto que contém informação sobre um evento. A informação contém o tipo de evento e seus atributos.
- Assinante – Um assinante é um objeto que expressa interesse em alguns tipos de eventos em outros objetos. Ele recebe notificações sobre tais eventos.
- Observador – A principal função de um observador é desacoplar um objeto de interesse de seus assinantes.
- Publicador – Objeto que declara que gerará notificações de determinado tipo de evento. Ele pode ser um objeto de interesse ou um observador.
- Serviço de Evento – Mantém uma relação de eventos publicados e de interesses de assinantes.

A figura 4.1 apresenta três situações:

1. Um objeto de interesse dentro do serviço de evento sem um observador. Ele envia notificações diretamente para os assinantes.
2. Um objeto de interesse dentro do serviço de evento com um observador. O objeto de interesse envia notificações para os assinantes por intermédio do observador.
3. Um objeto de interesse fora do serviço de evento. O observador consulta regularmente o objeto de interesse para descobrir a ocorrência de eventos. O observador envia notificações para os assinantes.

Um *middleware* baseado em eventos distribuídos é um tipo de sistema baseado em eventos distribuídos onde os assinantes são denominados de consumidores e os publicadores são chamados de produtores. As funções do observador são várias, a saber: filtragem de notificações, padrões de eventos, caixa-postal entre outras (COULORIS *et al.*, 2001). A filtragem reduz o número de notificações recebidas de acordo com algum predicado no conteúdo de cada notificação. Um padrão de eventos especifica uma relação entre vários eventos. A caixa-postal é usada quando um consumidor não está pronto para receber uma notificação.

Um *middleware* baseado em eventos distribuídos é largamente utilizado quando existe a necessidade de unir conjuntamente componentes heterogêneos e que mudam dinamicamente em grandes redes de nós. O *middleware* executa as funções de coletar mensagens de produtores, filtrar e transformar tais mensagens quando necessário e roteá-las para consumidores apropriados. Essa abordagem é comumente aplicada em domínios como finanças, automação de processos e transporte.

Um *middleware* baseado em eventos distribuídos também utiliza o paradigma *publish-subscribe*, em que um componente que gera eventos (produtor) publica os tipos de eventos que ele tornará disponível para a observação de outros componentes (consumidores). Os consumidores que desejam receber notificações de um produtor que publicou seus eventos assinam para os tipos de eventos que eles têm interesse. As notificações são enviadas de maneira assíncrona pelos produtores para os consumidores interessados. As notificações podem ser armazenadas, enviadas em mensagens e processadas de diversas formas. Quando um produtor dispara um determinado tipo de evento, os consumidores que se cadastraram para receber esse tipo de evento receberão as notificações.

Além da característica de comunicação assíncrona, esse paradigma é adequado para realizar a comunicação entre componentes que não foram construídos para trabalharem juntos. Tudo que é preciso é que os produtores gerem os eventos e os consumidores mostrem interesse pelos mesmos e forneçam uma interface para receber as notificações.

Esse tipo de paradigma é adequado para satisfazer a característica de implantação contínua da rede, não existindo possibilidade de parar a rede para se realizar atualizações e manutenções. Também suporta evolução dinâmica da rede integrando dinamicamente nós sensores novos para substituírem os danificados, os quais podem ser de diferentes fornecedores. O suporte a esses dois requisitos só é possível devido à comunicação assíncrona, heterogênea e anônima fornecida pela comunicação *publish-subscribe*, onde os dados são enviados e recebidos assincronamente baseados em assunto, independente da identidade e localização dos produtores e consumidores.

A comunicação *publish-subscribe* é anônima, assíncrona e *multicast,* ou seja, os dados são enviados e recebidos por difusão de maneira assíncrona, baseados em assunto, independente da identidade e localização dos produtores e consumidores. Esse tipo de comunicação tem propriedades desejáveis da perspectiva de uma RSSF, desde que a comunicação não é fim-a-fim então uma comunicação anônima com formação de grupo *multicast* específica da aplicação pode ser formada. Em relação à implementação, comunicação assíncrona contribui para preservar energia e aumentar o tempo de vida da rede, pois os nós sensores não operam continuamente.

Assim, a comunicação *publish-subscribe* atende ao requisito R1 sobre o modo de operação por ocorrência de eventos e ao requisito R2 sobre o endereçamento por atributo, além de considerar o requisito R5 sobre a conservação de energia.

4.3 Paradigma Multiusuário

O paradigma de interação cliente-servidor organiza um programa aplicativo para esperar passivamente que outro aplicativo inicie a comunicação. Os termos cliente e servidor se referem aos dois aplicativos envolvidos na comunicação. O aplicativo que começa ativamente o contato é chamado de cliente, ao passo que o aplicativo que espera passivamente por contato é chamado de servidor. As informações podem passar em uma ou ambas as direções entre um cliente e um servidor. Tipicamente, um cliente envia uma requisição para um servidor, e o servidor devolve uma

resposta para o cliente, como ilustra a figura 4.2. Essa forma de interação é chamada de requisição-resposta.

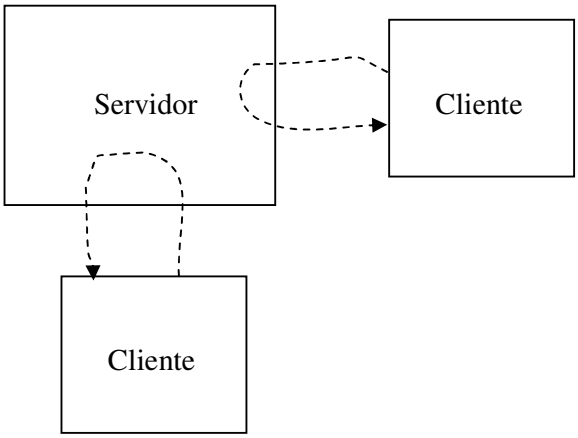


Figura 4.2: Aplicativo cliente-servidor.

Um aplicativo multiusuário apresenta uma pequena variação do aplicativo típico cliente-servidor. A diferença é que a informação passa de um cliente pelo servidor para outros clientes, e a figura 4.3 ilustra essa diferença quando comparada com a figura 4.2.

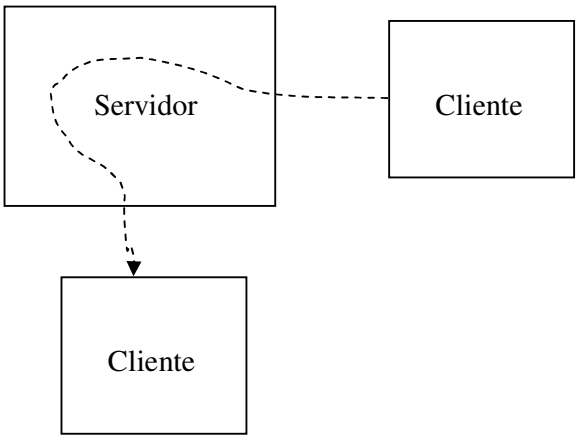


Figura 4.3: Aplicativo multiusuário.

Quando se cria um aplicativo multiusuário deve-se tanto quanto possível abstrair a rede, mesmo sabendo que há uma grande quantidade de trabalho extra entre o cliente e o servidor. Tenta-se minimizar o número de interações entre o cliente e o servidor. Assim, primeiro cria-se o servidor e uma interface cliente, e em seguida cria-se o encapsulamento para os vários protocolos de rede e sistemas de objetos remotos com os quais se

quer operar. A figura 4.4 mostra uma configuração de exemplo, onde o servidor pode ser acessado por *sockets* TCP (*Transmission Control Protocol*) e RMI.

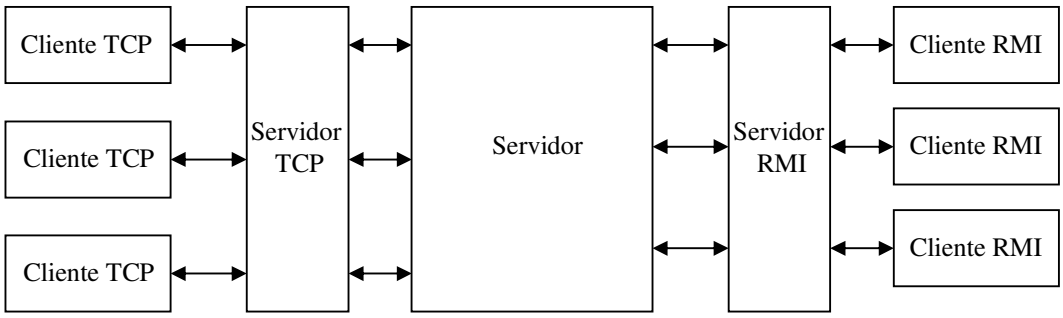


Figura 4.4: Configuração exemplo multiusuário.

No que diz respeito ao servidor, o protocolo de rede é a interface com o usuário para o servidor. Na verdade, segue-se o princípio da separação entre o aplicativo e a interface com o usuário. Como o servidor precisa invocar métodos no cliente, ele define uma interface que os clientes no sistema devem implementar. Tanto no servidor quanto na interface cliente não há menção a um protocolo de rede específico. Essas duas classes representam o núcleo do aplicativo. Se os aplicativos forem criados dessa forma, não haverá problemas para adicionar outros modos de acesso a eles.

Como o paradigma multiusuário utiliza a mesma forma de interação requisição-resposta, ele é adequado para suportar o modo de operação por consulta além de possibilitar a utilização de vários protocolos de comunicação de rede.

4.4 Linguagem de Restrição e de Consulta

Linguagens de restrição e de consulta são utilizadas para realizar consultas especificando restrições nos valores dos atributos e preferências na ordem em que é executado o *matching* dos dados entrantes. Uma declaração nessas linguagens é uma *string* que representa uma expressão.

As regras gramaticais que definem as construções de uma linguagem são descritas através de produções (regras que produzem, geram) cujos elementos incluem símbolos terminais (aqueles que fazem parte do código fonte) e símbolos não-terminais (aqueles que geram outras regras). Na

figura 4.5, as produções da linguagem de restrição são apresentadas na Forma Normal de Backus (BNF – Backus Naur Form) (AHO et al., 1988).

```
Programa ::= Expressao
        Expressão ::= Valor
                | ExpUnaria
                | ExpBinaria
        Valor ::= ValorInteiro | ValorBooleano
        ExpUnaria ::= "-" Expressao | "not" Expressao | "length" Expressao
        ExpBinaria ::= Expressao "+" Expressao
                | Expressao "-" Expressao
                | Expressao "and" Expressao
                | Expressao "or" Expressao
                | Expressao "==" Expressao
                | Expressao "!=" Expressao
                | Expressao " > " Expressao
                | Expressao ">=" Expressao
                | Expressao "<" Expressao
                | Expressao "<=" Expressao
```

Figura 4.5: BNF da linguagem de restrição.

A linguagem de restrição inclui apenas constantes (valores) e operações sobres valores. Valores e operações sobre inteiros, booleanos e *strings* são admitidos. Essa linguagem admite vários tipos de expressões que podem ser:

- Comparativas: = = (igualdade), != (desigualdade), >, >=, <, <=. Por exemplo, Temperatura < 36 implica somente considerar dados cujo atributo temperatura seja menor do que 36 graus Celsius.
- Booleanas: *and*, *or*, *not*. Por exemplo, Temperatura >= 26 *and* Temperatura <= 36 implica somente considerar dados onde o valor do atributo temperatura está na faixa entre 26 e 36 graus Celsius.
- Numéricas: com os operadores matemáticos + (adição), - (subtração), * (multiplicação) e / (divisão).

A linguagem de consulta tem sintaxe baseada em um subconjunto da sintaxe de expressão condicional SQL92 (SQL92, 1992), e é uma extensão da linguagem de restrição com novos operadores. Essa nova linguagem

incorpora identificadores que possuem um valor constante. É necessário um mapeamento entre identificadores e valores. Na avaliação de uma expressão a ocorrência de um identificador é substituída pelo valor associado ao identificador. A inclusão de novos operadores (*between, like, in, is, escape*) permite a construção de consultas semelhantes às usadas em bancos de dados que têm com linguagem de consulta o padrão SQL (*Structured Query Language*). Por exemplo, consultas do tipo -- Temperatura *between* 26 *and* 36 – são agora possíveis.

As linguagens de restrição e de consulta são necessárias para facilitar a programação de aplicações interativas pelos usuários. Essas aplicações acessam *on-line* as informações enviadas pelos nós sensores a partir de um computador de um usuário final. Assim, completa-se o atendimento do requisito R1 sobre o modo de operação por consulta.

4.5 Filtros de Aplicação

Filtros de aplicação são módulos de *software* específico de uma aplicação que interferem na difusão e no processamento de dados (HEIDEMANN *et al*., 2001). Os filtros são fornecidos antes da implantação de uma RSSF e cada um deles é especificado usando uma lista de atributos para possibilitar o *matching* com os dados entrantes.

Os filtros são usados para fazer agregação interna dos dados, processamento colaborativo de sinais, *caching*, e tarefas que controlam o movimento dos dados numa RSSF (HEIDEMANN *et al*., 2001). No SensorBus, filtros são utilizados para limitar a movimentação dos dados na rede, e se pode projetar um filtro que restrinja a faixa de valores de um determinado atributo. Por exemplo, a aplicação só tem interesse que o atributo Temperatura esteja entre os valores de 20 e 30 graus Celsius, não interessando dados que estejam fora dessa faixa. A filtragem descarta alguns dados não necessários, reduzindo a comunicação entre os nós. Essa redução diminui o consumo de energia nos nós sensores e completa-se assim o atendimento do requisito R5 sobre a economia de energia nos nós sensores.

4.6 Padrões de Projeto

Padrões de projeto são descrições de objetos e classes comunicantes que são customizados para resolver um problema geral de projeto num contexto específico (GAMMA *et al*., 2000). São arquiteturas de projeto repetitivas extraídas da experiência de um projetista de um determinado domínio. Um padrão nomeia, abstrai e identifica os aspectos chaves dessas estruturas, identificando classes e instâncias, suas colaborações e a distribuição de responsabilidades entre seus participantes. Padrões de projeto são utilizados no projeto de um *middleware* com intuito de aproveitar sua arquitetura de *software*. O SensorBus embute seis tipos de padrões descritos a seguir:

- Padrão *Observer* define uma dependência um-para-muitos entre objetos, de modo que quando um objeto muda de estado, os seus dependentes são automaticamente notificados e atualizados. Os objetos-chave nesse padrão são *subject* (assunto) e *observer* (observador). Um *subject* pode ter um número qualquer de observadores dependentes e esses observadores são notificados quando o *subject* sofre uma mudança de estado. Em resposta a isso, cada observador inquirirá o *subject* para sincronizar o seu estado com o estado do *subject*. Utilizou-se esse padrão para implementar o mecanismo de comunicação *publish-subscribe*, sendo que o *subject* é o publicador de notificações. Ele envia essas notificações sem saber quem são os seus observadores e um número qualquer de observadores pode inscrever-se para receber notificações.
- Padrão *Interpreter* define uma representação para gramática de uma determinada linguagem juntamente com um interpretador que usa essa representação para interpretar sentenças nessa linguagem. Esse padrão usa uma classe para representar cada regra da gramática e os símbolos do lado direito da regra são as variáveis de instância dessas classes. Utilizou-se esse padrão para definir a linguagem de restrição e de consulta.

- Padrão *Facade* define uma interface unificada (e de nível mais alto) para um conjunto de interfaces em um subsistema, tornando-o mais fácil de usar. Os clientes se comunicam com o subsistema através do envio de solicitações para interface *Facade*, a qual as repassa para os objetos apropriados do subsistema. Embora os objetos do subsistema executem o trabalho real, a *Facade* pode ter que efetuar trabalho próprio dela para traduzir a sua interface para as interfaces de subsistemas. Os clientes que usam a *Facade* não têm que acessar os objetos do subsistema diretamente. Utilizou-se esse padrão para disponibilizar primitivas de alto nível para os programadores de aplicação do SensorBus.
- Padrão *Mediator* define um objeto que encapsula a forma de interação de um conjunto de outros objetos. Ao evitar que os objetos se refiram uns aos outros possibilita as variações de suas interações independentemente. Os objetos enviam e recebem solicitações de um objeto *Mediator*. O mediador implementa o comportamento cooperativo pelo direcionamento das solicitações para os objetos apropriados.
- Padrão *Adapter* converte a interface de uma classe em outra interface esperada por outras classes. Esse padrão permite que classes com interfaces incompatíveis trabalhem em conjunto, o que de outra forma não seria possível. Ele se comporta similarmente a um adaptador de energia, convertendo um tipo em um outro tipo incompatível. Usando esse padrão, a aplicação mantém o uso de suas interfaces enquanto permite o uso de novos componentes.
- Padrão *Router* desacopla múltiplas fontes de informação dos alvos daquela informação e trabalha como um roteador em tráfego de rede de computadores. Ele recebe informação de uma fonte e determina, baseado de onde a informação está vindo, para que destinos a mensagem deve ser enviada. Para ser capaz de fazer isso o roteador mantém um mapeamento entre os vários canais de entrada (fontes) e seus destinos.

Os padrões *Mediator*, *Adapter* e *Router* são utilizados na construção do barramento de mensagens do SensorBus*,* sendo responsáveis pela implementação do mecanismo de comunicação permutável. Esse mecanismo permite que se permute o protocolo de roteamento de RSSFs, atendendo assim os requisitos R2, R3 e R4, pois são os requisitos que são tratados pelo protocolo de roteamento.

4.7 Considerações

Os requisitos levantados para aplicações de monitoramento ambiental que devem ser consideradas no modelo de *middleware* do SensorBus foram: R1 – O sistema deve ser capaz de operar em dois modos de operação, a saber, por consulta e por ocorrência de eventos; R2 – O sistema deve ser capaz de endereçar os nós sensores de maneira única e também de modo não único, isto é, por atributo; R3 – a mobilidade dos nós sensores deve ser considerada no modelo; R4 – a variação do tamanho de uma RSSF deve ser levada em conta no modelo; e R5 – os limitados recursos dos nós sensores devem ser considerados no modelo, principalmente a energia disponível no nó sensor.

Para considerar os requisitos levantados, o modelo de *middleware* do SensorBus é constituído de três mecanismos e uma abstração. Utilizam-se os mecanismos *publish-subscribe*, linguagens de restrição e de consulta e filtros de aplicação para atender os requisitos R1, R2 e R5. A abstração padrão de projeto é utilizada para atender os requisitos R2, R3 e R4.

5 Configuração de Políticas

5.1 Políticas e Suposições

A maioria dos algoritmos desenvolvidos para RSSFs é hipotética, isto é, eles ainda não foram testados em uma rede instalada no campo, embora tenham sido testados em redes de teste em laboratórios. Esta pesquisa não é diferente. Quando se fala em RSSFs, a referência é para redes de testes em laboratórios em vez das redes implantadas no campo. Essas redes apresentam características diversas de acordo com o propósito para qual foram projetadas. Focaliza-se a atenção apenas para organização de redes que são mais relevantes para as questões que se deseja considerar. Essa organização envolve três componentes principais: infra-estrutura, protocolo de rede e aplicações (SAMEER *et al*., 2003). A infra-estrutura é formada pelos nós sensores e pelo modo como eles foram espalhados em uma região. O protocolo de rede é responsável pela criação e manutenção dos caminhos de comunicação entre os nós sensores e a aplicação. As aplicações obtêm informações sobre determinado fenômeno através do sensoriamento dos nós. A seguir, abordam-se as políticas e suposições que foram feitas considerando esses aspectos.

5.1.1 Aplicações

A forma como as aplicações obtêm os dados dos nós sensores depende de como a rede foi projetada. Existem os seguintes modelos de transferência de dados entre os nós sensores e a aplicação: contínuo, dirigido a eventos, consulta e híbrido (SAMEER *et al*., 2003). No modelo contínuo, os nós sensores transferem seus dados continuamente em uma taxa pré-especificada. No modelo dirigido a eventos, os nós sensores

transferem informação somente quando um evento de interesse ocorreu. No modelo de consulta, a aplicação é responsável por emitir requisições sobre determinado fenômeno de interesse. Finalmente, as três abordagens podem coexistir na mesma rede, formando o modelo híbrido. O SensorBus suporta, como política, a escolha do modelo de consulta ou do modelo dirigido a eventos conforme exige o tipo de aplicação alvo apresentada no capítulo 4.

5.1.2 Protocolo de Rede

O desempenho do protocolo de rede é influenciado pelo modelo de comunicação adotado, modelo de transferência de pacotes de dados e mobilidade da rede. Para determinar como o protocolo de rede se comporta em diferentes cenários, é importante considerar esses três aspectos.

O modelo de comunicação em RSSFs é classificado em duas categorias (SAMEER *et al*., 2003): aplicação e infra-estrutura. A comunicação de aplicação consiste na transferência dos dados obtidos pelos nós sensores para informar o usuário sobre o fenômeno em estudo, e pode ser de dois tipos: cooperativa e não-cooperativa. Na comunicação cooperativa, os nós sensores se comunicam entre si antes de informar os dados observados. Na comunicação não-cooperativa, os nós sensores não cooperam para a disseminação da informação. A comunicação de infra-estrutura refere-se à comunicação necessária para configurar, manter e otimizar a operação da rede. De maneira estática, o SensorBus estabelece políticas para ambas categorias de comunicação. A configuração, manutenção e otimização da operação da rede são conseguidas através do envio de comandos para os nós sensores.

A transferência de pacotes de dados é um problema de roteamento do protocolo de rede e as abordagens de roteamento são classificadas em *flooding*, *unicast* e *multicast* (SAMEER *et al*., 2003). Na abordagem de *flooding*, nós sensores difundem suas informações para os nós vizinhos, que repassam essa difusão para novos nós vizinhos até encontrar o usuário. Na abordagem *unicast*, nós sensores se comunicam com o usuário diretamente, não só através de um roteamento *multi-hop*, mas também através de um cabeçalho de agrupamento usando roteamento *unicast* um-para-um. Finalmente, na abordagem *multicast*, os nós sensores são

formados em grupos e usam roteamento *multicast* para comunicação entre os membros dos grupos. O protocolo de roteamento se incumbirá de tratar o modo de transferência de pacotes e usando o SensorBus é possível utilizar, de maneira estática, diversos tipos de protocolos de roteamento.

Quanto à mobilidade, RSSFs são classificadas em redes estáticas e dinâmicas (SAMEER *et al*., 2003). Nas redes estáticas, não há movimento entre os nós sensores, o usuário e o fenômeno em estudo, os quais são móveis nas redes dinâmicas. Esse tipo de rede é ainda classificado conforme a mobilidade de seus componentes em redes dinâmicas com usuário móvel, nós sensores móveis e fenômeno móvel. Apesar da mobilidade ser tratada pelos diversos protocolos de roteamento, o SensorBus considera tanto RSSFs estáticas quanto dinâmicas, dependendo do protocolo de roteamento utilizado.

5.1.3 Infra-estrutura

Em relação à infra-estrutura, as suposições são sobre a localização e ponto de acesso enquanto que se estabelecem políticas sobre os recursos computacionais dos nós sensores. Nós sensores possuem localização conhecida e são espalhados sobre uma área bem definida. Assume-se que as informações são inseridas e recebidas através de um único ponto de acesso chamado de nó sorvedouro. Apesar de se tratar todos os nós com a mesma capacidade de recursos, nada impede que alguns nós tenham maior capacidade de memória, mais energia disponível ou maior facilidade de comunicação, possibilitando assim uma arquitetura em camadas.

A figura 5.1 mostra a arquitetura de comunicação da RSSF de testes.

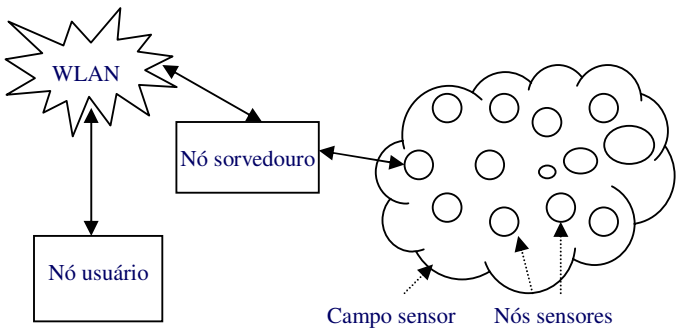


Figura 5.1: Arquitetura de comunicação.

Cada nó sensor espalhado no campo sensor possui a capacidade de coletar dados e enviá-los para o nó sorvedouro. O nó sorvedouro pode ser um computador fixo de maior capacidade computacional do que os nós sensores. O nó sorvedouro comunica-se com o nó usuário através de uma rede local sem fio convencional.

O SensorBus permite o estabelecimento de políticas para os seguintes recursos dos nós sensores: memória, bateria e intensidade do sinal de rádio. O usuário pode especificar quando determinado recurso atinge um valor limite, por exemplo, notificar quando o nível da bateria for menor do que 20% de seu nível inicial. O SensorBus obtém estas informações interagindo com uma variedade de sensores heterogêneos; por exemplo, energia disponível da bateria pode ser obtida invocando uma primitiva do sistema operacional. Nesta tese, não há interesse em analisar como o sensoriamento de contexto é executado; assume-se que cada sensor (memória, bateria, etc) fornece uma interface que o SensorBus pode usar para obter o valor do recurso associado. Se a interface existe, o recurso faz parte do contexto do SensorBus.

5.2 Processo de Configuração de Políticas

Chama-se de adaptação dirigida pela aplicação (*Application-driven adaptation*) a habilidade das aplicações configurarem estática e dinamicamente seu comportamento, com objetivo de se adaptar, de maneira reativa ou pró-ativa a diferentes condições de contexto e necessidades (CAPRA, 2003). Assim, surgem quatro tipos de configurações: configuração estática reativa, configuração estática pró-ativa, configuração dinâmica reativa e configuração dinâmica pró-ativa.

A configuração estática reativa refere-se à habilidade da aplicação definir quais aspectos de contexto são de interesse para ela; identificar que mudanças no contexto são relevantes, e associar comportamentos que a aplicação está disposta a aderir quando tais mudanças acontecerem. As aplicações controlam seus comportamentos em um conjunto de configurações de contexto pré-definidos. Por exemplo, ao se referir às aplicações em RSSFs uma aplicação pode precisar ser notificada quando alguma configuração de contexto pré-definida ocorrer. Isso pode acontecer

quando o nível de bateria alcança determinado valor limite, ou quando um determinado nó sensor esteja fora de operação por algum motivo.

A configuração estática pró-ativa refere-se à habilidade da aplicação: identificar os serviços que deseja adaptar para mudanças no contexto; decidir um conjunto de políticas que pode ser usado para fornecer o serviço; e especificar as configurações de contexto que devem ser mantidas para que uma política seja selecionada e aplicada quando o serviço for realmente requisitado. Por exemplo, em RSSFs a aplicação pode requerer a utilização de processamento cooperativo quando a largura de banda não é estável e a quantidade de memória disponível nos nós sensores é alta, enquanto que o processamento não-cooperativo poderia ser utilizado quando a largura de banda fosse estável e a quantidade de memória fosse baixa. Assim, aplicações definem tanto o conjunto de comportamentos que elas desejam seguir quanto as configurações de contexto que devem ser mantidas para que esses comportamentos sejam aplicados.

A configuração dinâmica reativa refere-se à habilidade da aplicação alterar dinamicamente a maneira como ela reage a mudanças no contexto. Desse modo as aplicações alteram dinamicamente o conjunto de possíveis comportamentos, assim como também as associações entre esses comportamentos e seus correspondentes contextos habilitados. Isso permite, por exemplo, suportar variação na necessidade das aplicações e condições de contexto não previstas. Considerando RSSFs, uma aplicação pode alterar o conjunto de configuração de contexto de interesse, e pode solicitar que seja notificada quando esteja executando com pouca memória.

A configuração dinâmica pró-ativa refere-se à habilidade da aplicação alterar dinamicamente a maneira com que os serviços são fornecidos em diferentes contextos. Isso se refere à habilidade de redefinir tanto o conjunto de serviços que requerem configuração quanto às associações entre os serviços configurados. Considerando aplicações em RSSFs, o conjunto de comportamentos associados com o serviço de comunicação pode ser alterado de um serviço com roteamento distribuído entre os nós sensores para um serviço com roteamento centralizado no nó sensor sorvedouro, caso ocorra desconexões de vários nós sensores que impossibilitem o roteamento satisfatório.

Observa-se que a configuração estática e a configuração dinâmica, apesar de estarem estritamente relacionadas, são dois processos diferentes. Pode-se ter configuração estática sem suportar configuração dinâmica, mas não o contrário.

É importante que aplicações adaptativas para RSSFs suportem tanto configuração estática quanto dinâmica de uma maneira transparente, de modo a minimizar o esforço dos programadores de aplicações em desenvolvê-las, e para isso um grau de automação deve ser fornecido. Esse objetivo é alcançado por meio de uma camada de *software* (*middleware*) que fornece automação explorando os princípios de metadados e comandos para controlar o processo de adaptação. Assim, tarefas como monitoramento do ambiente, detecção de trocas no contexto e assim por diante, devem ser executadas pelo *middleware* que esconde essas complexidades, fornecendo automação necessária para aplicação. Porém, somente as aplicações sabem como e quando devem realizar a adaptação adequada. O *middleware* não possui esse conhecimento *a priori*, devido à variedade de aplicações, às necessidades dos usuários e aos contextos diferentes. Assim, as aplicações devem permitir a qualquer tempo especificar dinamicamente quais informações de contexto lhes interessam, quais informações de contexto elas desejam atender, e assim por diante. Metadados e comandos suportam essa customização de uma maneira elegante e eficiente.

5.3 Configuração Estática de Políticas através de Metadados

Para facilitar o desenvolvimento de aplicações, um *middleware* se incumbe de realizar tarefas de baixo nível e repetitivas, tais como monitoramento de sensores físicos, detecção de trocas no contexto, seleção de comportamentos habilitados no contexto corrente, etc. Contudo, o conjunto de recursos que devem ser monitorados, as trocas que devem ser detectadas, os comportamentos que devem ser habilitados e assim por diante, apenas são conhecidas pela aplicação, e não pelo *middleware*. Informação de configuração, especificando como a aplicação deseja que o *middleware* se comporte em seu contexto corrente, deve ser codificada em metadados do *middleware* e usada por ele para executar suas tarefas.

A configuração de políticas através de metadados serve para construir um sistema de *middleware* para suportar aplicações adaptativas. Como mostra a figura 5.2, podem existir várias aplicações utilizando serviços do mesmo *middleware* e diferentes usuários usando a mesma aplicação.

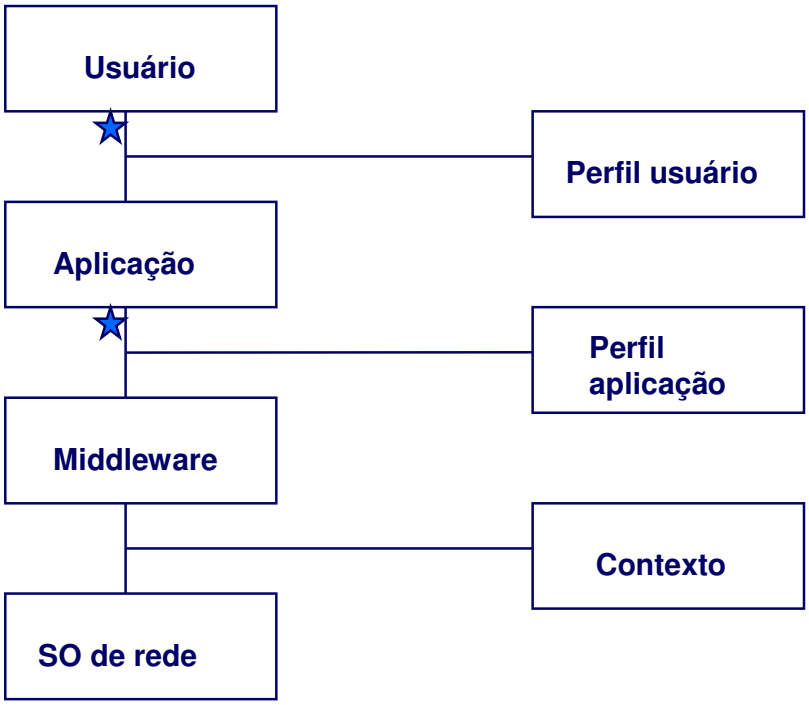


Figura 5.2: Perfis de usuário e aplicação.

Cada usuário pode customizar a aplicação de diversas maneiras. Ele pode, por exemplo, customizar uma barra de ferramenta da interface da aplicação usando determinado tipo de ícone em vez de outros; também pode fazer tarefas mais úteis, como automaticamente se desconectar da rede quando a energia da bateria estiver muito baixa, e assim por diante. Um usuário pode acessar a rede usando diferentes dispositivos, assumindo-se que a aplicação pode executar em diferentes dispositivos, cada um com capacidades diferenciadas. Em tal situação, o usuário pode configurar a aplicação para apresentar somente informação de texto quando estiver executando em um *laptop*, por exemplo, e possibilitar todas as funcionalidades quando estiver executando em *desktop*. Para conseguir esse objetivo, ele especifica um perfil de usuário que instrui a aplicação sobre como se comportar em diferentes casos.

Do ponto de vista da aplicação, chama-se dado, o assunto de sua própria computação ou de seus requisitos funcionais, como acontece com as variáveis ambientais em nosso estudo de caso. As informações contidas no perfil de usuário seriam os metadados da aplicação, a qual extrai as configurações que ela pode gerenciar sozinha em uma maneira independente de contexto (por exemplo, *layout* da barra de ferramenta), e transforma os outros em um perfil de aplicação que é então passado para o *middleware*. O perfil de aplicação relaciona principalmente os requisitos não-funcionais da aplicação, que em RSSFs seriam as características intrínsecas da rede. Em nosso estudo de caso, se confiabilidade é uma questão importante para a aplicação, então a utilização de roteamento centralizado é preferida à do roteamento distribuído, quando se experimenta perda de nós sensores.

Da perspectiva do *middleware*, o contexto de execução representa seus dados (por exemplo, valor da memória disponível, valor do nível de bateria, etc.) enquanto que o perfil de aplicação representa seus metadados, como mostra a figura 5.3.

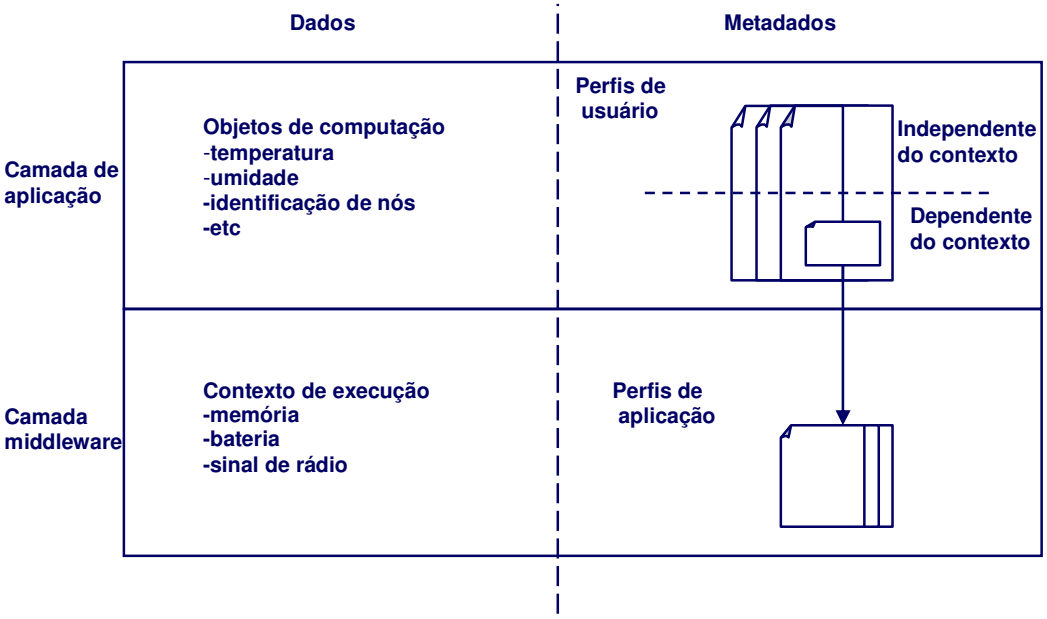


Figura 5.3: Dados/Metadados em aplicações e *middleware*.

O *middleware* fica com a responsabilidade de manter a representação válida do contexto de execução, interagindo diretamente com o ambiente

operacional da rede. Sempre que uma mudança no contexto é detectada, o *middleware* consulta seus metadados para descobrir como a aplicação solicitou que ele se comportasse em tal configuração.

5.3.1 Perfil de aplicação

Para adaptar mudanças no contexto, cada aplicação codifica como o *middleware* deve se comportar quando executa em um contexto de execução particular. Essa codificação é colocada em um perfil de aplicação, que de fato são os metadados do *middleware*.

Cada perfil de aplicação é dividido em duas partes, uma contendo metadados reativos para adaptação reativa, e outra contendo metadados pró-ativos para adaptação pró-ativa, como mostra a figura 5.4.

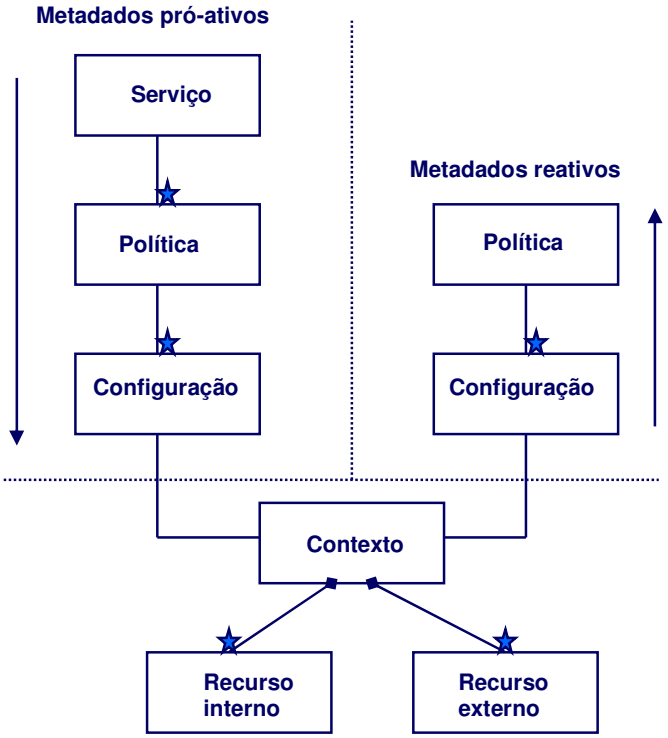


Figura 5.4: Diagrama Entidade-Relacionamento do perfil de aplicação.

Aplicações usam metadados reativos para solicitar ao *middleware* para atentar para trocas no contexto de execução e reagir de acordo com o que foi solicitado, independentemente da tarefa que a aplicação está executando no momento. Por exemplo, a aplicação pode solicitar ao

middleware que acione um alarme quando ela está executando com pouca memória, ou ser alertada quando a memória disponível cai abaixo de determinado valor. Portanto, estabelecem-se associações entre configurações de contexto particular que dependem do valor de um ou mais recursos que o *middleware* monitora, e políticas que o *middleware* deve acionar quando tais configurações ocorrem, como ilustrado no lado direito da figura 5.4. A seta na figura está apontando para cima a fim de representar o fluxo de execução que vai do *middleware* às aplicações. O *middleware* monitora contexto e quando uma reação a uma troca particular acontece, ele dispara políticas que a aplicação precisa. Como por exemplo, para simplesmente notificar que algo ocorreu, ou modificar o comportamento atual da aplicação.

Metadados pró-ativos especificam como a aplicação quer que um serviço seja fornecido em diferentes contextos. Em particular, cada aplicação cria associações entre serviços que a aplicação deseja customizar, e as políticas que podem ser aplicadas para fornecer os serviços, bem como as configurações de contexto que devem ser mantidas de maneira que a política seja aplicada, como mostrado no lado esquerdo da figura 5.4. Por exemplo, o serviço de mensagem pode ser fornecido usando uma política de roteamento distribuído quando a largura de banda está estável, ou uma política de roteamento centralizado quando a largura de banda está instável. Nesse caso, a seta na figura está apontando para baixo a fim de indicar que o fluxo de execução vai da aplicação para o *middleware*. Cada vez que uma aplicação solicita um serviço, o *middleware* consulta seu perfil de aplicação para determinar que política deve ser utilizada para fornecer o serviço solicitado no contexto atual.

Após a informação de configuração ter sido codificada em metadados e disponível para o *middleware*, o processo de adaptação é transparente para aplicações. O *middleware* se encarrega de usar essa informação para executar a adaptação reativa e pró-ativa do comportamento da aplicação.

5.3.2 Codificação de contexto

Configurações de contexto são representadas, para metadados reativos e pró-ativos, em termos dos recursos que o *middleware* monitora e

que interessam à aplicação. Estes recursos variam, ambos nas tecnologias de sensoriamento que o *middleware* estabelece interação para obter informação de estado de recurso, como também na própria informação obtida. Por exemplo, os recursos podem ser locais ao dispositivo, como disponibilidade de memória ou bateria cujos estados podem ser obtidos invocando uma primitiva do sistema operacional de rede. Recursos também podem ser externos ao dispositivo, como localização cujo estado normalmente é obtido interagindo com um sensor físico (não necessariamente disponível no dispositivo). Finalmente, recursos podem ser específicos da aplicação, como uma variável ambiental cujo estado pode ser obtido interagindo com um sensor específico de medição de dados ambientais, como é o caso da luminosidade.

Não somente o modo como a informação de estado é obtida varia grandemente, mas também a natureza dessa informação é extremamente heterogênea. Por exemplo, a disponibilidade de memória ou bateria pode ser expressa facilmente com um valor numérico que é interpretado aproximadamente da mesma maneira por qualquer aplicação (por exemplo, valor absoluto ou porcentagem). Informação de localização, ao invés, pode ser representada de modos de diferentes (por exemplo, coordenadas no espaço, proximidade a objetos físicos, etc.) por diferentes tecnologias de sensoriamento (por exemplo, GPS, infravermelho, freqüência de rádio, etc.), e pode requerer vários processamentos para tornar-se significativa para as aplicações.

Configurações de contexto codificado em um perfil isolam tanto a técnica específica de sensoriamento usada quanto a informação de estado heterogêneo e não-processado. Assim essas configurações assumem que a informação de contexto já foi coletada e processada e que uma representação uniforme de estado de recurso será usada. Detalhes da codificação do perfil de aplicação serão apresentados na seção 7.3.

5.4 Configuração Dinâmica de Políticas através de Comandos

Em certas situações, informações de configuração devem ser atualizadas em tempo de execução de uma aplicação. Exemplificando, uma aplicação pode desejar ser notificada com novas trocas no contexto como

resultado de configurações de contexto não previstas, ou ela pode querer se adaptar ao fornecimento de um serviço de diferentes maneiras, devido a trocas nas necessidades dos usuários da aplicação.

Da perspectiva da adaptação reativa, a configuração dinâmica refere-se à capacidade das aplicações alterarem dinamicamente a maneira que a aplicação reage a mudanças no contexto, isto é, altera o conjunto de recursos que compõem o contexto de interesse da aplicação, e redefine as associações entre configurações de contexto pertinentes e comportamentos de adaptação.

Semelhantemente, para adaptação pró-ativa, a configuração dinâmica recorre à habilidade de aplicações alterarem dinamicamente a maneira com que os serviços são fornecidos em contextos diferentes. Isso recorre à habilidade de proceder duas redefinições: redefinir o conjunto de serviços que requerem customização, juntamente com as associações entre serviços feitos sob encomenda; e redefinir comportamentos possíveis para entregar estes serviços, juntamente com configurações de contexto que habilitam os vários comportamentos.

6 ARQUITETURA SENSORBUS

O SensorBus é um *middleware* orientado à mensagem (MOM – *Message-Oriented Middleware*) que utiliza o paradigma *publish-subscribe* (COULORIS et al., 2001). Nesse paradigma, um componente gerador de eventos (produtor) publica os tipos de eventos que ele tornará disponível para a observação de outros componentes (consumidores). O MOM executa as funções de coletar mensagens de produtores, filtrar e transformar tais mensagens quando necessário e roteá-las para consumidores apropriados.

O SensorBus é constituído dos seguintes elementos principais: serviço de aplicação, serviço de mensagem e serviço de contexto, como mostra a figura 6.1. Nas figuras seguintes, utilizam-se diagramas de componentes UML (*Unified Modeling Language*) (RUMBAUGH *et al*., 1998) para ilustrar cada um dos serviços.

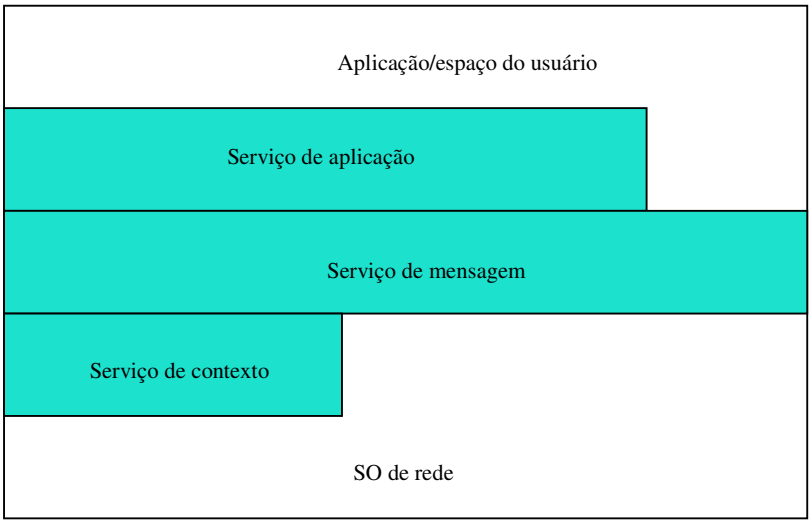


Figura 6.1: Arquitetura do SensorBus.

6.1 Serviço de Aplicação

O serviço de aplicação é responsável pelo fornecimento de uma API (*Application Programming Interface*) de alto nível que simplifica a construção de aplicações. Esse serviço é constituído de três componentes principais, ilustrados na figura 6.2:

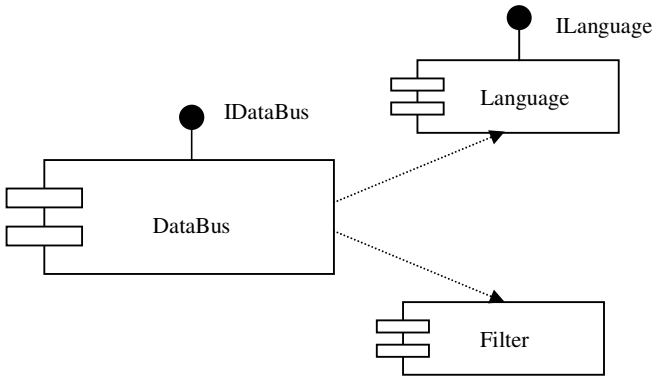


Figura 6.2: Arquitetura do serviço de aplicação.

- DataBus: é o componente que fornece operações para que produtores e consumidores se comuniquem através do barramento para: anunciar item de dados (produtor); encontrar um item de dados (consumidor); anunciar mudança nos dados (consumidor) e excluir itens de dados (produtor).
- Filter: é o componente responsável pela filtragem dos dados.
- Language: é o componente que implementa os comandos e o interpretador da linguagem de restrição/consulta.

6.2 Serviço de Mensagem

O serviço de mensagem é o responsável pela comunicação e coordenação dos componentes distribuídos, tornando essas questões transparentes para o usuário. Esse serviço é constituído de três componentes principais, como mostra a figura 6.3:

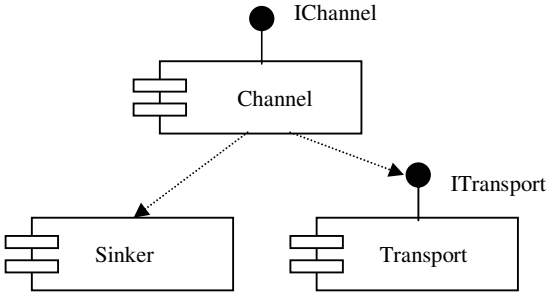


Figura 6.3: Arquitetura do serviço de mensagem.

- Channel: é o componente que possui operações para interagir com a implementação de transporte específica. Cada instância de Channel representa um simples canal no sistema. Channel mantém o estado global sobre os canais disponíveis e passa mensagens dos canais para a implementação de transporte de mensagem e recebe atualizações da implementação de transporte de mensagem.
- Transport: toda a comunicação entre os nós é feita através de uma implementação de transporte específica, tal como *sockets*. Cada implementação de transporte se comunica, através de um canal, com um servidor de passagem de mensagens, chamado de Sinker. As implementações de transporte implementam uma interface comum chamada de ITransport, o que as torna permutáveis.
- Sinker: é o componente responsável por rotear mensagens entre as instâncias da implementação de transporte, cada uma correspondendo a uma instância de Channel.

6.3 Serviço de Contexto

A perspectiva centrada nas aplicações inerentes a RSSFs impõe a captação de certas informações de contexto, como disponibilidade de recursos locais (capacidade de memória, energia da bateria, etc.), largura de banda, localização, informações específicas de aplicações (temperatura, pressão, etc.), entre outras. Um *middleware* obtém essas informações interagindo com uma variedade de sensores heterogêneos. Por exemplo, a energia disponível na bateria pode ser obtida executando uma primitiva do

sistema operacional, a localização pode ser calculada interagindo com várias tecnologias de comunicação (GPS, infravermelho e rádio freqüência), medidas de observação de interesses dos usuários podem ser obtidas através do tipo de sensores (acústico, temperatura, etc.) projetados para rede. Nesta pesquisa não se considera como o sensoriamento de contexto é executado; assume-se que cada sensor fornece uma interface que o SensorBus pode usar para obter o valor do recurso associado. Assim, de posse dessas interfaces, os recursos são tratados pelo serviço de contexto do SensorBus.

O serviço de contexto é responsável pelo gerenciamento de sensores heterogêneos que captam as informações do ambiente interno e externo ao nó sensor. Para cada recurso que o SensorBus controla, existe um adaptador que interage com o sensor físico processando sua informação e assim obtendo um valor que a aplicação precisa. Somente adaptadores de recursos que são necessários para aplicação em execução serão carregados para se evitar gastos dos escassos recursos computacionais dos nós sensores. A figura 6.4 ilustra um adaptador de energia interagindo com um sensor de energia (nesse caso, uma primitiva do sistema operacional de rede).

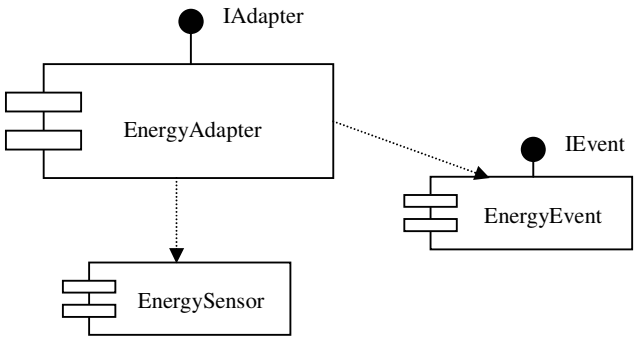


Figura 6.4: Arquitetura do serviço de contexto.

6.4 Distribuição de componentes na rede de sensores sem fio

A figura 6.5 mostra, através de um diagrama de implantação UML, a distribuição de componentes e serviços do SensorBus em uma RSSF. Nesse tipo de diagrama, as caixas em três dimensões representam nós de rede que podem ser dispositivos de *hardware* ou *software*. O nó User representa

um computador do tipo *desktop* ou *laptop* utilizado por usuários para acessar uma RSSF. O nó Sink representa dois dispositivos: um *desktop* interligado através da porta serial a uma placa de rede que contém um nó sensor sorvedouro. O nó Sensor representa os diversos nós sensores que constituem uma RSSF.

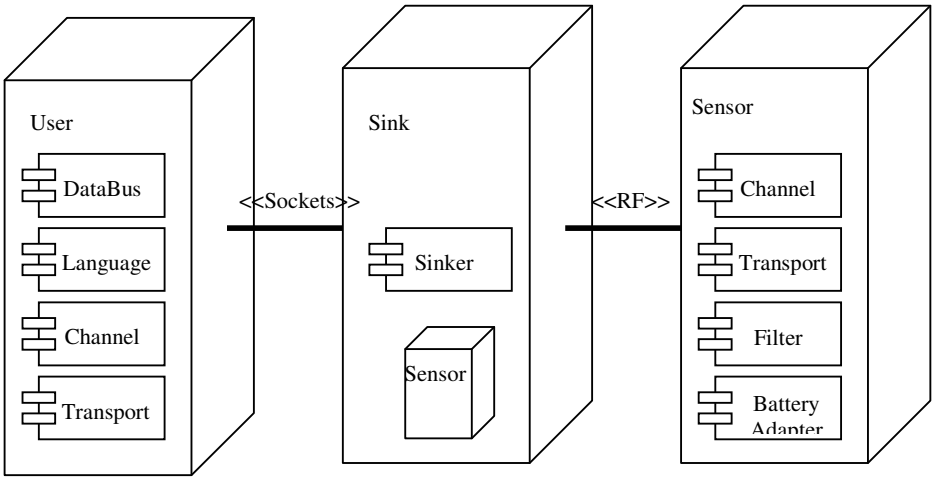


Figura 6.5: Diagrama de implantação UML.

Os serviços e componentes do SensorBus são distribuídos em três tipos distintos de nós de rede. O nó User contém os componentes DataBus, Language, Channel e Transport. O componente Sinker localiza-se no nó Sink. O nó Sensor contém, obrigatoriamente, os componentes Channel e Transport enquanto que o componente Filter e o serviço de contexto só serão carregados nesse tipo de nó caso a aplicação exija gerenciamento de energia e de outros recursos.

As conexões entre os nós de rede são feitas através de ligações físicas de baixo nível. A conexão entre o nó User e nó Sink pode ser feita através de um enlace Ethernet ou via rádio. Aqui, a conexão é apresentada através de *sockets* para facilitar a clareza do diagrama. A conexão entre o nó Sink e o nó Sensor é através de enlaces de RF, como indica o estereotipo <<RF>>.

6.5 Desenvolvimento de um aplicativo *publish-subscribe*

Desenvolver uma aplicação usando o SensorBus consiste em codificar as partes para o produtor e consumidor. O código do consumidor executa no nó de usuário enquanto que o código produtor executa nos nós sensores.

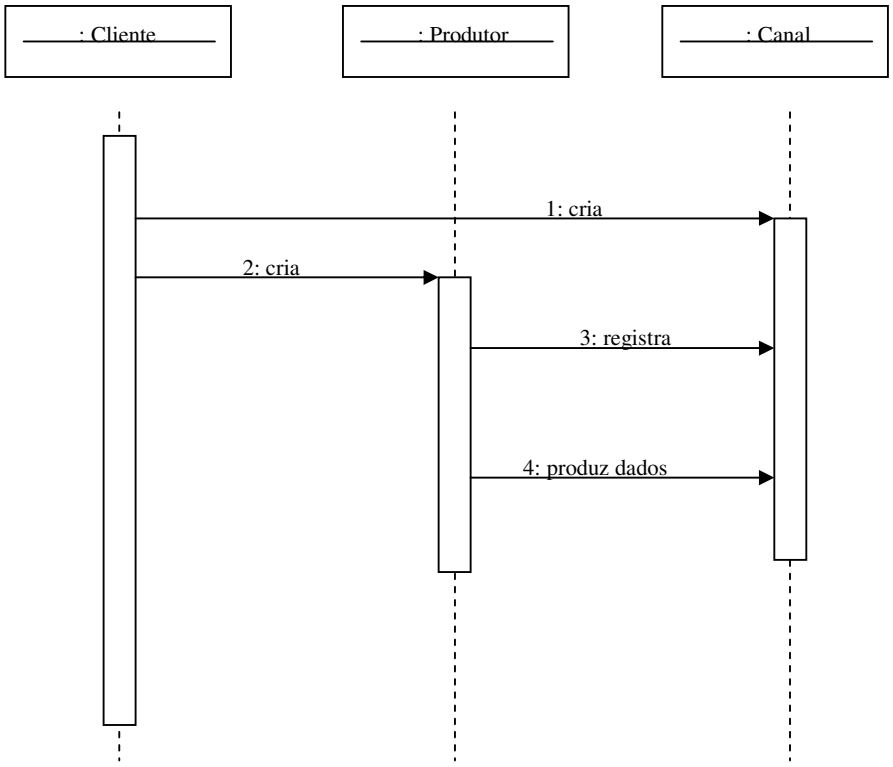


Figura 6.6: Procedimento para o produtor.

A figura 6.6 mostra o diagrama de seqüência UML para o procedimento do produtor. Os números que seguem correspondem aos números na referida figura:

1. cria uma instância um canal;
2. instancia um produtor;
3. registra o produtor para o canal criado; e
4. o produtor produz itens de dados e coloca no canal.

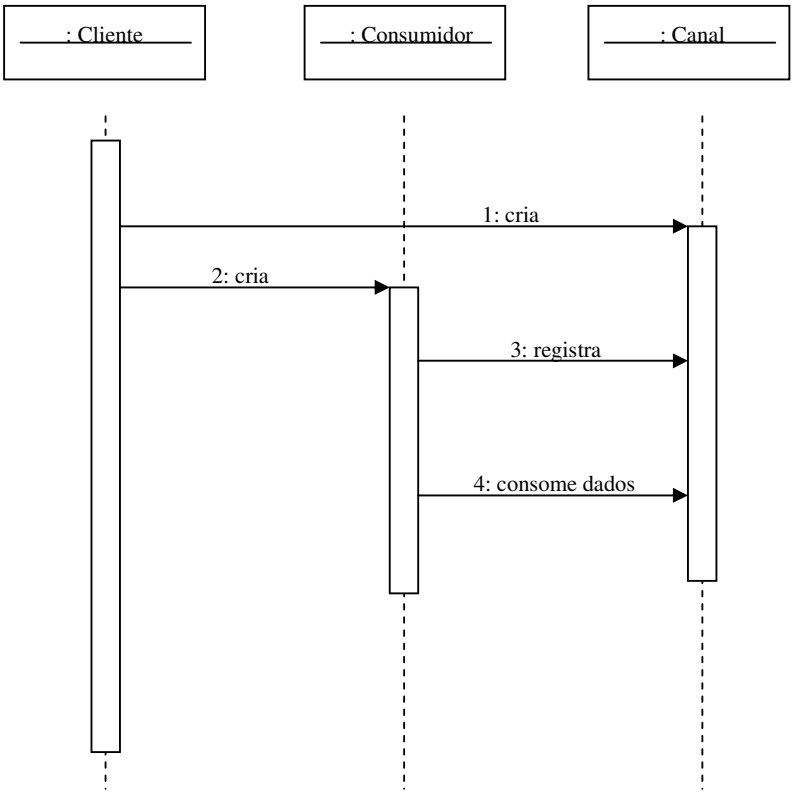


Figura 6.7: Procedimento para o consumidor.

A figura 6.7 mostra o diagrama de seqüência UML para o procedimento do consumidor. Os números que seguem correspondem aos números na referida figura:

1. cria uma instância de canal;
2. intancia um consumidor;
3. registra o consumidor para o canal criado; e
4. o consumidor encontra e consome itens de dados que foram colocados no canal.

O SensorBus oferece outras operações que podem ser implementadas, tais como listar os nomes dos canais disponíveis, adicionar novos canais, parar de receber novos canais, entre outras.

Observa-se que o código no nó sensor (produtor) deve ser implementado antes da instalação da rede. Se não for possível recuperar o nó sensor para a manutenção, os atributos dos dados enviados serão sempre os mesmos. Para contornar esse obstáculo utilizam-se as linguagens de restrição e de consulta que podem ser usadas para adicionar

novas consultas não previstas inicialmente. Essas consultas trafegam em forma de mensagens enviadas pelo consumidor (cliente) interessado.

Os filtros são usados na codificação implementada nos nós sensores. Logo depois de se instanciar um produtor, instancia-se um filtro e o registra para um determinado canal.

A figura 6.8 mostra os componentes que podem estar ativos em um determinado momento de execução de uma aplicação de monitoramento ambiental. Embora a maioria dos componentes seja os mesmos para uma dada aplicação, diferentes configurações ocorrem no serviço de contexto. A figura mostra apenas dois adaptadores executando no momento e interagindo com seus sensores associados (bateria e temperatura). Diferentes sensores podem ser usados dependendo da grandeza física que se deseja medir e do tipo de recurso computacional que se queira controlar.

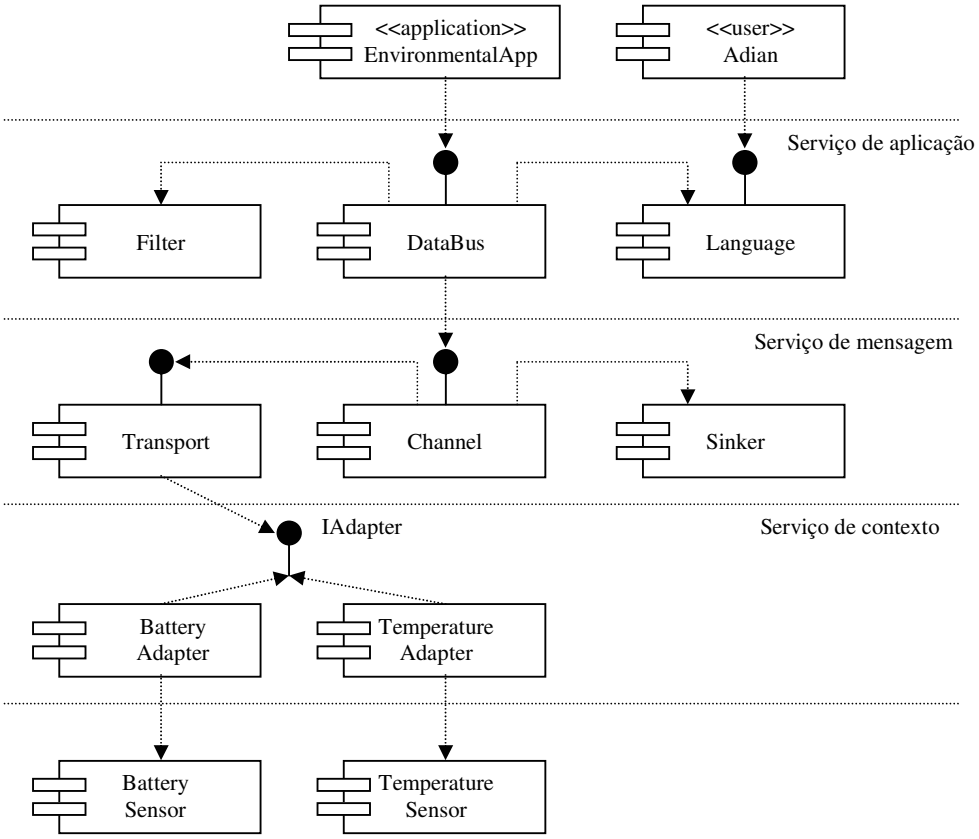


Figura 6.8: Exemplo de arquitetura do SensorBus.

6.6 Desenvolvimento de um aplicativo requisição-resposta

Para o desenvolvimento de um aplicativo multiusuário de requisição-resposta é necessário criar um aplicativo de consulta à RSSF e depois incluir um servidor *front-end* para permitir a outros clientes acessarem o aplicativo via um protocolo de comunicação como *sockets*, por exemplo. O servidor *front-end* é uma classe separada da classe original do aplicativo de consulta. Ele é apenas um configurador que aceita novas conexões e então cria objetos que interagem com o aplicativo de consulta e devolve o resultado para o cliente. Ele nunca interage com o aplicativo de consulta.

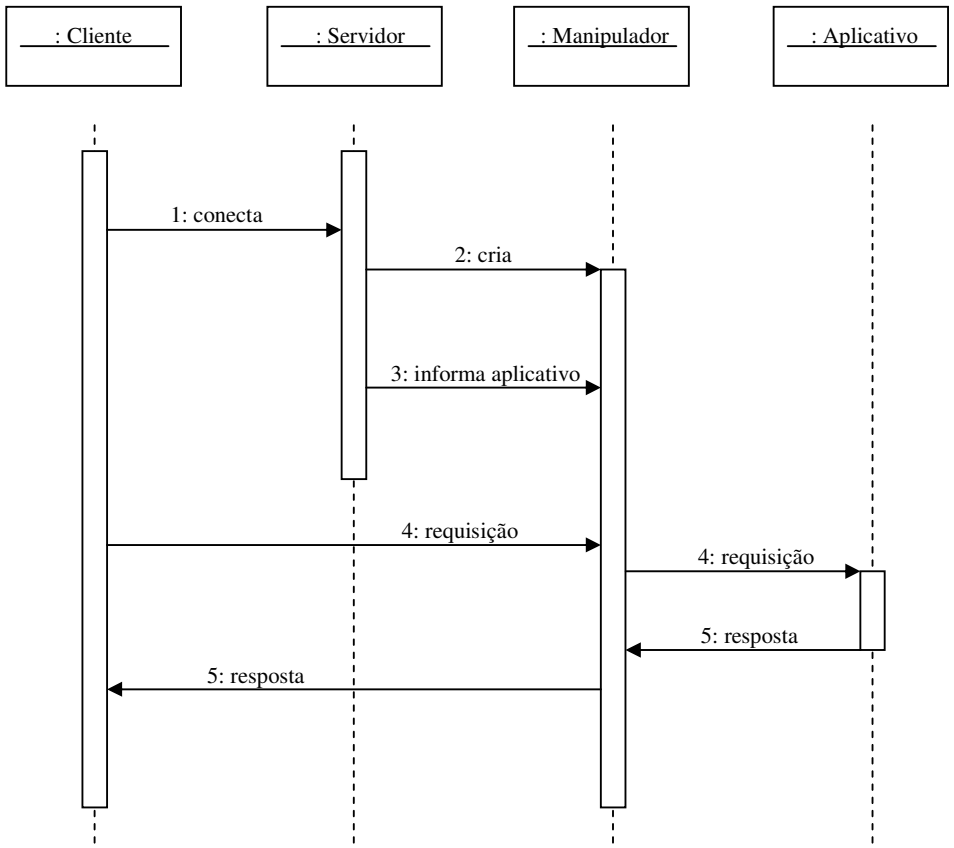


Figura 6.9: Aplicativo multiusuário de consulta.

A figura 6.9 ilustra o diagrama de seqüência UML da interação entre um cliente e o aplicativo de consulta. Os números que seguem correspondem aos números na referida figura:

1. um objeto Cliente para realizar uma consulta conecta-se ao Servidor *front-end*;

2. o Servidor cria um objeto Manipulador, que implementa a interface com o cliente do aplicativo de consulta;
3. o Servidor também informa ao Manipulador onde encontrar o objeto Aplicativo;
4. o objeto Cliente interage com o Aplicativo fazendo requisições através do objeto Manipulador; e
5. o objeto Aplicativo responde ao Cliente através do objeto Manipulador.

É importante salientar que o objeto Aplicativo faz a interação com uma RSSF. Após receber a solicitação, ele envia comandos para uma RSSF, recebe o resultado da consulta e retorna a resposta para o objeto Cliente.

7 IMPLEMENTAÇÃO E AVALIAÇÃO

7.1 Plataforma utilizada

A plataforma de *hardware* e *software* utilizada na implementação e avaliação do SensorBus é constituída de duas partes distintas. Uma parte constituída de PCs (*Personal Computers*) e outra parte constituída de nós sensores (*motes*), como mostra a figura 7.1

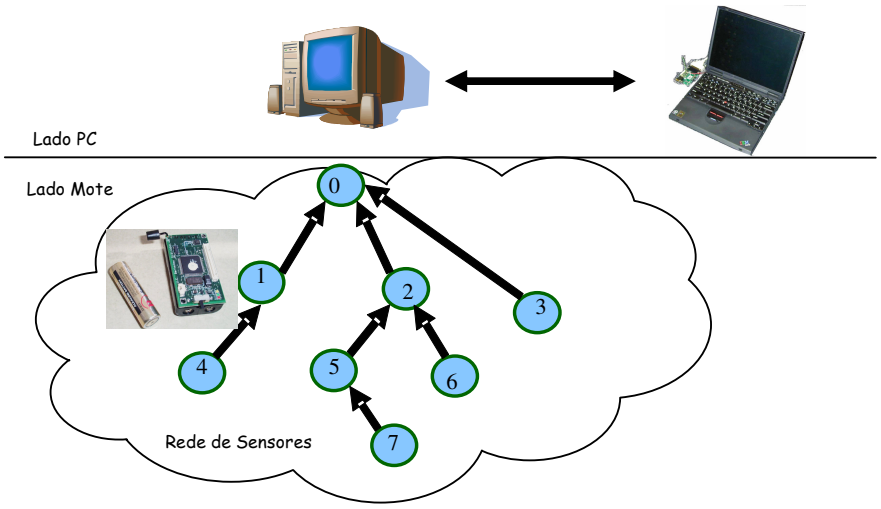


Figura 7.1: Cenário da plataforma utilizada.

A parte correspondente aos PCs é constituída de equipamentos com processadores Intel e interfaces 802.11b para comunicação sem fio. O PC que está conectado ao nó sorvedouro consiste em um *desktop* Dell Latitude equipado com 512 MB de RAM (*random access memory*) e processador com 1.6 GHz executando o sistema operacional Red Hat Linux 9.

Os nós sensores consistem de *motes* MICAz da Crossbow Technology, Inc equipados com processador ATMega129L (Atmel, 2005) de 7.3728 MHz

com 128 KB de memória de programa (FLASH) e 4KB de memória de dados (SRAM). Essa plataforma ainda possui 512 KB de memória externa (FLASH) para leitura de medições de sensores e um módulo de rádio CC2420 (Chipcon, 2005) de 2400 MHz capaz de oferecer uma largura de banda total de 250 kbps. Um *slot* de expansão acomoda uma variedade de placas de sensores tais como luz, temperatura, campo magnético, som, etc. A figura 7.2 ilustra esse tipo de dispositivo de sensoriamento.

Figura 7.2: Foto do nó sensor sem fio Micaz.

A extração de informações da RSSF foi realizada através da placa MIB510 (figura 7.3), que permite a conexão do PC a uma RSSF via porta serial. A figura 7.3 apresenta uma descrição dos equipamentos utilizados na RSSF:

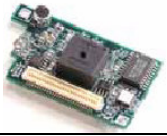
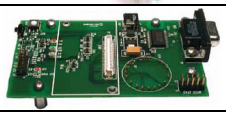
Equipamento	Descrição	Qte.
	MICAz (nó sensor): contém processador, memória, rádio transmissor/receptor e duas pilhas AA.	08
	MTS300CA (placa de sensor com sensores de luminosidade, temperatura, som).	03
	MTS310CA (placa de sensor com sensores de luminosidade, temperatura, som, vibração e aceleração).	04
	MIB510 (placa para comunicação com o PC via porta serial)	01

Figura 7.3: Plataforma de *hardware* utilizada na composição da RSSF.

A seguir apresenta-se a descrição da plataforma de *software* utilizada, a qual é subdividida em tecnologias de *software* que executam nos PCs e tecnologias de *software* que executam nos nós sensores MICAz.

7.1.1 Plataforma de *software* para PC

A plataforma de *software* para PC é constituída de tecnologias encontradas na maioria dos microcomputadores, tais como linguagens de programação, *parsers*, linguagens de marcação entre outras. A seguir, apresentam-se as tecnologias utilizadas nessa plataforma.

7.1.1.1 Java

Para o desenvolvimento dos componentes do SensorBus que executam em PCs, utilizou-se a linguagem de programação Java. Essa linguagem foi escolhida devido a suas várias características positivas para desenvolvimento de sistemas como (MARUYAMA *et al*., 1999):

- Independência de plataforma – Java é independente de plataforma, tais como arquiteturas de máquinas e sistemas operacionais. Os programas em Java são compilados em *bytecodes*, que por sua vez são interpretados por uma máquina virtual Java (JVM – *Java Virtual Machine*). Isso significa que os *bytecodes* podem executar em qualquer plataforma que tenha uma JVM.
- Alta produtividade – as características de Java que contribuem para alta produtividade incluem: separação entre a interface e a implementação; ausência de herança múltipla e sobrecarga de operador; suporte para *threads* e monitores.
- Suporte à Internet embutido – o pacote de rede Java (java.net) contém várias rotinas de conectividade à Internet. Em particular, as classes URL e URLConnection tratam de detalhes do protocolo HTTP (*HyperText Transfer Protocol*) e tornam simples a conexão com um servidor *Web*.

- Suporte a caracteres internacionais – Java usa *Unicode* como conjunto de caracteres. A máquina virtual Java (JVM) possui tabelas de conversões que contêm codificações de caracteres.

7.1.1.2Tecnologias XML

XML é uma linguagem de marcação que serve para descrever dados estruturados e semi-estruturados. Assim como outras linguagens de marcação do tipo generalizada, a XML lida com instruções embutidas no corpo de documentos chamadas *tags*, que permitem a descrição de dados.

A XML é um subconjunto da *Standard Generalized Markup Language* (SGML) e é padronizada pelo *World Wide Web Consortium* (W3C). A SGML não é usada para marcação de documentos; é uma meta-linguagem (uma meta-linguagem é uma linguagem que descreve outra linguagem) que é usada para criar linguagens de marcações para diferentes domínios de aplicações. As linguagens de marcação definidas usando SGML são conhecidas como aplicações de SGML. Existem diversas aplicações SGML, entre elas DocBook, uma aplicação SGML projetada pela *Association of American Publishers* (AAP) para livros, artigos e jornais. A *Hypertext Markup Language* (HTML) é uma outra aplicação SGML usada para descrever conteúdo de um documento *Web* (LEE *et al*., 2000).

Enquanto que em HTML definem-se *tags* que descrevem como os dados devem ser exibidos, se eles devem ser apresentados em negrito, em uma lista numerada ou de marcadores, em XML tem-se somente elementos e marcações. Folhas de estilos e *links* vêm em separado, e não escondidas no documento. Cada um pode ser acessado e alterado separadamente, quando preciso. A XML, ao contrário da HTML, permite que os desenvolvedores especifiquem uma série ilimitada de *tags* para abranger toda a riqueza do domínio a ser descrito (YOUNG, 2000).

Esquemas XML

Para criar conjuntos de *tags* que reflitam as necessidades de um domínio de conhecimento específico, deve-se criar uma DTD (*Document Type Definition*), que formalmente identifica o relacionamento entre os

vários elementos que formam um documento. As DTDs são opcionais e os dados enviados com uma DTD são conhecidos como dados XML válidos. Um analisador de documentos pode verificar os dados que chegam, analisando as regras contidas na DTD para ter certeza de que o dado foi estruturado corretamente. Os dados enviados sem DTD são conhecidos como dados bem formados. Nesse caso, o documento pode ser usado para implicitamente se autodescrever (Bourret, 1999) (Young, 2000).

Uma outra proposta da W3C para descrever a estrutura de um documento XML é o XML *Schema*, um padrão mais abrangente que uma DTD, permitindo expressar tipos de dados, herança, tipos abstratos, unicidade e chaves, entre outros (W3C, 2000). Esquemas são meta-descrições usadas para descrever: a ordem e a estrutura interna dos elementos; quais tipos de dados são aceitáveis; que valores estes tipos podem conter, em qual ordem e com qual freqüência eles podem ocorrer (W3C, 2000). O esquema especifica os tipos de dados no documento e como eles devem ser processados. É possível criar um documento sem um esquema, mas não se pode processá-lo. De fato, a maioria dos documentos segue um esquema definido previamente.

DOM

Document Object Model (DOM) (APPARAO *et al*., 1998) é uma recomendação que provê formas de acesso aos dados estruturados (documentos XML) utilizando *scripts*, permitindo aos desenvolvedores interagir e computar tais dados consistentemente. Os documentos XML são representados como uma estrutura hierárquica de árvore na memória (Byrne *et al*., 1998), a qual contém os elementos do documento, dos atributos, do conteúdo, etc. Um analisador baseado no DOM expõe (ou seja, torna disponível) uma biblioteca para uso em programas - denominada interface de programação de aplicação DOM – que permite aos dados em um documento XML serem acessados e modificados pela manipulação dos nodos em uma árvore DOM.

XPath

A XML fornece um modo de descrever dados de forma rica, flexível e eficiente através da marcação dos dados com marcas descritivas. Porém, ela não fornece um meio de localizar partes específicas de dados estruturados dentro de um documento. Por exemplo: um documento XML contendo dados sobre dissertações catalogadas na biblioteca central da UFPA necessitaria ser analisado e depois pesquisado elemento por elemento, de modo a encontrar uma dissertação específica. Para grandes documentos, esse processo é ineficiente e sujeito a erros.

A linguagem XML Path (XPath) (CLARK & DeRose, 1999) fornece uma sintaxe para localizar partes específicas (por exemplo, valores de atributos) de um documento XML efetiva e eficientemente. A XPath não é uma linguagem estrutural como XML, mas uma linguagem de expressões baseada em *strings*, usada por outras tecnologias XML tais como o DOM.

7.1.2 Plataforma de *software* para *motes* MICAz

Os nós sensores MICAz dispõem de uma plataforma de *software* bastante desenvolvida, considerando-se o fato de que uma RSSF é um novo paradigma de comunicação que ainda está se estruturando. A seguir, apresenta-se uma breve descrição dos principais *softwares* utilizados nessa plataforma.

7.1.2.1 TinyOS

Os *motes* MICAz executam o sistema operacional de código aberto TinyOS (*Tiny Microthreading Operating System*) (HILL *et al*., 2000), projetado especificamente para RSSFs. Trata-se de um escalonador de eventos bastante simples quando comparado com os atuais sistemas operacionais de computadores pessoais. Seu objetivo é reduzir a memória ocupada e o custo do sistema. Suas principais características são:

- Ocupação de espaço de memória limitado na ordem de 178 *bytes* dos 512 *bytes* disponíveis nos *motes*;
- Operações concorrentes intensas;

- Alocação dinâmica de memória;
- Modularidade eficiente;
- Diversidade no uso e operação robusta.

A figura 7.4 (HILL *et al*., 2000) mostra a arquitetura simplificada do TinyOS. Essa arquitetura é baseada em componentes que permite minimizar o tamanho do código necessário para executar no nó sensor, característica importante devido à limitação de memória inerente a RSSFs. A biblioteca componente do TinyOS inclui protocolos de rede, serviços distribuídos, *drivers* de sensor e ferramentas para aquisição de dados. Além de um sistema operacional, o TinyOS pode ser definido também como uma plataforma de desenvolvimento para uma RSSF, pois ele contém uma gama de programas que visam facilitar, na medida do possível, o trabalho do programador. Como exemplo desses programas destacam-se os compiladores NesC, Gcc, Assembler, entre outros. No desenvolvimento dos experimentos utilizou-se a versão 1.1.14 do TinyOS.

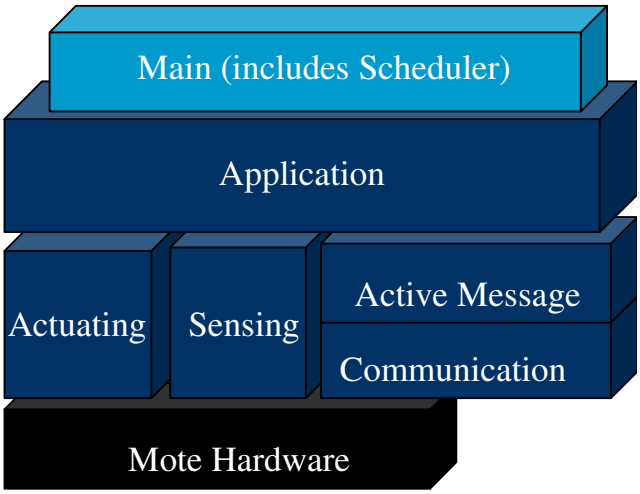


Figura 7.4: Arquitetura simplificada do TinyOS.

7.1.2.2NesC

Todo o sistema operacional TinyOS, as bibliotecas, e as aplicações são escritos na linguagem NesC (GAY *et al*., 2002), uma nova linguagem baseada em componente e estruturada. Essa linguagem foi definida

primeiramente para sistemas embarcados, tais como RSSFs. O NesC tem uma sintaxe semelhante à linguagem C mas suporta o modelo da simultaneidade do TinyOS, assim como os mecanismos para estruturar, nomear e ligar os componentes de *software* na rede de sistemas embarcados. O objetivo principal é permitir aos projetistas de aplicação construírem os componentes que possam facilmente ser compostos em sistemas completos e simultâneos. Nesta tese utilizou-se a versão 1.2 do NesC, a qual é pré-requisito para instalação da versão 1.1.14 do TinyOS.

7.1.2.3TinySchema

TinySchema é uma coleção de componentes TinyOS que gerencia um pequeno repositório de atributos, comandos e eventos que podem ser consultados, invocados ou assinalados de dentro ou de fora de uma rede de *motes* (HONG & MADDEN, 2003).

Um atributo é como uma coluna de uma tabela em um sistema de banco de dados tradicional, tem um nome e um tipo. O TinySchema permite, usando código TinyOS, obter ou estabelecer o valor do atributo. Uma vez que um atributo é criado, ele pode ser recuperado ou atualizado através de uma interface unificada fornecida pelo TinySchema. Existem três classes de atributos:

- Atributos de sensor – Podem ser leituras de nós sensores como temperatura, umidade, luminosidade, etc.
- Atributos introspectivos – São valores dos estados internos, tanto de *hardware* quanto de *software*, tais como o nó sensor pai em uma árvore de roteamento, tensão da bateria, etc.
- Atributos constantes – São valores constantes assinalados para um *mote* em tempo de compilação ou execução. Por exemplo, identificação de nó sensor, identificação de grupo de nó sensor, nome, localização, etc.

Um comando é como um procedimento armazenado de um sistema de banco de dados tradicional e consiste de um nome, uma lista de argumentos e um tipo de retorno. Associa-se código TinyOS para

determinado comando. O TinySchema fornece uma interface unificada para a invocação de comandos. Existem duas classes de comandos:

- Comandos de atuação – São comandos que causam alguma ação física em um *mote*, por exemplo, acender um LED, acionar um alarme, etc.
- Comandos de sintonização – São comandos que ajustam parâmetros internos da rede, por exemplo, política de roteamento, número de retransmissões, taxa de amostragem, etc.

Um evento é necessário para capturar eventos assíncronos em uma RSSF, por exemplo, detecção de um pássaro, acionar um botão de interface, etc. O TinySchema fornece interfaces para registrar e invocar eventos e também associar comandos com eventos como *callbacks* quando os eventos forem assinalados.

7.1.2.4TOSSIM e PowerTOSSIM

TOSSIM (LEVIS *et al*., 2003) é um simulador de evento discreto para RSSFs que usam o TinyOS. Em vez de compilar uma aplicação de TinyOS em um *mote*, os usuários podem compilá-lo na estrutura do TOSSIM, que funciona em um PC. Isso permite aos usuários eliminarem erros, testarem e analisarem os algoritmos em um ambiente controlado e repetível. Usando o TOSSIM os usuários podem examinar seu código de TinyOS usando *debuggers* e outras ferramentas de desenvolvimento.

PowerTOSSIM (SHNAYDER *et al*., 2004) é um módulo para cálculo do consumo de energia do simulador TOSSIM. Esse módulo fornece um consumo de energia estimado por dispositivo em cada nó com precisão entre 0,45 e 13% em relação a medições em nós sensores reais.

7.2 Implementação

A implementação atual do SensorBus é constituída de três módulos de *software*: um módulo que executa no PC, outro módulo que executa nos nós sensores e um módulo necessário para interligar os módulos anteriores. O módulo que executa no PC foi implementado em Java usando JDK 1.4.2

enquanto perfis de aplicação foram codificados em XML. A XML foi utilizada porque suporta uma representação de informação que é facilmente manipulada por computadores e é bem entendida por pessoas. Utilizou-se também algumas tecnologias XML relacionadas, em particular DOM e XPath. A utilização de *parsers* XML disponíveis reduziu consideravelmente o tempo de desenvolvimento do SensorBus.

Em linguagem nesC desenvolveu-se o módulo que executa nos nós sensores. Esse módulo é responsável pelo sistema de processamento de consulta para extrair informações da RSSF e foi desenvolvido com os componentes do TinySchema. Dada certa consulta, especificando os dados de interesses, esse módulo coleta os dados dos nós sensores no ambiente, filtra, agrega e faz o roteamento para o PC.

Implementou-se um *proxy* em Java para realizar a interligação entre a rede de PCs e a rede de *motes*. O *proxy* executa no PC conectado ao nó sorvedouro e acessa a rede de *motes* através da porta serial. Ele é usado para ler pacotes de dados que chegam pela porta serial e enviá-los através de uma conexão de porta TCP, de modo que programas possam se comunicar com rede de *motes* via nó sorvedouro. A ferramenta MIG (*Message Interface Generator*) (HILL *et al*., 2000) foi usada para gerar automaticamente classes Java que correspondem aos tipos de mensagens ativas usados nos componentes codificados em nesC, que executam nos *motes*. Usando MIG, contornou-se a dificuldade de traduzir os formatos de mensagens para utilização no *proxy* em Java.

7.3 Avaliação qualitativa

Um dos objetivos da avaliação qualitativa é suportar a tese de que o modelo de SensorBus fornece abstrações significantes aos programadores de aplicação, o que facilita o desenvolvimento de aplicações para RSSFs. Validar essa tese é difícil, pois não existem parâmetros quantitativos que se possam calcular quando da execução de um conjunto de experimentos. Em vez disso, verificou-se a usabilidade do SensorBus implementando uma aplicação de consulta de variáveis ambientais.

Uma das vantagens do modelo de SensorBus é que os programadores não têm que manusear explicitamente os sensores físicos que coletam as informações de contexto; a adaptação nas trocas de contexto é executada automaticamente pelo *middleware*, usando a informação codificada no perfil de aplicação. Então, a primeira preocupação que se apresenta aos desenvolvedores de aplicação em geral é fornecer um mecanismo para obter essa informação dinamicamente do usuário, e transformá-la em codificação XML usando a interface gráfica que o SensorBus fornece, como mostra a figura 7.5.

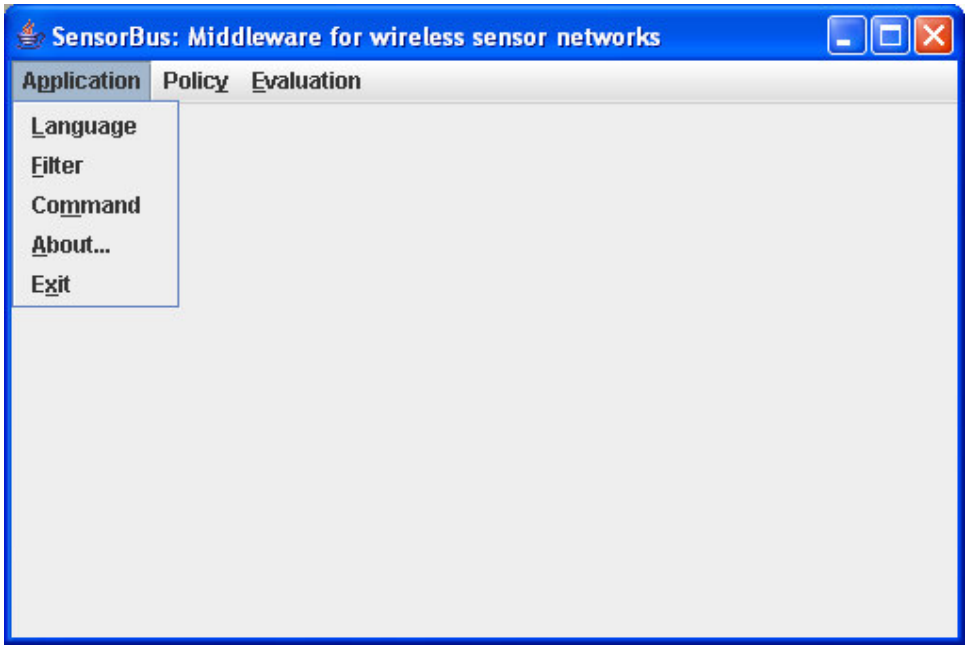


Figura 7.5: Interface gráfica do SensorBus.

Para aplicação de monitoramento ambiental, os atributos que podem ser coletados são temperatura, luminosidade, pressão e umidade. Para que os usuários especifiquem, em qualquer momento, os parâmetros a serem utilizados, uma janela de customização foi desenvolvida, como mostrado na figura 7.6.

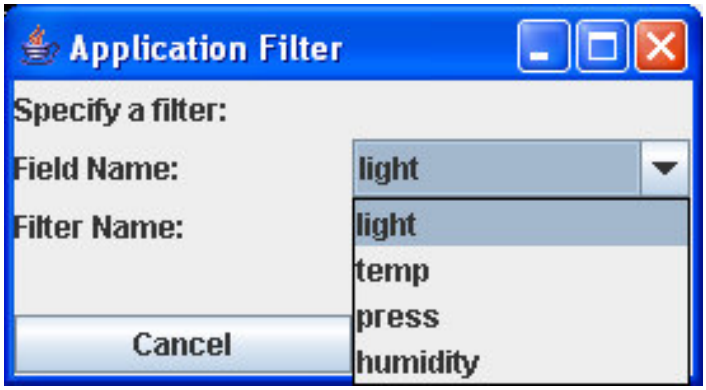


Figura 7.6: Aplicação de Monitoramento Ambiental.

Após, decidiu-se quais políticas deveriam ser implantadas nos perfis de aplicação. As políticas de aplicação estabelecidas foram as seguintes: período de amostragem, e se são ou não permitidas agregações, como mostra a figura 7.7. Em relação às políticas de comunicação, foram usadas: tipo de roteamento (central ou distribuído), tipo de consulta (dirigida a eventos ou contínua) e a especificação do intervalo de *broadcast*, como mostra a figura 7.8. Os níveis de bateria, memória e intensidade do sinal de rádio foram escolhidos como políticas de contexto. O usuário é notificado quando a disponibilidade desses recursos cai abaixo de um valor específico. A figura 7.9 mostra a janela customizada para entrada de políticas de contexto.

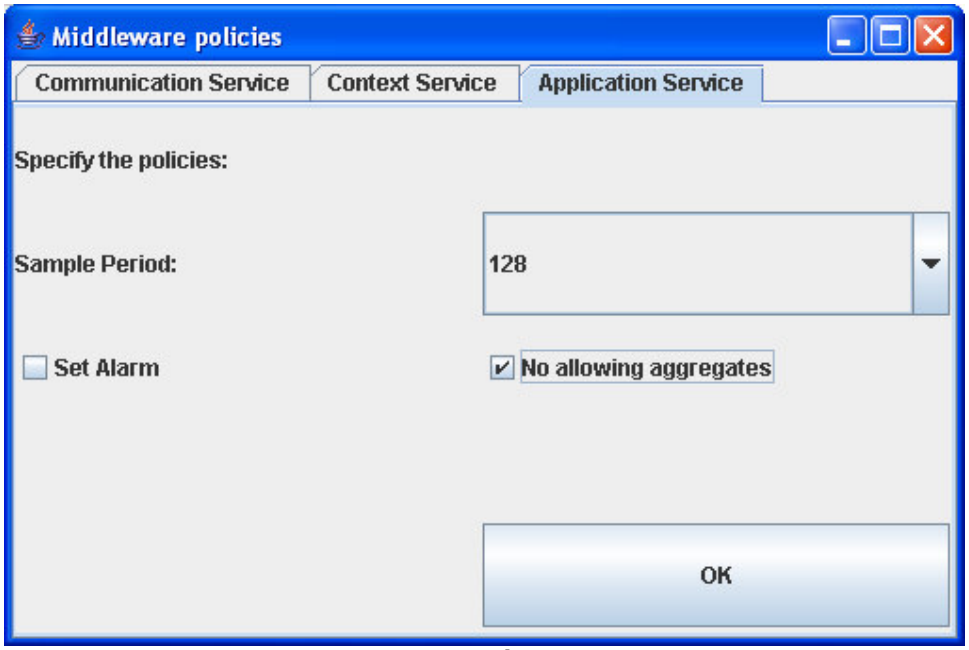


Figura 7.7: Políticas de aplicação.

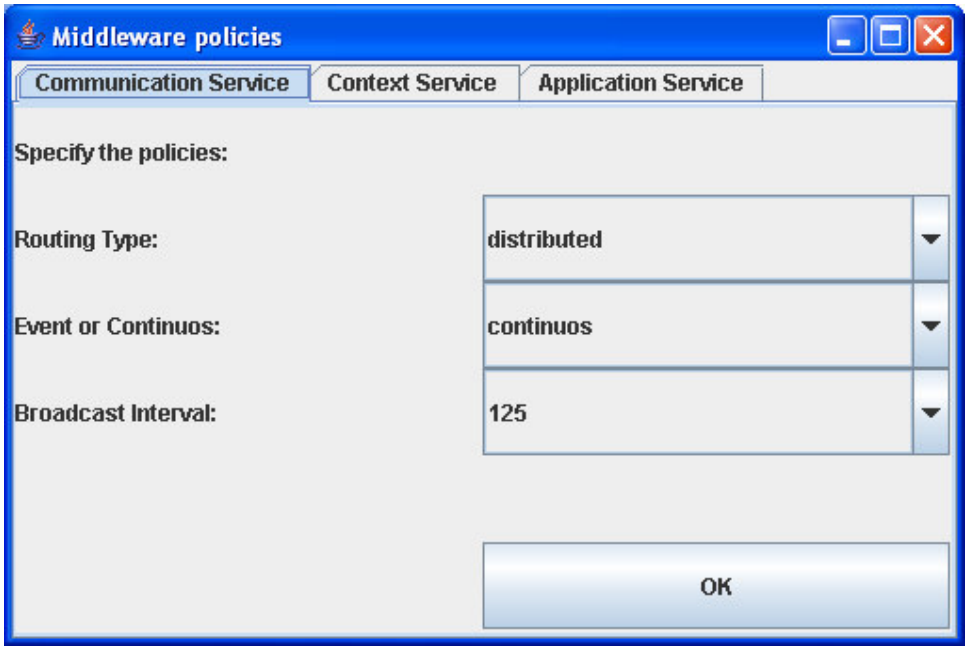


Figura 7.8: Políticas de comunicação.

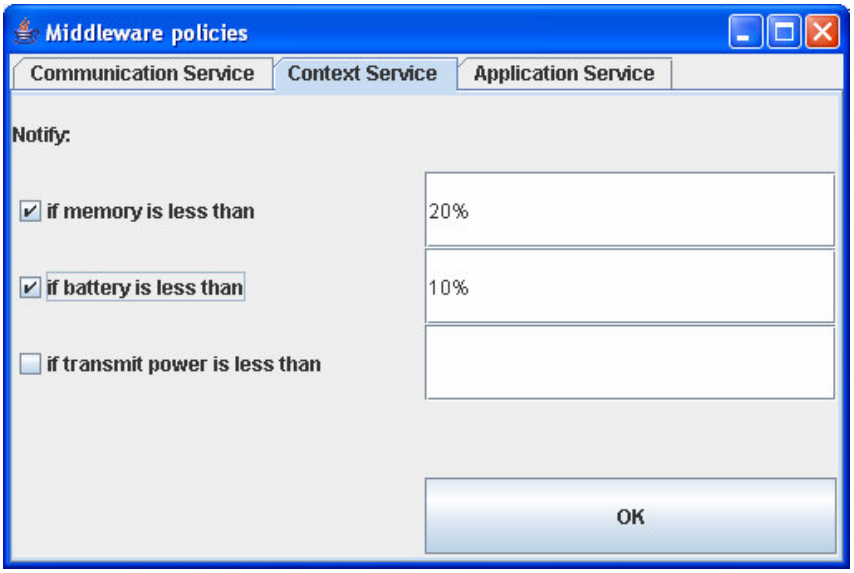


Figura 7.9: Políticas de contexto.

Baseado nessas preferências implementou-se um algoritmo para escrever o perfil de aplicação. Por exemplo, com referência às preferências expressas pelo usuário como mostrado na figura 7.9, o SensorBus fornece a codificação de metadados reativo como mostra a figura 7.10.

```
<REACTIVE frequency="5000">
 <POLICY name="lowMemoryAlert">
  <CONTEXT id="1">
   <RESOURCE name="memory">
    <OPERATOR name="lessThan"/>
    <STATUS value="20%"/>
   </RESOURCE>
  </CONTEXT>
 </POLICY>

 <POLICY name="lowBatteryAlert">
  <CONTEXT id="2">
   <RESOURCE name="battery">
    <OPERATOR name="lessThan"/>
    <STATUS value="10%"/>
   </RESOURCE>
  </CONTEXT>
 </POLICY>
</REACTIVE
```

Figura 7.10: Perfil XML – codificação reativa.

O usuário requer uma notificação quando a memória e o nível da bateria caem abaixo de 20% e 10% respectivamente de seus valores iniciais. Nesse caso não houve interesse pelos valores de intensidade do sinal de rádio. O atributo *frequency* representa o intervalo de tempo em milissegundos entre duas consultas de contexto consecutivas feitas pelo SensorBus. Esse atributo é definido pelo usuário e representa a velocidade de adaptação pelas mudanças de contexto.

Um esquema XML foi codificado através do XML *Schema* e define a gramática da linguagem usada para codificar o perfil de aplicação. A implementação do SensorBus permite a validação ser feita usando esse esquema ou através da interface que o *middleware* fornece aos desenvolvedores de aplicações. As aplicações não têm acesso direto à codificação XML.

Pelo exposto acima, observa-se que é bastante direta a utilização das abstrações e mecanismos que o SensorBus fornece. A tarefa mais

trabalhosa que o programador encontra é decidir quais políticas interessam ao usuário final. O programador deve desenvolver um algoritmo para mapear as preferências do usuário em perfis de aplicação. Essas tarefas podem ser simplificadas com ajuda de um especialista de um domínio de aplicação. A tarefa trabalhosa de consultar sensores heterogêneos para a coleta e manutenção de informações de contexto é completamente transparente para os desenvolvedores de aplicações.

7.4 Avaliação de desempenho

Para uma avaliação completa do SensorBus, levou-se em conta seus principais serviços: serviço de aplicação, serviço de mensagem e serviço de contexto. Cada serviço sofre o efeito que ocorre nas diversas camadas da pilha de protocolos que suportam a RSSFs. Como mostra a figura 7.11, nessas redes as camadas de protocolos essenciais são: protocolo MAC na camada de enlace de dados e o protocolo de roteamento na camada de rede. O protocolo MAC cria a topologia da rede e compartilha o meio de transmissão entre os nós sensores. O protocolo de roteamento permite a comunicação através de caminhos *multi-hop*. O protocolo de transporte que implementa o controle de fluxo fim-a-fim é raramente utilizado em RSSFs. A camada de *middleware* é equivalente à camada de apresentação no modelo OSI.

A figura 7.11 mostra que cada serviço do SensorBus está associado a um determinado conjunto de camadas. Por exemplo, o serviço de mensagen está acima do nível da camada de rede, sofrendo influências das camadas que estão abaixo. O serviço de aplicação sofre influência de todas as camadas da pilha enquanto que o serviço de contexto é responsável pelo gerenciamento de políticas.

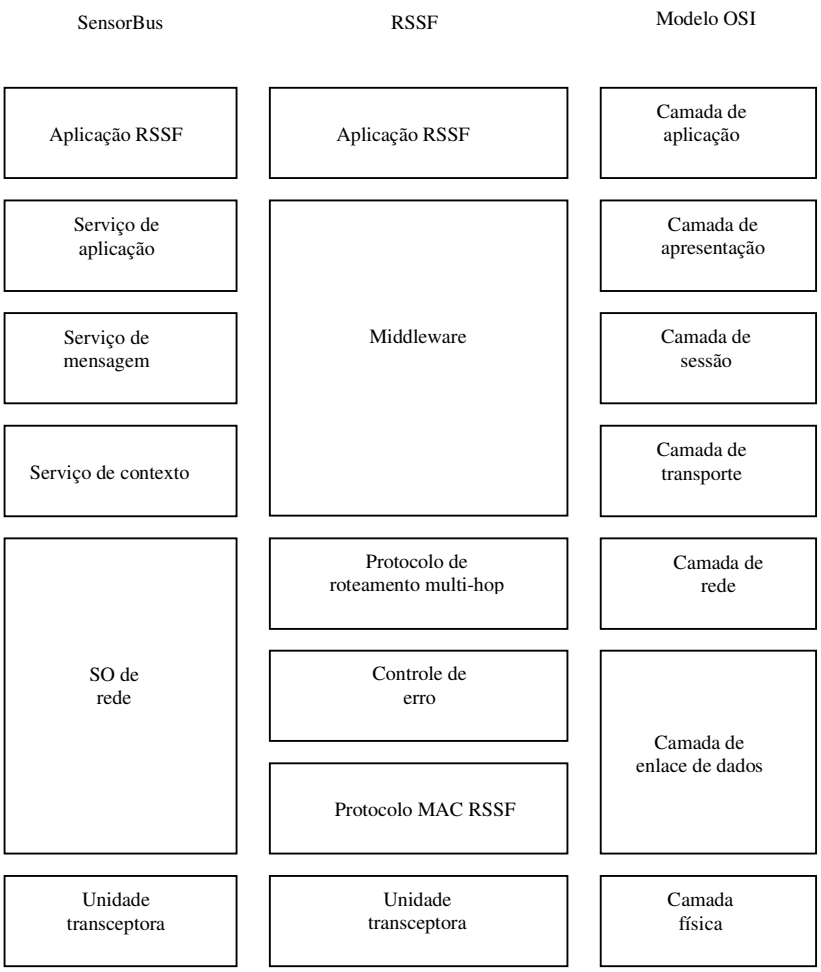


Figura 7.11: Modelo OSI, RSSF, e Serviços do SensorBus.

7.4.1 Serviço de Aplicação

Para avaliar o serviço de aplicação, o desempenho de duas consultas do SensorBus usando sua linguagem de restrição foram comparadas. Utilizou-se uma consulta simples e outra consulta com agregação. Utilizou-se o simulador TOSSIM para simular a parte da rede constituída de nós sensores (*motes*), ficando a outra parte da rede inalterada (parte constituída de PCs). As métricas de desempenho foram o tempo de resposta e o consumo de energia. Utilizou-se o módulo para cálculo do consumo de energia PowerTOSSIM do simulador e os parâmetros de simulação foram ajustados para corresponder a uma rede baseada no nó sensor MICAz.

7.4.1.1 Consumo de energia

Foram realizadas simulações em uma RSSF composta de 10 nós sensores estacionários e homogêneos. A carga utilizada correspondeu a duas consultas usando a linguagem de restrição do SensorBus. A Consulta 1 para amostrar a temperatura a cada 2048 ms e a Consulta 2 para amostrar a média da temperatura a cada 2048 ms usando um filtro de agregação. Gravou-se o consumo de energia para cada nó sensor fazendo 30 simulações de 300 segundos simulados cada uma. Esse número de simulações foi escolhido porque testes com um número maior de replicações não mostraram nenhuma variação significativa nos resultados. O resultado final se refere à média aritmética (consumo de energia) das 30 replicações e foram utilizados intervalos de confiança assintóticos ao nível de 95%.

Devido ao nó sorvedouro não ser usualmente alimentado por baterias, seu consumo de energia não foi considerado. O consumo de energia dos outros nós sensores é mostrado na figura 7.12. Em média, o uso de filtro de agregação (Consulta 2) consumiu 1 Joule menos energia do que a consulta simples (Consulta 1 - sem utilização de filtros). Em outras palavras, para uma dada carga de bateria idêntica, o uso de filtros de aplicação do SensorBus mantém uma RSSF ativa por 25 segundos a mais do que sem a utilização de filtros, nesse cenário descrito (tempo de simulação de 300 segundos e RSSF de 10 nós sensores). Em geral as médias das repetições apresentaram poucas variações, por isso os intervalos de confiança foram muito pequenos e sobrepostos, garantindo assim uma maior confiabilidade estatística aos resultados.

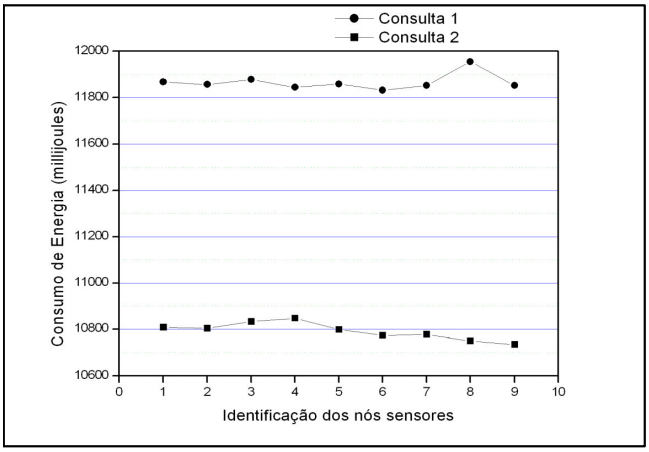


Figura 7.12: Consumo de Energia para as duas consultas: Consulta 1 e Consulta 2.

7.4.1.2Tempo de resposta

Nesse experimento, variou-se o número total de nós sensores simulados de 10 até 90 em passos de 10. A carga utilizada correspondeu às mesmas consultas anteriores. Gravou-se o tempo de resposta para 30 simulações de 10 segundos simulados cada uma. Esse número de simulações foi escolhido porque testes com um número maior de replicações não mostraram nenhuma variação significativa nos resultados. O resultado final se refere à média aritmética (tempo de resposta) das 30 replicações. O resultado final se refere à média (tempo de resposta) das médias calculadas para cada uma das 30 replicações e foram utilizados intervalos de confiança assintóticos ao nível de 95%.

A figura 7.13 mostra o resultado para 10, 20, 30 e 90 nós sensores, pois as demais simulações mantiveram o mesmo comportamento. Observou-se aqui também a eficiência da utilização de filtros no tempo de resposta em relação à consulta sem usar filtros. A eficiência aumenta conforme aumenta o número de nós sensores na RSSF, chegando a 0,5 segundo de diferença quando se usa uma RSSF de 90 nós sensores. As considerações sobre a confiabilidade estatística dos resultados são iguais às descritas para o experimento sobre o consumo de energia.

Número de nós sensores	Consulta 1	Consulta 2 (com agregação)
10	3980	3894
20	8180	8095
30	11990	11775
90	36320	35759

Figura 7.13: Tempo de resposta médio para duas consultas em milisegundos

7.4.2 Serviço de Mensagem

Para avaliação de desempenho do serviço de mensagem considerou-se o fornecimento de pacotes na camada física, camada MAC e camada de rede. Como o fornecimento de pacotes nessas redes é fortemente influenciado pelo protocolo de roteamento, analisou-se o impacto de dois protocolos *multi-hop* no serviço de mensagem enquanto controlaram-se os fatores que influenciam a camada física e camada MAC.

Na camada física, as funções de enquadramento e correção de erros são afetadas por vários fatores. Primeiro, características ambientais causam recepção do sinal por múltiplos caminhos, ou atenuação do sinal. Segundo, a distância entre emissor e receptor pode determinar a intensidade do sinal recebido. Finalmente, pequenas variações nos circuitos do emissor/receptor ou nos níveis da bateria podem afetar as funções da camada física. O experimento foi realizado em um mesmo ambiente fixo, sem interferências ambientais e em condições estáveis, para que as causas de influência nessa camada fosse a mínima possível.

Além dos fatores que influenciam a camada física, na camada MAC as funções de disciplina de acesso ao canal e detecção de erros são afetadas por dois fatores: a quantidade de mensagens geradas pelos nós sensores e a topologia (relação espacial entre nós sensores). No experimento, manteve-se a mesma topologia e carga para evitar a influência dessa camada. Foi utilizado o protocolo de enlace B-MAC devido a sua eficiência no fornecimento de pacotes decorrente da utilização de uma maior largura de banda, quando comparados com outros protocolos de enlace (POLLASTRE & CULLER, 2003).

Os protocolos de *multi-hop* usados no experimento foram: LEPSM (*Link Estimation and Parent Selection Method*) (HOHLT, 2005) e MintRoute (WOO et al., 2003). O algoritmo de roteamento *multi-hop* LEPSM é baseado no mecanismo de estimação de enlace e de seleção do pai (*Link Estimation and Parent Selection* - LEPS) para a execução *multi-hop*. O LEPS é responsável por monitorar todo o tráfego recebido no nó e recebe diretamente as mensagens de atualização de rota de um único salto. Essas mensagens podem ser enviadas dos vizinhos dentro da única escala de saltos. Internamente, o LEPS gerencia os vizinhos disponíveis mais próximos e decide o destino do salto seguinte baseado na semântica do menor caminho. Por padrão, o LEPS envia uma mensagem de atualização de rota a cada 10 segundos e recalcula após 50 segundos (5 mensagens de atualização de rota).

Outro protocolo de roteamento *multi-hop* abordado foi o MintRoute, também chamado WMEMA (*Window Mean with Exponentially Weighted Moving Average*). Esse protocolo é baseado na técnica de estimação de enlace que calcula a taxa média de sucesso sobre o período de tempo e ajusta através da técnica de ajustamento de curvas EWMA (*Exponentially Weighted Moving Average*) (WOO et al., 2003).

7.4.2.1 Ambiente de teste - prototipação

Para avaliação de desempenho dos protocolos *multi-hop* utilizou-se a aplicação Surge (Crossbow, 2004), uma aplicação que acompanha o TinyOS. Trata-se de uma aplicação para exemplificar o uso do protocolo *multi-hop* LEPSM e é dividida em dois módulos: O módulo desenvolvido para executar no nó sensor e o módulo que executa na estação base. A aplicação Surge executa no nó sensor, coleta informações de luminosidade dos sensores e envia pela rede para a estação base através do nó sorvedouro. O módulo que executa no PC é uma aplicação Java que pode ser usada para visualizar a topologia de rede lógica e as leituras de sensoriamento. Ambos os módulos da aplicação Surge foram usados na avaliação dos protocolos de roteamento *multi-hop*, no entanto algumas alterações foram necessárias para possibilitar o uso do protocolo MintRoute que se desejava avaliar. Por padrão, a aplicação Surge utiliza o protocolo

LEPSM, por isso para utilizar o protocolo MintRoute alterou-se as bibliotecas da aplicação Surge para configurá-lo de acordo com esse protocolo.

Utilizou-se a técnica de medição com as seguintes métricas: vazão, consumo de energia e taxa de pacotes fornecidos (*Packet Delivery Fraction* – PDF). A aplicação Surge forneceu a vazão e taxa de pacotes fornecidos. A vazão é obtida pelo número de mensagens que chegam a um determinado nó por unidade de tempo. Nesse experimento, como a aplicação Surge processa e envia mensagens, a vazão medida foi em mensagens enviadas por segundo em cada nó sensor. A taxa de pacotes fornecidos é obtida pela relação entre pacotes recebidos e enviados em cada nó sensor.

O consumo de energia foi obtido através da medida da corrente e a tensão de um nó sensor, utilizando um osciloscópio digital (GDS-800 Series) em um laboratório de medidas eletrônicas. Com a corrente e a tensão medida, a energia consumida por um dispositivo eletrônico é calculada pela equação:

$$E = V \; x \; I \; x \; \Delta T \qquad (1)$$

Onde E representa a energia em Joules; V, a tensão em Volts; I, a corrente em *Amperes*; e T, o tempo em Segundos. Para efeito das comparações realizadas neste trabalho, foi adotada a tensão de 3V (como tensão ideal fornecida pelas duas baterias AA de 1.5V), uma vez que o valor da tensão se mantém constante para todas as medições. Dessa forma, os cálculos de consumo de transmissão do rádio foram realizados utilizando o valor de 3V, tendo em vista que no nó sensor MICAz o VCC (3V) alimenta o microcontrolador Atmega128L e o rádio CC2420.

Para os cálculos relativos ao consumo de energia foram utilizados os valores expostos na figura 7.14.

Componente	Corrente (mA)	Tensão (V)
Rádio	27mA	3V

Figura 7.14: Consumo de energia no MICAz

Uma vez que os valores da tensão e da corrente na equação (1) são constantes (considerando a utilização dos valores da figura 7.14), conclui-se

que o acréscimo no consumo de energia está relacionado ao tempo de transmissão do pacote. O procedimento utilizado para obter o tempo de transmissão foi modificar o nível lógico de um dos pinos digitais de propósito geral no microcontrolador ATmega128L para sinalizar o início e o fim da marcação do tempo. No início da marcação, o pino é configurado para nível baixo (0 Volts) e no final, para nível alto (3 Volts). Para obter o intervalo de tempo (ΔT), utilizou-se o osciloscópio digital responsável pelo monitoramento da mudança no nível lógico do pino. A figura 7.15 mostra os equipamentos utilizados para as medições.

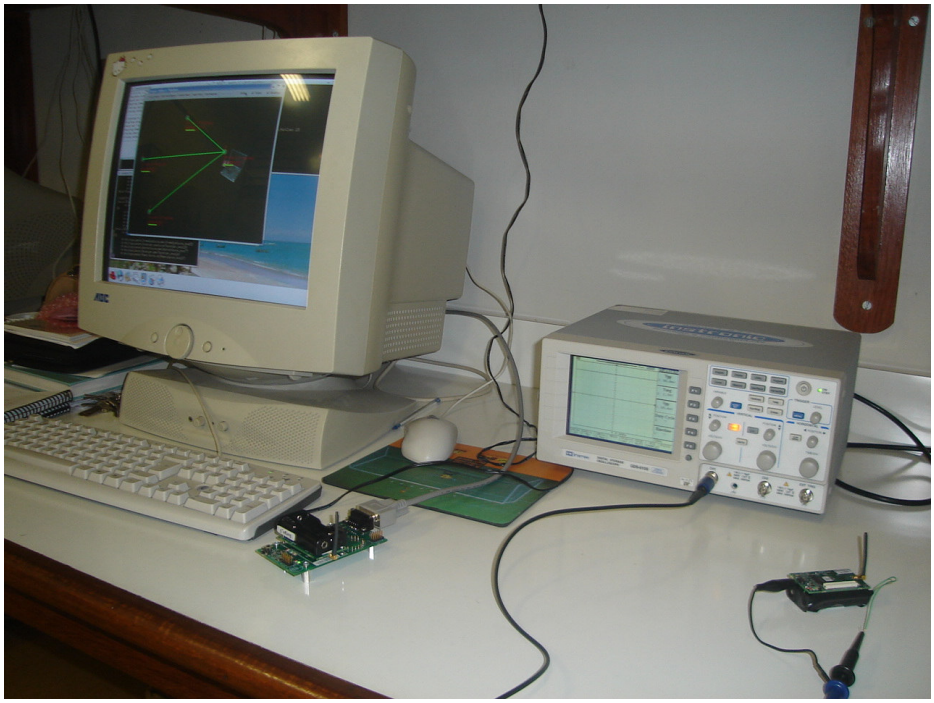


Figura 7.15: Conexão com o osciloscópio.

O consumo de energia drenada pelo rádio foi obtido através de 30 medições para cada nó sensor. O desvio padrão observado foi de aproximadamente 20%.

Para realizar as medições referentes ao rádio, foi utilizada uma rede composta por 8 nós sensores, incluindo o nó sorvedouro (ligado à estação base), transmitindo a uma freqüência de 2,048GHz e a uma potência de transmissão de 0 dBm. Os testes foram realizados em uma área de 10m x 50m.

Adotou-se uma configuração composta por três níveis, conforme mostra a figura 7.16. No nível 1, os nós sensores estão ligados diretamente ao nó sorvedouro. No nível 2, os nós sensores comunicam-se com um nó intermediário, até chegar ao nó sorvedouro. Já no nível 3, os nós sensores comunicam-se com dois nós intermediários, até chegar ao seu destino. O objetivo dessa configuração é forçar o roteamento dos pacotes pela rede. Outras configurações, variando a área e o número de níveis, foram testadas sem mudança significativa nos resultados devido ao pequeno número de nós sensores no experimento.

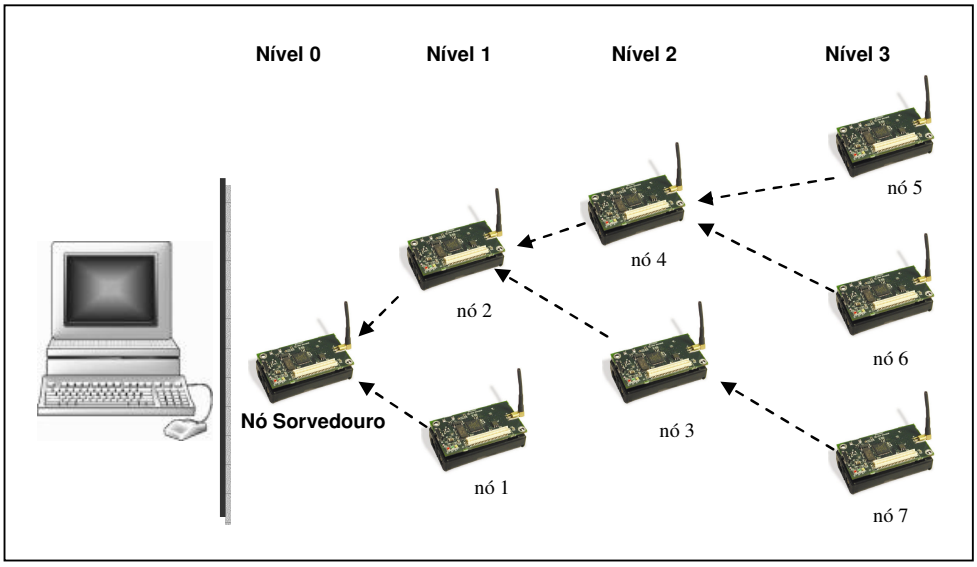


Figura 7.16: Configuração da RSSF.

7.4.2.2Resultados

Nesta seção serão apresentados os resultados obtidos referentes à avaliação de desempenho dos protocolos de roteamento LEPSM e MintRoute em uma RSSF real, considerando as métricas de vazão, porcentagem de pacotes entregues e energia consumida. Em todos os experimentos foram roteados aproximadamente 6.000 pacotes de tamanho de 8 bytes. A duração média dos experimentos foi de 2 horas para cada protocolo e cada nó sensor foi configurado para transmitir a uma taxa de 0,5 pacotes/seg.

Vazão média

A figura 7.17 apresenta a vazão média obtida para os protocolos LEPSM e MintRoute. Como pode ser observado, o desempenho do protocolo LEPSM foi superior na maioria dos nós sensores da rede, considerando a topologia apresentada na seção 7.4.2.1.

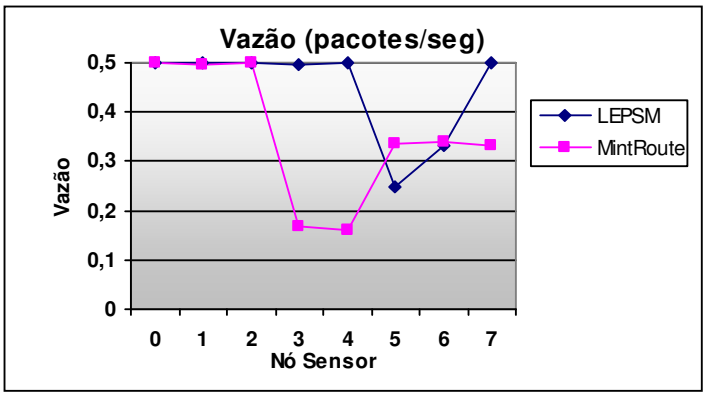


Figura 7.17: Vazão média para os protocolos.

Fazendo uma análise por nível, verificou-se que no nível 0 (onde está o nó sorvedouro) todos os protocolos mantiveram o desempenho esperado (vazão de 0,5 pacotes/seg), pois o nó sorvedouro está ligado diretamente ao PC, não havendo a possibilidade de perda de pacotes.

No nível 1, pelo fato dos nós estarem ligados ao nó sorvedouro (nó destino), não houve a necessidade de nenhuma complexidade no algoritmo para descoberta do próximo salto, dessa forma, ambos os protocolos apresentaram desempenho equivalente.

Nos níveis 2 e 3, em que o roteamento dos pacotes exige uma maior eficiência dos algoritmos de roteamento, o protocolo LEPSM apresentou um desempenho superior em relação ao protocolo MintRoute, demonstrando que o critério de descoberta do próximo salto através do menor caminho é mais eficiente que a estimação por qualidade do enlace. Isso porque em nós sensores afastados da estação base, a qualidade do enlace como critério pode provocar um laço no roteamento dos pacotes.

Percentagem de pacotes entregue

A figura 7.18 apresenta a percentagem de pacotes entregues para cada nó sensor, obtida para os protocolos LEPSM e MintRoute.

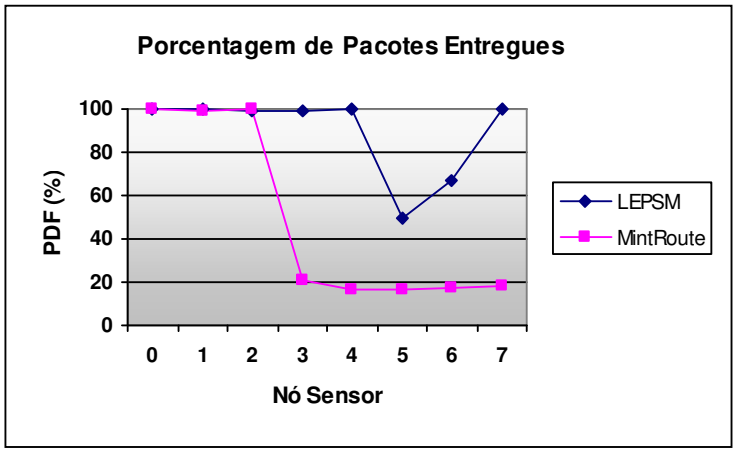


Figura 7.18: PDF para os protocolos.

De acordo com a figura, observou-se novamente que o desempenho dos protocolos analisados não divergem quando os nós sensores estão ligados diretamente ao nó sorvedouro, apresentando uma porcentagem de pacotes entregues próxima de 100%. Já nos níveis 2 e 3, foi observado desempenho superior do protocolo LEPSM em relação ao protocolo MintRoute. Isso se deve à grande perda de pacotes gerada em laços de comunicação ocorridos entre os nós mais distantes do nó sorvedouro (níveis 2 e 3).

Consumo de energia

Os dados apresentados na figura 7.19 representam o consumo médio de energia para cada nó sensor. É importante ressaltar que o nó sorvedouro (nó sensor 0) foi omitido dos cálculos de consumo de energia, devido ao mesmo ter sido alimentado diretamente através da rede de energia elétrica.

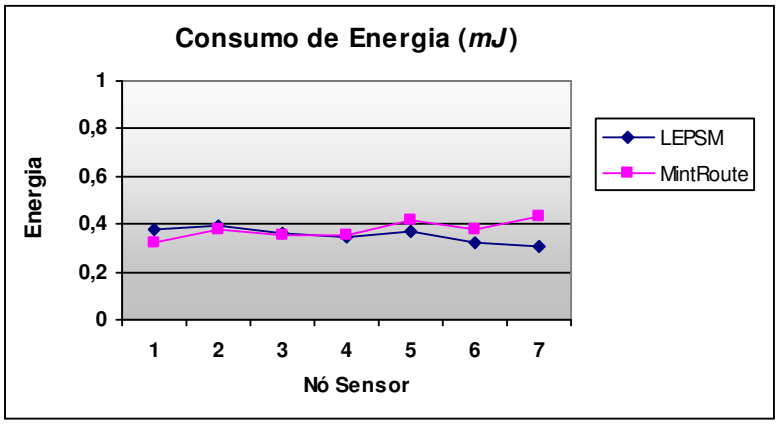


Figura 7.19: Consumo de energia para os protocolos.

De acordo com a figura, os dois protocolos apresentaram desempenho similar apenas para os nós 2, 3 e 4, divergindo para o nó 1 e para os nós do nível 3. Fazendo-se uma análise quantitativa, observou-se que o protocolo MintRoute proveu um consumo médio total 8% maior em relação ao provido pelo protocolo LEPSM. Esse desempenho inferior apresentado pelo protocolo MintRoute deve-se ao já mencionado laço de comunicação ocorrido nos níveis 2 e 3, gerados por um critério de descoberta de salto por estimação de qualidade do enlace.

7.4.3 Serviço de Contexto

O principal objetivo de avaliação de desempenho desse serviço é validar a tese que o SensorBus requer somente um pequeno acréscimo em termos de tempo decorrido para responder a uma requisição de serviço, comparada com uma abordagem que não usa políticas de customização em que os serviços fornecidos não são configurados para atender às preferências do usuário e às condições de contexto.

Para validar esse serviço, implementou-se um *benchmark* sintético que fornece como medida de desempenho o tempo decorrido. O tempo decorrido (em milissegundos) é medido entre o instante que uma requisição de serviço é emitida e o instante que uma política é selecionada e inicializada para atender à requisição.

Os resultados da avaliação de desempenho não consideram o tempo necessário para inicializar os sensores e o processamento que é executado

para obter a informação. Assume-se que as informações de contexto já estão disponíveis em uma base de dados. Assim que o serviço de contexto consulta os nós sensores sobre seus contextos correntes, o valor do recurso é imediatamente retornado. Nenhum custo adicional é necessário para processar e obter as informações dos nós sensores. Essa abordagem é plausível devido à grande variedade dos nós sensores e, portanto com a possibilidade de introduzir custos adicionais de diferentes ordens de magnitude que não podem ser comparados.

Nesse experimento, variou-se o número total de nós sensores de 10 até 100 em passos de 10. Gravou-se o tempo decorrido para 20 medições e o resultado final se refere à média dos valores medidos em cada uma das 20 replicações. O desvio padrão observado foi de aproximadamente 15%.

A figura 7.20 ilustra o efeito que a utilização de metadados tem no tempo decorrido de uma customização de política de contexto sobre o *middleware* básico sem utilização de políticas. A interseção da curva com o eixo Y representa o uso do SensorBus de maneira estática (sem políticas), pois o tempo decorrido é o mesmo para qualquer quantidade de nós sensores.

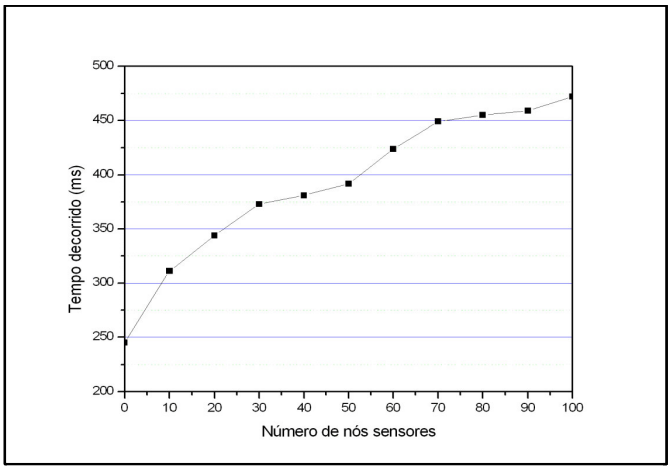


Figura 7.20: Efeito de customização de política de contexto.

Como a figura 7.20 mostra, o acréscimo no tempo decorrido é aproximadamente linear em relação ao número de nós sensores colocados na RSSF. Mesmo em uma situação com considerável número de nós, acerca

de 100, o desempenho do SensorBus não é afetado, ficando o tempo decorrido abaixo de 0,5 segundo.

8 TRABALHOS RELACIONADOS

A principal funcionalidade de um *middleware* para RSSFs é facilitar o desenvolvimento, manutenção, implantação e execução de aplicações de sensoriamento. Para essa facilidade ocorrer, é necessário incluir mecanismos para fornecer tarefas de sensoriamento complexas, comunicar essas tarefas para RSSFs, uma coordenação dos nós sensores para dividir uma tarefa e distribuí-la para nós sensores individuais, uma fusão de dados para concatenar as leituras dos sensores em um resultado de mais alto nível, e reportar o resultado de volta para o manipulador da tarefa. Além disso, mecanismos e abstrações para tratar com as heterogeneidades dos nós sensores devem ser fornecidos (YU *et al*., 2004).

Neste capítulo descrevem-se as principais plataformas para a programação de aplicações em RSSFs que seguem o paradigma de *middleware*. Os diversos tipos de *middleware* existentes serão introduzidos com base nos princípios de projeto descritos acima. Segundo esses princípios, o paradigma de *middleware* para RSSFs pode ser classificado em quatro categorias. Primeiro, o *middleware* segue uma abordagem orientada a dados, nessa categoria uma RSSF é representada como um sistema de base de dados de dispositivos. Em uma segunda categoria, o *middleware* segue uma abordagem baseada em eventos que manuseia com dados em tempo real em RSSFs. Em uma terceira categoria, o *middleware* segue uma abordagem orientada à qualidade de serviço (QoS – *Quality of Service*). Finalmente, na quarta categoria o *middleware* segue uma abordagem baseada em agentes.

Em RÖMER *et al*. (2002) apresenta-se uma descrição geral dos desafios que se enfrenta num projeto de *middleware* para RSSFs,

centrando-se nos aspectos de restrição de recursos e tempo de vida desses sistemas.

8.1 Abordagem orientada a dados

Na abordagem orientada a dados, os nós sensores executam a recuperação e a agregação dos dados de interesses através de operações em demanda para fornecê-los para aplicações externas. Essa abordagem deriva do modelo de banco de dados distribuído e segue o paradigma de comunicação assíncrono. Normalmente existe uma linguagem de consulta do tipo SQL. As técnicas de execução de consultas distribuídas são aplicadas para aumentar a capacidade de computação do dispositivo e reduzir sua capacidade de comunicação.

8.1.1 SQTL

Jaikaeo *et al*. (2000) desenvolveram uma linguagem de consulta (*query*), monitoramento e *tasking* (operação de incumbir um nó sensor apropriado para responder a uma consulta de usuário) para RSSFs, chamada de SQTL (*Sensor Querying and Tasking Language*). Basicamente é uma linguagem de *tasking* para RSSFs, mas segue o paradigma das linguagens de consultas (*queries*) de banco de dados. É uma linguagem de *scripting* procedural capaz de interpretar declarações de consultas simples.

Além das primitivas de comunicação, ciência de localização e acesso ao *hardware* do nó sensor, ela também fornece uma construção para tratamento de eventos. A SQTL suporta três tipos de eventos:

- Eventos gerados quando uma mensagem é recebida por um nó sensor;
- Eventos disparados periodicamente por um temporizador (*timer*);
- Eventos causados pela expiração de um temporizador.

Uma mensagem SQTL, contendo um *script*, pode ser interpretada e executada por qualquer nó na rede. Para endereçar o *script* para um específico receptor, ou um grupo de receptores, a mensagem deve ser encapsulada em um *wrapper* SQTL que atua como um cabeçalho de

mensagem indicando o transmissor, os receptores, uma aplicação particular executando nos receptores e também os parâmetros para a aplicação.

A rede é modelada como um conjunto de nós sensores colaborativos que transportam consultas e *tasking* programadas em SQTL. Um nó fonte recebe uma mensagem que encapsula um programa SQTL e se encarrega de fazer a difusão para os outros nós da rede. Cada nó sensor pode difundir o programa SQTL encapsulado para os outros nós sensores realizando as atividades de consultas e *tasking* colaborativamente.

8.1.2 SINA

Srisathapornphat *et al.* (2000) apresentaram SINA (*Sensor Information Networking Architecture*) que age como um *middleware* para facilitar consultas, monitoramento e *tasking* em RSSFs. A organização e o fornecimento de informação são realizados nas camadas inferiores enquanto que as atividades de consultas e *tasking* são feitas nas camadas superiores. A figura 8.1 (SRISATHAPORNPHAT *et al.,* 2000) mostra essa divisão.

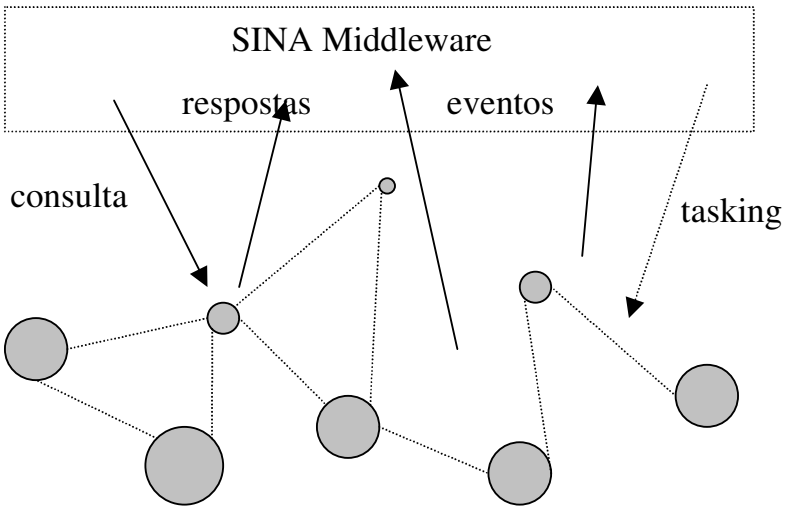


Figura 8.1: Arquitetura SINA

Os nós sensores são agrupados pelo nível de potência e proximidade de forma autônoma formando *clusters*. O processo de *clustering* é aplicado recursivamente formando uma hierarquia de *clusters*. Com essa arquitetura espera-se diminuir o nível de energia requerido para a troca de informação.

A figura 8.2 (SRISATHAPORNPHAT *et al.,* 2000) mostra a sua estrutura hierárquica.

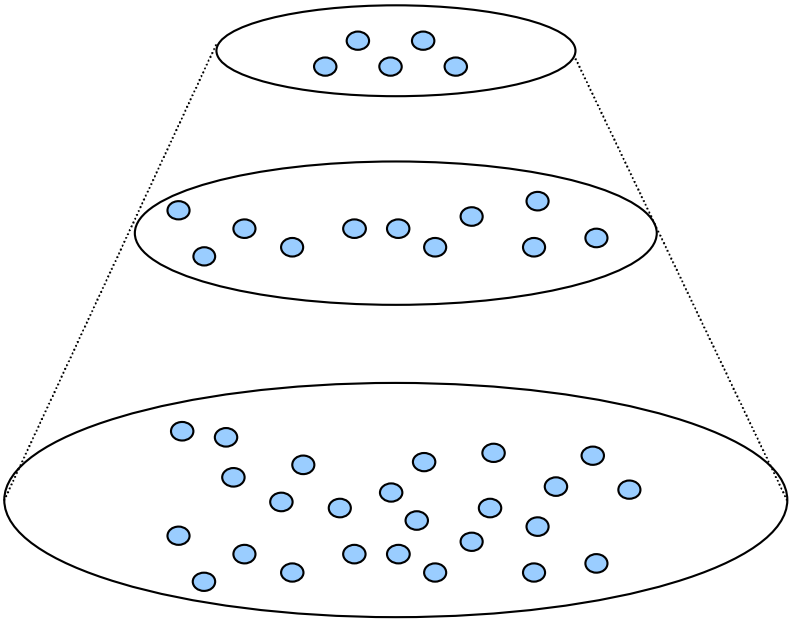


Figura 8.2: Clustering Hierárquico

Além disso, nessa arquitetura, os nós da rede são acessados através de endereçamento baseado em atributos em vez de endereçamento explícito. Assim em vez de acessar a informação de um nó em particular, a rede é acessada através de uma lista de nós que possuem a informação especificada. Usando esse tipo de consulta aumenta-se a eficiência da rede como um todo.

A SINA utiliza a linguagem SQTL para as tarefas de detecção programadas. Para aplicações que coletam informações do nó sensor, o usuário pode escolher entre invocar uma *query* pré-construída em vez de escrever explicitamente um *script* SQTL procedural.

Um ambiente de execução de sensor (SEE – *Sensor Execution Environment*), executando em cada nó sensor, é responsável por despachar mensagens entrantes, examinar as mensagens SQTL que chegam e realizar a apropriada operação para cada tipo de ação especificada nas mensagens.

Devido ao grande número de nós sensores, colisões resultantes de um grande número de respostas propagadas de volta para o nó sorvedouro (*sink node)* durante um pequeno período de tempo dão origem a um

problema chamado de implosão de resposta (SRISATHAPORNPHAT *et al.*, 2000). Para facilitar a coleta e disseminação de informação, o *kernel* de SINA, representado por um conjunto de SEEs, implementa mecanismos de comunicação cujo objetivo é minimizar o problema de implosão de resposta a fim de que diminua o consumo de recursos da rede.

8.1.3 PROJETO COUGAR

Em muitos *frameworks* de RSSFs existentes, os dados dos nós sensores são transmitidos e armazenados dentro de um servidor de *gateway* de acordo com um procedimento pré-programado. Em tal abordagem, um esquema de coleção de dados diferente requer uma mudança no programa. Além disso, o custo adicional devido à carga no servidor de *gateway*, e a transmissão de dados redundantes, não podem ser negligenciados.

Cougar (YAO *et al*., 2002) é uma arquitetura que trata uma RSSF como um banco de dados distribuído onde um número grande de nós sensores está conectado através de uma rede sem fios *multi-hop* e cada nó mantém dados de sensoriamento. Um otimizador de consulta fica situado no nó *gateway* para gerar planos de processamento de consultas distribuídas após receber consultas externas. O plano de consulta é criado de acordo com a informação de catálogo e a especificação da consulta. Tal plano de consulta especifica o fluxo de dados (entre nós sensores) e um plano de computação exata (para cada nó sensor). O plano é então disseminado para os nós sensores relevantes. Estruturas de controle são criadas para sincronizar o comportamento do nó sensor, e então a consulta é iniciada. Em tempo de execução, o registro de dados flui de volta ao nó *gateway* quando o processamento interno à rede acontece.

8.1.4 TinyDB

TinyDB (MADDEN *et al*., 2005) é um sistema de processamento de consultas para RSSFs que opera em TinyOS. Em TinyDB, o conceito de processamento de consulta aquisicional (*acquisitional query processing* - ACQP) é introduzido. Quando um processamento de consulta ocorre, o nó sensor faz sensoriamento para responder à consulta. Consultas de alto nível

são decompostas e distribuídas para redes e é necessário escrever um programa em linguagem C, correspondendo ao conteúdo da consulta em TinyOS e seu processamento. No ACQP do TinyDB, a SQL é aumentada para processamento de consulta, a qual é convertida em código interno, e executado para recuperação de dados e agregação. Por exemplo, considerando a seguinte consulta:

```
SELECT nodeid, light, temp
    FROM sensors
    SAMPLE PERIOD 1s FOR 10s
```

Essa consulta especifica que cada dispositivo deve reportar sua leitura de identificação, nível de luminosidade e temperatura (contida em tabela virtual de sensores) uma vez por segundo em 10 segundos. Resultados dessa consulta fluem para o nó sorvedouro da rede de maneira *on-line*, via topologia *multi-hop*. A saída consiste de um fluxo de tuplas, agrupadas em intervalos de tempo de 1 segundo. Cada tupla inclui uma marca de tempo correspondendo ao tempo em que ela foi produzida. Quando a consulta é convertida para código interno, ela otimiza o consumo de energia e a comunicação no TinyDB. O TinyDB é similar a Cougar, mas o TinyDB considera mais otimizações para ambientes de recursos escassos como uma RSSF.

8.1.5 TinyLIME

TinyLIME (CURINO *et al*., 2005) é uma extensão do *middleware* de compartilhamento de dados descentralizados LIME, o qual é uma extensão da linguagem de programação Linda, fornecendo memória compartilhada. O TinyLIME é uma alteração de um *middleware* de ambiente móvel para uma plataforma de RSSF.

O modelo TinyLIME foi projetado e implementado para a plataforma de *Motes Crossbow*, explorando as funcionalidades de TinyOS. Na parte que executa em computadores padrões (lado PC), o TinyLIME é implementado como uma camada no topo do *middleware* LIME sem nenhuma modificação. No TinyLIME, uma partição de espaço de tuplas reside em cada nó sensor, e um espaço de tuplas coordenado é formado quando há conexão entre o

host e os nós sensores. Os dados dos nós sensores podem ser lidos através de uma operação de leitura de Linda para esse espaço de tuplas coordenado. Assim, o TinyLIME abstrai a coleta de dados de uma RSSF como uma operação no espaço de memória compartilhado (espaço de tuplas). Isso funciona como um *middleware* oferecendo uma interface abstrata para os programadores de aplicação numa RSSF.

8.1.6 DFuse

DFuse (KUMAR *et al*., 2003) é um *framework* para desenvolvimento de aplicação que utiliza fusão de dados em redes de sensores distribuídas e descentralizadas. A arquitetura do DFuse enfatiza as seguintes características: API de fusão, e um algoritmo distribuído para implantação de função de fusão e relocação dinâmica.

A API de fusão oferece facilidade de programação para aplicações complexas que utilizam fusões de dados. A API permite operações de síntese em fluxos de dados que são especificadas como uma função de fusão, variando desde simples agregações (tais como min, max, sum, ou concatenação) até tarefas de percepção complexa (tais como analisar uma seqüência de imagens de vídeo).

A implantação de funções de fusão é feita através de um algoritmo baseado em heurística para encontrar um bom mapeamento (de acordo com alguma função de custo pré-definida) entre as funções de fusão e os nós sensores da rede. O mapeamento é reavaliado periodicamente para considerar tanto as trocas dinâmicas nos níveis de energia dos nós sensores quanto o comportamento da rede.

8.2 Abordagem baseada em eventos

Uma alternativa é usar uma abordagem de disparo de eventos onde a execução é dirigida por eventos. A comunicação dirigida por eventos é um paradigma assíncrono que desacopla emissores e receptores.

Nesta seção, descrevem-se tipos de *middleware* que suportam processamento de eventos em RSSFs e outros tipos de *middleware* que usam o paradigma *publish-subscribe.*

8.2.1 DSWare

DSWare (*Data Service Middleware*) (LI *et al*., 2003) é um *middleware* que tem uma abordagem centrada em dados definindo um serviço de dados comum e partes de serviço baseado em grupo de várias aplicações. Ele fica entre a camada de aplicação e a camada de rede provendo uma abstração de serviço de dados para aplicações.

O serviço de evento em tempo real trata da inconfiabilidade das informações fornecidas pelos nós sensores individuais, correlação entre observações de sensoriamento diferentes, e características de tempo real inerentes aos eventos. O serviço de evento suporta funções de confidência, que são baseadas na semântica dos dados, incluindo a importância relativa de subeventos e padrões históricos. Quando a taxa de falha é alta, o serviço de evento permite detecção parcial de eventos críticos para serem reportados de maneira pontual. O DSWAre realiza roteamento em tempo real levando em consideração o consumo de energia.

O DSWare fornece abstração de serviço de dados similar ao AutoSec (ver seção 8.3.3), mas em vez do serviço ser fornecido por um simples nó sensor, é fornecido por um grupo de nós sensores próximos. Assim, o DSWare gerencia, de maneira transparente, falhas nos nós sensores caso grande quantidade de nós sensores permaneçam na área para fornecer uma medida válida.

8.2.2 ENVIROTRACK

EnviroTrack é um *middleware* para sistemas distribuídos baseado em objeto que eleva o nível de abstração de programação para aplicações de rastreamento ambiental (ABDELZAER *et al*., 2003). Ele contém mecanismos que abstrai eficientemente grupos de nós sensores para objetos lógicos que mantêm estados ambientais agregados. Tais objetos são logicamente ligados em entidades em movimento no ambiente físico, de maneira que se possa monitorar o estado da entidade rastreada.

O EnviroTrack é o primeiro suporte de programação para RSSFs que suporta rastreamento de objetos móveis. Suas abstrações e mecanismos de suporte são bem adequados para monitorar alvos que se movem no mundo

físico. A administração de grupo dinâmico é realizada e um líder é responsável pelas trocas de mensagens periódicas entre os membros do grupo.

O EnviroTrack é uma camada de *middleware* que exporta um novo espaço de endereçamento numa RSSF. Nesse espaço, eventos físicos no ambiente externo são entidades endereçáveis. Esse tipo de endereçamento é conveniente para aplicações que necessitam monitorar eventos ambientais. Por exemplo, uma aplicação de vigilância que monitora os movimentos de veículos atrás da linha inimiga assinala rótulos únicos para veículos individuais. Seus estados podem então ser endereçados pelas referências a esses rótulos. Além disso, objetos de atuação ou computação podem ser agregados para endereços individuais.

8.2.3 CORTEX

CORTEX (BIEGEL & CAHILL, 2004) fornece um modelo de programação que suporta o desenvolvimento de aplicações através de objetos sensíveis móveis que consideram a provisão de garantias de confiabilidade e tempo real incremental. Ele possui uma arquitetura de sistema escalável e aberta que reflete a estrutura heterogênea e o desempenho das redes. O CORTEX é um *middleware* baseado no paradigma *publish-subscribe* para redes *ad hoc* que usa um modelo de objetos sensíveis para disseminar contexto e outros dados.

Um objeto sensível é uma entidade encapsulada com interfaces consistindo de sensores e atuadores. Os sensores são definidos como entidades que produzem eventos em reação aos estímulos do mundo real. Os atuadores são definidos como entidades que consomem eventos e reagem para alterar o estado do mundo real. Um objeto sensível contém um componente de captura sensível, que realiza fusão de dados baseado em redes bayesianas. Esse tipo de objeto fornece uma abordagem eficiente para raciocínio inteligente em uma hierarquia de contextos.

8.2.4 Mires

Mires (SOUTO *et al*., 2004) é um *middleware* baseado no paradigma p*ublish-subscribe* para sistema de mensagem em RSSF. A operação assume uma RSSF, de estrutura hierárquica, comunicando-se com nós de uma rede tradicional (rede com fio) em que aplicações residem, via um nó sorvedouro que funciona como um *gateway* entre uma RSSF e as aplicações. Inicialmente, os nós numa RSSF criam anúncios para seus tópicos disponíveis (por exemplo, umidade e pressão) coletados dos sensores locais. A seguir, as mensagens de anúncios são encaminhadas para o nó sorvedouro usando um algoritmo de roteamento *multi-hop*. Uma aplicação de usuário (via uma interface de usuário gráfica) conectada ao nó sorvedouro é capaz de selecionar os tópicos anunciados de interesse que estão sendo monitorados. As mensagens correspondentes para essas subscrições são então enviadas para os nós sensores na RSSF. Após receber as subscrições, os nós sensores publicam seus dados coletados via nó sorvedouro para as aplicações de usuário.

8.2.5 Scope

Scope (STEFFAN *et al*., 2004) é uma abstração genérica para definição de grupo de nós. RSSFs de múltiplos propósitos são heterogêneas na maioria dos casos: para cada aplicação somente um tipo específico de nó sensor ou algumas partes do ambiente monitorado são relevantes. O Scope propõe um *middleware* baseado em mensagens que segue o paradigma *publish-subscribe* dentro de um grupo de nós. O grupo de nós pode ser formado considerando a proximidade geográfica, tipo de nó sensor e assim por diante. As redes de múltiplos propósitos podem ser estabelecidas de duas maneiras. Por um lado, há implementações de baixo nível que focalizam na programação de nós sensores individuais e nas facilidades de comunicação e sensoriamento dos nós sensores, tais como o TinyOS. Por outro lado, há trabalhos em processadores de consultas declarativas oferecendo interfaces de alto nível que não permitem controle explícito em nós individuais. A dificuldade baseia-se em encontrar compensações entre a universalidade das interfaces de alto nível e o grau em que detalhes específicos da aplicação podem ser passados e utilizados para otimização de

roteamento e escalonamento de recursos. O Scope objetiva preencher a lacuna entre as interfaces de baixo e alto nível e permitir o particionamento das funcionalidades de RSSFs. O Scope como um bloco de construção de *middleware* facilita a construção de serviços customizados em RSSFs de múltiplos propósitos.

8.2.6 Impala

Impala foi construído com parte do projeto ZebraNet (LIU et al., 2004), em que os nós sensores são espalhados em um ambiente natural de vida selvagem para realizar estudos sobre migração de longo prazo em uma coleção de animais de um ecosistema. Ele consiste numa arquitetura de *middleware* que permite modularidade, adaptabilidade e reparabilidade de aplicações em RSSFs. O Impala explora técnicas de código móvel para alterar a funcionalidade do *middleware* que executa em um nó sensor. A chave da eficiência de energia no Impala é que as aplicações sejam tão modulares quanto possíveis, possibilitando pequenas alterações que requerem pouca energia de transmissão.

A adaptação é implementada através do modelo de programação baseado em evento do *middleware*, e ocorre em resposta a uma determinada faixa de eventos. Alguns eventos, como os de temporização, assinalam o tempo decorrido desde a última verificação de estado; o módulo responsável pela adaptação então escolhe consultar a aplicação ou o estado do sistema para determinar se alguma adaptação deve ser realizada. Outros eventos, como os de dispositivos, possuem fontes externas ao *middleware* Impala e assinalam um evento externo para o Impala responder, tal como a falha de um receptor de rádio específico; o módulo de adaptação deve então examinar o impacto da falha e determinar se despacha uma outra aplicação para contornar o problema.

8.3 Abordagem orientada a QoS

Nesta seção descrevem-se tipos de *middleware* que objetivam fornecer qualidade e confiabilidade aos dados coletados, dependendo dos requisitos da aplicação.

8.3.1 Milan

MiLAN (MURPHY & HEINZELMAN, 2002) foi desenvolvido para permitir configuração de rede dinâmica para atender os requisitos de desempenho das aplicações. As aplicações representam seus requisitos através de gráficos especializados que incorporam mudanças baseadas em estado em necessidades da aplicação. Baseado nessa informação, o MiLAN decide como controlar a rede e também os nós sensores para balancear QoS (*Quality of Service*) da aplicação e eficiência de energia.

Diferentemente dos tipos de *middleware* tradicionais que se situam entre a aplicação e o sistema operacional, o MiLAN possui uma arquitetura que engloba a pilha de protocolo de rede, possibilitando o uso de várias redes físicas. Como o MiLAN situa-se no topo de múltiplas redes físicas, uma camada de abstração é fornecida que permite *plug-ins* específicos de rede converter comandos MiLAN em comandos específicos de protocolos que são passados através da pilha de protocolo habitual. Portanto, o MiLAN pode continuamente se adaptar as características específicas de qualquer rede que está sendo usada para comunicação, de maneira a satisfazer as necessidades das aplicações com o passar do tempo.

8.3.2 QUASAR

Ao contrário dos sistemas de banco de dados distribuídos convencionais, uma arquitetura de dados de sensor deve controlar altas taxas de geração de dados de um grande número de pequenos componentes autônomos. E ao contrário do paradigma emergente de fluxo de dados, é impossível pensar que essa grande quantidade de dados possa ser recebida no local de processamento de consulta, devido à largura da banda do servidor, e a restrições de energia de nós sensores operados por bateria.

QUASAR (*Quality-Aware Sensing Architecture*) (LAZARIDIS *et al*., 2004) propõe uma arquitetura de dados de sensor que deve tornar-se ciente da qualidade, regulando a qualidade de dados em todos os níveis do sistema distribuído e suportando requisitos de qualidade das aplicações de usuário de uma maneira mais eficiente possível. Por exemplo, a QUASAR

objetiva processar consultas ciente da qualidade (Q_aQ_s) tais como "Recupere os Ids e temperaturas (dentro ± 5° C) dos nós sensores que apresentam temperatura acima de 30° em uma certa área R."

8.3.3 AutoSec

AutoSeC (*Automatic Service Composition*) (HAN *et al*., 2001), é um *middleware* de serviço dinâmico para utilização efetiva de recursos dentro de um ambiente distribuído. As aplicações distribuídas têm exigências de QoS que podem ser traduzidas nos recursos de nível de sistemas subjacentes. Nesse caso, o AutoSeC faz administração de recursos dentro de uma RSSF concedendo controle de acesso a aplicações em ordem satisfazer aos requisitos de QoS. A satisfação dessas exigências exige que o AutoSeC escolha dinamicamente uma combinação de informação e recursos que abasteçam políticas de um determinado conjunto baseado nas necessidades do usuário e estado de sistema. Desde que a escolha de políticas é feita pelo AutoSeC, o programador de aplicação ou o administrador de sistema é liberado da tarefa tediosa de escolher uma opção do conjunto de políticas disponíveis.

O AutoSec gerencia recursos em RSSFs fornecendo controle de acesso para aplicações de modo que a qualidade de serviço requisitada seja garantida. Essa abordagem é similar aos tipos de *middleware* para redes convencionais devido às restrições de recursos encontradas nos nós sensores, mas as técnicas para coletar as informações de utilização de recursos correntes são adequadas para RSSFs.

8.4 Abordagem baseada em agente

Nesta seção descrevem-se tipos de *middleware* que utilizam a tecnologia de agentes para desenvolvimento de aplicações em RSSFs.

8.4.1 SENSORWARE

Boulis & Srivastava (2000) desenvolveram SensorWare, um *framework* para redes ativas. O SensorWare é baseado na linguagem de *script* TCL (*Tool Command Language*) que foi estendida para suportar

código móvel. O SensorWare permite o desenvolvimento de aplicações na forma de *scripts* leves que se movem numa RSSF. O SensorWare utiliza-se de SensorSim (PARK *et al*., 2000) para o desenvolvimento de aplicações na forma de *scripts*. O principal objetivo da ferramenta é permitir o desenvolvimento de *scripts* que possam executar tanto em nós sensores reais como em nós sensores simulados.

O SensorSim é um simulador para RSSFs construído sobre o NS-2 (*Network Simulator* 2) (BRESLAU *et al*., 2000), um popular simulador de eventos discretos. A ferramenta é adequada para avaliação de desempenho de RSSFs. Além das capacidades oferecidas pelas ferramentas de simulação tradicionais, o SensorSim incorpora características para modelar uso de energia e estudo de novos protocolos de comunicação para nós sensores. A fim de conseguir esses objetivos, um modelo de nó sensor é derivado.

8.4.2 RUNES

O projeto RUNES (*Reconfigurable Ubiquitous Networked Embedded Systems*) (RUNES, 2005) tem como objetivo a criação de sistemas embarcados de redes com componentes heterogêneos e distribuídos, os quais interagem e se adaptam aos seus ambientes. O objetivo geral é fornecer uma plataforma de *middleware* adaptativo e ferramentas de desenvolvimento de aplicação que permitam aos programadores a flexibilidade de interagir com o ambiente onde for necessário, e ainda dispor de um nível de abstração que facilite a construção de aplicações. A abordagem RUNES implica em construir um *middleware* em termos de um modelo de componente bem definido e independente de linguagem, que é suportado por uma API mínima em tempo de execução. A heterogeneidade do modelo de componente para vários tipos de dispositivos é obtida provendo implementações diferentes da API em tempo de execução. Por exemplo, em um PDA executando um sistema operacional padrão, componentes podem ser implementados por classes Java ou como objetos Linux compartilhados, e em um microcontrolador de nó sensor, componentes podem ser implementados simplesmente como segmentos de código de máquina.

8.4.3 SMART MESSAGES

O projeto S*mart Messages* (SMART MESSAGES, 2003) propõe um modelo de computação distribuída baseado na migração de unidades de execução. As *Smart messages* são unidades migratórias que contêm dados e código. O objetivo do projeto *Smart Messages* é desenvolver um modelo computacional e uma arquitetura de sistema para sistemas de redes embarcadas (NES – *Networks Embedded Systems*). As aplicações para esses sistemas abrangem desde disseminação e coleta de dados até aplicações cooperativas complexas tais como colaboração de veículos para adaptar as condições de tráfego ou robôs com câmeras inteligentes realizando rastreamento de objetos distribuídos. O modelo *Smart Messages* é utilizado para programar aplicações distribuídas e definidas pelo usuário em redes de sistemas embarcados.

8.4.4 RSN

A abordagem de código móvel do projeto RSN (*Reactive Sensor Network*) (RSN, 2002) tem como objetivo principal fornecer uma arquitetura onde os nós sensores podem:

- Baixar executáveis e DLLs (*Dynamic Link Libraries*), identificadas por URLs (*Uniform Resource Locators*) de repositórios ou de suas *caches*;
- Executar o programa no nó local usando dados de entrada que podem estar remotamente localizados;
- Escrever os dados de saída em um local remoto.

O modelo RSN é baseado no modelo *applet* de Java generalizado para executáveis diversos e combinado com um serviço de procura (*lookup*). A RSN vê os nós sensores como *switches* de rede com protocolos adaptáveis dinamicamente, tentando mapear diretamente a motivação e os métodos das redes ativas em RSSFs.

8.5 Análise das propostas

Alguns desses trabalhos apresentados estão fracamente relacionados com o SensorBus. Trabalhos sobre simuladores (SensorSim) e linguagens

de consulta (SQTL) foram apresentados devido ao suporte que dão a outros trabalhos fortemente relacionados ou porque podem, de uma maneira mais trabalhosa, atingir os objetivos do SensorBus. A possibilidade de realizar simulações híbridas no SensorSim permite o desenvolvimento de aplicações em RSSFs, sendo essa a única particularidade comum com o SensorBus, visto que o SensorSim é basicamente um simulador para estudo de algoritmos para RSSFs.

Outros trabalhos, como as abordagens orientadas a QoS, apresentam propósitos bem distintos do SensorBus e por isso não serão consideradas nesta análise. Como são tipos de *middleware*, suas descrições apresentadas neste capítulo fizeram-se necessárias para oferecer uma completa descrição do estado da arte em tipos de *middleware* para RSSFs.

Uma comparação compreensiva das diversas propostas de *middleware* com o SensorBus mostra-se problemática devido à diversidade de plataformas, aplicações e implementações. Contudo, é possível discutir esta tese face as diversas propostas feitas de *middleware* para RSSFs, levando-se em consideração as contribuições originais que foram introduzidas com o SensorBus, ou seja, a adaptação da aplicação e do *middleware* utilizando o conceito de ciência do contexto de execução e a implantação de linguagem de restrição em RSSFs.

Para facilitar a discussão, a figura 8.3 mostra uma tabela comparativa das propostas de *middleware* apresentadas que estão mais relacionadas com o SensorBus, considerando as características principais que um *middleware* para RSSFs deve apresentar.

Middleware	Categoria	Fusão de dados	Paradigma de comunicação	Tipo de linguagem	Configuração adaptativa	Filtros	Tipo de endereçamento
Cougar	Orientada a dados	Suporta	Requisição-Resposta	Consulta tipo SQL	Não suporta	Não suporta	Único
SINA	Orientada a dados	Suporta	Requisição-Resposta	Consulta tipo SQL	Não suporta	Não suporta	Por atributo
SQTL	Orientada a dados	Suporta	Requisição-Resposta	Consulta tipo SQL	Não suporta	Não suporta	Único
Smart Messages	Baseada em agente	Suporta	Migração de código	Não suporta	Não suporta	Não suporta	Único
SensorWare	Baseada em agente	Não suporta	Migração de código	Não suporta	Não suporta	Não suporta	Único
Runes	Baseada em agente	Não suporta	Migração de código	Não suporta	Suporta	Não suporta	Único
RSN	Baseada em agente	Não suporta	Migração de código	Não suporta	Não suporta	Não suporta	Único
Impala	Baseada em eventos	Não suporta	Migração de código	Não suporta	Suporta	Suporta	Único

Figura 8.3: Tabela comparativa de *middlewares* relacionados

8.5.1 Linguagem de restrição

Nos tipos de *middleware* que seguem a abordagem orientada a dados, como Cougar, SINA e SQTL, são utilizadas linguagens de consulta para permitir que as aplicações extraiam informações do ambiente. Nessas abordagens, a rede é vista como uma grande base de dados de informações. Cougar e SINA fornecem uma interface de base de dados distribuída para RSSFs que utiliza uma linguagem de consulta para permitir que as aplicações executem funções de monitoramento. O Cougar gerencia a potência distribuindo a consulta entre os nós sensores para minimizar a energia consumida na coleta dos dados. A SINA incorpora mecanismos de baixo nível para construir agrupamentos hierárquicos de nós sensores visando agregação de dados eficiente e também prover protocolos que limitam a retransmissão de informação similar de nós sensores próximos.

A abordagem orientada a dados gasta muitos recursos quando a taxa esperada da ocorrência do evento é baixa. Por exemplo, um cliente que necessita de atualização instantânea da informação teria que realizar continuamente consultas aos provedores de informação acarretando sobrecarga na rede e congestionamento. Além disso, a energia é um recurso escasso e consultas desnecessárias de informação devem ser evitadas.

Os *middlewares* que seguem a abordagem orientado a dados para RSSFs têm os mesmos objetivos do SensorBus, isto é, possibilitar o desenvolvimento de aplicações em RSSFs, mas utilizam linguagens de consulta diferentes. O SensorBus apresenta uma linguagem de restrição que é própria de *middleware* para facilitar a programação enquanto que Cougar e SINA utilizam o conceito de *queries* de banco de dados com uma linguagem de *scripts*. As linguagens de consultas em forma de *scripts* são mais adequadas para serem utilizadas em sistemas gerenciadores de bancos de dados. Esse tipo de linguagem eleva a carga computacional de um *middleware* quando comparado com a inclusão de uma linguagem de restrição bem menor.

Com exceção de SINA, que se preocupa com a eficiência da energia nos nós sensores através de *clustering* hierárquicos e endereçamento

baseado em atributo, as demais soluções orientadas a dados não são adequadas para o desenvolvimento de aplicações eficientes do ponto de vista de economia de energia nos nós sensores. O SensorBus aumenta a eficiência do *middleware* através do uso de filtros, ao passo que as abordagens orientadas a dados existentes não utilizam a filtragem dos dados para diminuir o movimento dos mesmos numa RSSF.

Os *middlewares* que seguem a abordagem baseada em eventos para RSSFs geralmente têm o mesmo paradigma de comunicação distribuída do SensorBus, isto é, o paradigma *publish-subscribe*. Nesse paradigma os dados são disseminados por um componente produtor para todos os componentes que mostram interesse nesses dados. O SensorBus estende esse modelo de comunicação para suportar também o tradicional modelo requisição-resposta. Nesse modelo os dados são obtidos sob demanda quando determinado componente faz uma requisição. A linguagem de restrição do SensorBus dá suporte a esse modelo.

Outro detalhe é que os *middlewares* que seguem a abordagem orientada a dados ou baseada em eventos existentes fazem uso pesado de simulações para validar as propostas, mas a validação em redes reais ainda não foi comprovada. A utilização dessas soluções em experimentos concretos pode exibir comportamentos não previstos na simulação. Já o SensorBus foi aplicado em uma RSSF real.

8.5.2 Adaptação reativa da aplicação

Uma das primeiras questões que um *middleware* que suporta ciência do contexto deve se preocupar é com a captação da informação de contexto e seu processamento de maneira que seja útil para a aplicação. Vários pesquisadores (SCHILIT *et al*., 1994) têm estudado e desenvolvido sistemas que coletam informações de contexto, tais como localização; localização relativa; características de dispositivos como poder de processamento; características do ambiente físico como nível de ruído e largura de banda; e atividades dos usuários.

A capacidade de coletar e processar informações de contexto é somente o primeiro passo em direção ao desenvolvimento de aplicações

ciente de contexto. As aplicações, de fato, devem ser providas com mecanismos e primitivas para realizar a adaptação quando ocorram trocas no contexto. Vários sistemas existem que obtêm esse objetivo de diferentes maneiras. Como o SensorBus é um *middleware* para aplicações de RSSFs que faz reconfiguração, serão abordados apenas aqueles tipos de *middleware* que utilizam reconfiguração em seus projetos. Nessas soluções, a reconfiguração é obtida através de código móvel. Uma outra possibilidade de fornecer reconfiguração é através de reprogramação de memória.

Reprogramação de memória ainda não foi utlizada em *middleware* para RSSFs, sendo considerada apenas um modelo de computação. Por isso, as soluções que utilizam esse modelo não foram descritas neste capítulo. Apesar disso, duas soluções que utilizam esse modelo merecem ser mencionadas: XNP (Crossbow, 2004), uma API de programação em rede e Deluge (HUI & CULLER, 2004), um protocolo de disseminação de dados para programação em rede. XNP e Deluge suportam reconfiguração através da reprogramação da memória dos nós sensores, sendo que Deluge melhora a solução XNP através do suporte a protocolos *multi-hop*. As soluções que utilizam reprogramação de memória levam bastante tempo e consomem muita energia para transferir todo código através de uma RSSF e alterar as memórias dos nós sensores.

Os tipos de middleware que utilizam a tecnologia de agentes (S*mart Messages,* SensorWare, Runes e RSN) e Impala exploram técnicas de código móvel para alterar a funcionalidade do *middleware* que executa em um nó sensor. A chave da eficiência de energia é que as aplicações sejam tão modulares quanto possíveis, possibilitando pequenas alterações que requerem pouca energia de transmissão. As soluções por agentes móveis suportam fraca mobilidade, pois o estado de execução não é transferido.

O SensorBus contorna as limitações das soluções acima baseado no princípio de metadados. Através de metadados, obtem-se separação de conceitos, isto é, distingue-se o que o *middleware* faz do que como ele faz. Dessa maneira, é possível acrescentar reconfiguração no *middleware* através de um perfil codificado em XML.

Além das limitações citadas acima, uma limitação comum dessas abordagens reside na falta de suporte pelo que foi definido como adaptação pró-ativa para mudanças de contexto. Isto é, essas soluções não fornecem à aplicação os mecanismos para facilitar a configuração de serviços que a aplicação fornece aos seus usuários, baseado na troca de contexto. No caso de serviços que requerem adaptação para mudanças no contexto, a aplicação é forçada a realizar as tarefas repetitivas de consultar seu contexto para descobrir que política é mais adequada para o contexto corrente. O SensorBus estende a adaptação da aplicação, fornecendo comandos para transferir esse conhecimento para o *middleware*, assim automatizando essas tarefas.

8.5.3 Adaptação do *middleware*

Além da adaptação reativa da aplicação, o modelo do SensorBus adota uma perspectiva centrada no *middleware* e investiga princípios e mecanismos para obter a adaptação do *middleware* levando em consideração as mudanças no contexto. Nesse caso, estende-se a definição de contexto para incluir o paradigma de comunicação, os protocolos de roteamento, o tipo de roteamento e assim por diante.

As outras propostas de *middleware* apresentadas neste capítulo não possibilitam ao usuário informações sobre o contexto de execução do nó sensor. Já o SensorBus pode disponibilizar essas informações através de metadados, principalmente do nível de energia da bateria dos nós e com isso procedimentos de atualização da rede poderão ser tomados com base nessas informações.

9 CONCLUSÕES E TRABALHOS FUTUROS

O principal objetivo deste trabalho foi o desenvolvimento de abstrações e mecanismos que quando implementados em uma camada de *middleware*, efetivamente melhora a construção e execução de aplicações para RSSFs. Mais especificamente, a adaptação da aplicação é realizada através de um *middleware* que percebe mudanças no contexto de execução. O resultado final do trabalho foi o desenvolvimento do SensorBus, um MOM para desenvolvimento de aplicações distribuídas em RSSFs. O SensorBus permite a construção de aplicações eficientes do ponto de vista de economia de energia nos nós sensores através do uso de filtros. A permutação do mecanismo de comunicação permite o endereçamento das principais características de RSSFs. As linguagens de restrição e de consulta possibilitam o acesso *on-line* das informações coletadas pelos nós sensores. Uma implementação do SensorBus foi fornecida e aplicações foram construídas usando essa implementação, para demonstrar sua eficiência no desenvolvimento de aplicações para RSSFs e também sua adequação para ambientes que possuem recursos restritos. Neste capítulo, revisam-se as principais contribuições desta tese e discutem-se algumas questões em aberto que foram deixadas para um desenvolvimento futuro.

9.1 Contribuições

A contribuição original desta tese é o desenvolvimento de um modelo baseado em metadados e comandos que permite às aplicações para RSSFs serem adaptativas à troca de contexto tanto de maneira reativa quanto pró-ativa. A adaptação ocorre por meio de metadados ou perfis de aplicação que contêm as associações reativas e pró-ativas. As associações reativas relacionam configurações de contexto a políticas que devem ser tomadas

quando tais configurações ocorrem. As associações pró-ativas relacionam os serviços que as aplicações desejam customizar a políticas que devem ser usadas para fornecer tais serviços, e às configurações de contexto que devem ser aplicadas para uma política ser possível.

Foi fornecida uma definição de contexto ampla e flexível, que abstrai a heterogeneidade dos sensores físicos e as informações coletadas por eles. O *middleware*, e não as aplicações, se encarrega de lidar com uma diversidade de sensores e de codificar as informações obtidas de um modo uniforme. As aplicações acessam então essa codificação através de uma interface bem definida, bem como exploram essa abstração de contexto flexível para facilmente customizar o conjunto de recursos que formam o contexto de interresse em tempo de execução, possibilitando assim adicionar recursos específicos da aplicação também.

Esse modelo foi desenvolvido no *middleware* SensorBus e essa arquitetura foi implementada e avaliada para avaliar sua conveniência para RSSFs. Apenas um conjunto mínimo de componentes foi instalado tanto no PC conectado no nó sorvedouro quanto nos nós sensores. No lado do PC, foram selecionadas algumas políticas para construção do perfil de aplicação e no lado dos nós sensores, diversos componentes foram necessários para realizar vários tipos de consultas com intuito de coletar as informações ambientais da área investigada. De acordo com as funcionalidades que as aplicações demandam, esse conjunto de componentes pode ser facilmente estendido, isto é, novas políticas podem ser incorporadas e novos sensores podem ser monitorados.

A implementação do SensorBus foi avaliada através de um número de estudos de casos para avaliar a sua adequabilidade para o desenvolvimento de aplicações para RSSFs. Particularmente, demonstrou-se que o impacto do uso de metadados para configuração de políticas foi moderado, considerando-se as questões de desempenho. Sendo assim, o SensorBus mostra-se bem adequado para executar em RSSFs.

Para estimar a flexibilidade e usabilidade da abordagem adotada, implementou-se uma aplicação de consulta de variáveis ambientais sobre o *middleware* SensorBus e reportou-se essa experiência. Os resultados

obtidos mostraram que o tempo de desenvolvimento de aplicações é curto, pois as complexidades de manusear as mudanças que acontecem no contexto de execução estão confinadas no interior do *middeware*. Os projetistas de aplicações apenas têm que desenvolver uma pequena interface através da qual eles possam acessar e modificar as políticas que direcionam o comportamento do SensorBus.

9.2 Trabalhos futuros

O trabalho ora apresentado estabelece abstrações e mecanismos para melhorar o desenvolvimento e execução de aplicações para RSSFs. Baseadas nesse modelo, ampliações futuras dessa proposta poderão incluir:

- Aplicações que executam em RSSFs possuem poucos recursos disponíveis, de modo que elas precisam cuidadosamente explorar esses recursos, com intuito de fornecer aos usuários uma qualidade de serviço satisfatória. Quando mudanças no contexto e adaptação são implementadas, diferentes usuários apresentarão diferentes necessidades e as aplicações competirão para usarem os recursos disponíveis de acordo com suas necessidades. Quando esses conflitos aparecerem, um mecanismo de resolução de conflito deve ser implementado para que as aplicações estabeleçam um acordo sobre qual estratégia de adaptação aplicar, e assim cooperações corretas poderão ser estabelecidas. No modelo atual, o SensorBus assinala automaticamente a prioridade para as várias políticas, de modo que um conflito seja resolvido, selecionando e aplicando a política com maior prioridade associada entre as políticas conflitantes. Uma alteração no modelo de *middleware* do SensorBus consistiria em fornecer um mecanismo de resolução de conflito dinâmico que levasse em consideração as disponibilidades correntes dos recursos envolvidos.
- Apesar da definição de contexto apresentada neste trabalho ser bastante ampla para capturar um grande número de recursos disponíveis no ambiente físico, focalizou-se apenas a vista de contexto que é local ao dispositivo, isto é, a informação é captada

somente dos sensores internos aos dispositivos. Pode-se expandir essa vista para fora do dispositivo, a fim de incluir informações de contexto captadas pelos sensores de ambiente externo. Assim, haveria a possibilidade de customizar o *middleware* através de informações ambientais. Por exemplo, caso não se tivesse mais interesse em medidas de temperatura quando ela caísse abaixo de um determinado nível, se poderia configurar o *middleware* para finalizar o processo de consulta do dispositivo sensor, economizando-se assim a energia disponível em tal nó sensor.

- Embora não se tenha dado um maior destaque à questão da comunicação, é importante ressaltar que o paradigma de comunicação por passagem de mensagem utilizada no projeto do SensorBus pode ser melhorado, pois esse paradigma não suporta a persistência das filas de mensagens, o que significa que o SensorBus não suporta desconexões. Uma importante extensão consistiria integrar o Serviço de Mensagens Java (*Java Message Service* – JMS) na implementação do SensorBus. Atualmente, o JMS é utilizado apenas em redes sem fio estruturadas e uma questão de pesquisa importante seria portar tal serviço para redes *ad hoc*, principalmente para RSSFs.
- Na continuidade da pesquisa outras questões poderão ser investigadas, por exemplo, apesar do SensorBus utilizar o TinySec (KARLOF *et al*., 2000), um eficiente cifrador de blocos acoplado à pilha de rádio do sistema operacional TinyOS, existe a necessidade de se aprofundar a questão da segurança em outras versões do SensorBus, de modo que ele fique independente de sua atual plataforma de implementação.

APÊNDICE A

A.1 Definição do Esquema do Perfil de Aplicação

```
<?xml version="1.0" encoding="UTF-8"?>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">

<xs:element name="APPLICATION_PROFILE">
<xs:complexType>
<xs:all>
<xs:element ref="REACTIVE" minOccurs="1"/>
<xs:element ref="PROACTIVE" minOccurs="1"/>
</xs:all>
<xs:attribute name="name" type="xs:string" use="required"/>
</xs:complexType>
</xs:element>

<xs:element name="REACTIVE">
<xs:complexType>
<xs:sequence>
<xs:element ref="POLICY" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
<xs:attribute name="frequency" type="xs:short" use="required"/>
</xs:complexType>
</xs:element>

<xs:element name="PROACTIVE">
<xs:complexType>
<xs:sequence>
<xs:element ref="SERVICE" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
</xs:element>

<xs:element name="SERVICE">
<xs:complexType>
<xs:sequence>
<xs:element ref="POLICY" minOccurs="1" maxOccurs="unbounded"/>
</xs:sequence>
<xs:attribute name="name" type="xs:string" use="required"/>
</xs:complexType>
</xs:element>

<xs:element name="POLICY">
<xs:complexType>
<xs:sequence>
<xs:element ref="CONTEXT" maxOccurs="unbounded"/>
</xs:sequence>
<xs:attribute name="name" type="xs:string" use="required"/>
</xs:complexType>
</xs:element>

<xs:element name="CONTEXT">
<xs:complexType>
<xs:sequence>
```

```
<xs:element ref="RESOURCE" minOccurs="1" maxOccurs="unbounded"/>
</xs:sequence>
<xs:attribute name="id" type="xs:positiveInteger" use="required"/>
</xs:complexType>
</xs:element>

<xs:element name="RESOURCE">
<xs:complexType>
<xs:sequence>
<xs:element ref="OPERATOR" minOccurs="1" maxOccurs="1"/>
<xs:element ref="STATUS" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
<xs:attribute name="name" type="xs:string" use="required"/>
</xs:complexType>
</xs:element>
<xs:element name="OPERATOR">
<xs:complexType>
<xs:attribute name="name" type="xs:string" use="required"/>
</xs:complexType>
</xs:element>

<xs:element name="STATUS">
<xs:complexType>
<xs:attribute name="value" type="xs:string" use="required"/>
</xs:complexType>
</xs:element>

</xs:schema>
```

REFERÊNCIAS BIBLIOGRÁFICAS

ABDELZAHER T., BLUM B., CAO Q., EVANS D., GEORGE J., GEORGE S., HE T., LUO L., SON S., STOLERU R., STANKOVIC J., WOOD A. **EnviroTrack: Towards an Environmental Computing Paradigm for Distributed Sensor Networks.** Technical report, Department of Computer Science, University of Virginia, 2003.

AGRE, J. & CLARE, L. **An integrated architecture for cooperative sensing networks**. Computer, Volume 33 Issue 5, Pages 106-108, 2000.

AHO, A. V., SETHI, R. and ULLMAN, J. D. **Compiler Principles Techniques and Tools**. Addison-Wesley, 1988.

APPARAO, V., BYRNE, S., CHAMPION, M., ISAACS, S., JACOBS, I., HORS, A. L., NICOLl, G., ROBIE, J., SUTOR, R., WILSON, C., and WOOD, L. **Document Object Model (DOM) Level 1 Specification**. W3C Recommendation http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001, World Wide Web Consortium, 1998.

ARRUE, B. C., OLLERO, A., and de DIOS, J. R. M., **An intelligent system for false alarm reduction in infrared forest-fire detection**, IEEE Intelligent Systems, vol. 15, pp. 64-73, 2000.

ASTHANA, A., CRAVATTSAND, M., KRZYZANOWSKI, P. **An indoor wireless system for personalized shopping assistence**. In Proceedings of IEEE Workshop on Mobile Computing Systems and Applications, pages 69-74, Santa Cruz, California, December 1994.

Atmel inc. Disponível na Internet via http://www.atmel.com/products/avr/, 2005

BIEGEL, G. & CAHILL, V. **A Framework for Developing Mobile Context-aware Applications**. Proc. PerCom, 2004.

BOURRET, R. **Declaring Elements and Attributes in an XML DTD**. Disponível na Internet via WWW:URL: http://www.rpbourret.com/xml/xmldtd.htm.[Outubro 1999]

BOULIS A. & SRIVASTAVA M. B. **A framework for efficient and programmable sensor networks.** EE Department, University of California, 2001.

BRESLAU, L., ESTRIN, D., FALL, K., FLOYD, S., HEIDEMANN, J., HELMY, A., HUANG, P., Mc-CANNE, S., VARADHAN, K., XU, Y., and YU, H. **Advances in Network Simulation**. *IEEE Computer,* Vol. 33, No. 5, pp. 59-67, 2000.

BYRNE, S. *et al*. **W3C recommendation on Document Object Model.** [on line]. Disponível na Internet via WWW:URL:http://www.w3.org/TR/REC-DOM-Level-1 [1 Outubro 1998].

CAPRA, L., EMMERICH, W., MASCOLO, C. **Middleware for Mobile Computing**. Technical report, University College London, Dept. of Computer Science, 2001a.

CAPRA, L., MASCOLO, C., ZACHARIADIS, S., and EMMERICH, W. **Towards a mobile computing middleware: a synergy of reflection and mobile code techniques**. In Proceedings of the 8th IEEE Workshop on Future Trends of Distributed Computing Systems, 2001b.

CAPRA, L. **Reflective Mobile Middleware for Context-Aware Applications**, PHD thesis, University College London, Dept of Computer Science, 2003.

CHESS, D., HARRISON, C., KERSHENBAU, A. **Mobile agents: Are they a good idea?** Technical report, IBM Research Division, T. J. Watson Research Center, Yorktown Heights, New York, mar 1995.

Chipcon inc. Disponível na Internet via http://www.chipcon.com/, 2005

CLARK, J. & DeRose, S. **XML Path Language (XPath)**. Technical Report http://www.w3.org/TR/xpath, World Wide Web Consortium, 1999.

COHEN, N. H., PURAKAYASTHA, TUREK, J., WONG, L. and YEH, D. **Challenges in flexibe aggregration of pervasive data**. IBM Research Report RC 21942, IBM Research Division, Yorktown Heights, NY, USA, january 2001.

COULORIS, G., DOLLIMORE, J., and KINDBERG, T. **Distributed Systems: Concepts and Design**. Third edition. Addison-Wesley, 2001.

Crossbow Technology, Inc., XNP – X Network Programming. Disponível na Internet via http://www.tinyos.net/tinyos-1.x/, 2004.

CURINO, C. et al. **TinyLIME: Bridging Mobile and Sensor Networks through Middleware**. Proc. PerCom, 2005.

DAVIES, N., FRIDAY, A., WADE, S., and BLAIR, G. **L2imbo: A Distributed Systems Platform for Mobile Computing**. ACM Mobile Networks and Applications (MONET), Special Issue on Protocols and Software Paradigms of Mobile Networks, 3(2), 1998.

ENDLER, M. & SILVA, F. J. S. **Requisitos e Arquiteturas de software para computação móvel.** III Workshop de Comunicação sem Fio e Computação Móvel, Recife, PE, 2001.

EMMERICH, W., FINKELSTEIN, A., and SCHWARZ, W. **Markup Meets Middleware**. In 7th Int. Workshop on Future Trends in Distributed Systems, Capetown , South Africa, 1999.

EMMERICH, W. **Engineering Distributed Objects**. John Wiley & Sons, Apr. 2000a.

EMMERICH, W. **Software Engineering and Middleware: A Roadmap**. In The Future of Software Engineering – 22nd Int. Conf. On Software Engineering (ICSE2000), pages 117 –129. ACM Press, May 2000b.

ELIASSEN, F., ANDERSEN, A.,BLAIR, G. S., COSTA, F., COULSON G. GOEBEL, V., RAFAELSEN, O., SAIKOSKI, K. B., and YU, W. **Next Generation Middleware: Requirements, Architecture and Prototypes**. In Proceedings of the 7th IEEE Workshop on Future Trends

in Distributed Computing Systems, pages 60-65. IEEE Computer Society Press, Dec. 1999.

ESTRIN, D., GOVINDAN, R., HEIDEMANN, J. and KUMAR, S. **Next century challenges: Scalable Coordination in Sensor Networks**. Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking, Pages 263-270, 1999.

ESTRIN, D., GIROD, L., POTTIE, G., SRIVASTAVA, M. **Instrumenting the world with wireless sensor networks**. IEEE International Conference on Acoustics, Speech, and Signal Processing, pages 2033-2036, 2001.

Exolab.org. **OpenORB**. http://openorb.exolab.org/openorb.html, 2001.

FORMAN, G. & ZAHORJAN, J. **The challenges of mobile computing**. IEEE Computer, abr. 1994.

FpML. Introducing FpML: **A new standard for e-commerce**. http://www.fpml.org, 1999.

GAMMA, E., HELM, R., JOHNSON, R. e VLISSIDES, J., **Padrões de Projeto: Soluções Reutilizáveis de Software Orientado a Objetos**. Bookman, 2000.

GANESAN, D., KRISHNAMACHARI, B., WOO, A., CULLER, D., ESTRIN, D. and WICKER, S. **Complex Behavior at Scale: An Experimental Study of Low-Power Wireless Sensor Networks**. Technical Report CSD-TR 02-0013, UCLA, February 2002.

GAY, D., LEVIS, P., BEHREN R. von., **The nesC Language: A Holistic Approach to Networked Embedded Systems**, University of California, Berkeley, 2002.

GELERNTER, D. **Generative Communication in Linda**. ACM Transactions on Programming Languages and Systems, 7(1):80-112, 1985.

HAAHR, M., CUNNINGHAM, R., and CAHILL, V. **Supporting CORBA Applications in a Mobile Environment**. In Int. Conf. On Mobile Communications (Mobicom). ACM Press, Sept. 1999.

HAN, Q. et al. **AutoSeC: An Integrated Middleware Framework for Dynamic Service Brokering**. IEEE Distributed Systems Online, Vol.2, No.7, 2001.

HEIDEMANN, J., SILVA, F., INTANAGONWIWAT, C., GOVINDAN, R., ESTRIN, D., and GANESAN, D. **Building efficient wireless sensor networks with low-level naming**. In Proceedings of the Symposium on Operating Systems Principles, pages 146-159, Chateau Lake Louise, Banff, Alberta, Canadá, October 2001.

HEINZELMAN, W., KULIK, J., BALAKRISHNAN, H. **Negotiation-based protocols for disseminating information in wireless sensor networks**. Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking, 1999.

HEINZELMAN, W., CHANDRAKASAN, A. and BALAKRISHNAN, H. **Energy-efficient communication protocol for wireless micro sensor networks**. Proceedings of the 33rd Annual Hawaii International Conference on System Sciences, Pages 3005-3014, 2000.

HILL, J., SZEWCZYK, R., WOO, A., HOLLAR, S., CULLER, D., and PISTER, K. **System architecture directions for network sensors**. In ASPLOS, 2000.

HOHLT, B. A. **The Design and Evaluation of Network Power Scheduling for Sensor Networks**. A dissertation submitted in partial satisfaction of the requirements for the degree of Doctor of Philosophy in Computer Science. University of California, Berkeley. Spring., 2005

HONG, W. & MADDEN, S. **TinySchema: Managing Attributes, Commands and Events in TinyOS.** Disponível na Internet via http://www.xbow.com/Support/, 2003.

HUI, J. & CULLER, D. **The dynamic behavior of a data dissemination protocol for network programming at scale**. In Proceedings of the 2nd International conference on embedded networked sensor systems. ACM Press, 2004.

IBAMA/GTZ - Guia de Chefe – Dezembro de 2000. Disponível na Internet via http://www2.ibama.gov.br/unidades/guiadechefe/guia/t-1corpo.htm

IEEE1451. **Smart transducer interface for sensors and actuators**. Disponível na Internet via http://standards.ieee.org, 1998.

IMIELINSKI, T. & BADRIANTH R. B. **Mobile Wireless Computing: Challenges in data management**. Communications of the ACM, n° 10, p. 19-28, out. 1994.

INFINITY. **Infinity Network Trade Model Overview**. http://www.infinity.com/ntm/pdf/ntmOverview.pdf, 1999.

INTANAGONWIWAT, C., GOVINDAN, R., and ESTRIN, D. **Directed diffusion: A scalable and robust communication paradigm for sensor networks**. In Proceedings of the ACM/IEEE International Confererence on Mobile Computing and Networking, pages 56-67, Boston, MA, USA, Aug. 2000.

ISO10746-1. **Open Distributed Processing – Reference model**. Technical report, International Standardization Organization, 1998.

JAIKAEO, C., SRISATHAPORNPHAT, C. and SHEN, C., **Querying and tasking of sensor networks**, SPIE's 14th Annual International Symposium on Aerospace/Defense Sensing, Simulation, and Control, Orlando, Florida, April 2000.

JOHNSON, D. and MALTZ, D. **Dynamic Source Routing in Ad Hoc Wireless Networks**. Mobile Computing, pages 153-181, Kulwer Academic, 1996.

JPL Sensor Webs. Disponível na Internet via http://sensorwebs.jpl.nasa.gov/, 2002

KARLOF, C., SASTRY, N., WAGNER, D. **TinySec: A Link Layer Security Architecture for Wireless Sensor Networks.** Proceedings of ACM SenSys, 2000.

KON, F., ROMAN, M., LIU, P., MAO, J., YAMANE, T., MAGALHAES, L. C., and CAMBPELL, R. **Monitoring, Security, and Dynamic Configuration with the dynamic TAO Reflective ORB**. In International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware'2000), pages 121-143, New York, Apr. 2000.

KUMAR, R. et al. **DFuse: A Framework for Distributed Data Fusion**. Proc. SenSys, 2003.

KVM – **K Virtual Machine Spcification**. Disponível na Internet via http://java.sun.com./prioducts/kvm/ em Setembro/2003

LAZARIDIS, I. et al. **QUASAR: Quality Aware Sensing Architecture**. ACM SIGMOD Record, 2004.

LEDOUX, T., **OpenCorba: a Reflective Open Broker**. In Reflection'99, volume 1616 of LNCS, Saint-Malo, France, 1999.

LEE W. M., FOO S. M., WATSON K., WUGOFSKI, T. **Beginning WAP, WML, and WMLScript**. Wrox Press Ltd., 2000.

LEVIS, P. and CULLER, D. **Maté: A tiny virtual machine for sensor networks**. In ASPLOS, San Jose, CA, October 2002.

LEVIS, P., LEE N., WELSH, M. and CULLER, D. **TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications**, Proceedings of the First International Conference on Embedded Networked Sensor Systems, 2003.

LI, S. et al. **Event Detection Services Using Data Service Middleware in Distributed Sensor Networks**. Proc. Int. Workshop on Information Processing in Sensor Networks, 2003.

LINDSEY, S. & RAGHAVENDRA, C. S. **PEGASIS:Power-Efficient Gathering in Sensor Information Systems**. International Conference on Communications, 2001.

LIU, T. & MARTONOSI, M. **Impala: A middleware system for managing autonomic, parallel sensor systems**. In ACM SIGPLAN Symposium

on Principles and Practice of Parallel Programming (PPoPP´03), June 2003.

LIU, T. et al. **Implementing software on resource-constrained mobile sensors: experiences with Impala and ZebraNet**. Proc. MobiSys, 2004.

LONG, S., KOOPER, R., ABOWD, G. D., and ATKENSON, C. G. **Rapid prototyping of mobile context-aware applications: the Cyberguide case study**. In Proceedings of the Second Annual International Conference on Mobile Computing and Networking, pages 97-107, White Plains, NY, November 1996.

LOUREIRO, A. A. F., NOGUEIRA, J. M. S., RUIZ, L. B., MINI, R. A. F., NAKAMURA, E. F. e FIGUEIREDO, C. M. S., **Redes de Sensores sem Fio**. In 21° Simpósio Brasileiro de Redes de Computadores, Natal, RN, Brasil, Maio 2003.

MADDEN, S. et al. **TinyDB: An Acquisitional Query Processing System for Sensor Networks**. ACM Transactions on Database Systems, Vol. 30, No. 1, 2005.

MARUYAMA, H., TAMURA, K., URAMOTO, N. **XML and Java:Developing Web Applications**. ADDISON-WESLEY, 1999.

MANJESHWAR, A. & AGRAWAL, D. P. **TEEN: a routing protocol for enhanced efficiency in wireless sensor networks**. International Proceedings of 15th Parallel and Distributed Processing Symposium, Pages 2009-2015, 2001.

MURPHY, A. & HEINZELMAN, W. **MiLan: Middleware linking applications and networks**, University of Rochester, Tech. Rep. TR-795, 2002.

MURPHY, A. L., PICCO, G. P., and ROMAN, G. –C. **LIME: A Middleware for Physical and Logical Mobility**. In Proceedings of the 21st International Conference on Distributed Computing Systems (ICDCS-21), May 2001.

OMG. Object Management Group. **The Common Object Request Broker: Architecture and Specification**, Aug. 1996.

OMG. Object Management Group. **XML/Value Request for Proposals**. 492 Old Connecticut Path, Framingham, MA 01701, USA, Aug. 1999.

PARK, V. D. and CORSON, M. S. **A highly adaptive distributed routing algorithm for mobile and wireless networks**. In Proceedings of IEEE INFOCOM, pages 103-112, April 1997.

PARK, S., SAVVIDES, A . and SRIVASTAVA, M. B., **SensorSim: A simulation framework for sensor networks**, Proceedings of MSWiM 2000, Boston, MA, August 11, 2000.

PERKINS, C. E. and BHAGWAT, P. **Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers**. In Computer Comm. Review, pages 234-244, October 1994.

PERKINS, C. E. and ROYER, E. **Ad hoc on-demand distance vector routing**. IEEE Workshop on Mobile Computing Systems and Applications, pages 90-100, February 1999.

POLLASTRE, J. & CULLER, D. **B-mac: An adaptive csma layer for low-power operation**, UC Berkeley, Tech. Rep cs294-f03/bmac, December 2003.

POTTIE, G. & KAISER, W. **Wireless integrated network sensors**, Communications of the ACM, vol. 43, no. 5, pp. 551-558, May 2000.

REINSTORF, T., RUGGABER, R., SEITZ, J., and ZITTERBART, M. **A WAP-based Session Layer Supporting Distributed Applications in Nomadic Environments**. In Int. Conf. On Middleware, pages 56-76. Springer, Nov. 2001.

REYNOLDS, P. & BRANGEON, R. **Service Machine Development for an Open Long-term Mobile and Fixed Network Environment**. Disponível na Internet via http://www.fub.it/dolmen/, 1996.

RENTALA, P., MUSUNURI, R., GANDHAM, S. and SAXENA U., **Survey on Sensor Networks**. University of Texas, Dept. of Computer Science, 2002.

RÖMER, K. , KASTEN, O., MATTERN, F. **Middleware Challenges for Wireless Sensor Networks**. Mobile Computing and Communications Review, volume 6, number 2, 2002.

ROMAN, G. -C., MURPHY, A. L., and PICCO, G. P. **Software Engineering for Mobility: A Roadmap**. In The Future of Software Engineering – 22nd Int. Conf. On Software Engineering (ICSE2000), pages 243-258. ACM Press, May 2000.

RSN – **Reactive Sensor Network Project**. Disponível na Internet via http://strange.arl.psu.edu/RSN/ em Setembro/2002

RUMBAUGH, J. JACOBSON, I. and BOOCH, G. **The Unified Modeling Language Reference Manual**. Addison Wesley, 1998.

RUNES – **Reconfigurable Ubiquitious Networked Embedded Systems**. Disponível na Internet via http://www.ist-runes.org, 2005.

SAMEER, T., ABU-GHAZALEH, N. B. and Heinzelman W. A **Taxonomy of Wireless Micro-Sensor Network Models.** *Mobile Computing and Communications Review*, Volume 1, Number 2, 2003.

SCHILIT, B., ADAMS, N., and WANT, R. **Context-Aware Computing Applications**. In Proc. of the Workshop on Mobile Computing Systems and Applications, pages 85-90, Santa Cruz, CA. IEEE Computer Society Press, 1994.

SHNAYDER, V., HEMPSTEAD, M., RON CHEN, B., WERNER-ALLEN, G., and WELSH, M. **Simulating the power consumption of large-scale sensor network applications**. In ACM SenSys 2004, ACM Press, Nov. 2004.

Smart Dust. **Autonomous sensing and communication in a cubic milimeter**. Disponível na Internet via http://robotics.eecs.berkeley.edu/%7Epister/SmartDust, 2002.

Smart Messages project. Disponível na Internet via http://discolab.rutgers.edu/sm, 2003.

SOHRABI, K., GAO, J., AILAWADHI, V., POTTIE, G. J., **Protocols for self-organization of a wireless sensor network**. IEEE Personal Communications, Volume 7, Issue 5, Pages 16-27, 2000.

SOUTO, E. et al. **A message-oriented middleware for sensor networks**. Proc. workshop on Middleware for pervasive and ad-hoc computing, 2004.

SQL92 – **Database Language SQL** – July 30, 1992. Disponível na Internet via http://www-2.cs.cmu.edu/afs/andrew.cmu.edu/usr/shadow/www/sql/sql1992.txt

SRISATHAPORNPHAT, C., JAIKAEO, C., and SHEN, C., **Sensor Information Networking Architecture**, International Workshop on Pervasive Computing (IWPC00), Toronto, Canadá, August 2000.

STALLINGS, W. **Data & Computer Communications**, Prentice-Hall, Englewood Cliffs, NJ, USA, 6th edition, 2001.

STEFFAN, J. et al. **Scoping in wireless sensor networks**. Proc. workshop on Middleware for pervasive and ad-hoc computing, 2004.

SUN MicroSystems Inc. **Remote Method Invocation Specification**. Disponível na Internet via http://java.sun.com./prioducts/rmi/ em Setembro/2000.

TENNENHOUSE, D. and WETHERALL, D., *Towards na Active Network Architecture*. In Multimedia Computing and Networking (MMCN 96), San Jose, CA:SPIE, January 1996.

ULMER, C., **Wireless Sensor Networks**. Disponível na Internet via users.ece.gatech.edu/~grimace/research/sensorsimii/wireless_sensor_network.ppt, 2002.

WALDO, J. **Javaspaces specification 1.0**. Technical report, Sun Microsystems, March 1998.

WAP Forum. **Wireless Application Protocol Architecture Specification**. Disponível na Internet via http://www.wapforum.org em Jan/2001.

WINS. **Wireless integrated network sensors**. Disponível na Internet via http://www.janet.ucla.edu/WINS/, 2002.

WOO, A., T. TONG, and CULLER D. **Taming the Underlying Challenges of Reliable Multihop Routing in Sensor Networks**, Proc. ACM SENSYS, Los Angeles, CA. November, 2003.

WYCKOFF, P., McLAUGHRY, S. W., LEHMAN, T. J., and FORD, D. A. **T Spaces**. IBM Systems Journal, 37(3):454-474, 1998.

W3C. **XML Schema Part 0: Primer**. [on line]. Disponível na Internet via WWW:URL:http://www.w3.org/TR/2000/wd-xmlschema-0-20000407/.[14 Agosto 2000].

YAO, Y. et al. **The Cougar Approach to In-Network Query Processing in Sensor Networks**. ACM SIGMOD, Vol 31, No. 3, pp.9-18, 2002.

YE, F., CHEN, A., LIU, S. and ZHANG, L. **A scalable solution to minimum cost forwarding in large sensor networks**. Proceedings of Tenth International Conference on Computer Communications and Networks, Pages 304-309, 2001.

YOUNG, M. **XML Step by Step**. Redmond: Microsoft Press, 2000.

YU, Y., KRISHNAMACHARI B. and PRASANNA V. K., **Issues in Designing Middlewate for Wireless Sensor Networks**. Disponível na Internet via http://ceng.usc.edu/~prasanna, 2004.

ZHAO, Y. J., GOVINDAN, R. and ESTRIN, D., **Residual energy scans for monitoring wireless sensor networks**. In IEEE Wireless Communications and Networking Conference (WCNC'02), Orlando, FL, USA, March 2002.