MIDDLEWARE BASEADO EM SERVIÇOS PARA REDES DE SENSORES SEM FIO

Flávia Coimbra Delicato

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIAS EM ENGENHARIA ELÉTRICA.

Aprovada por:

_____
Prof. José Ferreira de Rezende, Dr.

_____
Profª. Luci Pirmez, D.Sc.

_____
Prof. Aloysio de Castro Pinto Pedroza, Dr.

_____
Prof. Luiz Fernando Gomes Soares, D.Sc.

_____
Prof. Fábio Moreira Costa, Ph.D.

_____
Prof. Valmir Carneiro Barbosa, Ph.D.

RIO DE JANEIRO, RJ – BRASIL
JUNHO DE 2005

DELICATO, FLÁVIA COIMBRA

Middleware Baseado em Serviços para Redes de Sensores sem Fio [Rio de Janeiro] 2005

XIII, 197 p. 29,7 cm (COPPE/UFRJ, D.Sc., Engenharia Elétrica, 2005)

Tese - Universidade Federal do Rio de Janeiro, COPPE

1. Redes de Sensores Sem Fio

2. Middleware

3. Serviços Web

I. COPPE/UFRJ II. Título (série)

Ao Paulo, com amor

AGRADECIMENTOS

Percorre-se um longo caminho até chegar ao final de um Doutorado. Muitas pessoas cruzam nossa vida ao longo desse caminho, algumas apenas passando ao largo, outras mudando rumos. Muitas dificultam o caminho, outras o tornam possível, outras ainda o deixam mais alegre. É impossível lembrar de todos os que, direta ou indiretamente, deixaram uma marca nesse longo trajeto. Porém, tentarei deixar aqui registrados meus agradecimentos àqueles que me acompanharam, de alguma forma ajudaram, e tornaram possível esse momento tão desejado.

Em primeiro lugar, por uma questão de cronologia, gostaria de agradecer ao meu pai, por ter sido a primeira pessoa que me acendeu a centelha da curiosidade científica. Foi ele quem me ensinou a ver o mundo com olhos indagadores, o que certamente despertou meu interesse pela pesquisa e moveu-me nessa direção.

A todos os meus professores, que deixaram sua preciosa contribuição de conhecimento, o único bem que podemos de fato adquirir nessa vida.

Aos meus chefes da UERJ, Professores Maurício Reis, Humberto Soriano, Luiz Biondi, Carlos Alberto Correia, Moacyr Carvalho Filho e José Pimenta de Carvalho, por terem me permitido cursar o Doutorado.

As secretárias do NCE, Lina e Adriana, e da COPPE, Solange, pela ajuda nas chatíssimas e necessárias tarefas burocráticas.

À Direção do NCE, por ter fornecido o suporte necessário (bolsa, máquinas, sala) para realizar o Doutorado.

Aos colegas do Mestrado e do labnet, pelas idéias e dificuldades compartilhadas.

Ao Alexandre e ao André, pela grande ajuda nas simulações e implementações.

Aos Professores Luiz Fernando, Valmir, Aloysio, Fábio Kon e Fábio Costa, por terem aceito participar de minhas bancas.

Ao Rust, por tudo que me ensinou desde o Mestrado.

Ao Fábio Protti, pelas conversas produtivas, essenciais para este trabalho ter atingido seu estágio final.

Ao Ageu, pelo exemplo de professor, pelo incentivo, pelas conversas, por se colocar sempre disponível para ajudar, por tornar o caminho mais alegre.

Ao Rezende, por ter aceito me orientar, tornando esse caminho possível. Agradeço-lhe pelos valiosos ensinamentos, como professor e como orientador, pela paciência que teve comigo nos momentos difíceis, quando parecia que tudo ia dar errado, e todos os obstáculos possíveis (reais ou imaginários) surgiam em minha frente. Pela sua boa vontade em lidar com um assunto que não é bem sua praia, e ainda assim procurar sempre dar o máximo de contribuição.

Seria preciso um capítulo a parte para agradecer à Luci, minha querida professora, orientadora e, espero não ser presunção minha considerar, amiga. Não tenho dúvidas de que ela foi a pessoa mais importante de minha vida acadêmica até aqui. Certamente, também, foi quem tornou esse momento possível. Não que ela não tenha, diversas vezes, tornado o caminho penoso, com seu rigor e suas exigências como professora. Graças a isso, porém, aprendi muito, e sei que levarei comigo esse aprendizado. Todas as portas que pôde ela sempre me abriu, todas as palavras de incentivo que precisei ela sempre me disse, todas as vezes em que estava errada ela me corrigiu, e quando estava certa ela me congratulou. Além de tudo isso, sempre esteve pronta para ouvir e aconselhar-me em meus problemas pessoais, que não foram poucos nesses anos. Não sei o que mais se pode esperar de um orientador. Eu sem dúvida tive mais do que esperava.

E finalmente, também por uma questão cronológica, porque termino um caminho e começo outros, gostaria de agradecer ao Paulo, que esteve sempre ao meu lado ao longo de toda essa jornada e sei que estará nas próximas. Não só ao meu lado como companheiro, mas como profissional, participando desde a concepção da idéia dessa tese, até a revisão do texto. Agradeço pelo incentivo nas horas em que eu esmorecia, por levantar-me quando eu estava por baixo, ou por não me deixar cair. Pela enorme compreensão e pelo carinho, principalmente nos últimos meses, em que a ansiedade é crescente e o humor nem sempre é dos melhores. Não há palavras suficientes para expressar minha gratidão e minha admiração, mas fica o desejo de que compartilhemos muitos trabalhos e realizações, e que o companheirismo e a compreensão mútua sejam sempre parte de nossas vidas.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Doutor em Ciências (D.Sc.)

MIDDLEWARE BASEADO EM SERVIÇOS
PARA REDES DE SENSORES SEM FIO

Flávia Coimbra Delicato

Junho/2005

Orientadores: José Ferreira de Rezende
Luci Pirmez

Programa: Engenharia Elétrica

Este trabalho propõe um sistema de middleware baseado em serviços para redes de sensores sem fio. O middleware proposto fornece uma camada de abstração entre aplicações e a infra-estrutura de rede subjacente e fornece serviços especialmente designados para atender às necessidades específicas das aplicações e dos desenvolvedores das aplicações de RSSFs. O middleware provê uma interface padrão para o acesso aos serviços da rede, o que confere um alto grau de flexibilidade e interoperabilidade ao sistema. O projeto da interface disponibilizada pelo middleware foi inspirado na área de Serviços Web, e utiliza tecnologias correlatas à área, como o protocolo SOAP e a linguagem XML, ambos considerados padrões ubíqüos na Web. O middleware toma decisões de baixo nível em favor da aplicação e fornece capacidades de adaptação e ciência de contexto, a fim de lidar com o alto dinamismo inerente ao ambiente das RSSFs. Ao esconder detalhes quanto a configuração e adaptação da rede dos desenvolvedores, o middleware minimiza os esforços de desenvolver aplicações para RSSFs. Além de prover um modelo de programação abstrato para RSSFs, o middleware balanceia da melhor forma os requisitos de qualidade da aplicação e os recursos de energia consumidos na rede, visando prolongar seu tempo de vida útil, enquanto satisfaz a aplicação.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Doctor of Science (D.Sc.)

SERVICE-ORIENTED MIDDLEWARE FOR
WIRELESS SENSOR NETWORKS

Flávia Coimbra Delicato

June/2005

Advisors: José Ferreira de Rezende
Luci Pirmez

Department: Electrical Engineering

This work proposes a service-oriented middleware for Wireless Sensor Networks (WSNs). The middleware supplies an abstraction layer between applications and the underlying network infrastructure and it provides services specifically tailored to meet the unique requirements of WSN applications and application developers. The middleware offers a standar interface for accessing the WSN services, which gives the system high degrees of interoperability and flexibility. The design of the middleware interface was leverage by the Web Services area, and it adopts correlated technologies, such as the SOAP protocol and the XML language, both considered as ubiquos Web standards. The middleware takes low-level decisions on behalf the application and it provides adaptation and context awareness capabilities, in order to deal with the high dynamics inherent to the WSN environments. By hiding the details regarding network configuration and adaptation from application developers, the middleware minimizes the efforts of developing WSN applications. Besides supplying an abstract programming model to WSN applications, the middleware keeps the balance between application QoS requirements and the network lifetime. It monitors both network and application execution states, performing a network adaptation whenever it is needed, aiming to extend the network lifetime, while meeting the application needs.

SUMÁRIO

Lista de Figuras

Lista de Tabelas

1 Introdução

Neste capítulo, apresenta-se o escopo, a motivação e os objetivos da presente tese. A Seção 1.1 aborda conceitos básicos sobre redes de sensores sem fio, necessários para a compreensão deste trabalho. As Seções 1.2 e 1.3 introduzem as principais motivações e os objetivos da tese. Finalmente, a Seção 1.4 contém a estrutura deste manuscrito.

1.1 CONCEITOS BÁSICOS DE REDES DE SENSORES SEM FIO

A área de redes de sensores sem fio (RSSFs) constitui um campo de pesquisa emergente com amplas implicações ao conectar o reino digital ao mundo físico. A íntima conexão com o ambiente físico permite que os sensores forneçam medições locais detalhadas as quais são difíceis de se obter através de técnicas de instrumentação tradicionais ou de sensoriamento remoto [84]. A utilização de redes de sensores traz novas e amplas perspectivas para o monitoramento de variáveis ambientais. Por outro lado, traz novos problemas e desafios.

RSSFs são compostas por dezenas a milhares de dispositivos de baixo custo e de tamanho reduzido, capazes de realizar sensoriamento, processamento e transmissão de informação através de enlaces sem fio. Os sensores atuam de forma colaborativa, extraindo dados ambientais e transmitindo-os para um ou mais pontos de saída da rede, chamados sorvedouros, para serem analisados e adicionalmente processados. Tais sensores podem ser instalados um a um em locais pré-estabelecidos, mas na maioria dos casos são lançados de forma *ad hoc* na área alvo, sem obedecer um plano de instalação pré-definido.

Os recursos de processamento, armazenamento e energia disponíveis nos sensores são bastante limitados. Em geral, sensores são alimentados por baterias não recarregáveis. Devido à sua enorme quantidade e à sua instalação em locais possivelmente de difícil acesso, os sensores devem operar sem assistência humana por longos períodos de tempo. A constante substituição de sensores que tenham suas baterias esgotadas é indesejável e diminui parte dos benefícios do uso de RSSFs, como a instalação *ad hoc* e a não perturbação do ambiente a ser monitorado. Portanto, é importante manter os sensores "vivos" o maior tempo possível. Dessa forma, o tempo de vida operacional das RSSFs é

severamente limitado pela capacidade de bateria de seus nós. Economia de energia torna-se, então, uma questão crucial nessas redes. Sendo assim, todas as etapas de projeto e funcionamento de RSSFs devem levar em conta o consumo de energia e procurar otimizá-lo.

RSSFs em geral caracterizam-se por possuir uma alta densidade de nós. Vários sensores monitoram o mesmo fenômeno, gerando dados redundantes e, muitas vezes, fornecendo à aplicação um nível de qualidade maior do que o necessário. Como a maior fonte de consumo de energia nos sensores é a transmissão de dados, grande parte dos esforços de pesquisa em RSSFs visa propor soluções para obter e rotear os dados de forma eficiente em energia, a fim de estender o tempo de vida global da rede. Por um lado, há propostas cujo enfoque é minimizar o número de transmissões e/ou o tamanho das mensagens, realizando o roteamento eficiente em energia [15,60,66,156,158]. Por outro lado, pesquisas recentes [101,102,103] mostram que, em vez de fornecer uma redundância de dados desnecessária para a aplicação, a alta densidade de nós pode ser aproveitada para obter significativa economia de energia. Vários métodos podem ser empregados a fim de se obter essa economia.

O primeiro método permite a modificação da quantidade de energia consumida com comunicação e computação. Nesse método é possível ajustar a taxa com que os sensores reportam seus dados, alterar o ciclo de operação dos sensores, ou mesmo alterar as velocidades de processamento utilizadas pelos nós. Mecanismos utilizados para alterar o modo de operação de um nó sensor são chamados "*knobs*" do sistema. Exemplos de *knobs* do sistema fornecidos por RSSFs atuais incluem gerenciamento dinâmico de potência (*dynamic power management* [111]), escalamento dinâmico de voltagem (*dynamic voltage scaling* [148]) e escalamento de modulação (*modulation scaling* [108]).

No segundo método, sensores redundantes podem ser completamente desligados por determinados períodos de tempo [22,102]. Para tal, é necessário desenvolver um esquema (estático ou dinâmico) que determina quais sensores devem estar ativos para sensoriar e/ou rotear dados e em que período de tempo. Nós não selecionados podem ser desligados.

Por fim, o último método consiste em empregar técnicas de agregação de dados dentro da rede (*in-network*) buscando a redução da quantidade de comunicação. Nesse

caso, operações de fusão de baixo nível são realizadas em dados de sensores vizinhos, antes deles serem enviados para nós sorvedouros [52,58,66].

Obviamente, há uma negociação entre o consumo de energia na rede e a qualidade global do dado entregue para a aplicação, quando da escolha do método a adotar. Qualquer abordagem adotada deve ser balanceada com os requisitos de qualidade requeridos pela aplicação.

Além dos recursos escassos, tanto computacionais quanto de energia, outra característica intrínseca das RSSFs é que sua estrutura topológica é altamente dinâmica. Sensores podem se deslocar, podem ser retirados da rede ao terem seus recursos esgotados, novos sensores podem ser adicionados após a rede estar operacional ou, conforme mencionado anteriormente, sensores podem ser temporariamente desligados por medida de economia de energia, alterando assim a topologia ativa da rede. Adicionalmente, tanto as condições do ambiente no qual está instalada a rede quanto os interesses da aplicação também podem mudar. Para prover suporte a tal grau de dinamismo, sem requerer a constante intervenção do usuário, RSSFs devem possuir algum grau de auto-organização e adaptação. Qualquer solução que busque otimizar o uso dos recursos da rede deve preferencialmente adotar mecanismos que permitam a adaptação dinâmica às possíveis variações, em vez de basear-se em conjuntos de parâmetros fixos e pré-estabelecidos.

Quanto às aplicações para RSSFs, uma ampla gama é mencionada na literatura [1,2,3,10,11,16,20,21,41,42,62,80,84,96,106,109,112,145] como, por exemplo, aplicações militares, médicas, vigilância, monitoramento ambiental, controle e inventário de equipamentos em fábricas, prédios inteligentes e muitas outras. Embora se estendam por um amplo espectro, a maioria das aplicações de RSSFs podem ser caracterizadas por vários aspectos comuns. De um modo geral, todas as aplicações caracterizam-se por serem inerentemente distribuídas, requererem alguma forma de cooperação para atingir sua meta e demandarem algum nível mínimo de qualidade de serviço (QoS) [61]. Entretanto, apesar dessas características comuns, os requisitos específicos de QoS, como o grau de cobertura de sensoriamento, latência e acurácia de dados, além de outros aspectos, como o modelo de entrega de dados solicitado [131], são altamente dependentes de cada aplicação. Vários trabalhos [55,131,160] destacam a importância da participação da aplicação no processo de comunicação em RSSFs. Restrições e otimizações específicas

da aplicação podem reduzir em muito o número de transmissões e, por conseguinte, o consumo de energia da rede.

Vários protocolos específicos para roteamento de dados em redes de sensores têm sido propostos nos últimos anos [23,59,60,66,75,79,152,153,156]. A maioria desses protocolos baseia-se na comunicação em múltiplos saltos (*multihop*) de curto alcance desde a origem dos dados sensoriados até os nós sorvedouros, e adotam algum mecanismo de agregação de dados dentro da rede. Como há uma extensa gama de possíveis aplicações, é improvável que um único protocolo de comunicação atenda a todas elas. Uma questão importante, então, passa a ser a seleção do protocolo mais adequado a cada aplicação. Uma vez decidido o protocolo de comunicação mais adequado a ser usado para uma determinada aplicação, são necessários mecanismos eficientes para seletivamente ativar os nós sensores mais apropriados para atender a essa aplicação. Os nós selecionados precisam cooperar entre si para distribuir a tarefa de sensoriamento solicitada, agregar os resultados e rotear a informação final para o sorvedouro. Os recursos da rede precisam ser gerenciados durante a execução da tarefa, de modo a otimizar seu uso ao mesmo tempo em que os requisitos de QoS da aplicação são satisfeitos. O gerenciamento da rede torna-se complexo devido à heterogeneidade dos sensores e à existência de mais de uma aplicação em execução simultaneamente. Dessa forma, os custos impostos à rede ao aceitar uma nova aplicação devem ser estimados, e o impacto da aceitação sobre outras aplicações já em execução deve ser avaliado. Tais desafios não são triviais, considerando as características *ad hoc* da rede, as capacidades limitadas dos nós sensores, as possíveis variações no sistema e nas condições ambientais, e o fato de que os sensores devem operar sem assistência.

1.2 MOTIVAÇÃO

Desenvolvedores de aplicações para RSSFs em geral são especialistas em seu domínio de conhecimento, não em redes. Portanto, a escolha do protocolo de comunicação a ser usado na rede e a implementação de rotinas para gerenciar o uso dos recursos e o escalonamento dos nós ativos é uma tarefa difícil para tais desenvolvedores.

Em computação distribuída tradicional, o uso de sistemas de middleware [89,90,98] facilita o trabalho de desenvolvedores de aplicações, liberando-os de lidar com a complexidade gerada pela distribuição. Atividades como controle de concorrência,

transações, replicação de dados, segurança, e outros serviços de infra-estrutura são realizados por uma camada de middleware situada entre as aplicações e o sistema operacional.

Sistemas de redes de sensores também poderiam se beneficiar do uso de uma camada de middleware que fornecesse um ambiente de execução genérico para as aplicações, provendo uma abstração das funcionalidades de infra-estrutura da rede. Entre outras funções, o middleware pode decidir o melhor protocolo a ser usado de acordo com os requisitos da aplicação, coordenar a operação dos sensores na realização do objetivo dessa aplicação e gerenciar o uso dos recursos da rede. A fim de prover de forma eficiente a qualidade requerida pelas aplicações, muitas vezes é necessário interagir com os níveis mais baixos da pilha de protocolos dos sensores, ou mesmo com dispositivos de hardware. O middleware pode realizar essa interação em benefício da aplicação. Todas essas funções facilitam as tarefas dos desenvolvedores de aplicações para RSSFs e dos gerentes de tais redes.

Entretanto, apesar das vantagens já comprovadas da utilização de tecnologias de middleware em sistemas distribuídos tradicionais, apenas recentemente começou-se a considerar seu uso no projeto de RSSFs [61,81,82,103]. A maior parte dos trabalhos encontrados na literatura têm como foco aplicações simples de aquisição de dados, e na maioria dos casos, esses trabalhos consideram uma única aplicação por rede. Portanto, o projeto dos protocolos de rede e o das aplicações são em geral altamente acoplados, ou mesmo combinados como um procedimento monolítico. Tal acoplamento é justificado para se obter uma alta eficiência em termos de consumo de energia. Entretanto, as estratégias de projeto são muitas vezes *ad hoc* e impõem interação direta da aplicação com o sistema operacional embutido subjacente, ou mesmo com componentes de hardware dos nós sensores. Além disso, essa abordagem de projeto gera sistemas rígidos, com RSSFs construídas especificamente para uma aplicação alvo. Dados os custos de construção e instalação de uma rede de sensores e seu tempo de vida útil, tal inflexibilidade de uso é indesejável a longo prazo.

Atualmente, vislumbra-se que as RSSFs do futuro, além de terem tempos de vida longos, atenderão concorrentemente a múltiplas aplicações, embora em geral pertencentes a um mesmo domínio, porém com necessidades específicas bastante variadas. Para acomodar tais cenários, o pleno desenvolvimento de RSSFs vai demandar métodos de

projeto sistemáticos baseados em abstrações padrão do sistema [161]. Assim, um sistema de middleware posicionado entre a camada de aplicações e as camadas subjacentes (que incluem o hardware da rede, o sistema operacional e a pilha de protocolos), faz-se necessário para fornecer: (i) serviços padronizados para diversas aplicações; (ii) um ambiente em tempo de execução que possa gerenciar múltiplas aplicações; e (iii) mecanismos para obter utilização adaptativa e eficiente dos recursos do sistema [161].

As tecnologias de middleware convencionais não são adequadas para redes de sensores sem fio. Sistemas de middleware empregados em sistemas distribuídos tradicionais (dispositivos fixos e interligados por redes cabeadas) têm sido construídos aderindo à metáfora da "caixa preta", isto é, a complexidade inerente à distribuição é escondida do usuário e dos desenvolvedores, de modo que o sistema apareça como uma única entidade computacional integrada. Em outras palavras, a distribuição torna-se transparente. Em geral, tais sistemas de middleware assumem a existência de conexões ponto a ponto permanentes e de alta largura de banda entre os nós comunicantes (por exemplo, para fornecer suporte a chamadas remotas de procedimentos), enquanto RSSFs operam com baixas larguras de banda e a disponibilidade dos nós é altamente variável.

Esconder completamente os detalhes de funcionamento da RSSF da aplicação, ou seja, fornecer transparência, pode ser pouco eficiente. Sistemas de RSSF precisam detectar e adaptar-se tanto a mudanças no ambiente como a mudanças nos interesses da aplicação. Para fornecer transparência total, o middleware deve tomar decisões em benefício da aplicação. Entretanto, a aplicação normalmente pode tomar decisões mais eficientes e de melhor qualidade, baseada em informações específicas que são de seu conhecimento. Com o intuito de obter um melhor desempenho do sistema, deve-se permitir que a aplicação afete direta e ativamente o comportamento dos protocolos usados na rede e dos próprios sensores. Entretanto, é indesejável que os desenvolvedores de aplicações tenham conhecimento detalhado da infra-estrutura da rede, ou mesmo que uma aplicação fique presa em tempo de projeto a uma infra-estrutura específica.

Portanto, a camada de middleware deve atuar como intermediária, traduzindo requisitos das aplicações em parâmetros de configuração de infra-estrutura da RSSF. Ao mesmo tempo, o middleware deve fornecer mecanismos que permitam à aplicação monitorar o contexto de execução atual através de uma interface de alto nível. Isto é, o

middleware deve permitir à aplicação um nível de controle sobre o sistema subjacente, mas tal acesso deve ser fornecido com um grau de abstração mais elevado.

O contexto de execução, nesse caso, é definido em termos do estado da rede (conectividade, largura de banda, energia residual, etc.) e do estado da aplicação (valores dos dados monitorados). A partir do monitoramento do contexto de execução, a aplicação pode decidir alterar dinamicamente algum comportamento da rede, ou seja, realizar alguma forma de adaptação. Portanto, sistemas de middleware para RSSFs devem fornecer ciência de contexto ("*awareness*"), ao invés de transparência, como os sistemas de middleware tradicionais. Princípios de middleware reflexivo [19], como por exemplo a capacidade de reificar o contexto de execução, podem ser adotados no projeto da RSSF para atender as necessidades de suas aplicações e para lidar com o dinamismo do ambiente em que atuam.

1.3 OBJETIVOS

A presente tese propõe um middleware para redes de sensores sem fio. O sistema proposto adota uma abordagem de serviços, onde a RSSF é vista como uma fornecedora de serviços para aplicações clientes. Tal abordagem foi inspirada nas recentes pesquisas com Serviços Web [53]. No projeto do sistema são utilizadas tecnologias que fazem parte da especificação de Serviços Web, como o protocolo SOAP [140] e a linguagem XML [135]. A escolha de linguagens baseadas em XML para representar as mensagens trocadas no sistema proposto foi motivada pela sua ampla aceitação como um padrão para troca de dados entre aplicações. Serviços Web apóiam-se fortemente na ubiqüidade do protocolo SOAP e da linguagem XML como meios fundamentais de comunicação entre sistemas que utilizam diferentes modelos de componentes, linguagens de programação e plataformas computacionais.

Usando a abordagem baseada em Serviços Web, o sistema de middleware proposto fornece uma camada de abstração entre as aplicações e a infra-estrutura de rede subjacente, facilitando a integração de diferentes aplicações com a RSSF. Além de fornecer um modelo de programação abstrato para aplicações de RSSFs, o middleware procura balancear os requisitos de QoS das aplicações e o tempo de vida global da rede, que é diretamente relacionado ao consumo de energia. O middleware é responsável por tomar decisões quanto aos protocolos de comunicação a serem adotados, os parâmetros de

configuração de tais protocolos, a organização topológica da rede, o escalonamento e os modos de operação dos sensores para a execução das tarefas de sensoriamento, entre outras coisas. O middleware monitora os estados de execução da rede e das aplicações, realizando procedimentos de adaptação quando necessário, com ou sem interferência da aplicação. Todas essas funcionalidades são fornecidas como serviços. Uma vez que o middleware implementa e, conseqüentemente, esconde dos desenvolvedores de aplicações a maior parte dos detalhes de configuração da rede e das decisões de baixo nível, ele minimiza os esforços de desenvolvimento de aplicações para RSSFs.

Os principais benefícios do sistema de middleware proposto são descritos a seguir.

- O sistema oferece uma camada de interoperabilidade entre as diferentes aplicações e a rede de sensores. Aplicações podem utilizar a rede com diferentes protocolos e topologias, sem tomar conhecimento dos detalhes de seu funcionamento.

- O middleware cria a abstração de uma rede de sensores genérica e otimizável, que fornece serviços para as várias aplicações. Esses serviços são acessados de forma flexível através de uma interface padrão e de alto nível para os usuários. Cada aplicação pode obter uma visão diferente dos dados que são coletados pela rede, de acordo com seus interesses e requisitos de QoS individuais.

- O sistema fornece um mecanismo para selecionar o protocolo de comunicação mais adequado e a configuração dos parâmetros desses protocolos, baseado nos requisitos de uma aplicação. Tal mecanismo é uma versão inicial, parcialmente validada, de um serviço robusto de decisão automatizada quanto à configuração da rede.

- O sistema realiza a seleção dos nós a serem ativados para a execução de uma tarefa de sensoriamento.

- O sistema gerencia o uso dos recursos da rede durante a execução da tarefa e adapta dinamicamente o funcionamento da rede de acordo com as variações ambientais e com a qualidade dos dados fornecidos para a aplicação.

- O sistema oferece uma interface que permite à aplicação inspecionar o estado da rede e participar ativamente do processo de adaptação. Tal interface utiliza o princípio da reflexão para capacitar à aplicação influenciar dinamicamente o comportamento do sistema.

- Em todas as suas atividades, o sistema busca atingir maior eficiência em termos de energia e maior tempo de vida da rede, enquanto satisfaz os requisitos da aplicação. Assim, através da utilização do sistema, a rede de sensores passa a funcionar como um mecanismo genérico e configurável de extração de dados ambientais, podendo adaptar-se a diferentes missões.

Os benefícios do sistema, bem como a viabilidade de sua implantação em redes de sensores reais foram avaliados através da realização de simulações e da implementação de um protótipo com várias das funcionalidades apresentadas nesta tese. Os resultados das avaliações foram promissores, demonstrando as vantagens de se adotar o middleware proposto na contrução de RSSFs.

1.4 ORGANIZAÇÃO DO TRABALHO

Este trabalho está organizado em 9 capítulos. O Capítulo 2 contém um embasamento teórico sobre redes de sensores sem fio, necessário para a compreensão da proposta. São apresentados conceitos relacionados à arquitetura, ao funcionamento e a procedimentos para a otimização de energia nessas redes. O Capítulo 3 aborda as tecnologias de middleware. Inicialmente, são enfatizadas as diferenças entre tecnologias desenvolvidas para sistemas distribuídos fixos tradicionais e tecnologias para sistemas móveis *ad-hoc* e, a seguir, são listadas as principais características que um sistema de middleware específico para RSSFs deve possuir. Como este trabalho adota várias tecnologias do paradigma de Serviços Web, tais tecnologias também são brevemente discutidas nesse capítulo. O Capítulo 4 apresenta os componentes lógicos do sistema de middleware para redes de sensores sem fio proposto, descrevendo os serviços fornecidos por tais componentes e as interfaces entre os componentes e entre estes e sistemas externos, como aplicações clientes, hardware do nó e protocolos de comunicação. O Capítulo 5 descreve os componentes físicos do middleware proposto, os quais são implementados como módulos de software de acordo com as soluções tecnológicas adotadas no projeto físico do sistema. O funcionamento do sistema obedece a um

conjunto de etapas que começam com a instalação da RSSFs e entrelaçam-se com a própria operação da rede. Essas etapas são descritas no Capítulo 6. O Capítulo 7 detalha dois dos principais serviços oferecidos pelo middleware proposto: o de decisão e o de seleção de nós ativos, e faz uma avaliação de seu desempenho. O Capítulo 8 contém a descrição do protótipo construído para o sistema proposto, incluindo o ambiente de desenvolvimento empregado e as principais classes implementadas. Finalmente, o Capítulo 9 apresenta as conclusões e os trabalhos futuros.

2 Redes de Sensores sem Fio

As redes de sensores sem fio (RSSFs) constituem um novo domínio da computação distribuída e têm sido alvo de grande interesse de pesquisa nos últimos anos. Uma RSSF é um tipo de rede *ad-hoc* com uma série de características e requisitos específicos. Seu principal objetivo é realizar tarefas de sensoriamento de forma distribuída em benefício de aplicações clientes. Portanto, essas redes funcionam como poderosos sistemas de aquisição de dados ambientais. Os dados são coletados através dos sensores distribuídos e entregues a pontos de saídas da rede, chamados nós sorvedouros, a fim de sofrerem análise e processamento adicional. As próximas seções fornecem conceitos básicos sobre RSSFs.

Quanto à sua estrutura, uma RSSF é organizada como um sistema com três componentes principais: (1) infra-estrutura; (2) pilha de protocolos; e (3) aplicação [132]. A infra-estrutura refere-se aos sensores físicos (suas características físicas e capacidades), ao número de sensores e à sua estratégia de instalação (como e onde eles são instalados). A pilha de protocolos refere-se ao software que implementa as diversas camadas de protocolos existentes em cada nó da rede. O componente de aplicação representa os interesses ou consultas do usuário, que são traduzidos em tarefas de sensoriamento a serem executadas pela rede. Vários procedimentos de otimizações envolvendo os três componentes organizacionais citados podem ser empregados na RSSF com o objetivo de aumentar o seu desempenho [131]. Por exemplo, características da aplicação podem influir na infra-estrutura da rede de sensores e nos protocolos utilizados. Portanto, informações no nível da aplicação devem ser aproveitadas pela rede para que ela possa alcançar uma maior eficiência em termos de energia. A Seção 2.1 descreve com detalhes os componentes organizacionais descritos acima.

Quanto ao modelo de entrega de dados, o qual é ditado pela aplicação, [131], as RSSF podem ser classificadas em contínuas, dirigidas a eventos, iniciadas pelo observador ou híbridas. Uma vez definido o modelo de entrega de dados a ser usado na rede para atender a uma determinada aplicação, protocolos que encaminhem de forma eficiente os dados desde sua origem (nós fontes) até o seu destino (nós sorvedouros) devem ser empregados.Vários protocolos específicos para RSSFs foram propostos nos

últimos anos. Exemplos de protocolos de disseminação de dados atualmente empregados em RSSFs são apresentados na Seção 2.2.

Quanto aos requisitos a serem atendidos no projeto e funcionamento das RSSFs, estes devem ser considerados sob as perspectivas das aplicações e da rede. Com o intuito de atender aos requisitos das aplicações clientes, um nível mínimo de qualidade de serviço (QoS) deve ser fornecido pela rede. A Seção 2.3 discute o conceito de QoS no contexto de RSSFs. Por outro lado, como a economia de energia é um requisito essencial em RSSFs, técnicas devem ser adotadas nos diferentes componentes da rede visando a otimização do consumo de energia. A Seção 2.4 apresenta várias dessas técnicas atualmente utilizadas.

2.1 ARQUITETURA DE REDES DE SENSORES SEM FIO

Esta seção tem por objetivo apresentar os três principais componentes organizacionais das RSSF, ou seja, a infra-estrutura, a pilha de protocolos e a aplicação, bem como descrever o modelo de comunicação e o modelo de entrega de dados adotados nessas redes.

2.1.1 Infra-estrutura

A **infra-estrutura** de uma RSSF consiste nos nós da rede e no seu estado de instalação no ambiente. Em geral, uma rede possui um ou mais nós de escoamento de dados, chamados de sorvedouros ou estações-bases, e diversos nós sensores. Sorvedouros geralmente são os nós da rede com maior poder computacional e sem restrições de energia. Esses nós fazem a interface entre a aplicação e a rede, servindo de ponto de entrada para a submissão dos interesses da aplicação e de concentrador dos dados enviados pelos nós sensores. Nós sensores contêm uma ou mais unidades de sensoriamento, e possuem capacidades de processamento e armazenamento limitadas. Sua função é coletar, eventualmente agregar, e transmitir seus próprios dados e os dos seus vizinhos para os nós sorvedouros.

O hardware de um nó sensor típico é composto por quatro subsistemas principais: o de sensoriamento, o de processamento, o de comunicação e o de energia (Figura 1). Dependendo da aplicação, podem existir outros subsistemas adicionais, como por

exemplo, o de localização (para determinar com precisão a posição de um nó) e o de movimento (para mover o nó para um local que permita a realização de uma tarefa) [114].

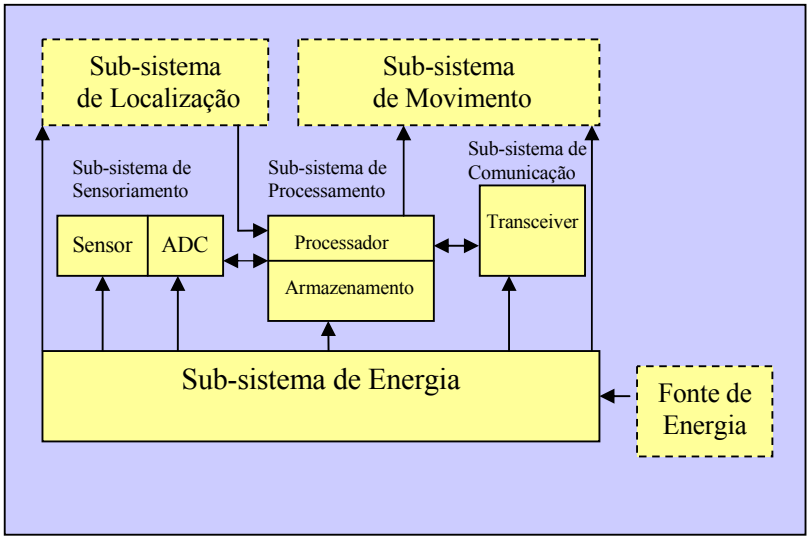


Figura 1: Componentes do hardware de um nó sensor.

O subsistema de sensoriamento (Figura 1) é geralmente composto pelos dispositivos de sensoriamento e pelo conversor de sinal analógico para digital (ADC). Uma RSSF pode conter muitos tipos diferentes de dispositivos de sensoriamento, como sísmicos, magnéticos, térmicos, visuais, infravermelhos, acústicos e radares. Tais dispositivos são capazes de monitorar uma ampla variedade de condições ambientais, incluindo: temperatura, umidade, movimento veicular, pressão, composição do solo, níveis de ruído, presença ou ausência de certos tipos de objetos, níveis de *stress* mecânico de objetos, e características como velocidade, direção e tamanho de um objeto. Os sinais analógicos produzidos pelos sensores a partir do fenômeno monitorado são convertidos em sinais digitais pelo ADC e são passados para o processador. O subsistema de processamento, que em geral está associado à unidade de armazenamento local, é responsável: (i) pela execução dos protocolos de comunicação e dos algoritmos de processamento de dados; (ii) pelo controle dos sensores; e (iii) pela gerência dos procedimentos necessários para que os nós atuem de forma colaborativa.

O subsistema de comunicação pode ser composto por um dispositivo ótico ativo ou passivo, ou por um dispositivo de rádio freqüência [3]. A maioria dos projetos de RSSFs atuais adota a comunicação via rádio freqüência. O subsistema de comunicação baseado em rádio freqüência é composto por um *transceiver*, uma antena e um conjunto de

componentes discretos para configurar as características da camada física, como sensibilidade e intensidade do sinal. Em geral, o transceiver pode operar em três modos: recepção, transmissão e desligado (*power-off*).

A fonte de energia dos sensores consiste de uma bateria e um conversor DC-DC. A bateria é um dispositivo complexo, cuja operação depende de diversas características como, por exemplo, suas dimensões, o material do qual é feita e a taxa de descarga. O conversor DC-DC é um dispositivo que recebe uma voltagem DC de entrada e produz uma voltagem DC de saída, tipicamente com um nível diferente do nível da entrada. Ele é responsável por fornecer uma tensão constante para o sensor e o seu fator de eficiência determina o tempo de vida da bateria [107].

Além dos componentes de hardware descritos acima, o projeto de um nó sensor pode incluir um sistema operacional rudimentar. Quando presente, o sistema operacional deve gerenciar a operação do nó sensor da forma mais eficiente possível. Um exemplo de sistema operacional desenvolvido especialmente para sensores, e utilizado em grande parte do hardware hoje existente, é o TinyOS [63].

O estado de instalação da rede diz respeito à localização dos sensores no espaço físico, à densidade da rede e a possíveis deslocamentos no caso de sensores móveis. Centenas a milhares de nós sensores são instalados na área a ser monitorada, distanciados no máximo algumas dezenas de metros uns dos outros. A densidade dos nós em uma RSSF pode ser tão alta quanto 20 nós/m^3 [123]. Gerenciar um elevado número de nós com alta densidade requer uma criteriosa tarefa de manutenção de topologia por parte dos protocolos da rede. A alta probabilidade de falhas ou o possível deslocamento dos sensores dificultam essa tarefa. Em [3] foram examinadas algumas questões relacionadas à manutenção e às mudanças na topologia que ocorrem durante as seguintes fases: (i) pré-instalação e instalação; (ii) pós-instalação e (iii) reinstalação de nós adicionais. Tais questões são discutidas brevemente a seguir.

Fase de Pré-instalação e Instalação. Nós sensores podem ser lançados em massa na área alvo por um veículo, míssel ou catapulta; podem vir pré-colocados de fábrica (sistemas embutidos); ou podem ser instalados individualmente por um ser humano ou robô. Embora o grande número de sensores e sua instalação sem assistência em geral impeçam a sua colocação de acordo com um plano cuidadoso, os esquemas para a

instalação inicial devem buscar (i) reduzir o custo de instalação; (ii) aumentar a flexibilidade de arranjo; e (iii) promover a tolerância a falhas [3].

Fase Pós-instalação. Após a fase de instalação, as mudanças de topologia ocorrem devido a alterações: (i) nas posições (sensores móveis) e no alcance dos nós (devido a ruídos, obstáculos e interferências); (ii) na energia disponível nos nós; (iii) nos requisitos das tarefas de sensoriamento ou devido ao mal-funcionamento ou à exaustão da fonte de energia dos nós [3]. Portanto, a topologia das RSSFs é propensa a freqüentes mudanças após a instalação. Para acomodar tais mudanças sem necessidade da constante intervenção do usuário, as RSSFs devem possuir mecanismos de adaptação e auto-reconfiguração.

Fase de Reinstalação de Nós Adicionais. Sensores adicionais podem ser instalados a qualquer momento para substituir nós danificados ou com mal-funcionamento ou, ainda, para atender a mudanças na tarefa da rede. A adição de novos nós requer a reorganização da rede. Tal reorganização também deve ocorrer, preferencialmente, com a mínima necessidade de participação do usuário.

2.1.2 Pilha de Protocolos

A pilha de protocolos usada pelos nós sorvedouros e sensores em uma RSSF é mostrada na Figura 2. Tal pilha consiste em cinco camadas horizontais e três planos verticais. As camadas são: aplicação, transporte, rede, enlace de dados e física. Os planos são: gerenciamento de energia, gerenciamento de tarefas e gerenciamento de mobilidade [3].

A camada de aplicação diz respeito a protocolos de nível de aplicação que são de uso comum para as diferentes aplicações. Dependendo da tarefa de sensoriamento, diferentes tipos de software de aplicação podem ser construídos e usados para interagir com a RSSF. Entretanto, apesar das diversas aplicações potencias para RSSFs, Akyildiz et al. [3] afirmam que os protocolos para a camada de aplicação ainda são pouco explorados. Os autores sugerem três possíveis protocolos para apoio às aplicações: (i) protocolo de gerenciamento de sensores (SMP); (ii) protocolo de anúncio de dados e designação de tarefas (TADAP); e (iii) protocolo de consulta e disseminação de dados (SQDDP).

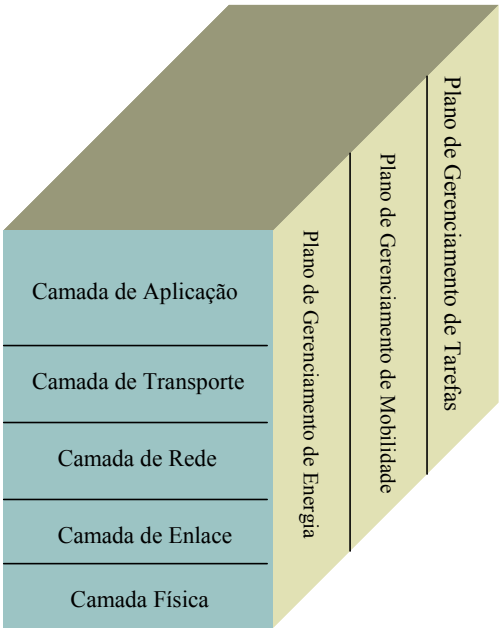


Figura 2: A pilha de Protocolos das RSSFs

O ***protocolo de gerenciamento de sensores (SMP)*** tem por finalidade tornar o hardware e o software das camadas mais baixas transparentes para as aplicações que gerenciam a rede de sensores. O administrador de sistemas interage com a RSSF usando o SMP. Diferente de outras redes, RSSFs são geralmente sem infra-estrutura (*ad hocs*) e compostas por nós que não possuem identificadores globalmente únicos, como endereços IP. Portanto, o SMP precisa acessar os nós usando endereçamento baseado em atributos desses nós, como sua localização ou o tipo de dado fornecido. O SMP fornece as operações necessárias para realizar as seguintes tarefas administrativas [3]:

- introduzir as regras relacionadas à agregação de dados, à nomeação baseada em atributos e à formação de grupos (*clusters)*;
- realizar a troca de dados relacionados a algoritmos de descoberta de localização;
- realizar a sincronização temporal dos sensores;
- gerenciar o movimento dos sensores;
- ligar e desligar os nós;
- consultar a configuração da RSSF e o estado dos nós e reconfigurar a rede, quando necessário; e

- realizar autenticação, distribuição de chaves e segurança na comunicação dos dados.

Outra operação importante em RSSFs é a submissão de consultas, representando interesses da aplicação, e a disseminação desses interesses na rede. Usuários enviam seu interesse para um nó sensor, para um subconjunto dos nós ou para a rede inteira. Esse interesse pode ser descrito em função de atributos do fenômeno monitorado, ou pode descrever um evento de disparo a partir do qual a aquisição de dados deve ter início. O ***protocolo de consulta e disseminação de dados (SQDDP)*** fornece, para as aplicações, interfaces para a emissão de suas consultas e para a coleta das respostas que chegam da rede. Uma abordagem alternativa à disseminação de interesses é o anúncio de dados disponíveis. Nessa abordagem, os nós sensores anunciam os dados disponíveis para as aplicações e elas, então, consultam os dados nos quais estão interessadas. O ***protocolo de anúncio de dados e designação de tarefas (TADAP)*** fornece à aplicação cliente da rede interfaces para conduzir a disseminação de seus interesses ou para receber anúncios de dados enviados pelos sensores. Além disso, o TADAP realiza a atribuição das tarefas aos sensores a partir dos interesses recebidos.

Apesar dos protocolos de gerenciamento de sensores (SMP), de consulta e disseminação de interesses (SQDDP), e de anúncio de dados e designação de tarefas (TADAP) serem sugeridos como protocolos de camada de aplicação, na verdade eles desempenham papéis de gerência e de apoio à construção de aplicações no topo da infra-estrutura da rede. Em sistemas distribuídos tradicionais, tais funcionalidades são tipicamente atribuídas a sistemas de middleware, que se situam entre o componente de aplicação e os componentes subjacentes. Essa abordagem de projeto também pode ser adotada no projeto de RSSFs, como será discutido posteriormente.

A camada de transporte é responsável por manter o fluxo de dados entre a origem e o destino, se a aplicação assim necessitar. Exemplos de protocolos da camada de transporte podem ser vistos em [7,117,142,143].

A camada de rede trata do roteamento dos dados. A Seção 2.2 aborda detalhadamente a questão do roteamento em RSSFs, exemplificando alguns dos protocolos de camada de rede existentes.

O objetivo da camada de enlace é assegurar conexões confiáveis em uma rede de comunicação. Tal objetivo é alcançado através das tarefas de multiplexação dos fluxos de dados, detecção dos quadros, acesso ao meio e controle de erros. Embora muitas dessas funções ainda não tenham sido completamente resolvidas no contexto de RSSFs, o protocolo de acesso ao meio (*Medium Access Control* – MAC) desempenha uma função crítica nessas redes. Como o ambiente é ruidoso e os nós sensores podem ser móveis, o protocolo MAC tem que ser capaz de minimizar as colisões entre vizinhos. Existem vários protocolos MAC para redes sem fio. Entretanto, tais protocolos não levam em conta as restrições e características específicas das RSSFs, principalmente seus recursos de energia extremamente limitados. Em Ye et al. [157], são apresentados os requisitos de protocolos MAC para RSSF e são identificadas três principais causas de desperdícios de energia nessa camada: colisões, recebimento de quadros endereçados a outros destinatários e espera ociosa. Os autores propõem um protocolo que procura minimizar tais problemas. Outros exemplos de protocolos de camada MAC específicos para RSSFs podem ser vistos em [25,32,72,83,100,115,123,128].

A camada física abrange as técnicas de transmissão, recepção e modulação utilizadas na rede, as quais devem ser simples, porém robustas. Segundo o modelo de propagação de rádio de primeira ordem apresentado em [60], a comunicação entre os nós (transmissão e recepção) é responsável pela maior parte do consumo de energia em RSSFs. Portanto, a eficiência em termos de energia assume uma importância significativa no projeto da camada física, afetando as decisões sobre a seleção da freqüência e da portadora, bem como dos mecanismos de detecção de sinal e modulação utilizados [107].

Em adição às camadas de protocolos horizontais apresentadas, os planos de gerenciamento de energia, de mobilidade e de tarefas monitoram a energia, os movimentos dos sensores e a distribuição de tarefas entre os nós, respectivamente. Esses planos permitem coordenar os nós sensores na realização das tarefas de sensoriamento e, ao mesmo tempo, controlar o consumo global de energia da rede, de forma a minimizá-lo tanto quanto possível.

O plano de gerenciamento de energia permite gerenciar a forma como os nós sensores utilizam sua energia. Por exemplo, um nó sensor pode desligar seu receptor após receber uma mensagem de um de seus vizinhos, evitando a recepção de mensagens duplicadas [3]. Um sensor com baixo nível de energia pode decidir não participar mais do

roteamento de mensagens e reservar o restante de sua energia para o sensoriamento. Com esse intuito, o sensor deve avisar seus vizinhos de sua decisão, enviando-lhes essa informação em *broadcast*. Todas essas decisões são responsabilidades do plano de gerenciamento de energia.

O plano de gerenciamento de mobilidade permite detectar e registrar o movimento dos nós sensores, de modo a manter sempre uma rota para o usuário e a guardar, para cada nó, a informação sobre quem são seus nós vizinhos.

O plano de gerenciamento de tarefas permite balancear e escalonar as tarefas de sensoriamento de uma região específica. Não é necessário que todos os sensores localizados na região alvo definida pela aplicação estejam ativos ou contribuam ao mesmo tempo para a realização da tarefa. Como resultado, alguns nós participam mais na realização da tarefa do que outros, dependendo do seu nível de energia ou do grau com que podem contribuir para a tarefa.

Em síntese, os planos de gerenciamento são necessários para que os nós sensores possam trabalhar juntos de forma eficiente em termos de energia, rotear dados em redes com sensores móveis e compartilhar recursos entre si. Sem eles, cada sensor iria trabalhar apenas de forma individual. Do ponto de vista da RSSF, é mais eficiente que os nós possam colaborar mutuamente, de modo que o tempo de vida global da rede possa ser prolongado.

2.1.3 Aplicações

Nós sensores podem ser usados para o sensoriamento contínuo de dados, para a detecção de eventos e para o controle local de atuadores, além de outras possíveis tarefas. Desse modo, o uso de RSSFs habilita uma ampla gama de aplicações potenciais, pertencentes a diversos ramos do conhecimento humano.

A aplicação é o componente de uma arquitetura de RSSF responsável por emitir um conjunto de consultas ou interesses, que descrevem as características dos fenômenos físicos que o usuário deseja analisar. Os interesses da aplicação devem indicar os tipos de dados desejados; a freqüência com que esses dados devem ser coletados; a necessidade ou não dos dados sofrerem algum tipo de agregação; os requisitos de QoS, como valores de atraso máximo ou acurácia mínima desejados; os limiares a partir dos quais os dados

devem ser transmitidos; ou ainda, eventos que podem disparar algum comportamento particular da rede, como a ativação de sensores específicos ou a alteração na taxa de sensoriamento.

As aplicações potenciais para RSSFs podem ser categorizadas nas seguintes áreas: militar, ambiental, de saúde, doméstica e comercial. É possível expandir essa classificação com categorias adicionais, como a exploração espacial, o processamento químico e o socorro a desastres [3]. A seguir são exemplificadas algumas dessas aplicações.

- ***Aplicações Militares*** - A rápida instalação e as características de auto-organização e tolerância a falhas das RSSFs fazem com que elas sejam uma ferramenta de sensoriamento bastante promissora para integrar sistemas militares de comando, controle, comunicação, computação, inteligência, vigilância, reconhecimento e mira (C4ISRT) [3]. Como as RSSFs são baseadas na densa instalação de nós de baixo custo e descartáveis, a destruição de alguns nós por ações inimigas não afeta uma operação militar tanto quanto a destruição de um sensor tradicional, de maior custo. Tal característica torna as RSSFs bastante adequadas para uso em campos de batalha. Algumas das aplicações militares de RSSFs são o monitoramento de forças, equipamentos e munições amigos; a vigilância de campo de batalha; o reconhecimento de forças e terrenos inimigos; a avaliação de danos de batalhas; a detecção e o reconhecimento de ataques químicos, biológicos e nucleares. Além disso, sensores podem ser incorporados a sistemas inteligentes de mira.

- ***Aplicações Ambientais*** - Aplicações ambientais de RSSFs incluem o rastreio de movimentos de pássaros, insetos e outros pequenos animais; o monitoramento das condições ambientais que afetam a colheita e o gado; o controle de irrigação; o monitoramento em grande escala da Terra e a exploração planetária; a detecção química e biológica; a agricultura de precisão; o monitoramento biológico e ambiental de águas, solos e da atmosfera; a detecção de incêndio em florestas; a pesquisa meteorológica e geofísica; a detecção de inundação; o mapeamento de bio-complexidade do ambiente, e o estudo da poluição [1,2,10,11,16,20,41,145].

- ***Aplicações de Saúde*** - Exemplos de aplicações de saúde para RSSFs são o monitoramento de pacientes; a realização de diagnósticos; a administração de

drogas em hospitais; o telemonitoramento de dados fisiológicos humanos; e o monitoramento de médicos e pacientes dentro de um hospital [96].

- ***Aplicações Domésticas*** - Nós sensores e atuadores inteligentes podem ser embutidos em eletrodomésticos, tais como aspiradores de pó, fornos de micro-ondas, geladeiras e vídeos-cassetes [49]. Esses nós podem interagir entre si e com redes externas via Internet ou satélite, permitindo que usuários gerenciem dispositivos domésticos local ou remotamente de forma fácil e integrada. Sensores sem fio também podem ser utilizados no projeto de Ambientes Inteligentes, como descrito em [62]. Nesses ambientes, os nós sensores podem ser embutidos em móveis e eletrodomésticos e comunicar-se uns com os outros e com um nó servidor do aposento. O servidor do aposento pode se comunicar com outros servidores de outros aposentos e aprender sobre os serviços que eles oferecem, por exemplo, impressão ou fax. Esses servidores e os nós sensores podem ser integrados com dispositivos embutidos existentes e tornarem-se sistemas auto-organizáveis e adaptativos baseados em modelos de teoria de controle [62].

- ***Aplicações Comerciais*** - Algumas das aplicações comerciais de RSSFs são: monitoramento de fadiga de material; construção de teclados virtuais; gerenciamento de estoque de fábricas; monitoramento de qualidade de produtos; construção de escritórios inteligentes; controle ambiental de edifícios inteligentes; controle de robôs; brinquedos interativos; museus interativos; automação e controle de processos fabris; monitoramento de áreas de desastres; diagnósticos de máquinas; detecção e monitoramento de roubos de automóveis; e detecção de veículos [42,109,112,145].

2.1.4 Modelos de Comunicação e Entrega de Dados em RSSFs

Geralmente, uma RSSF suporta dois tipos de comunicação [131]: comunicação de infra-estrutura e comunicação da aplicação.

A comunicação de infra-estrutura é responsável pela troca de mensagens entre os nós necessária para configurar, manter e otimizar a operação da rede. Nesse tipo de comunicação estão incluídas, por exemplo, as trocas de mensagens para descobrir os caminhos dos sensores aos nós sorvedouros, e as mensagens usadas para a formação de *clusters* e para a eleição de líderes. A comunicação de infra-estrutura é gerada pelos

protocolos de rede e de enlace em resposta aos requisitos da aplicação ou à ocorrência de eventos na rede.

A comunicação de aplicação é responsável pela transferência dos interesses da aplicação para a rede, a partir da qual a entrega de dados tem início, e pela transferência dos dados coletados sobre o fenômeno em estudo, desde sua origem até a aplicação destino.

Idealmente, o interesse da aplicação deve ser especificado em termos do fenômeno em estudo, evitando que o usuário tenha que tomar conhecimento quanto à infra-estrutura e aos protocolos de comunicação subjacentes. A adoção de um esquema de nomeação centrado em dados [57] permite a descrição de interesses na forma de conjuntos de atributos de baixo nível, tais como o tipo do nó sensor, a área geográfica que se deseja monitorar, e o intervalo desejado para coletar os dados.

A partir do interesse definido pela aplicação, a rede começa a coletar e a enviar dados. O modelo de entrega de dados utilizado pela rede irá determinar o tipo de tráfego da aplicação. Em [131], RSSFs são classificadas quanto ao modelo de entrega requerido pela aplicação, em quatro tipos: contínuas, dirigidas a eventos, iniciadas pelo observador e híbridas. No modelo contínuo, ou proativo, sensores comunicam seus dados continuamente a uma taxa pré-definida. No modelo dirigido a eventos, ou reativo, os sensores reportam informações somente se um evento de interesse ocorrer. Nesse caso, a aplicação está interessada apenas na ocorrência de um fenômeno específico. No modelo iniciado pelo observador (ou *request-reply*), os sensores reportam seus resultados em resposta a um pedido explícito (síncrono) da aplicação. A aplicação está interessada em obter uma visão instantânea (*snapshot*) do fenômeno monitorado. Finalmente, as três abordagens podem coexistir na mesma rede, gerando um modelo híbrido de entrega. O modelo da entrega de dados conduz a escolha do melhor tipo de protocolo de disseminação de dados a ser adotado na rede. A seguir serão descritos exemplos de protocolos de disseminação de dados para RSSFs.

2.2 PROTOCOLOS DE DISSEMINAÇÃO DE DADOS

Nos últimos anos, um grande número de protocolos de disseminação de dados especificamente projetados para RSSFs foram propostos. A maior parte desses protocolos baseia-se em algoritmos localizados [86,110] e na comunicação centrada em dados

[57,74], além de tirar proveito de conhecimento específico da aplicação na disseminação dos dados. Algoritmos localizados são um tipo especial de algoritmos distribuídos nos quais os nós participantes de um processamento distribuído interagem apenas com nós localizados dentro de uma vizinhança restrita, porém o sistema atinge uma meta global. Tais algoritmos têm boa escalabilidade e são robustos a partições da rede e a falhas nos nós [110]. Comunicação centrada em dados introduz um novo estilo de endereçamento no qual os nós são endereçados pelos atributos dos dados que eles geram (tipo de sensor) e por sua localização geográfica, em vez de serem endereçados pela sua localização topológica na rede (endereços IP) [57].

Há diversas classificações possíveis para os protocolos de disseminação de dados, cada uma enfatizando um aspecto de seu comportamento. Quanto à origem de sua inicialização, alguns protocolos são iniciados pelo emissor, ou seja, pelas fontes de dados [75], enquanto outros são iniciados pelo receptor, ou seja, pelos nós sorvedouros [66]. Quanto à topologia lógica da rede, a qual influencia diretamente a estratégia com que os nós disseminam os dados, os protocolos podem ser classificados em (i) baseados em comunicação direta; (ii) planos; ou (iii) hierárquicos [105]. Nos protocolos baseados na comunicação direta, cada nó sensor (fonte) envia seus dados diretamente para o sorvedouro. Com esses protocolos, se o diâmetro da rede é grande, a energia dos nós será drenada muito rapidamente [70]. Além disso, conforme o número de sensores aumenta, as colisões também aumentam, prejudicando a qualidade das transmissões. Em protocolos planos e hierárquicos tais problemas são minimizados. Portanto, eles são mais eficientes, principalmente considerando a larga escala da maioria das redes de sensores sem fio.

2.2.1 Protocolos Planos

Nos protocolos planos, todos os nós sensores da rede são considerados "iguais" em termos de roteamento. Quando um nó tem dados para enviar, ele deve encontrar uma rota, a qual consiste em múltiplos saltos (nós intermediários) até o sorvedouro. Um problema desses protocolos é que, normalmente, a probabilidade de participar do processo de transmissão de dados é mais alta para os nós ao redor do sorvedouro do que para os nós distantes do mesmo. Portanto, nós próximos ao sorvedouro tendem a ter sua energia esgotada mais rapidamente, o que pode limitar o tempo de vida global da rede, já que impede a entrega dos dados para a aplicação. A seguir são apresentados exemplos de protocolos planos de disseminação de dados em RSSFs.

2.2.1.1 Difusão Direcionada

A Difusão Direcionada (DD) [66] é um protocolo eficiente para a disseminação de dados em redes de sensores com topologias planas. Nesse protocolo, cada nó sensor individualmente transforma o sinal analógico gerado por um alvo detectado em uma descrição relativamente grosseira de um "evento". Tal descrição contém um conjunto de atributos. Aplicações que desejam obter dados da RSSF enviam interesses através de um ou mais nós sorvedouros. Tais interesses também são representados como conjuntos de atributos e são distribuídos na rede através de um método de inundação. Se os atributos dos dados gerados por nós sensores fontes combinam com os atributos de um interesse, um gradiente é configurado dentro da rede, o qual consiste em informação de estado indicando a direção (próximo salto) de outros nós interessados nos dados. Os dados são conduzidos através desse gradiente até os nós sorvedouros. A Difusão Direcionada é essencialmente um protocolo de disseminação de dados reativo, iniciado pelo receptor, orientado a interesses e baseado no uso de interações localizadas. Todas as decisões do protocolo baseiam-se em informações locais aos nós ou de seus vizinhos (não requer conhecimento global). Nós intermediários são capazes de realizar, além do sensoriamento, o *caching* e a agregação de dados.

O protocolo DD adota uma API declarativa para funções de publicação e subscrição (*publish/subscribe*) que isola os produtores e consumidores de dados dos detalhes dos algoritmos de disseminação subjacentes. A abstração chave dessa API é que os dados são identificados por um conjunto de atributos. Os produtores (ou fontes) *publicam* os dados que geram, e os consumidores (sorvedouros) *registram* seus interesses pelos dados. É função da implementação do protocolo DD garantir que os dados trafeguem dos produtores para os consumidores de forma eficiente. O DD encoraja as aplicações a influenciarem o fluxo de dados através do uso de filtros [58] e do processamento dentro da rede. Entretanto, muitas aplicações requerem apenas o envio dos dados selecionados pelos seus atributos, e permitem que o protocolo controle totalmente o roteamento.

Diferentes algoritmos podem ser usados para combinar produtores e consumidores, sem que seja necessária qualquer mudança na semântica ou na API de alto nível fornecidas pelo DD. Na primeira versão do protocolo DD, posteriormente denominada *two-phase pull* [66], consumidores de dados buscam ativamente por fontes de dados, e a seguir as fontes encontram o melhor caminho de volta até os consumidores. O protocolo

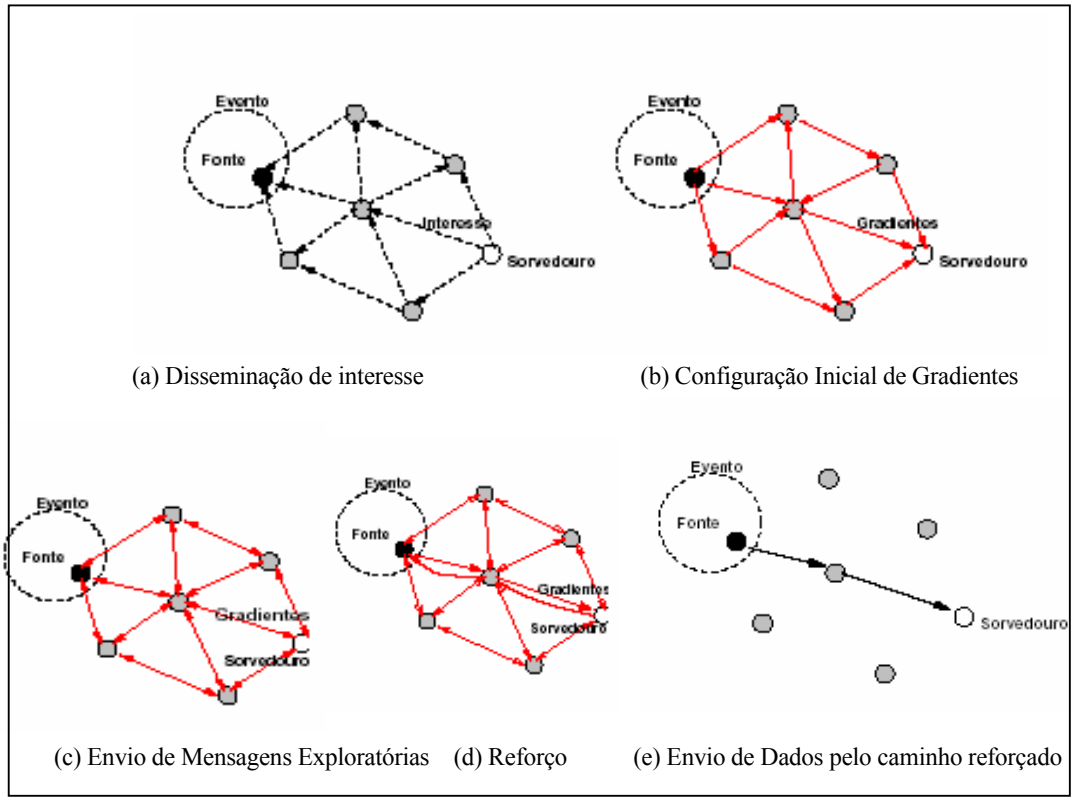
também fornece uma variação com roteamento geográfico e consciência de energia chamada GEAR (*Geographic and Energy Aware Routing Protocol* [159]). GEAR otimiza o processo de encontrar fontes de dados usando informação geográfica para restringir a área de busca. Dessa forma, ele melhora o desempenho do protocolo, evitando que mensagens de interesse inundem a rede inteira.

Embora a versão *two-phase pull* do protocolo tenha se mostrado eficiente para vários cenários, a crescente experiência de seus autores mostrou que essa versão comporta-se de forma ineficiente para alguns tipos de aplicações. Algumas aplicações possuem muitas fontes de dados e muitos nós sorvedouros fazendo subscrições cruzadas entre si, uma situação que resulta em uma grande quantidade de tráfego de controle, mesmo com o apoio de informação geográfica. Para tratar desse problema, em [56] é introduzido um algoritmo chamado *push diffusion*, que reverte o papel dos consumidores e produtores de dados, fazendo com que as fontes de dados busquem ativamente consumidores. Uma vantagem do modo *push* do protocolo DD é que ele requer somente uma fase na qual o tráfego de controle precisa ser disseminado na rede para encontrar sorvedouros, em contraste com o modo *two-phase pull,* que requer duas fases. A constatação da vantagem de se ter apenas uma fase de inundação de mensagens de controle motivou os autores a desenvolverem *one-phase pull* [56], uma terceira variação do algoritmo DD. O *one-phase pull* simplifica o *two-phase pull* eliminando uma fase do protocolo.

A seguir são descritos com maiores detalhes os três modos de operação do protocolo DD.

Two phase pull diffusion. Nesse modo de operação do DD, aplicações clientes executam a função de subscrição de interesses em nós sorvedouros. Na subscrição, o nó sorvedouro propaga pela rede, através de uma mensagem de interesse, os atributos que descrevem o interesse (Figura 3 (a)). A princípio, a distribuição de mensagens de interesse pode ser otimizada pelo uso de filtros específicos da aplicação ou de informações obtidas em execuções prévias do protocolo e mantidas em *cache*, como, por exemplo, informação geográfica quando se usa o GEAR. Sem essas informações de restrição, interesses têm que ser inundados na rede a fim de encontrar as fontes de dados existentes. Conforme os interesses são disseminados pelos nós, estes estabelecem os gradientes a serem usados na entrega dos dados (Figura 3 (b)). Quando um interesse chega em um nó fonte, tal nó

começa a produzir dados. Para conservar energia, os nós devem evitar produzir dados antes de receber interesses, ou podem produzir dados e armazená-los localmente. A primeira mensagem de dados enviada por um nó fonte é marcada como exploratória e enviada para todos os vizinhos que possuem gradientes configurados para aquele dado (Figura 3 (c)). Da mesma forma que mensagens de interesse, a transferência de mensagens de dados exploratórios pode ser otimizada usando informação contida no *cache* dos nós ou filtros específicos da aplicação. Mas, por *default,* mensagens exploratórias são enviadas para todos os nós, ou seja, são inundadas na rede. Quando mensagens de dados exploratórios alcançam o nó sorvedouro, o sorvedouro reforça seu vizinho preferencial, estabelecendo um gradiente reforçado em direção a ele (Figura 3 (d)). Na versão atual do protocolo, o vizinho preferencial é aquele de menor latência. Entretanto, a preferência pode ser modificada para incluir outros parâmetros, como a qualidade do enlace ou a energia residual do nó. O vizinho reforçado, por sua vez, reforça seu vizinho preferencial, e assim sucessivamente ao longo de todo o caminho de volta até a fonte ou fontes de dados. Tal procedimento resulta em uma cadeia de gradientes reforçados de todas as fontes para todos os sorvedouros.



(a) Disseminação de interesse (b) Configuração Inicial de Gradientes

(c) Envio de Mensagens Exploratórias (d) Reforço (e) Envio de Dados pelo caminho reforçado

Figura 3: Funcionamento básico do protocolo Difusão Direcionada

Mensagens de dados subseqüentes não são marcadas como exploratórias, e são enviadas somente por gradientes reforçados, em vez de o serem para todos os vizinhos (Figura 3 (e)). Gradientes são gerenciados como *soft-state*, portanto mensagens de interesse e de dados exploratórios são geradas e enviadas periodicamente para refrescar esse estado.

O modo *two-phase pull* funciona bem para aplicações onde há um pequeno número de nós sorvedouros. Um exemplo desse tipo de aplicação é um usuário consultando a rede sobre a detecção de algum objeto. Entretanto, para outros tipos de aplicações, esse modo pode não ser eficiente, como será visto a seguir.

Push diffusion. Algumas aplicações para RSSFs envolvem a comunicação sensor-sensor dentro da rede. Um exemplo desse tipo de aplicação consiste em ter sensores operando com uma carga leve de trabalho a maior parte do tempo. Entretanto, quando um sensor detecta algum evento, ele avisa os sensores vizinhos para se tornarem mais ativos e vigilantes (mudando seu modo de operação ou ciclo de trabalho). Uma característica desse tipo de aplicação é que há muitos nós interessados nos dados (os quais consistem em "gatilhos de ativação"), e muitos nós que podem publicar tais dados, mas a freqüência com que os dados realmente são enviados é muito rara. O modo *two-phase pull diffusion* comporta-se mal para esse tipo de aplicação, porque todos os sensores ativamente enviam interesses e mantêm gradientes para todos os outros sensores, mesmo quando nenhum evento é detectado.

Push diffusion foi projetado para esse tipo de aplicação. Nele, sorvedouros tornam-se passivos, com a informação sobre os interesses (subscrições) sendo mantida na memória local do nó. Nós fontes tornam-se ativos, com dados exploratórios sendo enviados através da rede sem a necessidade de gradientes criados pelo envio de interesses. Como ocorre com o *two-phase pull*, quando dados exploratórios chegam em um sorvedouro, uma mensagem de reforço é gerada e recursivamente passada de volta para a origem (fonte), criando um gradiente reforçado, e dados não exploratórios seguem somente esses gradientes reforçados.

O modo *push* é, portanto, otimizado para aplicações com muitas fontes e sorvedouros, mas onde as fontes produzem dados apenas ocasionalmente. *Push* não é uma

boa opção para aplicações com muitas fontes gerando dados continuamente, já que tais dados seriam enviados pela rede mesmo quando não fossem necessários.

One phase pull diffusion. Em redes grandes e sem o auxílio de informação geográfica, minimizar a inundação de mensagens pode trazer um benefício significativo. Com esse intuito, o *two-phase pull* foi alterado de forma a eliminar-se uma de suas fases de inundação, criando-se o modo *one-phase pull*. Como ocorre no *two-phase pull*, sorvedouros enviam mensagens de interesse que são disseminadas na rede, estabelecendo gradientes. Entretanto, quando um interesse chega em um nó fonte, ele não marca sua primeira mensagem de dado como exploratória, mas, em vez disso, envia dados somente pelo gradiente preferencial. O gradiente preferencial é determinado pelo vizinho que foi o primeiro a enviar o respectivo interesse, dessa forma sugerindo o caminho de menor latência. Portanto, *one-phase pull* não requer mensagens de reforço; o caminho de menor latência é implicitamente reforçado.

O modo *one-phase pull* possui duas desvantagens se comparado com o *two-phase pull*. A primeira desvantagem é que o *one-phase pull* assume que a comunicação sem fio é simétrica entre os nós, já que o caminho de dados (fonte-sorvedouro) é determinado pela menor latência no caminho do interesse (sorvedouro-fonte). O modo *two-phase pull* reduz o problema de assumir comunicação simétrica, já que o caminho de dados é determinado pelas mensagens exploratórias, enviadas na direção fonte-sorvedouro. Entretanto, *two-phase pull* ainda requer algum nível de simetria, já que as mensagens de reforço viajam através de enlaces reversos. Embora assimetria de enlaces seja um problema sério em redes sem fio, muitos outros protocolos requerem enlaces simétricos, incluindo o 802.11 [65] e protocolos que usam confirmação no nível de enlace. Os autores em [56] assumem que a camada MAC permitirá ao DD identificar enlaces assimétricos.

A segunda desvantagem é que o *one-phase pull* requer que mensagens de interesse transportem um identificador de fluxo (*flow-id*). Embora a geração de *flow-id* seja relativamente simples, esse requisito faz com que o **tamanho** da mensagem de interesse cresça com o número de sorvedouros. Porém, com o *two-phase pull* o **número** de mensagens de interesse cresce proporcionalmente com o número de sorvedouros. Um problema adicional é que o uso de *flow-ids* fim-a-fim significa que *one-phase pull* não usa apenas informação local nas decisões de disseminação dos dados, contrariando o paradigma original do DD.

2.2.1.2 Outros Protocolos Planos

Em [15] é descrito um método para rotear consultas (descrição e eventos de interesse) para nós que observaram um evento específico, em uma RSSF. A idéia básica é inicialmente criar caminhos que chegam em cada nó que observa um evento, isto é, criar "caminhos do evento". Dessa forma, quando uma consulta é gerada, ela é enviada por um caminho aleatório até encontrar o caminho do evento específico, em vez de ser inundada por toda a rede. Assim que é descoberto o caminho do evento para uma determinada consulta, a consulta pode ser roteada diretamente para o nó que observa o evento. Se o caminho não puder ser encontrado, a aplicação pode tentar submeter novamente a consulta ou, como último recurso, inundá-la. Os resultados obtidos em [15] mostram que, sob uma ampla gama de condições, é possível obter uma taxa de entrega de dados muito alta e a distribuição de consultas por inundação é uma ocorrência rara.

Os autores em [79] propõem modificações no protocolo hierárquico LEACH [59] visando aumentar seu desempenho em termos de consumo de energia. O trabalho apresenta PEGASIS, um protocolo que pressupõe topologia plana da rede e obtém resultados próximos ao ótimo (em termos de energia) na disseminação de dados em RSSFs. A idéia chave no PEGASIS é formar uma cadeia entre os nós sensores, de modo que cada nó recebe de e transmite para um vizinho próximo. A cada rodada do protocolo, os dados coletados movem-se de nó a nó, são agregados, e eventualmente um nó designado transmite-os para o sorvedouro. Nós se revezam na transmissão para o sorvedouro, de modo que o gasto médio de energia por nó, por rodada, seja reduzido. Construir uma cadeia para minimizar o tamanho total da rota é similar ao problema do caixeiro viajante, que é um problema de otimização «NP-completo», o que significa que, para um problema de dimensão razoável, como é o caso do número de nós em uma RSSF, há tantas hipóteses a considerar que procurar uma solução ótima seria demasiado custoso computacionalmente. Entretanto, os parâmetros da comunicação via rádio restringem o problema, e uma cadeia simples pode ser construída usando o algoritmo guloso. PEGASIS obtém entre 100 e 300% de economia de energia em comparação com o LEACH, para aplicações com modelo de entrega de dados contínuo.

O Protocolo EAR (*Energy Aware Routing* [121]) busca garantir a sobrevivência de redes de sensores com energia restrita. EAR também é um protocolo reativo, como a Difusão Direcionada, porém não se baseia na configuração de um único caminho ótimo

entre fontes e sorvedouros. Em vez disso, ele mantém um conjunto de "bons" caminhos e escolhe um baseado em uma probabilidade. Assim, a comunicação entre nós é efetuada usando diferentes caminhos em momentos diferentes, de forma a não esgotar a energia ao longo de um mesmo caminho. Esse protocolo tem resposta rápida ao movimento de nós entrando e saindo na rede, e tem um *overhead* mínimo de roteamento. Resultados de simulação apresentados nesse trabalho mostram uma melhoria no tempo de vida da rede de até 40% comparado a esquemas como a Difusão Direcionada. Os nós também consomem energia de uma forma mais eqüitativa ao longo da rede, garantindo uma degradação do serviço fornecido mais suave ao longo do tempo.

2.2.2 Protocolos Hierárquicos

Para redes de grande escala, adotar uma topologia hierárquica, agrupando os nós em *clusters* pode ser benéfico por várias razões [133]. Sob a perspectiva de roteamento, a clusterização permite que protocolos de rede operem em um modo hierárquico, quebrando as transmissões em vários níveis diferentes. Tal abordagem, além de ter maior escalabilidade, é mais tolerante a falhas, fornecendo melhor isolamento e recuperação de problemas na rede. A clusterização também pode ser benéfica para algoritmos de coleta de dados. Algumas aplicações não requerem que os dados sejam coletados por todos os nós durante todo o tempo. Membros de *clusters* podem colaborar quanto a medições recentes de dados e determinar a quantidade de informação que deve ser transmitida para a aplicação. Calculando a média de valores de dados coletados dentro do *cluster*, o algoritmo pode negociar precisão dos dados por energia de transmissão. Finalmente, a clusterização pode ajudar a lidar com distribuição não ideal de sensores na rede. Em áreas onde há um número redundante de sensores, um algoritmo de clusterização pode ser usado para selecionar quais nós representam melhores amostras de dados para a região e quais nós podem ser colocados em um modo de baixo consumo de energia, de modo a economizar energia e aumentar o tempo de vida da rede.

Em [59] e [60], é proposto o LEACH (*Low-Energy Adaptive Clustering Hierarchy*), um protocolo baseado em *clusters* que minimiza a dissipação de energia em RSSFs. LEACH é um protocolo adequado para aplicações proativas que assumem que a RSSF sempre tem dados para enviar. Nele, o processo de formação dos *clusters* é distribuído entre os sensores. Qualquer sensor pode se tornar um líder de cluster (*cluster head*) com probabilidade X. O líder de um cluster gera um esquema TDMA e transmite-o em

broadcast para todos os sensores pertencentes ao cluster. Cada sensor transmite seus dados para o respectivo líder apenas no *slot* de tempo que lhe foi alocado. Nos demais *slots*, os sensores podem permanecer em um modo de baixo consumo de energia. Os líderes são responsáveis por reunir os dados enviados pelos membros do cluster, aplicar um mecanismo de compressão nesses dados (para reduzir a comunicação global na rede) e enviar o resultado para o nó sorvedouro. Como nós líderes de *clusters* consomem mais energia que os demais, o LEACH realiza a rotação periódica de líderes a fim de distribuir de modo uniforme a utilização de energia entre os sensores na rede. Para isso, após algum tempo, cada sensor entra novamente na fase de formação de *clusters* e o ciclo é repetido até o completo esgotamento da energia dos sensores na rede.

O protocolo TTDD [156] adota uma hierarquia de dois níveis para disseminar dados em uma RSSF. Ele usa uma estrutura de grade, de modo que apenas sensores localizados em pontos de intersecção da grade precisam adquirir informação de encaminhamento de dados. Cada fonte de dados proativamente constrói a grade e configura a informação de encaminhamento nos sensores mais próximos aos pontos da grade (chamados nós de disseminação). Os nós de disseminação são análogos a líderes de *clusters*. Uma limitação do TTDD é o *overhead* para manutenção das grades. Grades são construídas periodicamente para cada fonte de dados na rede, mesmo se não houver interesses ativos. Portanto, TTDD não é adequado para ambientes de monitoramento onde poucos eventos ocorrem com alta freqüência mas a aplicação deseja receber dados apenas esporadicamente.

2.3 QUALIDADE DE SERVIÇO (QOS) EM RSSFS

Há uma ampla variedade de definições possíveis para o conceito de QoS em redes de sensores [47,67,101,103]. Em [47] QoS em uma RSSF é definida em termos da resolução espacial da rede. Segundo essa definição, há um número ótimo de sensores que devem estar ativos na rede, dependendo do estímulo ambiental presente a cada momento. Em [67] QoS significa fornecer confiabilidade de dados para a aplicação ao mesmo tempo em que os recursos de energia da rede são consumidos de modo eficiente. Nesse trabalho, os autores consideram que a confiabilidade dos dados está ligada ao número de sensores ativos na rede. Ambas as definições de QoS citadas estão bastante relacionadas à cobertura de sensoriamento, considerada em inúmeros trabalhos como o principal requisito de QoS em RSSFs. Outros trabalhos [160] consideram a latência como um

requisito crucial para várias aplicações. Outros ainda argumentam que, para alguns tipos de aplicações, o tempo de vida da rede é um requisito crítico.

De fato, o tipo da aplicação alvo influencia enormemente nos requisitos de QoS desejados. Nesse sentido, em [47] aplicações de RSSFs são categorizadas em duas classes: dirigidas a desempenho e dirigidas a custo. A primeira classe consiste basicamente em aplicações de detecção distribuída de eventos e caracteriza-se principalmente pelas restrições de energia e largura de banda, além de demandar dados em tempo real, ou seja, a latência é um requisito de QoS crítico. Exemplos típicos dessa classe são aplicações de vigilância militar [6].

Já a segunda classe caracteriza-se por demandar da rede amostragem espaço-temporal dos dados e por possuir requisitos de latência menos severos. O principal exemplo da segunda classe são as aplicações de monitoramento ambiental [84]. Aplicações de monitoramento ambiental não possuem requisitos rígidos de latência. Para essas aplicações, a maior restrição da rede é de energia e os requisitos mais importantes são o tempo de vida da rede e a acurácia do dado.

É importante definir o conceito de acurácia para aplicações de RSSFs. Em RSSFs, diferentemente de sistemas distribuídos tradicionais, há um forte acoplamento com o mundo físico, já que a rede monitora parâmetros de processos físicos. A rede, por sua própria natureza, possui apenas a capacidade de amostrar tais processos físicos, o que por sua vez implica que o resultado obtido é uma aproximação dos parâmetros reais observados. Dito isso, pode-se perceber que os dados em RSSFs possuem uma dimensão extra: a acurácia. Nesse contexto, a acurácia pode, então, ser definida como o grau com que o valor fornecido pela rede aproxima-se do valor "real" do fenômeno físico monitorado. Alguns fatores que contribuem para a acurácia do dado medido por um sensor são a sua precisão nominal e a sua distância em relação ao fenômeno, além do ruído ambiental.

2.4 OTIMIZAÇÃO DE ENERGIA EM RSSFS

Há inúmeros benefícios na instalação de RSSFs para a coleta de dados ambientais. Em primeiro lugar, a instalação da rede é simples, já que os sensores podem ser lançados aleatoriamente na área alvo e sua densidade pode ser variada de acordo com a classe de aplicação a que se destinam. Em segundo lugar, a manutenção do sistema pode ser

bastante facilitada pela capacidade de auto-configuração dos nós. Os nós podem se auto-organizar determinando configurações de topologia lógica e estados de operação que visem a eficiência em termos de energia. Em terceiro lugar, com os atuais avanços tecnológicos, é possível produzir sensores a baixo custo, possivelmente com componentes produzidos por terceiros ("de prateleria"). Além disso, não são necessárias dispendiosas modificações de infra-estrutura após a instalação da RSSF, como ocorre com redes cabeadas. Finalmente, com o tamanho reduzido dos nós, eles podem ser instalados com pouca ou nenhum perturbação para o meio ambiente que pretendem monitorar.

Entretanto, a questão do consumo de energia na rede pode anular grande parte desses benefícios. Se os nós não gastam sua energia de forma otimizada, o sistema perde sua vantagem de instalação arbitrária e sua facilidade de manutenção, já que torna-se necessário localizar e substituir com frequência nós com energia esgotada. Com a necessidade de se efetuar manutenção da rede, a capacidade de não perturbar o meio ambiente também é prejudicada.

Muitas aplicações requerem RSSFs com tempos de vida da ordem de meses a anos. Para atender tal requisito, o consumo de energia, que é o principal fator determinante do tempo de vida de uma rede, deve ser minimizado.

Todos esses fatores demonstram a importância da conservação de energia em RSSFs. Maximizar o tempo de vida de uma RSSF requer o uso de uma metodologia de projeto bem-estruturada e com uma abordagem holística, que capacite o desenvolvimento e a operação conscientes de energia em todos os aspectos da rede, desde a plataforma de hardware até o software da aplicação, incluindo os protocolos de rede. Assim, todas as soluções de hardware e software empregadas na rede devem adotar algum mecanismo para diminuir o consumo de energia, buscando racionalizar o seu uso. O projeto da rede deve incluir mecanismos que permitam negociar dinamicamente consumo de energia, desempenho e fidelidade operacional [113]. Essa abordagem, extremamente focada na eficiência em energia, consiste em um novo paradigma no projeto de redes e impõe vários desafios que precisam ser superados a fim de atingir plenamente o potencial das RSSFs.

A seguir são descritas as abordagens arquiteturais e algorítmicas que os projetistas podem utilizar para aumentar a consciência de energia em RSSFs. São apresentadas técnicas agressivas de otimização de energia em todos os estágios de projeto de uma rede

de sensores, considerando desde o nível do nó sensor individual até o nível da rede como um todo.

2.4.1 Otimizações de Energia no Nó

Esta seção descreve técnicas de otimização de energia que podem ser utilizadas no nível do nó sensor individual. São abordadas medidas para tornar o processamento e o software utilizado pelo nó mais eficientes em energia.

Processamento Consciente de Energia. Avanços no projeto de circuitos e sistemas de baixa potência [9] resultaram no desenvolvimento de vários microprocessadores e microcontroladores de ultrabaixa potência, que consomem pouca energia e podem ser utilizados no projeto do nó sensor. Além disso, o uso de gerência dinâmica de energia (DPM) [9] pode reduzir ainda mais o consumo de energia, aumentando o tempo de vida da bateria. Um esquema de gerência de energia comumente utilizado baseia-se no desligamento de componentes ociosos. Nesse esquema, o nó (ou partes dele) é desligado ou colocado em um estado de baixo consumo de energia nos momentos em que nenhum evento relevante está ocorrendo. A questão central em DPM baseada no desligamento é decidir a política de transição de estados [9], já que diferentes estados são caracterizados por diferentes gastos de energia e as transições gastam uma quantidade de energia não desprezível, além de incorrerem em *overhead* de tempo.

Além das técnicas baseadas no desligamento de componentes ociosos do nó, economia adicional pode ser obtida no estado ativo através do uso de *dynamic voltage scaling* (DVS) [104]. A maioria dos sistemas baseados em microprocessadores possui uma carga computacional variável no tempo e a carga de pico do sistema nem sempre é necessária. A técnica de DVS explora esse fato, adaptando dinamicamente a voltagem fornecida para o processador e a sua freqüência de operação para satisfazer apenas o requisito instantâneo de processamento, dessa forma trocando recursos computacionais não utilizado por economia de energia. Gerenciamento de energia baseado em DVS, quando aplicável, tem mostrado uma eficiência em energia significativamente mais alta em comparação com gerenciamento baseado em desligamento [114].

Software Consciente de Energia. Maior eficiência em energia é obtida através da utilização de plataformas de hardware construídas especificamente para as aplicações. Entretanto, as vantagens da flexibilidade oferecida por microprocessadores e sistemas

baseados em DSP (*Digital Signal Processing*) resultaram no uso crescente de soluções programáveis durante o projeto de sistemas de RSSFs [113].

O tempo de vida da rede pode aumentar significativamente se o software do sistema, incluindo o sistema operacional, a aplicação e a pilha de protocolos da rede, forem todos projetados para serem "conscientes de energia".

O sistema operacional pode implementar políticas de gerenciamento de energia baseadas no desligamento de componentes e em DVS, já que ele possui conhecimento global do funcionamento do nó e dos requisitos de qualidade das aplicações. Portanto, o sistema operacional pode controlar diretamente os recursos de *hardware* subjacentes, fazendo o ajuste fino dos parâmetros de configuração ("*knobs*" do sistema) disponíveis, com o intuito de negociar desempenho-energia. No núcleo do sistema operacional há um escalonador de tarefas, responsável por escalonar um determinado conjunto de tarefas a serem executadas no sistema, enquanto garante que as restrições temporais são satisfeitas. O tempo de vida do sistema pode ser consideravelmente aumentado incorporando consciência de energia no processo de escalonamento de tarefas [114,154].

Algumas RSSFs incluem um sistema de middleware entre os componentes de aplicações e os componentes subjacentes. O sistema de middleware, quando presente, assume as responsabilidades do sistema operacional citadas acima e a gerência de energia passa a ser uma de suas funções.

O compromisso energia-qualidade de dados pode ser explorado projetando-se o componente de aplicação para ser escalável em energia. Com esse intuito, o software do componente de aplicação deve ser modificado de modo que o processamento mais significativo seja realizado primeiro. Dessa forma, terminar o algoritmo prematuramente devido a restrições de energia não causaria impactos severos no seu resultado. Por exemplo, aplicações para rastreamento de alvos envolvem o uso intenso de algoritmos de filtragem de sinais, como o filtro de Kalman [149]. Transformar os algoritmos de filtragem para serem escaláveis em energia significa negociar precisão computacional (e, portanto, precisão do rastreamento) por consumo de energia. Várias transformações de algoritmos DSP para aumentar a escalabilidade em energia são apresentadas em [125].

2.4.2 Comunicação sem Fio Consciente de Energia

Embora o gerenciamento de energia no nível dos nós sensores individuais reduza o consumo de energia na rede, é importante que a comunicação entre os nós também seja conduzida de uma forma eficiente em energia. Uma vez que a transmissão de dados no meio sem fio representa uma grande parcela do consumo total de energia em RSSFs, decisões sobre gerenciamento de energia que levam em conta o efeito da comunicação entre os nós propiciam economias de energia significativamente altas. Para conseguir comunicação consciente de energia é necessário identificar e tirar proveito dos vários compromissos desempenho-energia que existem no subsistema de comunicação.

A tecnologia de rádio específica usada no enlace sem fio entre os nós sensores desempenha um papel importante em questões de energia. A escolha do esquema de modulação influencia muito o compromisso entre consumo de energia, precisão de dados e latência (requisito da aplicação). Técnicas de *modulation scaling* [118] podem ser usadas como parte do gerenciamento de energia do rádio. Tais técnicas consistem basicamente em adaptar dinamicamente o nível de modulação usado na comunicação via rádio, de acordo com a carga de tráfego observada na rede a cada instante.

Ao explorar os compromissos energia-desempenho-qualidade, restrições de confiabilidade de dados também têm que ser consideradas. Decisões quanto à confiabilidade são geralmente tomadas na camada de enlace, já que ela é responsável pela detecção e correção de erros. Esquemas de correção de erros adaptativos são propostos em [77]. Tais esquemas visam reduzir o consumo de energia, enquanto mantêm a especificação de taxa de erros de bits (BER) solicitada pelo usuário. Para um dado requisito de BER, podem-se usar esquemas de controle de erros que reduzem a potência de transmissão necessária para enviar um pacote, às custas de processamento adicional no transmissor e no receptor. O uso desses esquemas é particularmente útil para transmissões de longa distância até os nós sorvedouros, que envolvem alta potência de transmissão.

2.4.3 Otimização Global na Rede

No nível global de uma RSSF surge a questão de como o tráfego é encaminhado das fontes de dados para os sorvedouros. A alta densidade dos nós típicas de RSSFs favorece o estabelecimento da comunicação em múltiplos saltos (*multihop*) através de nós intermediários entre a origem e o destino. Em [48] é apresentado um modelo do consumo

de energia com a comunicação via rádio em função da distância de transmissão. Observou-se que, conforme a distância de transmissão aumenta, torna-se vantajoso, em termos de gasto de energia, aumentar o número de nós intermediários. Entretanto, quando a comunicação envolve distâncias pequenas (menores do que 30m no modelo utilizado), a transmissão direta foi mais eficiente do que o encaminhamento através de múltiplos saltos. Como em RSSFs os nós sorvedouros tipicamente são posicionados em locais distantes da área de monitoramento, em geral a comunicação utilizando nós intermediários em rotas com múltiplos saltos é a solução mais eficiente [107]. Portanto, a disseminação da informação em RSSFs é feita salto a salto (*hop-by-hop*), onde nós que realizam o sensoriamento dos dados repassam seus dados para os nós vizinhos, e assim sucessivamente, até que tais dados alcancem algum nó sorvedouro na rede. Os dados enviados por diferentes nós podem ser agregados em nós intermediários entre a origem e o destino (sorvedouro), a fim de reduzir redundâncias e minimizar o tráfego na rede, diminuindo ainda mais o consumo total de energia.

Por exemplo, considere-se a aplicação de rastreamento de alvos. Devido à alta densidade dos nós, um alvo é detectado não apenas por um único nó, mas por uma nuvem inteira de nós vizinhos, levando a um alto grau de redundância nos dados obtidos. Combinar a informação dos nós nessa nuvem via processamento dentro da rede (*in-network*) pode aumentar a confiabilidade de detecção do alvo e, ao mesmo tempo, reduzir enormemente a quantidade de tráfego na rede. Uma opção é combinar as leituras dos sensores de diferentes nós de uma forma coerente via técnicas de *beam-forming* [127]. Alternativamente, combinação não-coerente, também conhecida como fusão ou agregação de dados, pode ser usada. Técnicas de combinação não-coerentes não requerem sincronização entre os sensores, mas são menos poderosas que técnicas coerentes. Várias alternativas têm sido propostas para selecionar os nós que irão realizar a combinação de dados, tais como eleição de vencedor [127] ou clusterização [60].

Um aspecto importante da disseminação de dados em RSSFs é a escolha da rota entre origem e destino. Várias propostas têm sido apresentadas, visando selecionar caminhos que minimizam o consumo total de energia na rede. A Seção 2.2 apresentou exemplos de protocolos de disseminação de dados em RSSFs que são eficientes em energia.

A distribuição eficiente do tráfego realizada pelos protocolos de roteamento essencialmente explora a redundância em macro-escala de possíveis rotas entre origem e destino. Em cada rota, entretanto, há uma redundância em micro-escala gerada pela existência de nós que podem ser considerados equivalentes do ponto de vista da escolha do caminho em múltiplos saltos. Tal redundância é gerada pelo fato de que, em cenários típicos de instalação, uma alta densidade de nós é necessária para garantir a cobertura adequada para o sensoriamento e para aumentar a tolerância a falhas da rede [22]. Nesse contexto, vários compromissos entre energia e qualidade também podem ser explorados. Por exemplo, em aplicações de rastreamento de alvos, redes mais densas levam a uma maior precisão nos resultados. Entretanto, se o tempo de vida da rede é mais crítico do que a precisão do rastreamento, a tarefa de sensoriamento pode ser realizada usando-se amostras de dados de menos sensores. Além de reduzir a complexidade computacional, tal procedimento também reduz os requisitos de comunicação dos nós não participantes, já que eles não precisam mais enviar seus dados para serem processados.

Um ponto importante destacado em [113] é que altas densidades de nós não resultam imediatamente em um maior tempo de vida da rede, já que o consumo de energia do rádio em modo ocioso (*idle*) não difere muito dos modos de transmissão e recepção. Somente mudando o rádio para o estado desligado (*power-off* ou s*leep*) pode-se conservar energia dos nós temporariamente inativos. Entretanto, nesse estado, os nós não podem se comunicar e são de fato removidos da rede, mudando a sua topologia ativa. Então, a questão crucial é gerenciar de forma inteligente as transições de/para o estado *sleep* enquanto se fornece operação robusta e sem interrupção da rede.

Estratégias inteligentes de roteamento e de manutenção da topologia garantem que a carga do tráfego é distribuída entre os nós de uma forma eficiente em energia, aumentando o tempo global de vida da rede. Exemplos de protocolos de controle de topologia que buscam otimizar o consumo de energia da rede podem ser vistos em [22,127,151,153].

3 Tecnologias de Middleware

A tarefa de construir aplicações distribuídas diretamente no topo da camada de rede é extremamente tediosa e propensa a erros. Os desenvolvedores de aplicações têm que lidar explicitamente com vários requisitos não funcionais dos sistemas distribuídos, como replicação e localização de dados, tratamento de falhas da rede, gerência de transações e concorrência, entre outros, tornando o desenvolvimento e a manutenção de aplicações extremamente lentos.

Com o objetivo de dar suporte aos desenvolvedores, uma camada de middleware é inserida entre o sistema operacional e a aplicação distribuída. O principal objetivo de um sistema de middleware é possibilitar a comunicação entre componentes distribuídos, escondendo das aplicações a complexidade do ambiente de rede subjacente e livrando-as da manipulação explícita de protocolos e serviços de infra-estrutura. Para isso, um sistema de middleware fornece aos desenvolvedores um alto nível de abstração dos serviços relacionados à distribuição. A utilização de sistemas de middleware acelera o desenvolvimento e a implantação de aplicações, permitindo aos desenvolvedores concentrarem-se nos requisitos funcionais de seus projetos, ou seja, no desenvolvimento dos componentes específicos da lógica do negócio.

Tecnologias de middleware tradicionais foram desenvolvidas considerando os requisitos de sistemas distribuídos fixos, e não são adequadas para ambientes de redes sem fio *ad-hoc*, móveis ou não. A Seção 3.1 apresenta uma caracterização dos sistemas distribuídos, apontando as principais diferenças entre sistemas fixos e sistemas móveis *ad-hoc*. A Seção 3.2 apresenta um modelo de referência para sistemas de middleware e descreve as principais características de middleware para sistemas distribuídos fixos e sistemas móveis. As Seções 3.3 e 3.4 apresentam soluções de middleware existentes para sistemas distribuídos tradicionais e móveis, respectivamente. Como o sistema proposto nessa tese baseou-se em vários conceitos da área de Serviços Web, a sub-Seção 3.3.1 descreve sucintamente as principais características e os componentes dessa tecnologia de middleware. Finalmente, a Seção 3.5 aborda os requisitos que um middleware especificamente projetado para RSSFs deve atender e comenta sobre o estado da arte nessa área, apresentando os principais trabalhos desenvolvidos até o momento.

3.1 CARACTERIZAÇÃO DE SISTEMAS DISTRIBUÍDOS

Um sistema distribuído consiste em uma coleção de componentes, distribuídos entre vários computadores conectados via uma rede. Esses componentes precisam interagir entre si, a fim de trocar dados ou acessar os serviços uns dos outros. Embora essa interação possa ser construída diretamente no topo das primitivas do sistema operacional, isso seria extremamente complexo para muitos desenvolvedores de aplicações. Em vez disso, usa-se o suporte de sistemas de middleware, localizados entre componentes do sistema distribuído e componentes do sistema operacional, e cuja tarefa é facilitar as interações entre esses componentes.

A definição de sistemas distribuídos apresentada acima se aplica tanto a sistemas fixos quanto a sistemas móveis. As principais diferenças entre os dois tipos de sistemas podem ser analisadas segundo três conceitos, os quais irão também influenciar o tipo de sistema de middleware a ser utilizado: o conceito de dispositivo, de conexão de rede e de contexto de execução.

Quanto ao tipo de dispositivo, sistemas distribuídos fixos são compostos por dispositivos fixos, que podem variar de PCs a estações de trabalho Unix e computadores de grande porte. Sistemas distribuídos móveis são compostos por dispositivos móveis que variam de PDAs a celulares e *smartcards*. Enquanto os primeiros são em geral máquinas poderosas, com grande quantidade de memória e processadores rápidos, os últimos têm capacidades limitadas, como baixa velocidade de CPU, pouca memória, baixa potência de bateria, entre outras características.

Quanto ao tipo de conexão de rede, dispositivos fixos são em geral conectados permanentemente à rede através de enlaces contínuos de grande largura de banda. As desconexões, quando ocorrem, são realizadas explicitamente por razões administrativas ou são causadas por falhas imprevistas. Essas falhas são consideradas esporádicas e, portanto, tratadas como exceção ao comportamento normal do sistema. Já os sistemas móveis sem fio podem obter largura de banda razoável apenas se os dispositivos estiverem dentro do alcance de poucas centenas de metros de sua estação base. Caso o número de dispositivos conectados simultaneamente aumente muito, a largura de banda cai rapidamente. Além disso, se o dispositivo se mover para uma área sem cobertura ou com alta interferência, a largura de banda pode cair para zero e a conexão pode ser

perdida. A ocorrência de desconexões imprevistas passa a não ser mais exceção, e sim parte do comportamento normal para comunicações sem fio.

Quanto ao tipo de contexto de execução, esse conceito pode ser definido como tudo o que influencia o comportamento de uma aplicação, incluindo recursos internos ao dispositivo, como quantidade de memória, energia disponível, e recursos externos, como largura de banda, qualidade da conexão de rede, localização, dispositivos (ou serviços) vizinhos, etc. Em um ambiente distribuído fixo, o contexto é mais ou menos estático: a largura de banda é alta e contínua, a localização quase nunca muda, dispositivos são adicionados ou removidos do sistema com pouca freqüência. Os serviços disponíveis podem mudar, mas a descoberta de serviços é facilmente realizada forçando-se os fornecedores de serviços a se registrarem com um serviço de localização bem conhecido.

Por outro lado, em sistemas móveis o contexto é extremamente dinâmico. Dispositivos podem entrar e sair do sistema com freqüência, e os serviços que estão disponíveis quando ocorre uma conexão na rede podem não estar mais lá quando ocorrer uma nova conexão. A procura de serviço é muito mais complicada em cenários móveis. A localização do dispositivo não é fixa e, dependendo de onde ele está e se há ou não movimento, a largura de banda e qualidade da conexão podem também variar muito.

De acordo com o tipo de dispositivo, da conexão de rede e do contexto sobre o qual é construído um sistema distribuído, podem-se distinguir três classes de sistemas distribuídos: tradicionais, *ad-hoc* e nômades. Segundo essa classificação, sistemas distribuídos fixos são categorizados como sistemas tradicionais. Os principais requisitos não funcionais para esses sistemas são a escalabilidade, a heterogeneidade dos componentes, a tolerância a falhas e o compartilhamento de recursos. Sistemas distribuídos móveis *ad-hoc* (ou simplesmente *ad-hoc*) consistem em um conjunto de dispositivos móveis, conectados à rede de modo intermitente através de enlaces de qualidade altamente variável e executando em ambientes altamente dinâmicos. Eles diferenciam-se dos sistemas tradicionais por não terem uma infra-estrutura fixa. Entretanto, os requisitos não funcionais listados para sistemas distribuídos tradicionais permanecem os mesmos, alguns problemas sendo ainda mais graves.

Já os sistemas distribuídos nômades situam-se entre os sistemas fixos tradicionais e os sistemas ad-hoc. Eles são baseados em um núcleo de roteadores, *switches* e

computadores fixos. Porém, na periferia dessa infra-estrutura de rede fixa, estações-base com capacidades de comunicação sem fio controlam o tráfego de mensagens de/para configurações dinâmicas de dispositivos móveis.

Como a infra-estrutura física é diferente, as soluções de middleware propostas para sistemas tradicionais não podem ser empregadas com sucesso em ambientes *ad-hoc* ou nômades. São necessárias novas soluções, especialmente projetadas para tais ambientes.

3.2 SISTEMAS DE MIDDLEWARE: UM MODELO DE REFERÊNCIA

Em [85] é apresentada uma classificação dos sistemas de middleware, levando em conta três aspectos: o tipo de carga computacional, o tipo de paradigma de comunicação, e o tipo de representação de contexto (Figura 4). A seguir esses três aspectos são analisados.

- ***Tipo de carga computacional*** - a carga computacional do middleware pode ser leve ou pesada, e depende do conjunto de requisitos não funcionais satisfeitos pelo sistema. Considere-se, por exemplo, que um dos objetivos de qualquer middleware é possibilitar a comunicação entre pares em um sistema distribuído, permitindo que um usuário requisite um serviço remoto. Nesse caso, o que distingue a carga computacional de diferentes sistemas de middleware é a confiabilidade com a qual essas requisições são manipuladas. É muito mais caro em termos de recursos consumidos garantir que uma solicitação seja sempre executada, em vez de fornecer apenas o serviço de melhor esforço, isto é, a solicitação pode ou não ser executada.

- ***Tipo de paradigma de comunicação*** - um sistema de middleware pode prover dois tipos de paradigmas de comunicação: síncrono ou assíncrono. O primeiro requer que, tanto o cliente que solicita um serviço, quanto o servidor que fornece aquele serviço, estejam conectados e em execução ao mesmo tempo, a fim de que a solicitação seja processada com sucesso. Já a comunicação assíncrona não requer que o emissor e o receptor de uma solicitação estejam conectados simultaneamente.

- ***Tipo de representação de contexto*** - a informação sobre o contexto de execução pode ser mostrada para as aplicações (translucência ou *awareness*) ou pode ser

mantida escondida dentro do próprio middleware (transparência). O middleware interage com o sistema operacional de rede subjacente e coleta informações sobre o contexto de execução, como a localização de um dispositivo, sua energia residual, serviços que estão disponíveis no mesmo, a largura de banda atual, entre outras. *Transparência* é obtida quando tais informações de contexto são usadas de forma privada pelo middleware e não são expostas para as aplicações. Já o conceito de translucência (*awareness*) significa que a informação de contexto (ou parte dela) é passada para as aplicações em execução, que passam a ser, portanto, cientes do contexto e responsáveis por tomar parte nas decisões estratégicas, dividindo essa responsabilidade com o middleware.

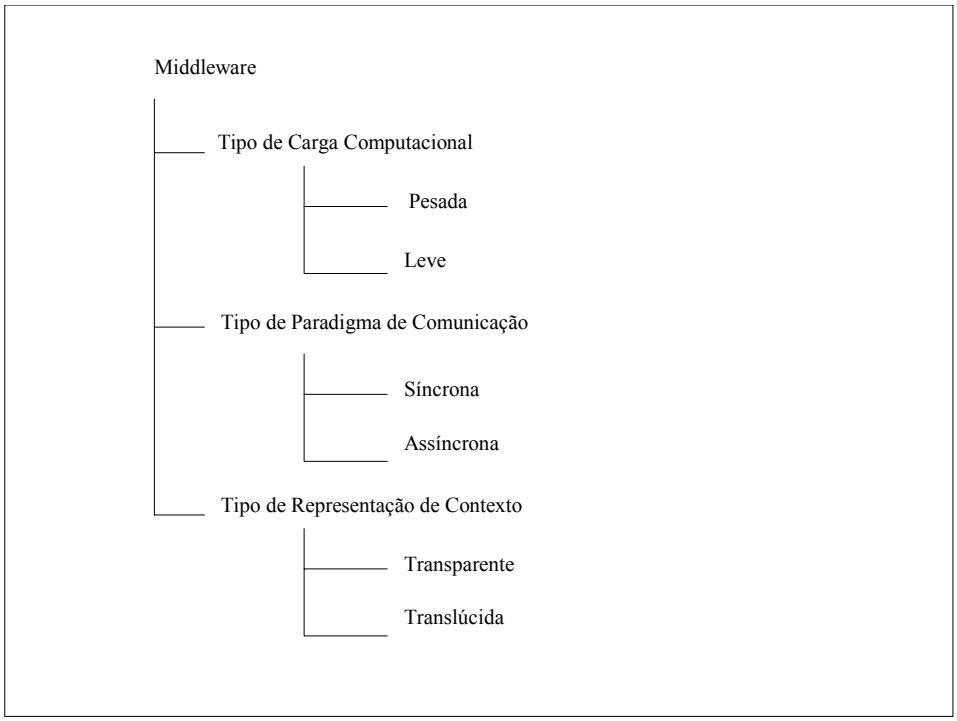


Figura 4: Caracterização de sistemas de middleware

As seções a seguir mostram as características que um sistema de middleware deve ter, segundo a classificação apresentada, a fim de serem utilizados em sistemas tradicionais e em sistemas *ad-hoc* e nômades.

3.2.1 Características de Middleware para sistemas distribuídos tradicionais

Com relação ao modelo conceitual de referência apresentado, middleware para sistemas distribuídos fixos podem ser descritos como sistemas que demandam recursos

computacionais (carga computacional pesada), cujo paradigma de comunicação é principalmente síncrono e que escondem informações de contexto das aplicações tanto quanto possível (transparência). A seguir será analisada em maiores detalhes a relação entre a infra-estrutura dos sistemas distribuídos fixos e as características do middleware associado.

- Dispositivos Fixos - carga computacional pesada. Como visto, sistemas distribuídos fixos em geral consistem de dispositivos ricos em recursos. Ao se construírem aplicações distribuídas no topo dessa infra-estrutura, vale a pena aproveitar todos os recursos disponíveis a fim de oferecer a melhor qualidade de serviço para a aplicação. Quanto mais alta a qualidade de serviço oferecida, mais pesado se torna o middleware executando sob a aplicação.

- Conexão Permanente - comunicação síncrona. Sistemas distribuídos fixos são em geral permanentemente conectados à rede através de enlaces estáveis e de alta largura de banda. Em outras palavras, o solicitante de um pedido e seu fornecedor estão em geral conectados ao mesmo tempo. A conexão permanente permite uma forma síncrona de comunicação.

- Contexto estático - transparência. O contexto de execução de um sistema distribuído fixo é essencialmente estático. O conhecimento no nível da aplicação pode ser transferido para o middleware em uma única interação inicial, a fim de que o middleware possa aproveitar tal conhecimento ao realizar escolhas estratégicas, que visem fornecer a melhor qualidade de serviço para a aplicação. Esconder informações de contexto dentro do middleware facilita o esforço dos desenvolvedores de aplicações.

3.2.2 Características de Middleware para Sistemas *Ad-Hoc* e Nômades

Middleware para sistemas distribuídos ad-hoc ou nômades precisam ser leves, permitir comunicação assíncrona entre componentes e tornar os projetistas de aplicações cientes do contexto de execução, pelas seguintes razões [85]:

- Dispositivos são móveis – a carga computacional nesses dispositivos deve ser leve. Devido às limitações de recursos, não é viável a execução de sistemas de middleware pesados nesses dispositivos. Portanto, é necessário escolher o melhor

compromisso entre a carga computacional e os requisitos não funcionais atendidos pelo middleware.

- Conexão é intermitente – a comunicação entre os dispositivos móveis é assíncrona, isto é, os dispositivos conectam-se à rede oportunisticamente por curtos períodos de tempo, principalmente para acessar algum dado ou requisitar algum serviço. Mesmo durante esses períodos, a largura de banda disponível é muito mais baixa do que em sistemas distribuídos fixos e pode subitamente cair para zero se o dispositivo entrar em uma área sem cobertura de rede. Portanto, freqüentemente o cliente que solicita um serviço e o fornecedor do serviço podem não estar conectados ao mesmo tempo. A fim de permitir esse tipo de interação, o middleware deve fornecer uma forma assíncrona de comunicação.

- Contexto é dinâmico – necessidade de ciência de contexto ("*awareness")*. Sistemas *ad hoc* e nômades executam em um contexto extremamente dinâmico quanto à largura de banda e à disponibilidade de recursos e de serviços. Portanto, não é viável para o desenvolvedor de aplicações prever todos os possíveis contextos de execução e instruir o middleware a priori sobre como se comportar em cada situação. Como não há um conhecimento estático que possa ser aproveitado, o middleware não pode tomar decisões de forma transparente com relação à aplicação e, ao mesmo tempo, garantir a melhor qualidade de serviço. Em vez disso, o middleware tem que interagir com a aplicação, tornando-a consciente de mudanças no contexto de execução e ajustar dinamicamente seu próprio comportamento usando informações que a aplicação lhe passa em retorno. Tornar a aplicação ciente do contexto de execução e lidar explicitamente com mudanças que ocorrem no ambiente adiciona um nível extra de complexidade aos desenvolvedores de aplicações, de modo que esforços de pesquisa têm que ser direcionados na busca de uma representação facilmente compreensível do contexto e de interfaces simples que os desenvolvedores possam usar para interagir dinamicamente com o middleware subjacente.

Após a exposição do modelo conceitual de referência com as principais características de middleware para os diferentes tipos de sistemas distribuídos, as seguintes seções apresentam exemplos de soluções de middleware existentes, as quais são de relevância para o presente trabalho.

3.3 SOLUÇÕES DE MIDDLEWARE PARA SISTEMAS DISTRIBUÍDOS TRADICIONAIS

CORBA [98], J2EE [129], COM [88] e .NET [89] são exemplos de tecnologias de middleware tradicionais, desenvolvidas levando em conta as necessidades de sistemas distribuídos fixos. Com a crescente demanda por interoperabilidade requerida pelo desenvolvimento de aplicações para a Web, a tecnologia de Serviços Web vem sendo utilizada como infra-estrutura de middleware apropriada para atender a essa demanda. Serviços Web são uma tecnologia de middleware baseada em XML que oferece um alto grau de interoperabilidade e flexibilidade para as aplicações, devido principalmente à ubiqüidade dos protocolos e soluções adotados.

Um Serviço Web pode ser definido como uma aplicação autocontida, cujas interfaces e ligações são definidas, descritas e localizadas por artefatos que utilizam a linguagem XML. Um Serviço Web deve ser capaz de interagir com outras aplicações através da troca de mensagens baseadas em XML, utilizando os protocolos de comunicação padrão atualmente disponíveis na Internet. Serviços Web podem ser descobertos e invocados através da Internet ou de uma intranet corporativa. Da mesma forma que componentes de software, Serviços Web expõem uma interface que pode ser reutilizada sem que se tome conhecimento sobre a forma como o serviço é implementado. Seu principal objetivo é facilitar a integração de sistemas através da Web, permitindo a interoperabilidade de aplicações desenvolvidas por diferentes grupos e em diferentes plataformas. Tecnologias baseadas em componentes, como CORBA [98], EJB [129], COM [88] e DCOM [90], resolvem muitos dos problemas de integração, porém ainda há barreiras que impedem seu pleno uso na Web. Uma das principais barreiras é que tais tecnologias exigem que as aplicações adotem o uso de um modelo de programação específico, por exemplo, baseado em objetos. Diferentemente de tais tecnologias, Serviços Web não requerem que as aplicações sejam desenvolvidas segundo um determinado modelo de programação. A plataforma de Serviços Web requer apenas o uso de um conjunto mínimo de protocolos de comunicação para permitir a integração de aplicações através da Web. Além disso, os protocolos e formatos de dados adotados são ubíqüos, independentes de fabricante e considerados padrões *de facto* da Web, tais como *Hypertext Transfer Protocol* (HTTP[44]), SOAP [140] e XML [135].

O presente trabalho adota uma abordagem de serviços, desenvolvida a partir do paradigma de Serviços Web, e utiliza várias das tecnologias envolvidas na área, como a

linguagem XML [135] e o protocolo SOAP [140]. A seguir serão detalhados os principais componentes da arquitetura de Serviços Web.

3.3.1 Arquitetura de Serviços Web

Os Serviços Web baseiam-se em uma arquitetura flexível orientada a serviços, denominada SOA (*Service-Oriented Architecture*) [53]. Os componentes da arquitetura SOA são essencialmente coleções de serviços que se comunicam através da troca de mensagens. São definidos três papéis na arquitetura: fornecedor de serviços, solicitante de serviços e registro de serviços. Um fornecedor de serviços é responsável pela descrição e a publicação de um Serviço Web no registro dos serviços. O fornecedor também é responsável por descrever as informações de ligação do serviço usadas para sua chamada. As informações estão representadas em um documento XML escrito na linguagem padrão WSDL (*Web Services Description Language*) [141]. O solicitante, ou consumidor de serviços é responsável por encontrar uma descrição de um Serviço Web publicada em um ou mais registros de serviços e usar tal descrição para se ligar ao respectivo provedor e invocar o serviço. O registro de serviços mantém a lista de todos os serviços e suas descrições, como nome, fornecedor e categoria. Ele é responsável por anunciar descrições de Serviços Web publicadas por fornecedores de serviços e por permitir que solicitantes de serviços inspecionem a coleção de descrições existente no registro. O papel do registro de serviços é servir de intermediário entre solicitantes e fornecedores de serviços. O padrão adotado na arquitetura de Serviços Web para registro é o UDDI (*Universal Description, Discovery and Integration*) [8].

A interação entre os três papéis envolve três operações básicas: a publicação da informação sobre um dado serviço (*publish*), a descoberta dos serviços disponíveis (*find*) e a ligação a esses serviços (*bind*) [51]. A Figura 5 ilustra como os papéis e as interações, representadas pelas operações, estão relacionados.

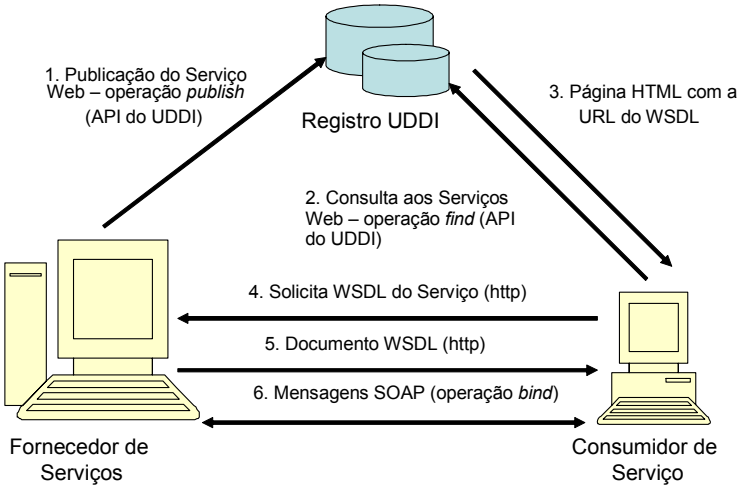


Figura 5**:** Modelo de Serviços Web, papéis e suas interações.

3.3.2 Componentes dos Serviços Web

Além de se conformar à arquitetura SOA, a tecnologia de Serviços Web pode ser descrita segundo três conjuntos de especificações: (i) descrição de serviços, (ii) protocolos de comunicação, e (iii) publicação e descoberta de serviços. Os três conjuntos de especificações têm em comum o uso da linguagem XML, o que permite a integração e troca de dados entre componentes distintos e garante a interoperabilidade necessária para a arquitetura.

A descrição de serviços é utilizada para definir as operações, mensagens e os tipos de dados de um serviço, além de manter as informações sobre como acessar os serviços. A especificação dos protocolos de comunicação contém a descrição dos formatos e protocolos para serem utilizados no estabelecimento da comunicação entre aplicações. A especificação da publicação e descoberta de serviços contém os protocolos que possibilitam a localização da descrição dos serviços. Nas próximas seções há uma breve descrição de cada conjunto de especificações.

3.3.3 Descrição de serviços

A linguagem WSDL (*Web Services Description Language*), atualmente na versão 1.2, é uma linguagem baseada em XML que descreve de forma padronizada e

independente de plataforma como e onde os Serviços Web podem ser conectados e utilizados através da rede [141].

Um documento WSDL representa um contrato entre o provedor de serviços e seus clientes. Com WSDL, um desenvolvedor descreve a interface de um Serviço Web, ou seja, o conjunto de operações fornecidas pelo serviço, incluindo os tipos de objetos esperados como entrada e saída de tais operações e as várias ligações (*bindings*) para esquemas concretos de codificação de dados.

Em um documento WSDL, serviços são definidos como uma coleção de portas na rede. Os documentos WSDL apresentam uma clara separação entre a definição de serviços e mensagens - parte abstrata - e sua implementação - parte concreta, o que permite o reuso das definições [42].

Um documento WSDL é composto por sete elementos XML que representam as partes abstrata e concreta. Na parte abstrata há quatro elementos:

1. ***types*** - fornece a definição dos tipos de dados (independente de plataforma ou linguagem) usados para descrever as mensagens trocadas entre aplicações; normalmente representada por um documento XSD (XML *Schema* [137]);
2. ***message*** - descreve os dados trocados, utilizando as definições dos tipos de dados;
3. ***porttype*** - é um conjunto de operações oferecidas por um ou mais *endpoints*, cada operação se refere a mensagens de entrada, saída ou erro;
4. ***operation*** - descreve uma ação fornecida pelo serviço;

Na parte concreta estão os outros três elementos:

1. ***binding*** - define uma especificação de protocolo e formato de dados para as operações e mensagens descritas em um *porttype*; aqui é que se define o protocolo de comunicação de alto nível que será usado entre um cliente e um servidor específicos. Na especificação atual do WSDL [141] são incluídos *bindings* para os seguintes protocolos e formatos de mensagens: SOAP 1.2 [136], HTTP GET/POST [44] e MIME [46];
2. ***port*** - é um único *endpoint*, definido como uma combinação de um *binding* e um endereço de rede;

3. ***service*** - é uma coleção de elementos *ports* relacionados. Cada elemento *port* se relaciona com um elemento *binding* particular, indicando qual interface e qual protocolo de comunicação estão sendo utilizados nessa implementação.

3.3.4 Descoberta e Publicação de serviços

A especificação do UDDI [8], atualmente na versão três, tem um papel de destaque no paradigma dos Serviços Web. UDDI é um protocolo para comunicação com registros de serviços. Ele fornece um serviço de diretórios o qual provê uma maneira uniforme de fornecedores descreverem seus serviços e clientes descobrirem esses serviços, entendendo os detalhes de como se conectar e interagir com o software que implementa o serviço. UDDI permite a publicação, descoberta e integração de serviços, através da definição de estruturas de dados para a descrição e classificação de serviços, e de uma interface (API) baseada no protocolo SOAP que permite o acesso a essas informações.

Resumidamente, o registro de serviços UDDI possui dois tipos de cliente. O primeiro envolve as aplicações de desejam publicar serviços e suas interfaces, o segundo tipo envolve os clientes que desejam obter e se ligar a Serviços Web.

A especificação do UDDI define quatro estruturas de dados, ou "registros", descritas como documentos XML: *businessEntity*, *businessService*, *bindingTemplate* e *tModel* [94]. O *businessEntity* é uma estrutura de alto nível (páginas brancas) que contém, para cada serviço, as informações (nome, categoria, identificadores, entre outros) sobre a organização que publicou o Serviço Web. O *businessService* contém informações descritivas sobre Serviços Web (páginas amarelas), tais como nome e descrição do serviço publicado. O *bindingTemplate* contém informações sobre como acessar e quais os endereços dos pontos de entrada do Serviço Web (páginas verdes). Finalmente, o *tModel* é o mecanismo usado para a troca de definições abstratas (metadados) sobre um Serviço Web. Ele aponta opcionalmente para um documento WSDL que descreve a interface do serviço.

3.3.5 Protocolos de comunicação

O SOAP é um protocolo de "transporte" que rege a troca de mensagens entre aplicações em ambientes distribuídos e descentralizados [42]. A especificação do protocolo SOAP tem quatro partes [139]: (i) envelope SOAP, (ii) infra-estrutura de

ligação, (iii) regras de codificação e (iv) SOAP RPC. A construção chamada de *envelope* SOAP define uma infra-estrutura global para expressar o que está contido na mensagem (o que é a mensagem), quem deve tratá-la, se ela é opcional ou obrigatória, e como sinalizar erros. A infra-estrutura de ligação define uma infra-estrutura abstrata para trocar envelopes SOAP entre pares comunicantes, usando um protocolo subjacente para o transporte. As regras de codificação SOAP definem um mecanismo de serialização que pode ser usado para trocar instâncias de dados definidos pela aplicação, vetores e tipos compostos. A representação SOAP RPC define uma convenção que pode ser usada para representar chamadas remotas a procedimentos e suas respectivas respostas.

```
POST /VirtualBooks HTTP/1.1
Host: localhost
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: http://equipe.nce.ufrj.br/fdelicato
<soapenv:Envelope>
<soapenv:Header>
  <soapenv:criptograph
      soapenv:mustUnderstand="0" xsi:type="xsd:string">yes
  </soapenv:criptograph>
  <soapenv:priority
      soapenv:mustUnderstand="0" xsi:type="xsd:string">high
  </soapenv:priority>
 </soapenv:Header>
 <soapenv:Body>
   <validateInfo
    soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <Username xsi:type="xsd:string">Sergio </Username>
    <BirthDate xsi:type="xsd:string">03/02/1965 </BirthDate>
    <IDCardType xsi:type="xsd:string">CPF </IDCardType>
    <IDCardNumber xsi:type="xsd:string">45608 </IDCardNumber>
   </validateInfo>
 </soapenv:Body>
</soapenv:Envelope>
```

Figura 6**:** Mensagem SOAP contendo os elementos ***Envelope***, ***Header*** e ***Body***.

Toda mensagem SOAP é um documento XML o qual contém, obrigatoriamente, os elementos *Envelope* e *Body* e, opcionalmente, os elementos *Header* e *Fault*. O elemento *Envelope* é a raiz do documento XML e representa a mensagem propriamente dita. O elemento *Body* contém a carga de informações (operações e parâmetros) que são entregues ao destinatário da mensagem. Esse elemento pode conter um elemento *Fault*, que, quando presente, pode ser utilizado no processamento de falhas do Serviço Web. O elemento *Header* expande uma mensagem SOAP, definindo algumas características opcionais e acordos negociáveis entre as partes. Seu conteúdo deve ser aceito pelas aplicações que estiverem se comunicando. A Figura 6 ilustra uma mensagem SOAP sendo enviada através do método POST do protocolo HTTP v 1.1.

A especificação SOAP, a rigor, não depende do protocolo HTTP, apesar de SOAP/HTTP ser a combinação mais amplamente utilizada atualmente. Contudo, SOAP pode ser utilizado em conjunto com outros protocolos como, por exemplo, FTP ou SMTP.

3.4 SOLUÇÕES DE MIDDLEWARE PARA SISTEMAS DISTRIBUÍDOS MÓVEIS

Como visto, sistemas de middleware desenvolvidos para sistemas distribuídos tradicionais não podem ser empregados com sucesso em ambientes móveis (*ad-hoc* ou nômades), porque geram uma carga computacional pesada, utilizam um paradigma de comunicação síncrona e/ou foram projetados para aderir ao princípio de transparência.

Como os requisitos para construir aplicações distribuídas móveis são consideravelmente diferentes dos requisitos impostos por aplicações fixas, os pesquisadores perceberam a necessidade de projetar novos sistemas middleware, desenvolvidos desde o início tendo em mente atender a tais requisitos [85].

Dentre as soluções desenvolvidas até o momento, sistemas de middleware reflexivos e cientes de contexto ("*context-aware*") parecem ser os mais promissores, satisfazendo a maioria dos requisitos que um middleware para computação móvel deve atender.

3.4.1 Middleware Ciente de Contexto (*Context-aware*)

Para permitir que aplicações se adaptem à heterogeneidade de dispositivos e redes, bem como a variações no ambiente do usuário, sistemas de middleware têm que fornecer para aplicações móveis informações sobre o contexto no qual elas estão sendo utilizadas. O contexto do usuário inclui, mas não é limitado a: (i) localização do dispositivo, com precisão variável dependendo do sistema de posicionamento usado; (ii) localização relativa a outros dispositivos; (iii) características do dispositivo, como capacidade de processamento e bateria disponível; (iv) ambiente físico, como nível de ruído e largura de banda; e (v) atividade do usuário, por exemplo se está em movimento ou não.

Computação ciente de contexto (*context-aware*) tem sido pesquisada há um longo tempo, principalmente com relação a consciência de localização. Existem vários sistemas que obtêm informação de contexto e adaptam-se a mudanças nesse contexto. A maioria desses sistemas interage diretamente com o sistema operacional subjacente para extrair informação de localização, processá-la e apresentá-la em um formato conveniente para o

usuário. Uma das suas principais limitações diz respeito ao fato de tais sistemas não lidarem com a heterogeneidade de informações e, portanto, diferentes versões têm que ser entregues para interagir com tecnologias específicas de sensores para posicionamento, tais como GPS para ambientes externos (*outdoor*) e infravermelho ou rádio freqüência para ambientes internos (*indoor*). A fim de facilitar o desenvolvimento de serviços e aplicações baseados em localização, sistemas de middleware foram construídos para integrar diferentes tecnologias de posicionamento, fornecendo uma interface comum para os diferentes sistemas.

3.4.2 Middleware Reflexivo

O conceito de reflexão computacional foi introduzido por Smith em 1982 [126] como um "princípio que permite a um programa acessar, refletir a respeito e alterar sua própria interpretação". O papel da reflexão em sistemas distribuídos relaciona-se com a introdução de maior abertura e flexibilidade nas plataformas de middleware. Em middleware tradicional, a complexidade inerente à distribuição é manipulada de forma transparente para o desenvolvedor de aplicações, que lida com um nível maior de abstração. Detalhes de implementação e de funcionamento do sistema distribuído são escondidos de usuários e desenvolvedores e encapsulados dentro do middleware. Como visto, essa abordagem sofre de sérias limitações quando aplicada a ambientes móveis.

Esconder detalhes de implementação significa que toda a complexidade é gerenciada internamente pela camada de middleware. O middleware é responsável por tomar decisões em benefício da aplicação, sem deixar que esta influencie em suas escolhas. Tal abordagem pode levar a sistemas de middleware computacionalmente pesados, caracterizados por grandes quantidades de código e dados que são usados para lidar de modo transparente com qualquer tipo de problema e encontrar a solução que garanta a melhor qualidade de serviço para a aplicação. Sistemas pesados não podem rodar eficientemente em um dispositivo móvel. Além disso, em uma configuração móvel, nem sempre é possível, ou mesmo desejável, esconder todos os detalhes de implementação do usuário. Escondendo detalhes de implementação, o middleware tem que tomar decisões para a aplicação; a aplicação pode, entretanto, ter informações vitais que poderiam levar a decisões mais adequadas ou eficientes.

Ambas as limitações, ou seja, a geração de sistemas de middleware pesados e a não disponibilização de informações internas do middleware para o usuário, podem ser contornadas usando-se os princípios de middleware reflexivo [27,29,30,73]. Segundo a definição apresentada em [19]: "Reflexão capacita a inspeção e a adaptação dos sistemas em tempo de execução. A inspeção permite que o estado corrente do sistema seja observado e a adaptação permite que o comportamento do sistema seja alterado para melhor se ajustar ao ambiente vigente de operação do sistema". Um middleware reflexivo pode tornar explícitos e acessíveis para a aplicação alguns aspectos de seu comportamento interno, através de um processo chamado reificação [18,73]. As aplicações têm, então, permissão para inspecionar e alterar tal comportamento em tempo de execução. Assim, um núcleo de middleware com apenas um conjunto mínimo de funcionalidades pode ser instalado em um dispositivo móvel, e a aplicação é responsável por monitorar e adaptar o comportamento do middleware de acordo com suas próprias necessidades. A aplicação pode modificar parâmetros nos procedimentos existentes no middleware ou pode mesmo ativar novos procedimentos dinamicamente.

As possibilidades abertas por essa abordagem são notáveis: middleware leve pode ser construído fornecendo ciência de contexto (*context-awareness*). Informação de contexto pode ser mantida pelo middleware em suas estruturas de dados internas e, através de mecanismos reflexivos, aplicações podem adquirir informações sobre seu contexto de execução e ajustar o comportamento do middleware de acordo. Nenhum paradigma de comunicação específico é relacionado ao princípio de reflexão. A escolha do paradigma a ser adotado fica a cargo do sistema específico a ser construído. O middleware reflexivo permite, inclusive, a reconfiguação dinâmica do paradigma de comunicação adotado.

3.5 MIDDLEWARE PARA REDE DE SENSORES SEM FIO

A maior parte dos projetos de RSSFs assume a existência de um forte acoplamento entre o componente de aplicação e os níveis subjacentes, ou seja, a pilha de protocolos e a infra-estrutura. Tal acoplamento é justificado pela necessidade de se obter sistemas eficientes em termos de energia. Porém, esse acoplamento produz sistemas rígidos e, conseqüentemente, gera redes construídas especificamente para atender umas poucas aplicações alvo. Dados os custos de construção e instalação de uma rede de sensores e seu tempo de vida útil, tal inflexibilidade de uso não se mostra desejável a longo prazo.

RSSFs podem se beneficiar da existência de um sistema de middleware situado entre o componente que engloba as aplicações e os componentes subjacentes. Tal middleware deve atender aos requisitos específicos das redes de sensores e oferecer funcionalidades que facilitem a integração de aplicações com a rede e o trabalho dos desenvolvedores de aplicações. Exemplos dessas funcionalidades são o fornecimento de uma interface de alto nível para acessar a rede e obter seus dados, e a tradução dos requisitos recebidos das aplicações para operações de baixo nível, como a seleção do protocolo a ser usado, a configuração da topologia lógica da rede e o escalonamento dos nós sensores na realização da tarefa recebida. Dadas as características dinâmicas em que operam as RSSFs e a importância da participação da aplicação no processo de comunicação na rede, o middleware deve prover mecanismos que permitam à aplicação configurar, inspecionar e adaptar dinamicamente o comportamento da rede.

As redes de sensores sem fio são uma categoria de redes sem fio *ad-hoc*, compartilhando muitas das características dessas redes e possuindo requisitos e restrições adicionais. Com relação à carga computacional, os dispositivos em RSSFs possuem capacidades de armazenamento e processamento extremamente restritos. Portanto, um middleware para RSSF deve ser ainda mais leve do que para redes móveis *ad-hoc*. Além disso, como sensores não possuem fontes de alimentação externa, podendo ter seus recursos de energia esgotados, o middleware deve possuir mecanismos para otimizar agressivamente o uso dos recursos de energia disponíveis. Além da intermitência de conexão característica de redes sem fio, a disponibilidade dos sensores na rede é altamente variável. Apesar de geralmente serem fixos, os sensores estão sujeitos às dinâmicas do meio ambiente, podendo ser destruídos ou deslocados de modo imprevisível. A topologia da rede também é alterada com freqüência pelos protocolos de controle de topologia, que decidem quando e quais nós desligar para economizar energia. Tais características mostram o alto grau de dinamismo na disponibilidade dos nós e do contexto de execução das RSSFs. Portanto, o estilo de comunicação adotado na rede deve ser preferencialmente assíncrono e mecanismos de ciência de contexto ("*awareness*") devem ser fornecidos pelo middleware.

Uma característica particular das RSSFs é que elas podem ser compostas por milhares de nós, os quais, em geral, não possuem um endereço único global, mas são endereçados por atributos que descrevem suas características. Portanto, todas as soluções

de middleware usadas nas redes de sensores devem ser escaláveis e adotar mecanismos de endereçamento centrados em dados.

Como visto, um middleware para RSSF deve ser projetado obedecendo a todos os requisitos de middleware para redes móveis *ad-hoc*. Porém, requisitos adicionais devem ser atendidos a fim de comportar as particularidades das redes de sensores. A próxima seção fornece uma visão mais detalhada sobre características de middleware específico para RSSFs.

3.5.1 Requisitos e Princípios de Projeto para Sistemas de Middleware para RSSFs

Considerando a arquitetura de RSSFs exposta na Seção 2.1, um middleware para RSSFs atua nos três planos de gerenciamento descritos: de energia, de mobilidade e de tarefas, ou seja, possui atribuições as quais se encaixam naquelas descritas para esses planos. Os principais propósitos do middleware são dar suporte ao desenvolvimento e à execução de aplicações baseadas em sensoriamento e gerenciar a utilização dos recursos da rede ao atender às aplicações. Todas as responsabilidades mencionadas na Seção 2.1 para os protocolos de apoio às aplicações podem ser atribuídas a um sistema de middleware. Assim, um middleware para RSSFs deve incluir, adicionalmente, mecanismos para formular tarefas de sensoriamento complexas de alto nível, comunicar tais tarefas para a rede, coordenar os nós sensores na execução das tarefas, agregar os dados coletados individualmente pelos sensores em um resultado de alto nível e reportar o resultado em uma forma compreensível para a aplicação que emitiu a tarefa.

O middleware deve isolar as aplicações das características de infra-estrutura e dos protocolos de comunicação subjacentes. Para isso, abstrações apropriadas e mecanismos para lidar com a heterogeneidade dos nós sensores, com relação aos seus tipos, capacidades e aos protocolos utilizados, devem ser fornecidos [116]. O middleware deve tomar decisões de infra-estrutura, como escolha e configuração de protocolos de comunicação e topologia lógica dos nós, de forma transparente para as aplicações, porém considerando seus requisitos de QoS. Ele deve encapsular os diferentes protocolos de comunicação existentes, fornecendo para a aplicação uma interface de alto nível.

O middleware deve também tomar decisões quanto à seleção dos nós que irão participar de cada tarefa de sensoriamento, sempre levando em conta o compromisso

entre atender aos requisitos da aplicação e otimizar o consumo de energia na rede. Dependendo da granulosidade de controle desejada, o middleware pode ainda decidir os níveis de voltagem utilizados para processar cada tarefa alocada a um sensor, caso o processador do sensor permita DSV, e pode variar dinamicamente o esquema de modulação de dados, sempre com o intuito de otimizar o consumo de energia.

Todos os serviços fornecidos pelo sistema de middleware devem respeitar as características particulares das RSSFs, principalmente robustez, escalabilidade e a eficiência em termos de energia.

Uma característica adicional diz respeito aos conceitos de tempo e localização dos eventos sensoriados. Como as RSSFs monitoram dados do mundo real, informações de tempo e espaço são relevantes, sendo elementos chave para fundir leituras individuais de sensores. Portanto, suporte para gerenciamento de tempo (sincronização) e de localização pode também ser integrado em um middleware para RSSFs.

Em [161] foram propostos princípios a serem adotados no projeto de um middleware para RSSFs. Tais princípios foram seguidos no presente trabalho e são apresentados a seguir.

- O middleware deve fornecer mecanismos centrados em dados para o processamento e a consulta de dados dentro da rede.

- Conhecimento da aplicação deve ser usado para otimizar o funcionamento da rede. É, então, importante integrar conhecimento da aplicação aos serviços fornecidos pelo middleware.

- Todas as soluções devem, de preferência, ser baseadas na utilização de algoritmos localizados. Tais algoritmos fornecem robustez e escalabilidade ao sistema.

- O middleware deve ser leve em termos de requisitos de comunicação e computação. Tal requisito indica a necessidade de usar heurísticas simples e eficientes que gerem soluções sub-ótimas para as decisões tomadas pelo middleware.

- Devido aos recursos limitados, é muito provável que os requisitos de desempenho de todas as aplicações em execução não possam ser simultaneamente satisfeitos. Portanto, é necessário que o middleware negocie inteligentemente a QoS de várias aplicações umas contra as outras.

O conhecimento da aplicação é utilizado pelo middleware em decisões de vários níveis, incluindo o roteamento e a agregação de dados. Entretanto, devido às RSSFs poderem suportar uma ampla classe de aplicações, compromissos precisam ser explorados entre o grau de especificidade de aplicação e a generalidade do middleware. Uma política prática é embutir as características particulares de uma aplicação na especificação de uma categoria de aplicações, construindo-se um perfil. A informação embutida no perfil da aplicação pode ser interpretada pelo middleware e utilizada para direcionar suas operações.

Devido ao ambiente extremamente dinâmico no qual opera, para fornecer uma qualidade de serviço razoável para seus usuários, uma RSSF não pode basear seu comportamento em parâmetros estáticos definidos a priori. Decisões de configuração tomadas inicialmente pelo middleware devem poder ser alteradas dinamicamente, com ou sem a participação da aplicação. O contexto de execução deve ser continuamente monitorado e as aplicações precisam ser "cientes" de seu contexto [18] e participar ativamente do processo de tomadas de decisões.

Para isso, o middleware deve expor para as aplicações parte de seu conhecimento sobre o contexto, permitindo-lhes participar de decisões de infra-estrutura e da ativação de estratégias de adaptação, quando necessário.

Segundo [19], sob o termo geral "ciência de contexto", podem-se identificar dois níveis mais específicos de ciência: ciência de dispositivo e ciência de ambiente. Ciência de dispositivo refere-se a tudo o que reside no dispositivo físico, ou seja, seus recursos internos. Em RSSFs, o dispositivo físico é o sensor e seus recursos internos são sua bateria, capacidades de memória e processamento. Ciência de ambiente refere-se a tudo que está fora do dispositivo, isto é, largura de banda, conexão de rede, localização e outros dispositivos ao alcance (recursos externos). Em RSSFs, tanto os recursos internos quanto os externos dos dispositivos dizem respeito ao "estado da rede", considerando que a rede engloba os nós sensores e os enlaces sem fio. Há ainda um outro nível de ciência de

contexto em RSSFs que diz respeito aos valores dos dados monitorados pela rede e, portanto, reflete o "estado da aplicação".

Os autores em [18] argumentam que o uso de reflexão [126] provê uma forma poderosa de construir sistemas de middleware que permitem o desenvolvimento de aplicações cientes do contexto ("*context-aware*"). Um middleware para RSSFs pode utilizar princípios reflexivos para interagir com a infra-estrutura da rede subjacente, manter informação de contexto atualizada em suas estruturas de dados e disponibilizar essa informação para as aplicações. Aplicações especificam para o middleware o comportamento desejado da rede através de perfis, nos quais declaram requisitos de QoS a serem respeitados nos diferentes contextos de execução. Através de mecanismos reflexivos, perfis podem ser acessados (inspecionados) e alterados dinamicamente.

3.5.2 Estado da Arte

Nos últimos quatro anos, surgiram várias propostas de sistemas de middleware especialmente projetados para redes de sensores sem fio, cada um com diferentes objetivos e abordagens. Além disso, sistemas de middleware tradicionais com capacidades para fornecer suporte parcial aos requisitos de RSSFs foram adaptados para tal uso.

Em [61] é apresentada uma classificação detalhada de sistemas de middleware, com exemplos de projetos existentes em cada categoria. Nas seções a seguir são exemplificados sistemas de middleware que podem ser utilizados para RSSFs, indicando-se em que categoria eles se enquadram segundo a classificação apresentada e descrevendo-se suas principais características.

3.5.2.1 Sistemas de Middleware Tradicionais com Suporte Parcial aos Requisitos de RSSFs

Quanto à classificação proposta em [61] e apresentada na Figura 7, "middleware reativo" refere-se a sistemas que são capazes de reagir a mudanças no comportamento da rede, enquanto "middleware proativo" refere-se a sistemas que mudam proativamente a funcionalidade da rede. Já o conceito de "capacitar aplicações reativas" refere-se a sistemas que fornecem ganchos, isto é, procedimentos, de modo a permitir que as próprias aplicações reajam a mudanças no ambiente, enquanto "capacitar aplicações proativas"

refere-se a sistemas de middleware que aceitam informação da aplicação sobre como responder de forma proativa a mudanças na rede.

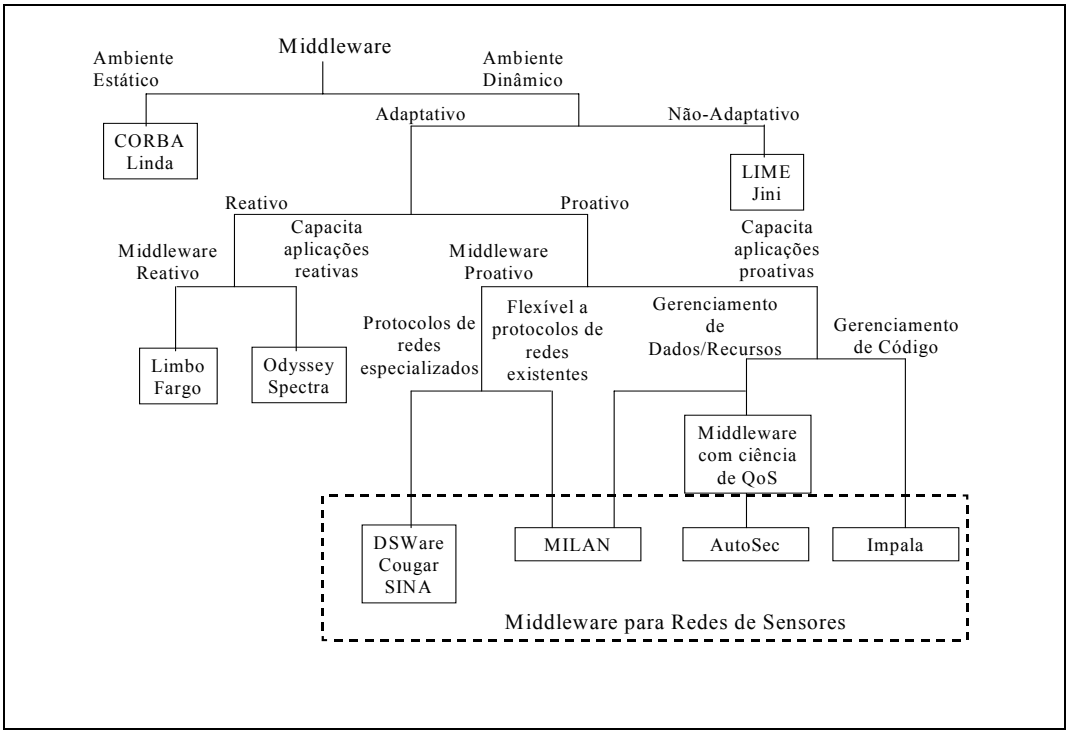


Figura 7: Relação entre diferentes sistemas de middleware. Fonte: [61].

Jini [38], e Lime[91] são exemplos de sistemas de middleware genéricos não adaptativos e adequados a ambiente dinâmicos. O protocolo de descoberta de serviços e o mecanismo de arrendamento (*leasing*) do Jini permitem que aplicações clientes descubram novos serviços e gerenciem conexões cliente-servidor conforme o conjunto de serviços disponíveis muda com o tempo. Descoberta de serviços é útil em RSSFs dinâmicas, pois permite descobrir quais sensores e/ou serviços estão disponíveis a cada momento. Entretanto, o acesso aos serviços no Jini é baseado em objetos, de forma similar a CORBA [98]. Conseqüentemente, é fornecido acesso aos sensores individuais através de objetos, de forma que os ganhos potenciais de energia que podem ser obtidos com agregação de dados são perdidos. O middleware LIME, por sua vez, adota uma API diferente, não baseada em objetos. Tal API consiste em um esquema de memória compartilhada para componentes móveis ad-hoc através de um espaço de tuplas semelhante ao usado em Linda [50]. Jini e LIME não consideram as restrições de energia das RSSFs, e seus protocolos são muito pesados quando comparados com protocolos construídos especificamente para atender tais redes.

Há sistemas de middleware genéricos com funcionalidades que os tornam capazes de prover suporte a alguns dos requisitos das redes sem fio, como a necessidade de adaptação a ambientes dinâmicos. Limbo [33] e FarGo [64] são exemplos de sistemas de middleware adaptativos e reativos que reordenam as trocas de dados ou realocam componentes para responder a mudanças nas condições da rede, tais como disponibilidade de banda ou confiabilidade do enlace.

Mobiware [17] e Odyssey [95] são exemplos de sistemas adaptativos que "capacitam aplicações reativas". Mobiware é um sistema de middleware que dá suporte a vários níveis de qualidade de serviço, adaptando fluxos dentro da rede através do uso de filtros ativos instalados nos roteadores. Outros sistemas de middleware fornecem ganchos para permitir à aplicação adaptar-se. Por exemplo, aplicações construídas sobre a plataforma Odyssey podem registrar-se para receber notificações de mudanças na taxa de dados da rede subjacente. Similarmente, o componente Spectra [45] do sistema Aura [49] monitora as condições da rede e os recursos de computação acessíveis, decidindo onde a computação deve ser realizada. Tal decisão é baseada: (i) na quantidade de transmissão requerida na rede para completá-la; e (ii) na comparação entre o custo da computação em nós móveis e fixos.

Quanto à natureza proativa, um sistema pode ser classificado em middleware proativo ou em sistema que "capacita aplicações proativas". Um sistema que "capacita aplicações proativas", por sua vez, pode ser classificado quanto ao tipo de gerenciamento que realiza: gerenciamento de dados e/ou recursos e gerenciamento de código. QoS-Aware [92] é um exemplo de sistema de middleware que gerencia dados/recursos. Tal middleware é projetado para dar suporte a aplicações para ambientes ubíqüos e heterogêneos. Essas aplicações, da mesma forma que aplicações de RSSFs, possuem requisitos mínimos de QoS. QoS-Aware é responsável por gerenciar recursos do sistema operacional local, com base nos requisitos da aplicação especificados para o middleware. A informação de QoS da aplicação é compilada em um perfil de QoS para guiar o middleware nas tomadas de decisão quanto ao uso dos recursos.

O sistema OpenCOM v2 [31] fornece um plataforma de propósito geral para o desenovolvimento de sistemas, a qual consiste em um modelo de componentes em tempo de execução e uma infraestrutura de middleware reflexivo e sensível ao contexto. O OpenCOM v2 pode ser utilizado em uma ampla gama de plataformas de execução, desde

ambientes tradicionais como PC/Windows ou PC/Unix até em PDAs com recursos limitados, sistemas embarcados sem sistemas operacionais, ou em hardware de processadores de redes de alta velocidade. Tal flexibilidade é conseguida através de seus pequenos requisitos de memória, e da sua independência de linguagem e de políticas. O OpenCOM v2 é construído no topo do núcleo de *kernels* adequados para cada dispositivo alvo. O modelo de componentes é dinamicamente configurável, o carregador de componentes pode ser substituído e os meta-modelos do sistema podem ser carregados e utilizados apenas quando necessários. O sistema é leve o suficiente para ser utilizado em RSSFs. Com relação às categorias apresentadas na Figura 7, o sistema pode ser classificado como um middleware adaptativo, reativo e que capacita aplicações reativas.

Todos os avanços fornecidos pelos sistemas de middleware mencionados anteriormente são aplicáveis a redes de sensores sem fio. Entretanto, tais sistemas não consideram os detalhes dos protocolos sem fio de baixo nível, nem integram os protocolos de agregação de dados específicos de RSSFs. A seguir são exemplificados sistemas especificamente projetados para RSSFs.

3.5.2.2 Sistemas de Middleware para RSSFs

O sistema AutoSec (*Automatic Service Composition* [54]) gerencia os recursos em uma rede de sensores, fornecendo controle de acesso para aplicações, de modo que os requisitos de QoS sejam mantidos. Essa abordagem é similar a middleware para redes convencionais, e as restrições de recursos são satisfeitas considerando cada sensor individualmente. Porém, as técnicas para coletar os dados sobre a utilização atual de recursos na rede foram adaptadas para RSSFs. Já Impala [81,82] é outro exemplo de sistema de middleware para RSSFs que "capacita aplicações proativas". Porém, seu foco é realizar o gerenciamento de código. Impala adota uma abordagem dirigida a eventos e seu principal objetivo é fornecer capacidades de modularidade e adaptação às aplicações. A chave para eficiência em energia no Impala é que as aplicações nos nós sensores sejam tão modulares quanto possível, capacitando pequenas atualizações quando necessário, demandando assim pouca energia de transmissão. Um código de aplicação pode ser instalado e atualizado em sensores em tempo de execução. Impala considera o procedimento que implementa os diferentes protocolos de comunicação como o software da aplicação a ser executado nos nós sensores. Ele fornece um componente para adaptação, que seleciona um protocolo de comunicação dentre diferentes protocolos já

disponíveis nos sensores, de acordo com o contexto de execução atual. O foco do trabalho realizado com Impala é avaliar os ganhos obtidos quando se adota a migração de módulos de código através do meio sem fio, em contraste com o uso de código monolítico.

O sistema DSWare [78] é um exemplo de middleware proativo. Ele fornece um tipo de abstração de serviço de dados similar ao fornecido pelo AutoSec. Porém, em vez do serviço ser fornecido por um único sensor, ele pode ser fornecido por um grupo de sensores geograficamente próximos. Portanto, DSWare pode gerenciar de forma transparente as falhas de sensores, desde que uma quantidade suficiente de sensores permaneça na área em questão para fornecer uma medida válida.

Embora cada um desses sistemas de middleware sejam projetados para o uso eficiente de RSSFs, de uma maneira geral eles ignoram as propriedades da rede em si. Em outras palavras, a maioria dessas abordagens não tenta modificar os parâmetros de rede a fim de gerenciar o consumo de energia, nem são flexíveis o suficiente para acomodar diferentes pilhas de protocolos ou diferentes requisitos de QoS das aplicações. Recentemente, vários trabalhos têm tratado do desenvolvimento de sistemas de middleware especificamente projetados para satisfazer os requisitos de RSSFs, focando principalmente nos aspectos de longevidade e restrição de recursos desses ambientes.

Em contraste com a abordagem baseada em serviços, proposta no presente trabalho, há trabalhos que propõem middleware para RSSFs seguindo abordagens de bancos de dados ou modelos de programação dirigidos a eventos. Abordagens baseadas em bancos de dados são apresentadas, por exemplo, em [11,12,28,52,155]. Pode-se considerar que tais abordagens foram a primeira tentativa de se propor uma camada de middleware rudimentar para o projeto de RSSFs. No sistema COUGAR [28,155] as capacidades de processamento dos sensores são aproveitadas para executar parte do processamento de consultas dentro da rede, em vez de centralizar tal processamento apenas em nós sorvedouros. Em [12], uma linguagem de consulta declarativa baseada em SQL é proposta para usuários submeterem suas consultas para a RSSF. Ambos são exemplos de sistemas de middleware proativos baseados em protocolos de rede especializados.

Em [122] é apresentada uma arquitetura para redes de sensores chamada SINA, a qual fornece mecanismos para consulta, monitoramento e submissão de tarefas. SINA desempenha o papel de um middleware que abstrai os nós de uma RSSF como uma

coleção de objetos distribuídos. Usuários acessam informações dentro de uma RSSF utilizando consultas declarativas, ou solicitam tarefas usando scripts de programação. SINA adota a linguagem de script procedural proprietária SQTL como a interface entre aplicações e o middleware SINA. Uma abordagem dirigida a eventos é adotada como modelo de programação.

Há trabalhos recentes em middleware para redes sem fio que tratam da questão de fornecer algum grau de ciência de contexto ou capacidade de reflexão. MILAN [61] é um middleware para RSSFs que recebe uma descrição de requisitos da aplicação e escolhe a melhor configuração de rede de acordo com tais requisitos, enquanto procura maximizar o tempo de vida da rede. MILAN incorpora mudanças nos interesses da aplicação baseadas em informação de estado e gerencia as condições da rede ao longo do tempo. Ele pode ser classificado tanto como um sistema proativo quanto como um sistema que capacita aplicações proativas, realizando o gerenciamento de recursos da rede. Em sua primeira versão, MILAN adota uma abordagem totalmente centralizada, adequada para redes de sensores de pequena escala, como as utilizadas para monitorar pacientes em hospitais.

CARISMA [18] é um middleware para redes sem fio móveis que explora os princípios de reflexão para permitir a construção de aplicações adaptativas e cientes de contexto. Em CARISMA, o middleware mantém uma representação válida do contexto de execução e disponibiliza parte dessa informação para as aplicações. Aplicações podem alterar dinamicamente a política de execução dos serviços solicitados, o que pode gerar conflito entre solicitações de diferentes aplicações. O middleware possui mecanismos capazes de gerenciar a resolução de tais conflitos, de modo a balancear os requisitos das diferentes aplicações.

Nos últimos anos, vários pesquisadores têm investigado o problema de gerenciamento de sensores como sendo um dos principais objetivos de um middleware para RSSFs. A maior parte dos trabalhos de gerenciamento em RSSFs [21,23,80,86,87,132,153] visa balancear a eficiência em energia da rede com os requisitos das aplicações clientes. O principal requisito de qualidade para a aplicação nesse contexto refere-se à cobertura de sensoriamento oferecida pela rede.

Em [102] e [103] os autores tratam do problema de seleção de nós ativos a fim de maximizar o tempo de vida de uma RSSF, enquanto se satisfaz um nível mínimo de

qualidade para a aplicação. Em ambos os trabalhos, o problema de seleção dos nós é tratado em conjunto com o roteamento, como um problema de fluxo máximo generalizado. Os trabalhos apresentam uma solução ótima e heurísticas para a solução sub-ótima. É adotada uma abordagem totalmente centralizada, baseada em informação global da rede.

O middleware proposto no presente trabalho, segundo a classificação apresentada na Figura 7, pode ser categorizado tanto como Reativo quanto como Proativo, já que ambas as abordagens podem ser adotadas pelo mecanismo de adaptação do sistema. Por outro lado, ele pode também capacitar aplicações proativas, já que possui mecanismos para aceitar informações da aplicação sobre como responder a mudanças na rede. Adicionalmente, ele inclui a capacidade de fornecer ganchos que permitam à própria aplicação reagir a mudanças no ambiente. O sistema proposto é totalmente independente do protocolo de disseminação de dados usado na camada de rede subjacente. Portanto, ele é flexível a diferentes protocolos existentes. Possui capacidade de gerenciar os recursos da rede e consciência dos requisitos de QoS das aplicações.

O presente trabalho possui algumas características que o distinguem de outras propostas existentes. Primeiro, ele propõe uma abordagem baseada em serviços para o projeto de RSSFs. De acordo com a abordagem proposta, todas as interações entre aplicações e a rede são baseadas em uma relação cliente-fornecedor. Tal abordagem foi inspirada na área de Serviços Web [53] e aproveita várias tecnologias pertencentes à área, como a linguagem XML [135] e o protocolo SOAP [140]. A principal vantagem dessa abordagem é oferecer um modelo de programação genérico e flexível, com grande capacidade de interoperabilidade entre diferentes aplicações e a rede. A segunda característica particular do trabalho é a proposta de um mecanismo padrão para representação de dados e formatação de mensagens, em vez de basear-se em formatos de trocas de mensagens ou linguagens proprietárias, ditadas pelas aplicações clientes ou pelos projetistas dos protocolos da rede. O uso dos padrões ubíqüos XML e SOAP com esse objetivo fornece alta portabilidade, extensibilidade e permite o uso de ferramentas automáticas para gerar mensagens para a rede, a partir de diferentes linguagens de programação. O impacto do uso de XML no limitado hardware das RSSFs foi avaliado nesta tese e está descrito no Capítulo 8. Através da exposição das funcionalidades dos sensores como serviços, a proposta oferece uma arquitetura mais flexível em comparação com abordagens baseadas em bancos de dados e consultas SQL. SQL é basicamente uma

linguagem de consulta, enquanto XML pode ser usada tanto como linguagem de consulta quanto de submissão de tarefas sendo, portanto, mais adequada para uso em RSSFs. A adoção de linguagens proprietárias para acessar os dados da rede, como SQTL, dificulta a interoperabilidade entre diferentes aplicações e redes de sensores, levando a uma solução com baixo grau de flexibilidade e portabilidade.

A terceira característica é que o middleware proposto adapta dinamicamente, e de forma transparente para a aplicação, a configuração da rede e dos protocolos utilizados aos requisitos específicos de cada aplicação. Apesar de MILAN compartilhar esse objetivo, o problema de como receber e interpretar os requisitos da aplicação não foi tratado nesse trabalho. Além disso, em contraste com a abordagem distribuída que pode ser adotada no presente trabalho, o projeto de MILAN é totalmente centralizado, o que limita a escalabilidade de sua solução. Outra característica relevante do middleware proposto é o uso de mecanismos para adaptação do sistema e para fornecer ciência de contexto. Tal funcionalidade também é fornecida por CARISMA que, no entanto, foi projetado para redes sem fio genéricas, não abordando os requisitos específicos de RSSFs, como a necessidade de mecanismos de agregação de dados, de escalonamento de nós ativos e de seleção e configuração de protocolos. Todos esses requisitos foram incluídos como serviços fornecidos pelo sistema de middleware proposto.

4 Modelo Lógico do Middleware para RSSFs Proposto

Para facilitar o desenvolvimento e permitir a interoperabilidade entre diferentes aplicações e diferentes infra-estruturas de redes, as aplicações de RSSFs devem ser construídas no topo de protocolos e infra-estruturas físicas existentes. Um sistema de middleware deve encapsular os diferentes protocolos, fornecendo uma abstração de alto nível do seu funcionamento, e atuar de forma a variar os parâmetros desses protocolos ao longo do tempo, a fim de atender aos requisitos das aplicações. Além da escolha e da parametrização dos protocolos a serem usados na rede, outra questão de infra-estrutura que pode ser tratada pelo middleware é a seleção dos nós sensores que devem ser ativados para realizar uma tarefa de sensoriamento. Uma vez tomadas as decisões de infra-estrutura necessárias, o sistema de middleware deve gerenciar a execução das tarefas de sensoriamento e a utilização dos recursos da rede.

Este trabalho propõe um sistema de middleware baseado em serviços para redes de sensores sem fio. No sistema proposto, considera-se que várias aplicações podem rodar na mesma rede, ao mesmo tempo ou em momentos diferentes. Cada aplicação pode otimizar a rede de forma diferente, de acordo com suas necessidades. Para isso, os requisitos da aplicação são usados para construir um perfil de aplicação. O perfil é usado pelo middleware para ditar parâmetros para os diferentes serviços fornecidos e para instruir como a rede deve se comportar em diferentes estados de execução. Para acomodar a natureza dinâmica das RSSFs, o middleware permite que as aplicações acessem seus perfis para consulta e alteração, em tempo de execução.

Neste trabalho, a estrutura de uma RSSF pode ser representada em termos de seus componentes físicos e de seus componentes organizacionais. A Seção 4.1 a seguir descreve tais componentes. Além disso, como na abordagem proposta a RSSF é vista como uma fornecedora de serviços para aplicações, os componentes arquiteturais da rede conformam-se ao padrão SOA [53], usado no projeto de arquiteturas orientadas a serviços. A Seção 4.2 descreve como a arquitetura de RSSFs considerada no trabalho adere ao padrão SOA [53], incluindo as tecnologias utilizadas no projeto do sistema proposto. Quanto ao sistema de middleware proposto, ele pode ser descrito segundo os modelos lógico e físico. O modelo lógico do sistema é apresentado na Seção 4.3 e inclui a

descrição dos serviços que devem ser disponibilizados pelo middleware para dar suporte à execução de aplicações de RSSFs, a especificação dos componentes lógicos que fornecem tais serviços, bem como a descrição das interfaces entre os componentes e destes com as aplicações e com a infra-estrutura de rede subjacente. Já o modelo físico do sistema, apresentado no Capítulo 5, inclui a descrição detalhada dos seus componentes, segundo as tecnologias escolhidas para sua implementação.

4.1 ESTRUTURA FÍSICA E ORGANIZACIONAL DAS RSSFS

Esta seção tem por objetivo descrever a arquitetura de RSSF considerada no trabalho. Quanto à estrutura física da rede, neste trabalho considera-se uma RSSF formada por dois tipos de nós: sensores e sorvedouros. Considera-se que todos os nós sensores são homogêneos com relação a suas capacidades de processamento e armazenamento, bem como quanto a seus recursos de energia. Entretanto, cada nó sensor pode possuir diferentes dispositivos de sensoriamento.

Sorvedouros são nós da rede com maiores recursos computacionais e ligados à rede elétrica, portanto, sem restrições de energia. Eles atuam como interfaces, através das quais aplicações clientes submetem suas consultas para a rede e obtêm os dados resultantes. Sorvedouros podem ser acessados por aplicações de forma local ou remota, por exemplo, através da Internet.

Quanto aos componentes organizacionais da rede, seus nós podem pertencer a uma dentre três categorias: (i) nó sorvedouro; (ii) nó comum; e (iii) nó líder de *cluster*. O sorvedouro, por representar o ponto de acesso à rede e a interface com as aplicações, possui uma arquitetura física bastante distinta da dos demais nós. Nós sorvedouros são dispositivos mais robustos computacionalmente e neles rodam os componentes mais pesados do sistema. Nós comuns e líderes são ambos nós sensores, portanto, fisicamente idênticos. Tal distinção entre as duas categorias deve-se à adoção, no presente trabalho, da organização da rede em *clusters*, para fins de gerenciamento.

Devido à sua simplicidade, flexibilidade, escalabilidade e robustez, arquiteturas baseadas em *cluster* têm sido amplamente usadas no projeto e na implementação de protocolos para RSSFs, como em [28,59,124,158]. Os princípios de projeto de middleware para RSSFs, descritos na Seção 3.5, destacam a necessidade de adotar algoritmos localizados na rede. Arquiteturas baseadas em *cluster* naturalmente restringem

a interação dos nós sensores e, portanto, o *overhead* de coordenação e controle, à uma vizinhança restrita, formada pelo *cluster*. A adoção de algoritmos localizados passa a ser uma escolha natural em tais arquiteturas. Comparadas com redes móveis, que incorrem em um alto custo para manter *clusters* na rede, RSSFs em geral consistem em nós sensores estacionários. Portanto, o custo de sobrepor uma arquitetura lógica de *clusters* sobre a rede física é compensador, dadas as vantagens potenciais oferecidas pelos *clusters* para o projeto de algoritmos escaláveis e localizados. Tais características motivaram a adoção desse tipo de arquitetura no projeto do middleware proposto.

Na arquitetura proposta, as funcionalidades do middleware são distribuídas entre os nós conforme sua categoria. Os nós líderes de *cluster* possuem responsabilidades adicionais em relação aos nós comuns.

Independentemente da estratégia de roteamento dos dados adotada, que pode basear-se em topologias planas ou hierárquicas, o middleware requer a criação de uma topologia virtual baseada em *cluster*, para sua execução. Na maior parte dos casos, a topologia virtual irá coincidir com a física, mas não é um requisito obrigatório.

4.2 COMPONENTES LÓGICOS DO SISTEMA

A construção de um sistema de middleware é influenciada pelos requisitos das aplicações e por questões relacionadas à infra-estrutura de comunicação sobre a qual o middleware irá executar. A descrição do sistema de middleware proposto inicia com a definição dos serviços que devem ser fornecidos para atender às necessidades das aplicações e para se adequar ao ambiente das RSSFs. A seguir, será descrito como as aplicações de sensoriamento interagem com o sistema e como novos serviços podem ser adicionados ao middleware.

4.2.1 Principais serviços do Middleware

O serviço básico fornecido por uma RSSF é a entrega dos dados coletados pelos sensores para as aplicações clientes. Tal entrega depende da descoberta das capacidades de sensoriamento disponíveis nos nós da rede, da solicitação dos dados pelas aplicações e da forma como se dá a comunicação entre os nós produtores de dados e a aplicação (consumidora de dados). O middleware proposto fornece para a aplicação uma abstração

desse serviço de entrega, de modo que o mesmo possa ser configurado de acordo com as diferentes necessidades de cada aplicação.

A descoberta das capacidades de sensoriamento fornecidas pela RSSF é realizada através do **serviço de descoberta** do middleware. A abstração do serviço básico de entrega de dados é provida pelo **serviço de comunicação** do middleware que, entre outras funções, fornece para a aplicação interfaces para acesso à rede. O comportamento do serviço de entrega é controlado pelos **serviços de configuração**, de **gerência** e de **adaptação** do middleware. Além disso, o sistema oferece **serviços genéricos** que atendem às necessidades básicas de todas as aplicações de RSSFs. A seguir os serviços fornecidos são detalhados. Cabe ressaltar que os serviços fornecidos individualmente pelos nós depende de sua categoria (sorvedouro, comum ou líder de cluster).

Serviço de Comunicação - Como visto na Seção 3.5, a comunicação do tipo síncrona (*Request/Response*) não é adequada para satisfazer os requisitos das redes de sensores. A comunicação assíncrona (por passagem de mensagem ou *Publish/Subscribe*) e dirigida a dados é mais adequada para o modelo de disseminação de dados requerido pelas aplicações de RSSFs. Neste tipo de comunicação, um fornecedor da informação (*Publisher*) publica as mensagens, que são enviadas a um ou mais assinantes (*Subscribers*). Esse modelo permite que mensagens sejam associadas a assuntos ou tópicos. Os assinantes recebem somente as mensagens referentes ao assunto o qual assinaram. O serviço de comunicação do middleware proposto segue o modelo *Publish-Subscribe* e é responsável por: anunciar os assuntos fornecidos pelos nós, ou seja, os tipos de dados que eles são capazes de sensoriar; publicar as mensagens que contêm os dados relacionados aos assuntos anunciados; receber as assinaturas por assuntos enviadas pelas aplicações clientes e intermediar a comunicação entre os demais serviços disponíveis no middleware. Sendo assim, o serviço de comunicação é responsável pelo estabelecimento de dois níveis de comunicação: (i) entre o middleware e o "mundo externo", repressentado pelas aplicações clientes e pelos protocolos de rede e dispositivos subjacentes; e (iii) entre os diversos componentes do middleware. Esse serviço está presente nas três categorias de nós, embora sua implementação difira de acordo com a categoria, como será visto no Capítulo 5.

Serviço de Descoberta – Há dois níveis de descoberta de serviços em uma RSSF: interno e externo à rede. O nível interno de descoberta de serviços é utilizado para que os

nós sorvedouros tenham conhecimento de todas as capacidades dos nós sensores que compõem a rede. A funcionalidade desse nível de descoberta é fornecida pelo serviço de comunicação, o qual, como visto, é responsável pelo anúncio dos assuntos fornecidos pelos nós. Já o nível externo de descoberta de serviços é usado por uma aplicação para descobrir quais RSSFs fornecem os serviços desejados por ela, e como acessar tais serviços. A descoberta da localização da RSSF e dos serviços disponibilizados por ela é controlada pelo serviço de descoberta do middleware, e seu funcionamento é fortemente acoplado às decisões de implementação do sistema, sendo, portanto, descrito no Capítulo 5. O serviço de descoberta é disponibilizado apenas pelos nós sorvedouros.

Serviço de Configuração – Esse serviço tem como função tomar decisões quanto à configuração da rede de sensores, ou seja, os protocolos de comunicação e a topologia lógica a serem adotados, a fim de satisfazer os requisitos de uma aplicação alvo. O fornecimento de um serviço de configuração pelo middleware visa facilitar o trabalho dos desenvolvedores de aplicações, liberando-os da tomada de decisões de baixo nível, referentes à infra-estrutura da rede. O serviço de configuração é fornecido pelos nós sorvedouros

Serviços de Gerência de Recursos e Gerência de Tarefas – Esses serviços tratam do gerenciamento dos recursos da rede de sensores e da alocação das tarefas de sensoriamento recebidas. Um dos principais objetivos do gerenciamento em RSSFs é otimizar o uso dos limitados recursos da rede, a fim de prolongar seu tempo de vida útil. O serviço de gerência de recursos tem como principal função selecionar os nós da rede que devem ser ativados para a execução da tarefa submetida por uma nova aplicação. Essa seleção deve procurar maximizar a relevância, do ponto de vista da aplicação, dos nós escolhidos para participar da tarefa de sensoriamento recebida. A inerente redundância da informação fornecida por redes de sensores densas deve ser explorada para encontrar o melhor compromisso entre qualidade do dado fornecido e consumo de energia. Tal serviço pode ser fornecido tanto pelos sorvedouros, em uma abordagem centralizada, quanto pelos sorvedouros, em uma abordagem distribuída. Já o principal objetivo do serviço de gerência de tarefas é controlar a admissão de novas aplicações, considerando seu custo, em termos de recursos da rede, de modo a garantir que todas as tarefas de sensoriamento em execução tenham seus requisitos de QoS respeitados. O controle de admissão é responsável por estabelecer o custo para a rede de cada tarefa de sensoriamento submetida por uma nova aplicação. Caso a rede possua recursos para

realizar tal tarefa com a QoS solicitada, a aplicação é admitida. Caso não tenha recursos suficientes, a aplicação pode ser rejeitada ou os requisitos de QoS negociados. Esse serviço é fornecido pelos nós sorvedouros.

Serviço de Inspeção e Adaptação – Esse serviço fornece para a aplicação uma representação de alto nível da RSSF e lhe permite inspecionar seu comportamento dinamicamente. Informações sobre o contexto de execução corrente são monitoradas pelo middleware e disponibilizadas para a aplicação através desse serviço, dessa forma tornando-a ciente do contexto. Além disso, esse serviço ativa políticas de adaptação sempre que for detectado um estado indesejável da rede ou se for solicitado pela aplicação. Esse serviço é principalmente fornecido pelos nós sorvedouros, entretanto a parte de monitoramento necessária para seu funcionamento é realizada por todas as categorias de nós.

Serviços Genéricos – Exemplos de serviços genéricos em uma RSSF são: nomeação, localização, segurança e agregação. O serviço de nomeação é usado sempre que for necessário um identificador único para cada nó da rede. Apesar de grande parte das aplicações de RSSFs não requerer a identificação individual de cada nó da rede, para muitos protocolos de nível de rede ou enlace, tal identificação torna-se necessária. Uma opção, nesses casos, é utilizar um número serial de dispositivo ou o endereço MAC do nó. Porém, tais identificadores possuem um tamanho grande em bytes, aumentando o tamanho das mensagens e, conseqüentemente, o consumo de energia nas transmissões. Como não há necessidade de identificadores globalmente únicos, mas sim únicos apenas dentro de uma RSSF, pode-se adotar um esquema de nomeação que forneça identificadores sob demanda para nós ativos. Nesses esquemas, um mesmo identificador pode ser reutilizado, quando o nó que o possui passa para o estado inativo [39]. Já o serviço de localização é útil quando é necessário que todos os nós conheçam sua posição geográfica e nem todos possuem um receptor GPS. Nesses casos, um algoritmo de localização, por exemplo, baseado em triangulação [93], pode ser implementado como um serviço fornecido pelo middleware, e executado sob demanda. Quanto ao serviço de segurança, além dos serviços típicos, como criptografia, confidencialidade, integridade, autenticação e não repúdio, alguns requisitos específicos são necessários em RSSFs, tais como atualidade ("*freshness*") dos dados e detecção de intrusos. Nos últimos anos, mecanismos de segurança adaptados para RSSF foram desenvolvidos e testados por vários grupos de pesquisa [105]. Tais mecanismos também podem ser implementados

como serviços do middleware, e executados sob demanda. O serviço de agregação implementa funções de agregação a serem aplicadas nos dados sensoriados. A agregação em RSSFs consiste em combinar dados oriundos de diferentes nós sensores a fim de ampliar o entendimento do fenômeno monitorado, eliminar dados espúrios gerados eventualmente por um sensor individual e, ao mesmo tempo, diminuir a quantidade de transmissões redundantes, dessa forma economizando energia. O sensoriamento de um fenômeno comum e a alta densidade dos nós pode gerar um elevado nível de redundância, o que favorece a agregação dos dados durante seu roteamento. O mecanismo de agregação pode ser uma operação trivial de supressão de dados duplicados ou pode envolver funções básicas, como valor máximo, mínimo ou médio de dados com a mesma semântica. Pode, ainda, incluir técnicas provenientes de modelos de fusão de dados, baseadas em métodos matemáticos e heurísticos, como as técnicas paramétricas de Inferência Bayesiana e de Dempster-Shafer, que permitem combinar dados com diferentes significados [107]. Funções de agregação também podem ser implementadas como serviços genéricos fornecidos pelo middleware. A categoria de nó responsável por fornecer um serviço genérico irá depender do serviço específico. Por exemplo, serviços de segurança e agregação devem estar disponíveis em todas as categorias de nós, enquanto um serviço de nomeação deve ser fornecido de forma centralizada, pelo sorvedouro ou por nós líderes de clusters.

A Figura 8 apresenta os componentes lógicos do sistema de middleware proposto. Na abordagem de projeto adotada, o middleware está localizado acima da camada de rede e de outras camadas de protocolos de nível inferior, bem como do sistema operacional e do hardware do nó. O sistema encapsula detalhes internos de funcionamento dos protocolos e dos dispositivos de hardware dos nós, fornecendo serviços de alto nível para as aplicações. O principal componente do middleware é o serviço de comunicação. Todos os demais serviços são diretamente conectados ao serviço de comunicação. Serviços adicionais podem ser incluídos no sistema desde que se conectem com o serviço de comunicação através das interfaces fornecidas, as quais serão descritas na Seção 4.2.2.

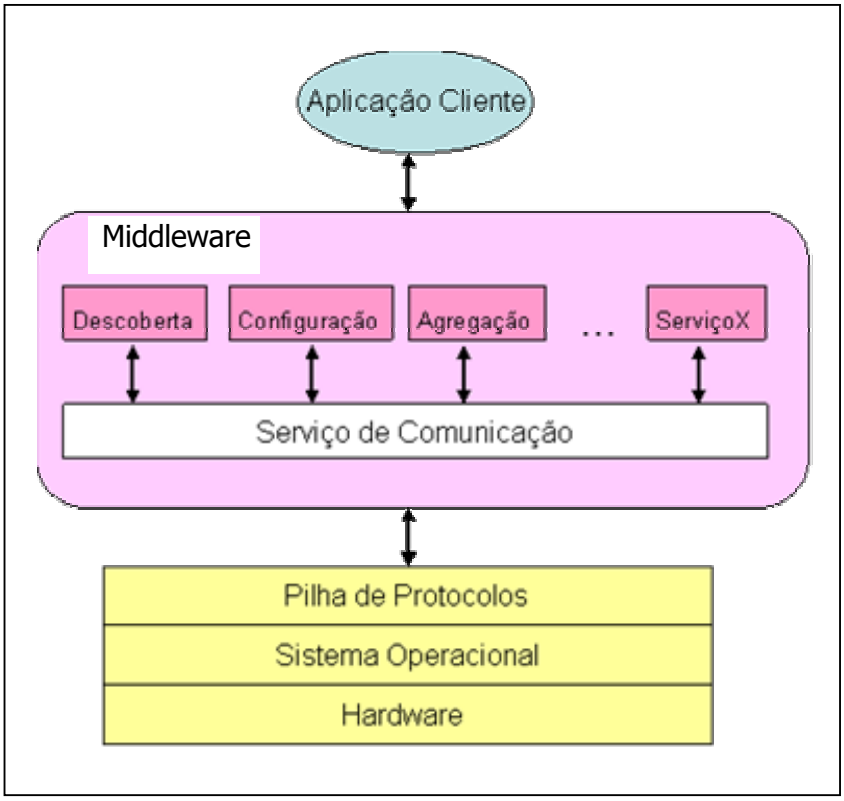


Figura 8: Componentes lógicos do sistema

4.2.2 Interfaces e Componentes Lógicos do Sistema

As funcionalidades dos serviços descritos na Seção 4.2.1 são fornecidas pelos diversos módulos componentes do middleware. A Figura 9 apresenta os principais componentes lógicos do sistema proposto e as interfaces entre os componentes e entre estes e o mundo exterior (aplicações, dispositivos sensores e protocolos da rede). Todas as interfaces são fornecidas pelo serviço de comunicação. A seguir são descritas as funções de cada interface, bem como as primitivas ou métodos de cada uma. Os componentes lógicos são implementados por módulos de software e serão detalhados no Capítulo 5, que descreve o modelo físico do sistema proposto.

Interface `Subscribe` – utilizada por aplicações clientes, que fazem o papel de assinantes (*Subscribers*), segundo o modelo de comunicação *Publish-Subscribe* adotado. Contém as primitivas, ou métodos, listadas abaixo.

- `SubscribeInterest` – primitiva usada pela aplicação para submeter um interesse, ou seja, uma assinatura por um assunto. Essa primitiva pode ser implementada de forma síncrona, no caso de uma consulta instantânea sobre o

fenômeno monitorado, ou de forma assíncrona, quando se deseja monitorar eventos ou realizar consultas de longa duração. No caso de uma consulta síncrona, uma chamada a primitiva `SubscribeInterest` recebe como retorno os dados consultados.

- `ReceiveResults` – primitiva usada para notificar a aplicação sobre os dados publicados pelos sensores, no caso de consultas assíncronas.

- `SubmitExecPolicy` – primitiva usada pela aplicação para submeter uma política de execução, a qual contém informações sobre o comportamento do sistema ao executar o serviço solicitado. Políticas de execução serão detalhadas na Seção 6.2.

- `SubmitQoSParameters` – primitiva usada pela aplicação para submeter os parâmetros de QoS a serem respeitados durante a coleta e publicação dos dados solicitados.

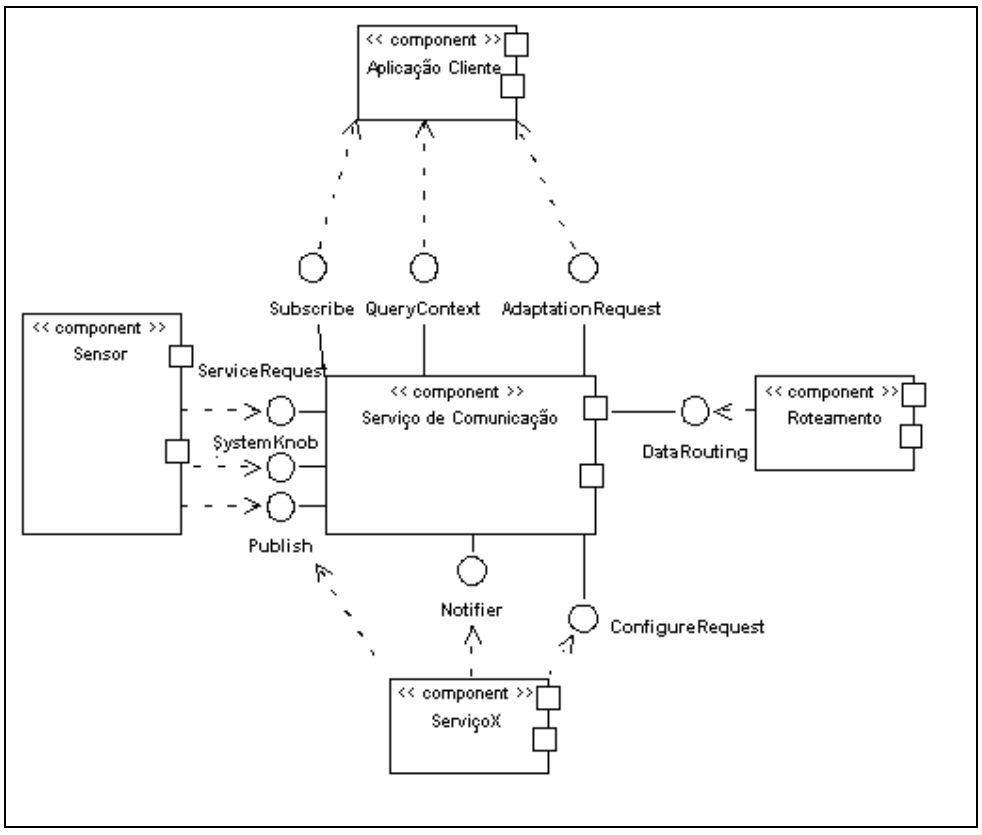


Figura 9: Componentes lógicos e interfaces do sistema proposto

Interface `QueryContext` – usada pelas aplicações clientes para inspecionar o estado do sistema. Contém apenas uma primitiva, `Request_Inspection`, com parâmetros de entrada e parâmetros de retorno.

Interface `AdaptationRequest` – usada pelas aplicações clientes para solicitar a ativação de alguma política de adaptação. Contém apenas uma primitiva, `Request_Adptation`, apenas com parâmetros de entrada.

Interface `Publish` – usada pelos sensores para anunciar suas capacidades e publicar seus dados, e por serviços do middleware para publicar os resultados de seus processamentos. Composta pelas primitivas listadas abaixo.

- `PublishConfiguration` – primitiva usada pelos nós sensores para anunciar suas capacidades de sensoriamento, ou seja, os assuntos que eles estão aptos a fornecer.

- `PublishData` - primitiva usada pelos nós sensores para publicar seus dados.

- `PublishResults` – primitiva usada pelos serviços do middleware para publicar para o serviço de comunicação o resultado do processamento realizado em dados recebidos (do nó local ou provenientes da rede).

Interface `Notifier` – usada para notificar os serviços do middleware sobre a chegada de dados ou a chegada de um assunto assinado por uma aplicação. Composta pelas três primitivas listadas abaixo.

- `LocalDataArrival` – primitiva usada para notificar os serviços do middleware sobre a geração de um dado no nó local.

- `NetworkDataArrival` - primitiva usada para notificar os serviços do middleware sobre a chegada de um dado da rede (proveniente de algum nó vizinho).

- `InterestArrival` - primitiva usada para notificar os serviços do middleware sobre a chegada de um interesse de uma aplicação, ou seja, de uma assinatura por um assunto.

Interface `ConfigureRequest` – usada por um serviço para indicar o formato a ser usado pelos nós sensores para solicitar a sua execução (no caso de serviços que requeiram sua solicitação explícita, como, por exemplo, o serviço de nomeação, no qual um nó solicita ao serviço um nome único dentro da rede). Possui uma única primitiva `Configure`.

Interface `DataRouting` – usada para a troca de dados entre os protocolos de roteamento e o middleware. Suas primitivas são listadas abaixo.

- `Configure` – primitiva usada pelo serviço de comunicação para enviar para a camada de rede informações de configuração, geradas pelo serviço de configuração do middleware.

- `SendMHop` - primitiva usada pelo serviço de comunicação para enviar dados para a rede utilizando o protocolo *multihop* de disseminação de dados selecionado pelo serviço de configuração.

- `SendBCast` - primitiva usada pelo serviço de comunicação para enviar dados em *broadcast* na rede.

- `Receive` – primitiva usada pelo serviço de comunicação para receber dados endereçados ao nó local.

- `Intercept` - primitiva usada pelo serviço de comunicação para interceptar dados provenientes da rede e destinados a outros nós.

Interface `ServiceRequest` – usada pelos sensores para solicitar algum serviço do middleware que requeira um pedido explícito para sua ativação, por exemplo, o serviço de nomeação. Contém apenas uma primitiva, `Request_Service`, cujos dados de entrada consistem no identificador do serviço solicitado e nos parâmetros necessários para o serviço.

Interface `SystemKnobs` – permite a interação com os dispositivos de hardware do sistema. Possui as primitivas listadas a seguir.

- `ChangeProcessorState` – usada pelo middleware para alterar o estado do processador do nó sensor local. Recebe como parâmetro os valores "*On*" ou

"*Off*", indicando se o processador deve ser ligado ou desligado e, opcionalmente, um valor para a voltagem a ser usada, no caso de sistemas que permitem DVS [104].

- `ChangeReceiverState` - usada pelo middleware para alterar o estado do receptor do nó sensor local, entre ligado e desligado ("*On*" ou "*Off*").

- `ChangeTrasmitterState` - usada pelo middleware para alterar o estado do transmissor do nó sensor local. Recebe como parâmetro: (i) os valores "*On*" ou "*Off*", indicando se o transmissor deve ser ligado ou desligado; (ii) um valor indicando a taxa de transmissão a ser usada; (iii) e um valor indicando a potência a ser usada no sinal, quando o sistema permite controle de potência.

- `ChangeSensorState` - usada pelo middleware para alterar o estado do dispositivo de sensoriamento do nó local. Recebe como parâmetro os valores "*On*" ou "*Off*", indicando se o dispositivo deve ser ligado ou desligado, e um valor indicando a taxa de sensoriamento a ser usada.

Após definir os principais componentes e interfaces lógicos do sistema, a próxima seção descreve como a arquitetura de RSSFs considerada no trabalho conforma-se ao padrão SOA [53].

4.3 ARQUITETURA DA RSSF SEGUNDO O PADRÃO SOA

Como visto anteriormente, na abordagem proposta a RSSF é vista como uma fornecedora de serviços para aplicações clientes. O objetivo desta seção é mostrar como os componentes arquiteturais da RSSF aderem ao padrão SOA [53], usado no projeto de arquiteturas orientadas a serviços. Mostra-se como as operações e os papéis definidos no padrão SOA são mapeados para a arquitetura de rede utilizada no presente trabalho.

Uma aplicação cliente consultando dados de uma RSSF desempenha o papel de um solicitante de serviços. Nós sorvedouros atuam primariamente como fornecedores de serviços para o meio externo. Seu endereço de rede (URL) é registrado em um serviço de registros UDDI como sendo o ponto de acesso para o Serviço Web oferecido pela rede (operação *publish*). Nós sorvedouros fornecem as descrições de serviços da rede como um todo, através de um documento WSDL, e oferecem acesso a tais serviços. Ao mesmo tempo, sorvedouros atuam como solicitantes para nós sensores, requisitando seus serviços

especializados a fim de satisfazer as necessidades das aplicações. Nós sensores são fornecedores de serviços, fornecendo o serviço básico de entrega de dados e serviços específicos, implementados pelos módulos do middleware, tais como segurança e agregação. Sensores enviam suas descrições de serviços para sorvedouros, dessa forma executando uma operação de publicação (*publish*). Sorvedouros mantêm um repositório com descrições de serviços de cada tipo de sensor existente na rede. Aplicações clientes realizam uma operação de descoberta de serviços (*find*) junto a um registro UDDI, a fim de localizar uma RSSF que atenda a seus propósitos. Como resultado da operação *find*, a aplicação obtém o endereço do sorvedouro da rede desejada. A partir de então, ela pode contatar diretamente o sorvedouro, obter o documento WSDL da rede, e submeter sua consulta. Quando uma aplicação submete uma consulta para a RSSF, ela na verdade está executando uma operação *bind* com os serviços fornecidos pelos nós sensores. Entretanto, a aplicação interage apenas com o nó sorvedouro (Figura 10).

Como visto na Seção 3.3.1, além de aderirem ao padrão SOA, Serviços Web podem ser decompostos segundo três conjuntos de especificações. No sistema de middleware proposto, a comunicação é realizada pelo protocolo SOAP e um protocolo de disseminação de dados subjacente. A descrição de serviços é baseada em documentos WSDL e Esquemas XML. As funcionalidades de publicação e descoberta são realizadas pelo módulo de descoberta de serviços.

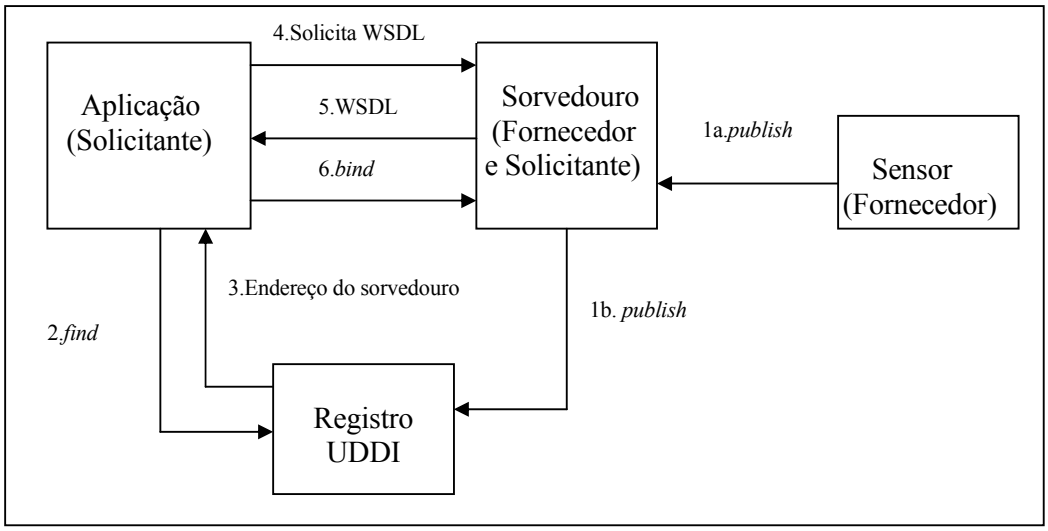


Figura 10: Operações segundo o padrão SOA

Após apresentar os componentes físicos e organizacionais das RSSFs considerados neste trabalho, bem como o modelo lógico do sistema proposto, no próximo capítulo apresenta-se a solução adotada em seu projeto físico.

5 Modelo Físico do Middleware para RSSFs Proposto

Tecnologias da área de Serviços Web foram empregadas na especificação concreta e na implementação dos componentes do middleware proposto [34,37]. Tal decisão de projeto foi motivada pelo fato das tecnologias de Serviços Web basearem-se em protocolos e padrões ubíqüos na Web, o que facilita a integração do sistema com as aplicações clientes. Apesar de alguns usuários estarem interessados em acessar os dados da RSSF através de uma interface gráfica, que disponibiliza um número fixo de consultas pré-definidas, grande parte das aplicações deseja obter o conjunto de dados gerados e pré-processados pela rede como entrada para seus módulos de análise e tratamento de informações. Tal tipo de interação caracteriza uma interface do tipo aplicação-aplicação. A arquitetura de Serviços Web fornece uma solução viável para capacitar esse tipo de interoperabilidade de aplicações.

Ao se usar as tecnologias de Serviços Web no projeto físico e na implementação do sistema de middleware, os diferentes serviços oferecidos pelo sistema são expostos como Serviços Web. As interfaces lógicas descritas na Seção 4.2.2 são descritas fisicamente através de documentos WSDL [141] e de Esquemas XML (XML *Schemas* [137]) e as mensagens trocadas entre os componentes externos e internos do sistema (ou seja, as primitivas de serviços) são implementadas como mensagens SOAP ou XML.

Os componentes do modelo físico do middleware proposto diferem em função da categoria dos nós (sensor/líder/sorvedouro). Todas as categorias de nós possuem um módulo de comunicação, que implementa o serviço de comunicação, e vários módulos implementando os demais serviços disponibilizados pelo middleware (Figuras 11, 12 e 13). Os nós sorvedouros, por serem mais robustos e representarem o ponto de acesso à rede, foram projetados com base na especificação da arquitetura de Serviços Web [53]. Portanto, nesses nós a implementação do módulo de comunicação é baseada no protocolo SOAP. A fim de evitar o *overhead* imposto pelo uso do SOAP, o módulo de comunicação dos nós sensores é baseado no uso de mensagens XML formatadas segundo Esquemas especialmente definidos para torná-las o mais compactas possível. Portanto, dentro da rede, mensagens XML são diretamente encapsuladas na carga útil dos pacotes de dados do protocolo de camada de rede. Opcionalmente, a fim de reduzir ainda mais o *overhead*,

o formato XML binário, WBXML (*Wireless Binary XML* [138]), pode ser utilizado dentro da rede, exclusivamente para as mensagens trocadas entre os sensores. O uso de WBXML dentro da rede foi explorado na fase de implementação do protótipo do sistema, e está descrito no Capítulo 8.

Também faz parte da arquitetura do middleware proposto um *proxy* SOAP, usado para implementar a comunicação com as aplicações clientes, e um conjunto de *drivers* XML para implementar a comunicação com os protocolos de rede e dispositivos subjacentes. As Seções 5.1 e 5.2 detalham os módulos componentes do modelo físico do sistema proposto, destacando, sempre que necessário, as diferenças entre cada categoria funcional de nó criada no presente trabalho.

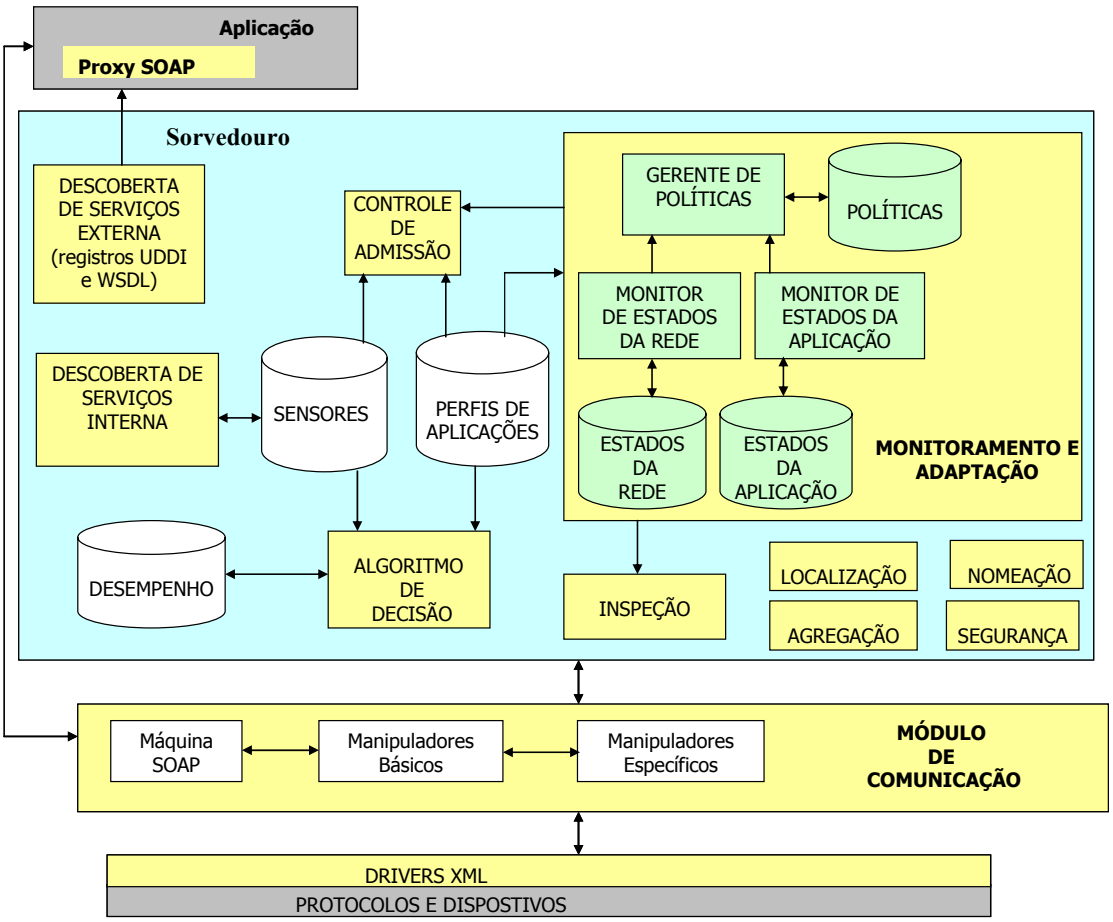


Figura 11: Módulos Componentes do Middleware no Nó Sorvedouro

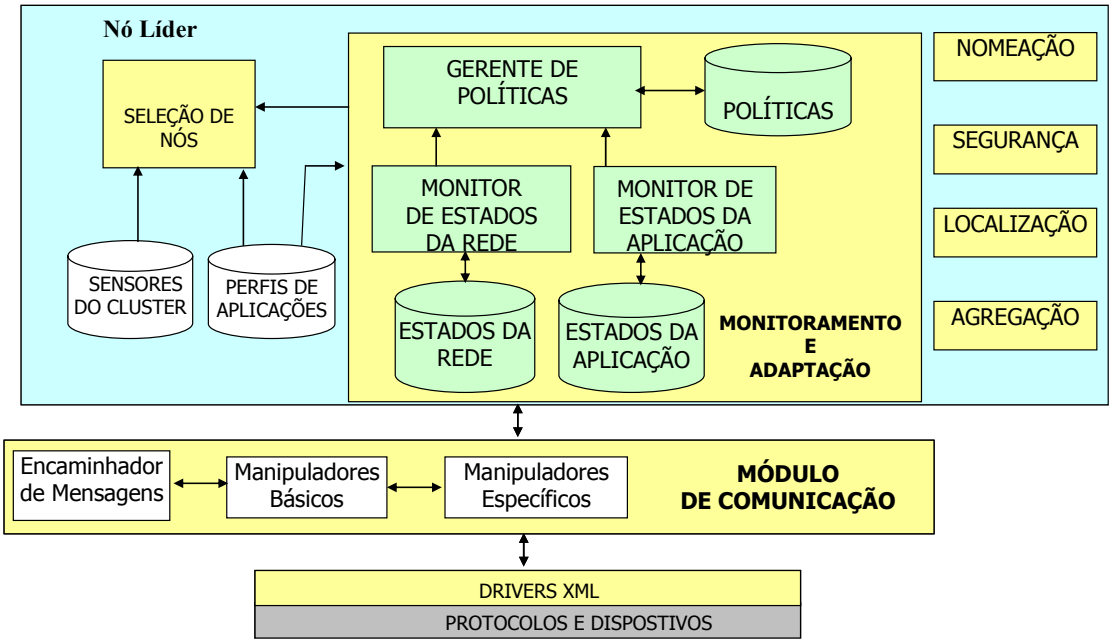


Figura 12: Módulos Componentes do Middleware no Nó Líder de *Cluster*

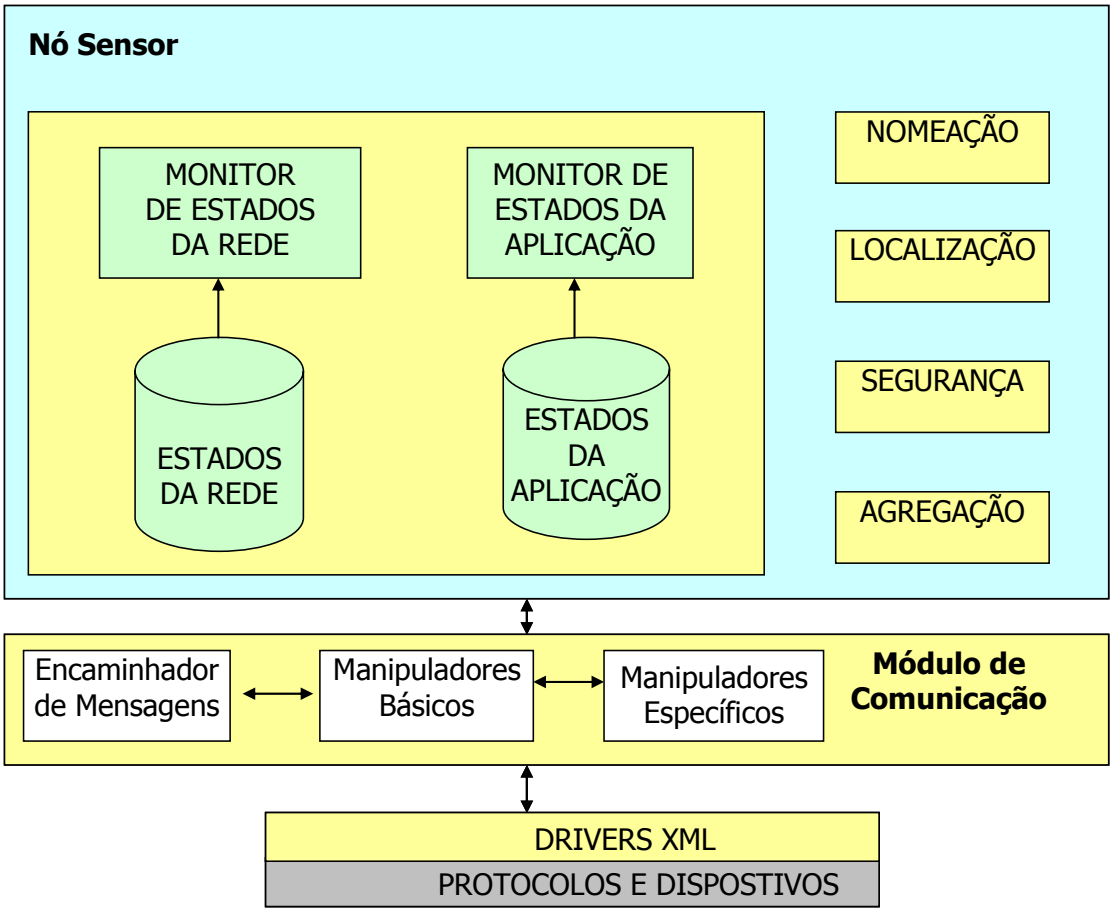


Figura 13: Módulos componentes do middleware no Nó Sensor

5.1 MÓDULO DE COMUNICAÇÃO

É o módulo responsável pela implementação do serviço de comunicação do middleware. No sistema proposto, a comunicação com a aplicação é baseada no protocolo SOAP. O módulo inclui um *proxy* SOAP para o estabelecimento da comunicação com a aplicação e *drivers* XML para a interação do middleware com os protocolos de rede e dispositivos subjacentes. O tratamento das mensagens e seu encaminhamento para os serviços do middleware é feito por componentes baseados em SOAP nos nós sorvedouros e por componentes baseados em XML nos nós sensores (comuns e líderes).

***Proxy* SOAP**. A comunicação da RSSF com a aplicação ocorre nos nós sorvedouros. Todas as mensagens trocadas entre aplicações e a RSSF são representadas em linguagem XML e encapsuladas como mensagens SOAP. *Proxies* SOAP são programas que transformam chamadas de funções feitas na linguagem de programação da aplicação em mensagens SOAP de invocação de operações dos serviços da rede. Da

mesma forma, mensagens SOAP de resposta são convertidas para dados e/ou chamadas de funções na linguagem da aplicação. Os *proxies* SOAP fazem uso das primitivas das interfaces lógicas `Subscribe`, `QueryContext` e `AdaptationRequest`, as quais são implementadas como operações do módulo de comunicação e descritas no documento WSDL da rede. Os *proxies* invocam tais operações em benefício da aplicação e recebem suas respostas. Como SOAP é um protocolo Web padrão, há várias ferramentas comercialmente disponíveis [130,147] que geram automaticamente *proxies* SOAP para muitas linguagens de programação, a partir de um documento WSDL fornecido. *Proxies* SOAP, apesar de considerados componentes do sistema de middleware, são implementados como parte do programa da aplicação.

***Drivers* XML**. Para a interoperabilidade do middleware com as várias implementações de protocolos de comunicação existentes para RSSFs, um conjunto de *drivers* é implementado. Os *drivers* fazem a tradução de mensagens e comandos do middleware para a linguagem específica dos protocolos e vice-versa. Para a maior flexibilidade do sistema, a interface com o middleware (API) é baseada em XML. Então, similarmente a *proxies* SOAP, *drivers* para diferentes protocolos consistem em módulos de software que convertem chamadas de funções, segundo a linguagem de programação de um dado protocolo de comunicação, em chamadas de operações definidas pela API do middleware, e vice-versa. Da mesma forma, os *drivers* convertem mensagens XML geradas pelo middleware em formatos de dados proprietários de um dado protocolo de comunicação, e vice-versa. Os *drivers* implementados possuem uma arquitetura em duas camadas. A primeira camada representa a comunicação com o lado do middleware e corresponde à implementação das primitivas da interface lógica `DataRouting`, fornecida pelo módulo de comunicação. As primitivas fornecem a abstração necessária para ativar a execução e configurar os protocolos de comunicação de acordo com a tarefa da rede. Essa camada tem uma implementação fixa, e depende apenas do Esquema XML fornecido pelo middleware. O Esquema XML define a gramática, ou seja, as regras que devem ser seguidas para escrever as mensagens XML no sistema (nomes de *tags*, estrutura e modelo de conteúdo). Enquanto uma mensagem XML contém a informação estruturada a ser trocada entre os pares comunicantes, o Esquema XML define a sintaxe dessa estrutura.

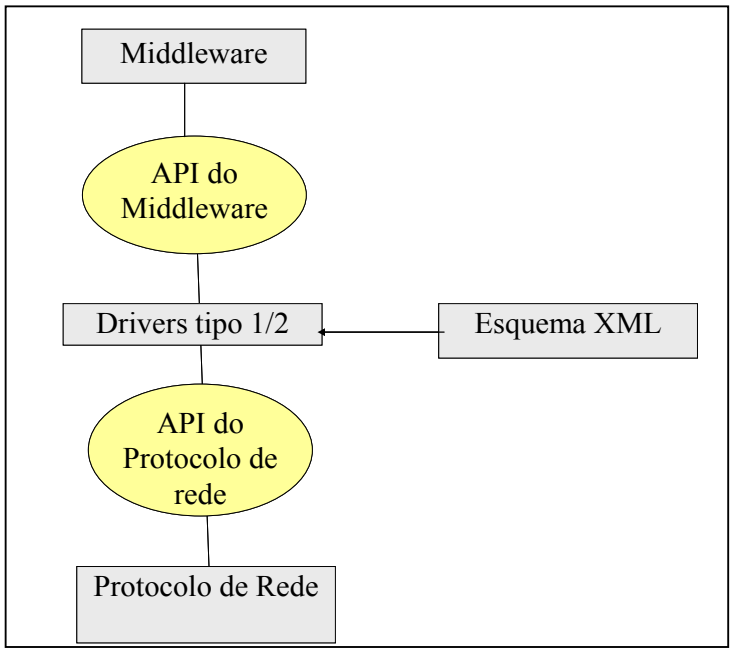


Figura 14: *Drivers* XML

A segunda camada representa a comunicação com o lado dos protocolos. Ela é totalmente dependente do protocolo em uso (portanto, é variável) e consiste em uma camada de software que implementa a lógica da conversão para os comandos e formatos de dados dos protocolos subjacentes (Figura 14). Essa conversão é feita a partir da API fornecida pelos protocolos de rede.

A segunda camada define também os procedimentos necessários para o transporte dos dados pelos protocolos. Tais procedimentos são responsáveis por encapsular mensagens XML na carga útil do pacote do protocolo de disseminação de dados subjacente, de acordo com o formato do pacote. Há duas situações possíveis. Na primeira, o protocolo de rede subjacente utiliza informações do conteúdo da mensagem para o seu algoritmo de roteamento. Nesse caso, a estratégia mais eficiente consiste em converter a mensagem XML para o formato adotado pelo protocolo (*driver* tipo 1). Na segunda situação, o protocolo transporta a carga útil sem tomar conhecimento de seu conteúdo em suas decisões de encaminhamento de mensagens. Nesse caso, o *driver* apenas configura os campos necessários do cabeçalho e insere a mensagem XML diretamente no corpo do pacote de dados do protocolo (*driver* tipo 2).

Para fornecer a QoS solicitada pela aplicação e permitir a adaptação do comportamento da rede aos vários contextos de execução, o middleware deve ser capaz de interagir também com os dispositivos de hardware, como sensores e transmissores. A

interação com dispositivos sensores ocorre quando dados são gerados localmente e enviados para o middleware, e quando algum mecanismo de controle *(knob)* do sistema tem que ser ativado. Os mecanismos de controle implementados para sensores permitem que o transdutor seja ligado e desligado e que a taxa de sensoriamento de dados seja definida. Os mecanismos de controle implementados para transmissores e receptores permitem: (i) alterar seu estado, entre ligado e desligado, (ii) modificar a taxa de transmissão, e (iii) configurar a potência usada no sinal para transmitir.

Da mesma forma que ocorre para a comunicação com os protocolos de rede, o middleware especifica uma API que abstrai o comportamento dos *knobs* do sistema. Essa API consiste na implementação das interfaces lógicas `Publish` e `SystemKnobs`, descritas na Seção 4.2.2. Os fabricantes de dispositivos devem fornecer uma API para interagir com os mesmos. *Drivers* para a conversão entre a API do middleware e a dos dispositivos podem, então, ser construídos a partir das especificações fornecidas.

Componentes SOAP. A comunicação baseada em SOAP é implementada apenas em nós sorvedouros e o módulo de comunicação nesses nós é composto pela máquina SOAP e por um conjunto de manipuladores. A máquina SOAP é responsável por coordenar o fluxo das mensagens SOAP através dos vários manipuladores e por garantir que a semântica do protocolo SOAP seja respeitada. Os manipuladores representam a lógica de processamento das mensagens e agem como despachantes para os vários serviços fornecidos pelo middleware. Manipuladores comportam-se como interceptadores de mensagens SOAP e podem ser invocados antes ou depois da mensagem ser entregue ao componente que implementa o serviço. Um manipulador analisa os campos do cabeçalho das mensagens os quais indicam os serviços que devem ser executados e despacha os pacotes para os componentes que implementam tais serviços. Cadeias de manipuladores podem ser especificadas para indicar a ordem de execução quando vários serviços devem ser executados em uma mesma mensagem. No sistema proposto, há um conjunto de manipuladores básicos e um conjunto de manipuladores específicos. Os manipuladores básicos são responsáveis pela serialização de mensagens, pelo processamento de cabeçalhos e pelas conversões de dados. Um analisador e um escritor de mensagens são implementados como parte dos manipuladores básicos. O analisador tem a função de extrair e interpretar a informação transportada por uma mensagem SOAP. O escritor tem a função de construir mensagens SOAP sempre que necessário. Os

manipuladores específicos são definidos para cada serviço especializado implementado pelo middleware. Por exemplo, pode ser definido um manipulador para encaminhar as mensagens de dados recebidas da rede para um serviço de agregação, ou pode ser definido um manipulador para encaminhar partes das mensagens de interesses para um serviço de criptografia. Os manipuladores específicos são responsáveis por configurar e buscar, no cabeçalho das mensagens, informações que forneçam indicações sobre o tratamento que deve ser dado às mesmas.

Componentes XML. Para evitar o *overhead* do uso do protocolo SOAP, optou-se por trafegar dentro da RSSF mensagens XML formatadas segundo uma especificação baseada em SOAP, porém mais compacta, tornando seu processamento mais leve. Um exemplo de mensagem XML no formato utilizado pode ser visto na Figura 15 e seu respectivo Esquema XML pode ser visto no Anexo 1. Como a questão de garantir a interoperabilidade pode ser relaxada no interior da rede, assume-se o uso de um mesmo espaço de nomes (*namespace*), previamente conhecido, em todas as mensagens trocadas pelos sensores. Dessa forma, os *namespaces* foram eliminados da mensagem. Além disso, as *tags* que definem a estrutura da mensagem foram abreviadas, tornando a mensagem mais compacta. O módulo de comunicação baseado em XML é, então, o componente nos nós sensores similar ao módulo baseado em SOAP em nós sorvedouros. Ele é composto por um encaminhador de mensagens e por um conjunto de manipuladores. O encaminhador de mensagens é uma versão mais leve da máquina SOAP, sendo responsável por coordenar o fluxo das mensagens através dos vários manipuladores dentro do módulo de comunicação baseado em XML. Os manipuladores são os despachantes para os serviços fornecidos pelo middleware nos nós sensores. Os manipuladores básicos desempenham o mesmo papel que seu correspondente no módulo baseado em SOAP. Um analisador XML e um escritor XML são implementados como parte dos manipuladores básicos.

```
<Env>
<Header>
</ Header>
<Body>
  <PublishSensorDesc>
    <parameter ID="NODE_MAC_ADDRESS" NetworkID="NODE_NETWORK_ID">
        <TTL unit="Seconds">3600</TTL>
        <Type>Motion</Type>
        <DataDomain>
          <Value>Four Legged Animal</Value>
          <Value>Two Legged Animal</Value>
          <Value>Creeper Animal</Value>
        </DataDomain>
        <GeographicLocation unit="LatLong">
```

```
          <x>35.00</:x>
          <y>-23.00</y>
        </GeographicLocation>
        <Energy unit="J">1</Energy>
        <Confidence>
          <Max>1.0</Max>
          <Min>0.2</Min>
        </Confidence>
        <DataRate unit="mSeconds">
          <Max>10</Max>
          <Min>1000</Min>
        </DataRate>
      </parameter>
    </ PublishSensorDesc >
  </Body>
</Env>
```

Figura 15: Formato XML Compacto – Mensagem `PublishSensorDescription`

Na versão atual do middleware, foram definidos manipuladores específicos que tratam da comparação entre dados gerados por sensores e interesses emitidos por aplicações. Essa função de comparação é útil para uma grande gama de protocolos de disseminação de dados em RSSFs. O manipulador `Matching_Data` foi especificamente designado para o envio e a recepção de mensagens através de um protocolo subjacente baseado em interesses e centrado em dados, como por exemplo os protocolos [39,66,74]. Ele é responsável por verificar se o conteúdo semântico das mensagens de dados recebidas ou geradas em um nó combina com o conteúdo semântico do interesse emitido por uma aplicação. O resultado da comparação é usado pelo protocolo de roteamento em suas decisões de encaminhamento. O manipulador `Parse_Interest` realiza uma função de comparação entre as características de um sensor (tipo de dispositivo de sensoriamento e localização) e um interesse recebido.

Os dois manipuladores, `Matching_Data` e `Parse_Interest`, somente são usados quando o protocolo subjacente não fornece sua representação própria para dados e interesses. Nesses casos, o middleware oferece uma representação padrão, baseada em SOAP e XML. Nessa representação, interesses são representados por mensagens SOAP quando passados da aplicação para o nó sorvedouro e são representados em XML dentro da rede. Dados são representados por mensagens XML dentro da rede e por mensagens SOAP quando transferidos para a aplicação. Por outro lado, se o protocolo usa um formato proprietário para dados e interesses, as funções de comparação são realizadas em nível de rede, pelo próprio protocolo.

Outro manipulador específico definido é o `Matching_Aggregation`. Tal manipulador é responsável por representar a ativação de funções de agregação

implementadas como componentes de serviços do middleware. Um serviço de agregação é executado sempre que dados que correspondam a um conjunto de atributos especificados são gerados ou recebidos em um nó sensor. Solicitações por funções de agregação são enviadas como parte dos interesses submetidos pela aplicação e são registradas nos nós que participam da tarefa de sensoriamento corrente da rede. Quando um sensor gera um dado, os atributos do dado são comparados com os atributos definidos em todas as solicitações de agregação registradas no nó. Essa função de comparação é realizada pelo `Matching_Aggregation`. Se o resultado da comparação for positivo, a mensagem é despachada para o serviço de agregação correspondente à solicitação, antes de ser enviada para outros nós pelo protocolo de comunicação subjacente. O mesmo procedimento é aplicado aos dados recebidos em um nó sensor. Como a agregação é implementada como um serviço do middleware, todos os dados que devem sofrer agregação precisam ser convertidos para XML.

O manipulador específico `Monitor_State` foi definido para os serviços de inspeção e adaptação. Em resposta a uma solicitação da aplicação para inspecionar o sistema, ou a uma solicitação do middleware para verificar o estado atual da rede e a necessidade ou não de se efetuar um procedimento de adaptação, informação de contexto deve ser enviada pelos nós sensores. Ao inserir informação de contexto em uma mensagem de dado, o nó configura um campo no cabeçalho, indicando a existência dessa informação, e o seu tipo, ou seja, se a informação refere-se ao estado da rede ou da aplicação. O manipulador `Monitor_State` verifica o cabeçalho de todas as mensagens de dados em busca do conteúdo desse campo. Caso exista informação de contexto na mensagem, o manipulador a despacha para o componente de monitoramento responsável pelo seu processamento.

5.2 MÓDULOS DE SERVIÇOS DO MIDDLEWARE

No sistema proposto, as interfaces de serviços fornecidas pelo middleware para o meio externo, ou seja, aplicações e outros serviços, são descritas através de documentos WSDL. Desta forma, através de um documento WSDL é definido o formato das mensagens usadas para submeter interesses e requisitos de QoS das aplicações, bem como das mensagens solicitando inspeção e adaptação do sistema (Figura 16). Além disso, o sistema proposto define um documento WSDL que descreve as interfaces disponíveis para desenvolvedores de novos serviços a serem incorporados ao middleware, bem como

esquemas XML que descrevem as interfaces disponíveis para desenvolvedores de protocolos de roteamento para RSSFs. As primitivas descritas nas interfaces lógicas correspondem a definições de operações, as quais são invocadas através de mensagens SOAP ou XML.

Os vários serviços fornecidos pelo middleware são implementados como componentes de software modulares. Como foram projetados em módulos independentes, dependendo dos recursos computacionais dos nós, um diferente conjunto de serviços pode ser instalado. Dependendo da categoria funcional do nó (sensor ou líder), diferentes módulos podem ser carregados e descarregados dinamicamente do ambiente de execução.

Como visto na descrição da arquitetura lógica do sistema, o serviço de descoberta possui dois níveis: um nível de descoberta interna, que faz parte do módulo de comunicação, e um nível de descoberta externa, que será descrito a seguir. Os documentos WSDL da rede (Anexos 2 e 3) fazem parte do módulo de descoberta de serviços externa.

O serviço de configuração, utilizado para auxiliar o desenvolvimento de aplicações para RSSFs, é implementado pelo módulo de decisão. O módulo de decisão é responsável por tomar as decisões de configuração necessárias para satisfazer um determinado conjunto de requisitos de aplicação.

O serviço de inspeção e adaptação é implementado como dois módulos independentes: o módulo de inspeção permite à aplicação inspecionar o comportamento da rede dinamicamente, fornecendo uma representação do seu estado corrente. O módulo de monitoramento e adaptação é responsável pelo monitoramento dos estados do sistema (rede e aplicação) e pela ativação de políticas de adaptação sempre que for necessário ou desejado pela aplicação.

O serviço de gerência de recursos e tarefas também é implementado como dois módulos independentes: um módulo de controle de admissão e um módulo de seleção de nós ativos.

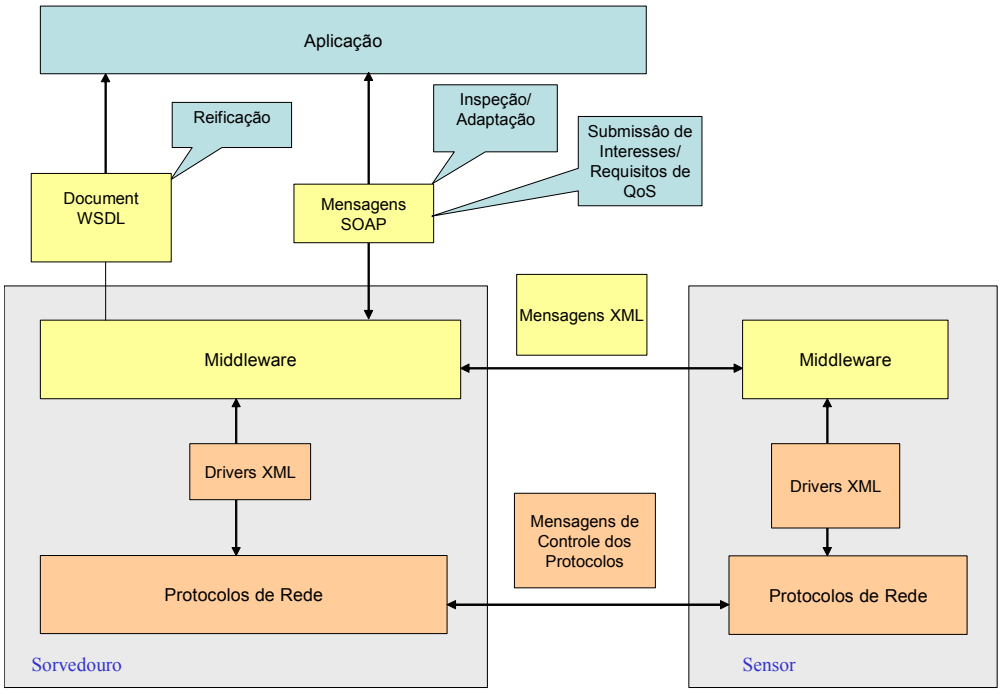


Figura 16: Comunicação entre os componentes do sistema

Cada serviço genérico fornecido pelo middleware é implementado como um módulo separado no sistema proposto. A implementação dos módulos de serviços genéricos pode ser fornecida por terceiros. O uso pelo middleware de APIs baseadas em XML permite que os serviços sejam facilmente incorporados ao sistema, desde que implementem as interfaces disponíveis. Qualquer serviço fornecido pelo middleware que tenha que ser executado sobre todos os pacotes da rede, como por exemplo, segurança (criptografia), precisa ter um manipulador específico correspondente.

A seguir são descritos os módulos de serviços especificados para o sistema proposto e em que categoria de nó cada módulo deve ser implementado.

Módulo de Descoberta Externa de Serviços: A utilização de SOAP e XML, ambos parte da arquitetura de Serviços Web [53], faz com que o padrão UDDI [8] seja uma escolha natural como protocolo de descoberta de serviços a ser usado pelas aplicações. Como visto na Seção 3.3.1, UDDI é um protocolo para comunicação com registros de serviços [42]. Para permitir o acesso e uso da RSSF por aplicações, os nós sorvedouros implementam um Serviço Web com todas as funcionalidades oferecidas pela rede e mantêm um repositório com os documentos WSDL que descrevem as interfaces de

tal serviço (Anexo 2). As aplicações clientes, após localizarem o nó sorvedouro utilizando o protocolo UDDI, obtêm o documento WSDL a fim de aprenderem o formato das mensagens para acessar a rede. Da mesma forma, desenvolvedores de novos serviços obtêm o documento WSDL que descreve o formato das mensagens para permitir a integração de seus serviços com o middleware (Anexo 3). O módulo de descoberta externa é composto pelos documentos WSDL e pela especificação dos registros necessários para a publicação do Serviço Web da rede no UDDI.

Módulo de Decisão. Para tomar as decisões quanto à configuração da rede, o sistema possui um módulo de decisão [36], que se comunica com o módulo SOAP para receber as mensagens das aplicações. Esse módulo é implementado apenas em nós sorvedouros. A configuração da rede consiste na escolha do protocolo de disseminação de dados e da topologia lógica da rede a serem utilizados. Tal escolha é realizada por um algoritmo de decisão, apresentado no Capítulo 7. A construção do algoritmo foi baseada nos resultados reportados na literatura e por simulações realizadas. Além do algoritmo de decisão, o módulo acessa uma base de dados de desempenho e uma base de dados de perfis das aplicações. A base de dados de desempenho contém valores médios de parâmetros de desempenho da rede, como latência, acurácia e gasto de energia para cada aplicação em execução ou anteriormente executada na rede. A base de dados com os perfis das aplicações contém a descrição dos interesses de cada aplicação ativa na rede ou já executada, seus requisitos de QoS e políticas de execução e adaptação utilizadas. Os perfis são estruturas de dados compartilhadas pelos módulos de decisão, seleção, adaptação e inspeção. O módulo de decisão usa informação das bases de dados como entrada para seu algoritmo de decisão. Tal informação é freqüentemente refinada quando novos valores são reportados pela aplicação ou a execução de uma aplicação é encerrada.

Módulo de Monitoramento e Adaptação. É o módulo responsável por permitir a adaptação dinâmica do comportamento da rede em função dos diferentes contextos ou estados de execução. Para isso, é preciso realizar o monitoramento das variáveis que determinam tais estados. Como no sistema proposto o estado de execução é definido em termos do estado da rede e do estado da aplicação, há um componente de monitoramento para as variáveis de cada uma dessas categorias de estado. As informações de estado obtidas são armazenadas em uma base de dados do estado da aplicação e em uma base de dados do estado da rede.

Além dos componentes de monitoramento, há um gerente de políticas, responsável por selecionar a política de execução inicial da rede, conforme solicitada pela aplicação, e as possíveis políticas de adaptação, quando necessário (devido à violação de alguma parâmetro de QoS, por exemplo) ou quando solicitado pela aplicação. O sistema mantém uma base de dados contendo as descrições das diferentes políticas existentes. O documento WSDL (Anexo 2) fornece para a aplicação o formato da mensagem SOAP para solicitar a ativação de alguma política de adaptação. A invocação dessa mensagem SOAP corresponde a uma chamada à primitiva `Request_Adaptation`, da interface `AdaptationRequest`.

Há duas possíveis abordagens para a estratégia de adaptação: centralizada ou distribuída. Na abordagem centralizada, apenas os componentes de monitoração dos estados precisam ser implementados em todos os nós. Esses componentes contêm informação de estado referente ao próprio nó e a lógica de ativação de envio de mensagens de monitoramento. O gerente de políticas, bem como suas estruturas de dados, são mantidas apenas nos sorvedouros. Essa solução é viável para redes com nós em pequena quantidade e localizados próximos uns dos outros, por exemplo, redes para monitorar pacientes [61]. Em redes de maior escala, soluções distribuídas são mais eficientes. No presente trabalho, adota-se a divisão lógica da rede em *clusters*, para fins de gerência. Nesse caso, componentes de monitoração precisam ser implementados em todos os nós e o gerente de políticas e estruturas de dados necessárias são mantidas nos nós designados como líderes de *clusters*.

Módulo de Inspeção. Um documento WSDL (Anexo 2) fornece para a aplicação o formato da mensagem SOAP para inspecionar os aspectos do sistema acessíveis ao meio externo. Essa mensagem representa a invocação da primitiva `Request_Inspection` da interface `QueryContext`. O módulo de inspeção é implementado nos nós sorvedouros por um módulo de software que, a partir da mensagem SOAP recebida da aplicação, acessa o perfil da aplicação e informa os parâmetros desse perfil que se encontram em utilização no momento. Caso a aplicação tenha solicitado inspecionar o contexto de execução corrente, o módulo obtém tais informações interagindo com o módulo de adaptação e suas bases de dados de estados de execução.

Módulo de Seleção de Nós Ativos. A princípio, pode parecer que uma infra-estrutura mais densa leva a uma rede de sensores mais eficiente, porque com um maior

número de nós uma acurácia de dados mais alta pode ser obtida e a quantidade global de energia da rede é maior [132]. Entretanto, se não for apropriadamente gerenciada, uma rede mais densa pode levar a um número maior de colisões e até mesmo à presença potencial de congestionamentos na rede, aumentando a latência dos dados e diminuindo a eficiência em energia da rede. Adicionalmente, o grande número de amostras de dados reportadas pelos sensores pode exceder a acurácia requerida pela aplicação. Portanto, o número de sensores que efetivamente devem participar de cada tarefa deve ser balanceado levando em conta, para cada configuração, o seu custo para a rede, e os requisitos de qualidade (QoS) solicitados pela aplicação. Uma forma de se atingir esse objetivo é adotar um esquema de rodízio do trabalho realizado pelos nós da rede, ativando alternativamente diferentes sub-conjuntos desses nós. O sub-conjunto selecionado para permanecer ativo em cada momento deve ser capaz de atender aos requisitos de QoS solicitados pelas aplicações em execução. O estabelecimento de um esquema de rodízio entre os nós é o objetivo do módulo de seleção de nós ativos. Esse módulo é composto pelo software que implementa o algoritmo de seleção e por várias interfaces com outros módulos do middleware. Para a seleção dos nós, o módulo precisa ter acesso às estruturas de dados contendo informações sobre os sensores (suas características físicas e seu estado corrente), sobre os requisitos da aplicação em execução e sobre as políticas de execução e de adaptação utilizadas para tal aplicação. O algoritmo de seleção será detalhado no Capítulo 7.

Em redes de pequena escala, o serviço de seleção dos nós pode ser executado de forma centralizada, nos nós sorvedouros. Na abordagem baseada em *clusters*, adotada no presente trabalho, a seleção é realizada pelos líderes dos *clusters*. Como cada nó é um potencial candidato a ser líder, o módulo que implementa o algoritmo de seleção está presente em todos os nós sensores (comuns ou líderes), sendo carregado no ambiente de execução sob demanda. Entretanto, as estruturas de dados necessárias como entrada para o algoritmo (informações de estado referentes aos sensores do *cluster* e aplicações em execução) só são mantidas pelos nós correntemente eleitos como líderes.

Módulo de Controle de Admissão. Além de racionalizar a utilização dos recursos da rede para atender a uma aplicação, o gerenciamento de uma rede de sensores deve também levar em conta a existência de múltiplas aplicações em execução simultaneamente. Trabalhos atuais consideram que RSSFs possuem longos tempos de vidas e atendem a múltiplos usuários, possivelmente com diferentes interesses, embora

em geral ligados a um mesmo domínio de aplicação. “Múltiplas solicitações de informações ambientais diferentes, chegando na rede em momentos diferentes” traduz-se em “múltiplas aplicações diferentes executando concorrentemente na rede, enquanto múltiplas solicitações para a execução de novas aplicações são constantemente recebidas” [14]. Esse cenário traz à tona a seguinte questão: “dada uma quantidade finita de energia e uma seqüência desconhecida de solicitações de aplicações (escolhidas a partir de um conjunto de aplicações candidatas, com probabilidades de ocorrência, custos de energia e requisitos de QoS conhecidos), como decidir aceitar/rejeitar aplicações na rede, de modo a maximizar o uso da rede e ao mesmo tempo atender aos diferentes requisitos ?” Tratar dessa questão é o objetivo do módulo de controle de admissão. Esse módulo é composto por (i) um conjunto de descritores da aplicação, obtidos a partir do perfil da aplicação; (ii) um componente de software que, a partir dos descritores, calcula o custo para a rede da nova aplicação submetida; (iii) um componente que realiza o processo de medição do uso corrente de recursos da rede; e (iv) um componente que implementa a política de controle de admissão de aplicações, composto por um conjunto de regras de admissão e pelo software que realiza a lógica de decisão baseada nas regras. É importante ressaltar que os recursos exigidos por uma nova aplicação muitas vezes são compartilhados por aplicações já em execução. Então, o custo de aceitar a nova aplicação deve ser analisado para cada componente individual do sistema (sensores, transmissores, processador, etc).

O módulo que realiza o processo de medição do uso dos recursos utiliza informações sobre o contexto de execução corrente, obtidas do módulo de monitoramento e adaptação. O módulo de controle de admissão roda apenas em nós sorvedouros.

Módulos de Serviços Especializados. O projeto do middleware proposto prevê a existência de componentes para fornecer serviços genéricos para RSSFs. A implementação desses componentes pode ser fornecida por terceiros, desde que implementem as interfaces disponibilizadas pelo serviço de comunicação do sistema. Como visto na Seção 4.2, exemplos relevantes de serviços genéricos são nomeação, segurança, localização e agregação. Na arquitetura proposta, tais serviços, quando presentes, devem ser implementados nos nós sorvedouros e em todos os nós sensores da rede (comuns e líderes).

Um exemplo de esquema de nomeação para RSSFs é descrito em [39] e pode ser implementado como o componente de serviço de nomeação do middleware. Tal esquema

permite a reutilização de identificadores quando nós passam para modos inativos. Já o serviço de localização pode consistir na implementação de um algoritmo de triangulação, como o proposto em [93], e é executado na fase de inicialização da rede ou sob demanda pelo nó caso ele se movimente. Ambos os serviço de nomeação e de localização utilizam a primitiva `Configure` da interface `ConfigureRequest` para indicar como um nó sensor pode solicitar sua execução.

No caso do serviço de segurança, além do módulo que implementa os diferentes mecanismos, como os propostos em [105], é necessário um manipulador específico que intercepte cada mensagem e decida se ela deverá ser encaminhada para o serviço de segurança ou não, nesse caso notificando o serviço da chegada do dado (interface `Notifier`). Por exemplo, pode ser que apenas mensagens de dados devam ser autenticadas, mensagens de controle não. Ou podem ser definidos diferentes níveis de segurança, conforme o conteúdo da mensagem, ou, ainda, pode-se optar por criptografar-se apenas alguns campos das mensagens de dados. O manipulador é o componente responsável por analisar cada mensagem e encaminhar as que forem necessárias, ou parte delas, para o serviço de segurança.

O módulo de agregação, presente nas duas categorias de nós sensores, consiste no software que implementa a função de agregação desejada e no manipulador que decide se uma mensagem deve ser encaminhada para o serviço de agregação. O manipulador responsável por essa tarefa é o `Matching_Aggregation`.

Todos os módulos de serviços especializados fazem uso da interface `Publish` para enviarem seus resultados para o serviço de comunicação do middleware.

6 Funcionamento do Sistema

O funcionamento básico do sistema de middleware proposto consiste em uma série de fases que são entrelaçadas com o funcionamento da própria rede de sensores. As fases são: (i) descoberta de serviços; (ii) submissão de interesses e requisitos da aplicação; (iii) controle de admissão; (iv) seleção dos nós ativos; (v) construção de perfis da aplicação; (vi) configuração da rede; (vii) disseminação de dados; (viii) inspeção do sistema; e (ix) adaptação do sistema.

As próximas seções detalham as etapas de funcionamento do sistema.

6.1 ETAPA DE DESCOBERTA DE SERVIÇOS

Antes que o sistema possa entrar em operação, é necessário que os serviços oferecidos pela RSSF sejam conhecidos. Serviços de nomeação e localização podem ser necessários durante a instalação física da rede. Para possibilitar a execução desses serviços, uma fase inicial de descoberta de serviços interna é necessária. Para que as aplicações sejam capazes de acessar a RSSF, uma fase de descoberta de serviços externa deve ser realizada.

A descoberta de serviços interna começa com uma fase inicial de configuração, durante a qual os nós usam o serviço de comunicação do middleware para trocar mensagens descrevendo seus serviços. Tais mensagens de configuração, chamadas `PublishSensorDescription` (Figura 15), incluem o identificador do nó, uma estampa de tempo, os tipos dos dispositivos sensores, localização geográfica, energia residual, graus de confiança máximo e mínimo, intervalos de aquisição (taxa de dados) máximo e mínimo. Mensagens de configuração são enviadas em *broadcast* na rede, a fim de alcançar pelo menos um nó sorvedouro, de preferência o mais próximo ao nó emissor. A Figura 17 apresenta o diagrama de seqüência da etapa de descoberta de serviços interna. O diagrama descreve a interação entre os componentes lógicos do middleware (descritos na Seção 4.2.2) através da troca de primitivas, as quais representam métodos das interfaces descritas para os componentes. Essas primitivas são implementadas como operações do Serviço Web da rede invocadas através de mensagens XML.

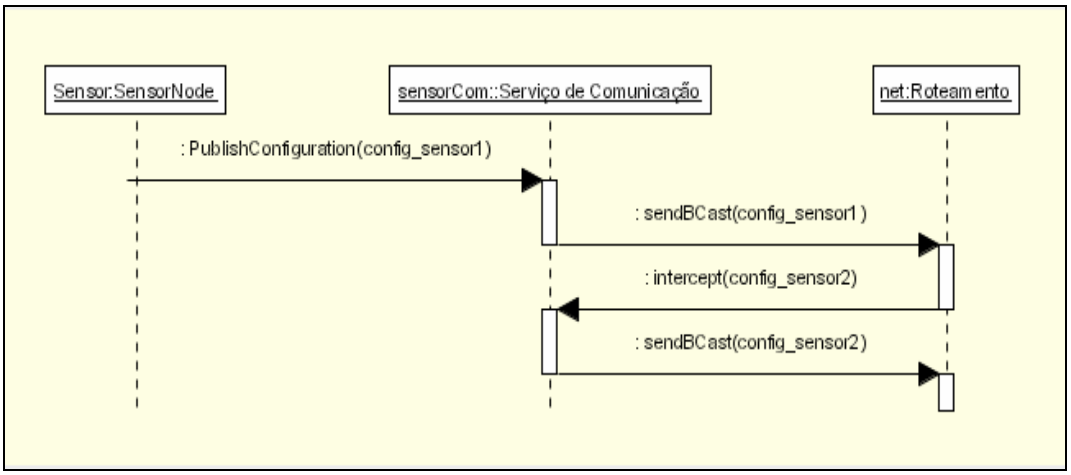


Figura 17: Diagrama de Seqüência da Descoberta de Serviços Interna

É importante notar que essa fase ocorre apenas uma vez, durante a inicialização do sistema. Portanto, ainda não se utiliza o protocolo de comunicação otimizado para a aplicação específica. Nós sorvedouros armazenam o conteúdo de mensagens de configuração recebidas em um repositório local. Eles periodicamente trocam mensagens, de modo que todos os sorvedouros em uma rede possuam o mesmo conteúdo.

A descoberta de serviços externa à rede é usada por uma aplicação para descobrir que RSSFs fornecem os serviços desejados por ela, e também para descobrir como invocar tais serviços. Para isso são utilizados o serviço de registro UDDI e um documento WSDL.

A fim de disponibilizar uma rede de sensores como um Serviço Web, o administrador da rede deve registrá-la em um serviço de registros UDDI. Para isso, ele deve criar quatro diferentes tipos de "registros": o *businessEntity* (registra a instituição proprietária da rede), o *businessService* (registra o serviço específico fornecido pela rede), o *bindingTemplate* e o *tModel* [53]. O *bindingTemplate* inclui informação sobre como e onde acessar um Serviço Web específico. Por exemplo, pode ser especificado no registro que o serviço é disponível via SOAP em uma dada URL, no caso do presente trabalho, a URL do nó sorvedouro. Já o *tModel*, ou modelo técnico, é usado para fornecer ponteiros para especificações técnicas externas. Por exemplo, um *bindingTemplate* para um serviço fornece informações sobre onde acessar o *binding* SOAP necessário para utilizá-lo, mas não fornece informação alguma sobre como interagir com ele. O elemento *tModel* fornece um ponteiro para uma especificação externa que, no caso presente, consiste nos documentos WSDL da rede (Anexos 2 e 3).

6.2 ETAPA DE SUBMISSÃO DE INTERESSES E REQUISITOS PELA APLICAÇÃO

Na fase de descoberta de serviços, a aplicação passa a conhecer o endereço do sorvedouro, os serviços disponíveis na RSSF e o correspondente formato das mensagens para invocá-los. Uma vez conhecendo esse formato, a aplicação pode invocar as operações fornecidas pelo Serviço Web da rede (descritas como elementos ***operation*** no documento WSDL), através do envio de mensagens SOAP. A primeira operação invocada pela aplicação tem por objetivo conhecer as características específicas dos sensores disponíveis na rede. Essa operação, denominada `Query_Sensors`, é representada por duas mensagens SOAP: uma mensagem de entrada, sem parâmetros, para invocar a operação, e uma mensagem de saída, contendo a resposta (Figuras 18 e 19). Após conhecer os sensores existentes, a aplicação pode submeter seus requisitos. Os requisitos da aplicação incluem seus interesses (descritores da (s) tarefa (s) de sensoriamento), seus requisitos de QoS e uma lista de políticas de execução para cada tarefa solicitada. Os requisitos da aplicação são usados inicialmente para realizar o controle de admissão da aplicação e para selecionar o sub-conjunto de nós que devem permanecer ativos. Na versão atual do presente trabalho, o serviço de controle de admissão não é tratado. A seguir, os requisitos da aplicação são usados para construir o perfil inicial da aplicação e para decidir o melhor protocolo de disseminação e topologia lógica da rede a serem adotados para a tarefa requisitada. A Figura 20 apresenta o diagrama de seqüência lógico da fase de submissão de interesses.

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <m:QuerySensorIn xmlns:m="http://namespace.example.com">
      <parameter> </parameter>
    </m:QuerySensorIn>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figura 18: Mensagem SOAP `Query_SensorsIn`: mensagem de entrada para uma operação `Query_Sensors`

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <m:QuerySensorOut xmlns:m="http://namespace.example.com">
      <parameter>
        <m:SensorType>Motion</m:SensorType>
        <m:Confidence> <m:Max>1.0</m:Max>  </m:Confidence>
        <m:DataRate unit="mSeconds"> <m:Max>10</m:Max> </m:DataRate>
      </parameter>
    </m:QuerySensorOut>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figura 19: Mensagem SOAP `Query_SensorsOut`: mensagem de saída para uma operação `Query_Sensors`

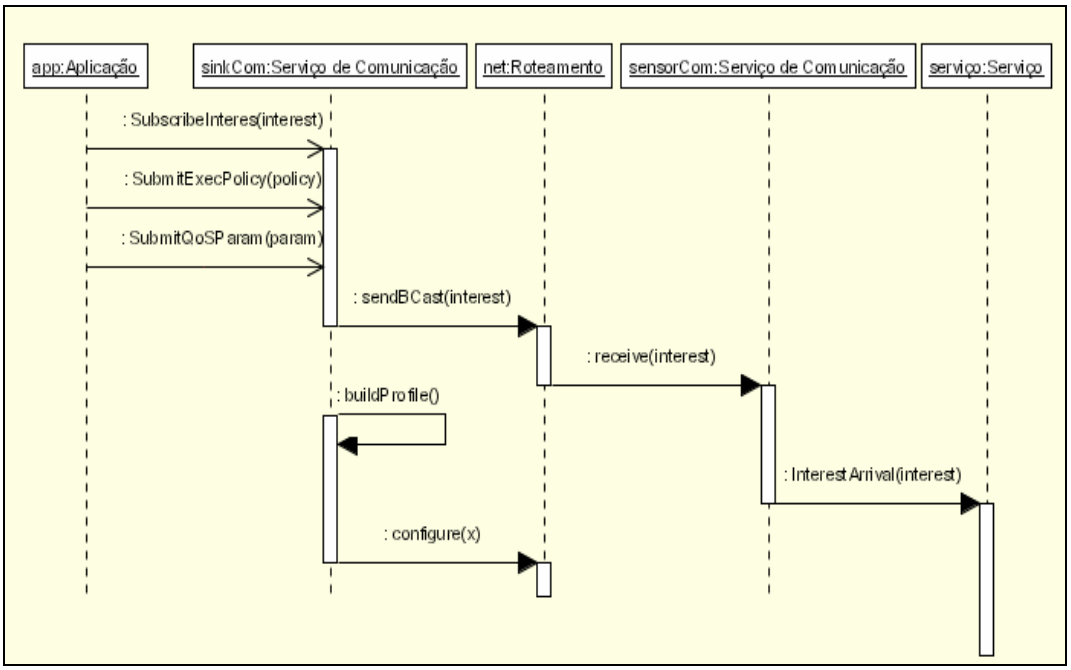


Figura 20: Diagrama de Seqüência da fase de submissão de interesses

Interesses podem ser classificados em *síncronos* ou *assíncronos*. Existem diferentes formatos de mensagens SOAP para anunciar diferentes tipos de interesses.

Um interesse síncrono corresponde a uma operação simples de consulta ao estado corrente de algum fenômeno monitorado pela rede. Um exemplo de um interesse síncrono seria: "qual a temperatura máxima da área A ?". Portanto, uma mensagem SOAP de interesse síncrono, denominada `Subscribe_Synch_Interest` (Anexo 4), contém o tipo do sensor e a região geográfica alvo, além de poder incluir opcionalmente uma restrição ao valor a ser reportado, como por exemplo, reportar apenas valores de dados acima de um limiar. Interesses assíncronos correspondem a consultas de longa duração ou a consulta sobre a ocorrência de algum evento específico. Um exemplo de consulta de longa duração é: "qual a temperatura média nas próximas 24 horas na área A ?". "Avise quando um elefante passar pela área A" é um exemplo de consulta sobre a ocorrência de evento específico. Mensagens SOAP de consultas de longa duração, denominadas `Subscribe_LongRunning_Interest`, devem indicar o tipo do sensor, a área alvo, a duração da consulta e a taxa de aquisição (Anexo 5). Mensagens SOAP de consultas sobre a ocorrência de algum evento (`Subscribe_Event_driven_Interest`) devem indicar a área, o tipo de sensor (por exemplo, de movimento) e o evento a ser monitorado, por exemplo, "elefante" ou "temperatura > 37" (Anexo 6). Aplicações podem, ainda,

desejar que alguma ação seja realizada em resposta ao evento, por exemplo, ativar um maior número de sensores para aumentar a precisão dos dados obtidos, ativar sensores especializados que anteriormente estariam inativos ou aumentar a taxa de aquisição dos sensores já ativos. Quando houver disponível previamente conhecimento sobre variações temporais do fenômeno monitorado, a aplicação pode solicitar diferentes taxas de aquisição para diferentes horas do dia ou dias da semana/mês, a fim de ajustar as taxas para capturar as mudanças de dados mais significativas [84]. Para representar as ações dirigidas a eventos mencionadas a mensagem SOAP `Triggering_Interest` foi definida (Anexo 7).

Na submissão de seus interesses, aplicações podem informar valores mínimos de parâmetros de QoS a serem respeitados, como acurácia de dados e latência, e podem solicitar a execução de funções de agregação. Além de interesses, aplicações podem enviar, para cada serviço solicitado, uma lista de políticas de execução, indicando os parâmetros a serem usados na aquisição dos dados e os requisitos de QoS a serem respeitados em cada contexto de execução.

O contexto de execução é representado por um conjunto de parâmetros da aplicação e parâmetros da rede. Parâmetros da aplicação compreendem informações conhecidas por uma aplicação em execução e são definidos em termos dos valores dos dados sensoriados. Parâmetros da rede representam informação conhecida pelo middleware e incluem: (i) nível de energia da rede; (ii) alcance/potência/taxa de dados do transmissor e (iii) a posição geográfica dos nós [81].

Além de poder definir mais de um requisito de QoS para um serviço solicitado, a aplicação pode decidir qual dos requisitos priorizar em detrimento dos demais. Por exemplo, uma aplicação de monitoramento ambiental pode escolher priorizar o tempo de vida em favor da acurácia ou, ao contrário, priorizar a acurácia em favor do tempo de vida, ou pode, ainda, optar por balancear ambos os parâmetros. De acordo com a escolha feita pela aplicação quanto à prioridade dos requisitos de QoS, um perfil de aplicação pode ser classificado em:

- Baseado no desempenho - quando a aplicação decide priorizar um requisito de desempenho, como cobertura de sensoriamento, latência ou acurácia de dados;

- Baseado no tempo de vida - quando a aplicação decide priorizar o tempo de vida da rede; e

- Baseado na razão - quando a aplicação opta por balancear o tempo de vida e o desempenho da rede, ou seja, busca a melhor razão custo/beneficio entre gasto de energia e latência, acurácia de dados ou cobertura.

Um exemplo de especificação de política de execução para o serviço básico de entrega de dados é representado pela mensagem XML mostrada na Figura 21. Na política especificada na mensagem, a aplicação varia o parâmetro "taxa de aquisição de dados" (*data rate*) e o requisito de QoS "latência" (*delay*), conforme o contexto de execução, dado pela energia residual e a largura de banda da rede. A aplicação escolheu dar prioridade ao requisito latência, ou seja, o perfil gerado será do tipo baseado no desempenho. Como as políticas de execução são definidas em termos de parâmetros de rede e parâmetros da aplicação, com freqüência os valores correntes de tais parâmetros podem se encaixar em mais de uma única política, dando origem a um conflito quanto à decisão sobre a política a ser aplicada. Para resolver esse tipo de conflito, criou-se um valor de prioridade a ser usado pela aplicação na especificação de políticas. Esse valor, representado pela TAG XML <`conflictResolutionRules`> (Figura 21) define uma ordem de prioridade a ser aplicada na seleção das políticas. No exemplo, a aplicação define que um valor crítico do dado de temperatura (parâmetro da aplicação) deve ditar a política de execução a ser adotada, independente dos parâmetros da rede. Na ausência desse valor crítico, o parâmetro "latência" tem prioridade sobre o parâmetro "largura de banda". Outro tipo de conflito que pode ocorrer é entre diferentes aplicações rodando na RSSF ao mesmo tempo. A resolução desse tipo de conflito é mais complexa e será deixada para trabalhos futuros.

```
<service>
  <priorityQoS>
<param type="delay"/>
</priorityQoS>
<execPolicy id ="1">
    <context id ="1">
      <network>
          <param type="average energy" operator="gt" value="5"/>
          <param type="bandwidth" operator="gt" value="100"/>
      </network>
      <application>
        <param type="temperature" operator="lt" value="30"/>
      </application>
    </context>
    <qos>
      <param type="delay" operator="lt" value="5"/>
      <param type="data rate" operator="gt" value="10"/>
```

```
    </qos>
  </execPolicy >
  <execPolicy id ="2">
    <context id ="1">
      <network>
          <param type="average energy" operator="lt" value="4"/>
          <param type="bandwidth" operator="lt" value="100"/>
      </network>
      <application>
        <param type="temperature" operator="lt" value="30"/>
      </application>
    </context>
    <qos>
      <param type="delay" operator="lt" value="6"/>
      <param type="data rate" operator="gt" value="5"/>
    </qos>
  </execPolicy>
  <execPolicy id ="3">
    <context id ="1">
      <network>
          <param type="average energy" operator="gt" value="5"/>
          <param type="bandwidth" operator="gt" value="100"/>
      </network>
      <application>
        <param type="temperature" operator="gt" value="30"/>
      </application>
    </context>
    <qos>
      <param type="delay" operator="lt" value="2"/>
      <param type="data rate" operator="gt" value="20"/>
    </qos>
  </execPolicy>
  <conflictResolutionRules>
    <rule id="1" criticalParam="temperature" over="all"/>
    <rule id="2" criticalParam="delay" over="bandwidth"/>
  </conflictResolutionRules>
</service>
```

Figura 21: Exemplo de Mensagem XML de especificação de políticas de execução para um serviço de entrega de dados

Como outro exemplo, pode-se considerar uma aplicação que solicita dois serviços da rede: entrega de dados e agregação. A aplicação pode decidir dar prioridade ao tempo de vida da rede, determinado pelo seu nível de energia. Portanto, a aplicação pode definir o tempo de vida mínimo da rede necessário para seus requisitos e pode definir dois valores de latência máxima tolerada para o serviço de entrega de dados, um para cada valor corrente da energia residual da rede. Requisitos de QoS para o serviço de agregação podem ser definidos de acordo com dois parâmetros: grau de agregação (razão entre o número de mensagens recebidas e o número de mensagens enviadas) e atraso de agregação. Atraso de agregação é o atraso gerado pelo tempo que um sensor deve esperar pela chegada de mensagens a serem agregadas, antes de transmitir a mensagem agregada resultante. Um valor menor para esse atraso indica que provavelmente menos mensagens serão agregadas, gerando um menor grau de agregação. Valores maiores de atraso geralmente implicam um maior grau de agregação, ou seja, um maior número de

mensagens são agregadas, o que pode diminuir o tráfego na rede e, por conseguinte, economizar energia.

É importante observar que graus muito altos de agregação podem comprometer a acurácia do dado, enquanto atrasos de agregação muito altos podem comprometer a latência. Portanto, deve haver um compromisso entre os parâmetros de QoS estabelecidos para diferentes serviços. No exemplo em questão, a aplicação pode definir dois valores diferentes para o atraso de agregação, um para cada valor de energia residual da rede. Entretanto, o middleware deve gerenciar o valor de latência de dados, a fim de garantir que ele se mantenha dentro do limite estabelecido pela aplicação, em qualquer caso.

6.2.1 Construção do Perfil da Aplicação

Um perfil de aplicação é inicialmente estabelecido a partir de dados fornecidos pela aplicação, quando da submissão de seus interesses, e posteriormente gerenciado pelo middleware subjacente. Eventualmente, a aplicação pode inspecionar e alterar em tempo de execução as informações contidas em seu perfil.

O perfil da aplicação consiste em estruturas de dados contendo a descrição dos requisitos da aplicação em termos de seus interesses de sensoriamento e as políticas de execução para cada serviço solicitado. O middleware acrescenta ao perfil da aplicação suas decisões quanto ao protocolo de comunicação e configuração topológica adotados para a rede. Adicionalmente, as diferentes políticas de adaptação aplicadas (se houver) durante a execução da aplicação podem ser incluídas no perfil, para compor um banco de dados de informações históricas do sistema.

O middleware interage com a infra-estrutura de rede subjacente a fim de manter atualizada a informação de contexto. Sempre que ocorre uma mudança no contexto, o componente gerente de políticas do sistema verifica nos perfis das aplicações ativas se os requisitos de QoS estão sendo satisfeitos. Se algum requisito não estiver sendo atendido para o contexto atual, o middleware aplica uma política de adaptação pré-definida, tomando as ações apropriadas.

6.3 ETAPA DE CONFIGURAÇÃO DA REDE

Em geral, desenvolvedores de aplicações para RSSFs são especialistas em seu próprio domínio de aplicação, não em redes. Portanto, é difícil para tais desenvolvedores

escolher a configuração de rede mais adequada para suas necessidades. O middleware proposto oferece um processo de decisão automatizada, utilizando os requisitos da aplicação para selecionar a melhor configuração, isto é, protocolo e topologia a serem usados na rede, para tal aplicação. O processo de decisão faz parte do serviço de configuração e é fornecido pelo módulo de decisão.

Ao receber os requisitos de uma nova aplicação, o módulo de decisão verifica, nas suas estruturas de dados, se há algum perfil de aplicação existente cujas características e requisitos sejam os mesmos da nova aplicação. Caso haja, são adotados os mesmos parâmetros descritos no perfil. Caso contrário, é executado o algoritmo de decisão.

O algoritmo de decisão, detalhado no Capítulo 7, é responsável por escolher o modelo de entrega de dados, o protocolo de disseminação de dados, e a topologia lógica da rede que melhor atendam a uma determinada aplicação. As decisões resultantes da execução do algoritmo são reunidas com as decisões tomadas pelo módulo de seleção de nós ativos (no caso centralizado) e passadas para o módulo de comunicação (SOAP). Nesse módulo, elas são convertidas em informações de configuração do protocolo de rede subjacente e passadas para o mesmo com o uso da primitiva `Configure` da interface `DataRouting`. O protocolo de rede gera, então, toda a comunicação de infra-estrutura necessária para configurar a topologia ativa da rede e estabelecer a estratégia de disseminação (roteamento) dos dados. Como as mensagens geradas pela comunicação de infra-estrutura são dependentes da linguagem e formato dos protocolos previamente instalados nos sensores, elas não são representadas em XML. Somente a comunicação da aplicação é baseada em XML.

Caso o protocolo de disseminação de dados selecionado para ser usado na rede seja baseado em *clusters*, a topologia hierárquica criada para o encaminhamento dos dados será a mesma usada pelo middleware. Caso o protocolo escolhido adote uma topologia plana, um mecanismo de clusterização será necessário para fins de gerência da rede. Uma topologia virtual de nós será criada sobre a topologia física real, e os líderes de *clusters* serão eleitos para implementar algumas das funcionalidades de serviços fornecidos pelo middleware, como a seleção dos nós ativos.

Após o middleware usar as informações contidas na mensagem SOAP de anúncio de interesse para decidir sobre a configuração da rede, essa mensagem é propagada para

os nós sensores, de acordo com a estratégia de disseminação de dados adotada. A partir desse momento, a rede passa a realizar sua tarefa de sensoriamento e entrega (publicação) de dados, de acordo com o interesse recebido.

6.4 ETAPA DE SELEÇÃO DOS NÓS ATIVOS

A tarefa de sensoriamento submetida por uma aplicação para a rede pode ser atendida por diferentes conjuntos de sensores. O processo de seleção de nós ativos pode ser expresso como o algoritmo que decide dinamicamente quais sensores deverão permanecer ativos para a execução dessa tarefa. O algoritmo busca maximizar o tempo de vida útil da rede ao mesmo tempo em que a qualidade de serviço desejada pela aplicação é fornecida.

Nessa abordagem, nós não selecionados são escalonados para "dormir", ou seja, permanecer em um estado de baixo consumo de energia. É importante ressaltar que o consumo de energia em um sensor no estado ocioso é apenas ligeiramente menor do que no estado de recepção de dados. Portanto, os nós escalonados para "dormir" devem ser completamente desligados, resultando na alteração da organização topológica da rede. Após a seleção, a rede passa a ter uma topologia virtual, composta pelos nós selecionados, sobrepondo-se à topologia real correspondente aos nós fisicamente instalados. Todas as tarefas da rede, como roteamento e coleta de dados serão realizadas apenas pelos nós pertencentes à topologia virtual ativa no momento

Na abordagem baseada em *clusters* adotada no presente trabalho, os nós líderes são responsáveis por selecionar os nós de seu *cluster* que deverão permanecer ativos para realizar a tarefa. Como qualquer nó da rede pode ser selecionado como líder, o serviço de seleção de nós ativos do middleware é implementado em todos os nós, sendo carregado no ambiente de execução apenas quando o nó se torna líder. No caso da aplicação requerer um número relativamente pequeno de sensores para a tarefa, a seleção pode ser realizada pelos nós sorvedouros, de forma centralizada.

O algoritmo de seleção de nós ativos é executado pela primeira vez quando os interesses de uma nova aplicação são submetidos para a rede. A tarefa requerida começa logo após a execução do algoritmo. O algoritmo pode ser executado novamente como uma estratégia de adaptação do sistema, nos seguintes casos:

- sob demanda pela aplicação, quando a mesma deseja alterar algum parâmetro de QoS;

- proativamente pelo middleware, como medida de conservação de energia da rede;

- reativamente pelo middleware quando detecta a violação de algum requisito de QoS.

Neste trabalho, o processo de seleção de nós ativos foi interpretado como um problema de maximização e modelado como um Problema da Mochila (*knapsack problem* [26]), adotando-se uma abordagem gulosa para resolvê-lo. O Capítulo 7 detalha a solução adotada.

6.5 ETAPA DE DISSEMINAÇÃO DOS DADOS

Quando um sensor gera um dado, ele é passado para o módulo de comunicação, onde um *driver* converte o dado em uma representação baseada na linguagem XML. Manipuladores são responsáveis por notificar a chegada do dado aos serviços do middleware (primitiva `Local_Data_Arrival`). Os serviços registrados para o tipo de dado gerado são, então, executados e o resultado da execução é passado de volta para o módulo de comunicação (primitiva `PublishResults`), que se encarrega de encaminhá-lo para o protocolo de roteamento. Ao passar por nós intermediários em direção ao sorvedouro, os dados recebem o mesmo tratamento que dados gerados localmente. A Figura 22 apresenta o diagrama de seqüência da fase de publicação de dados.

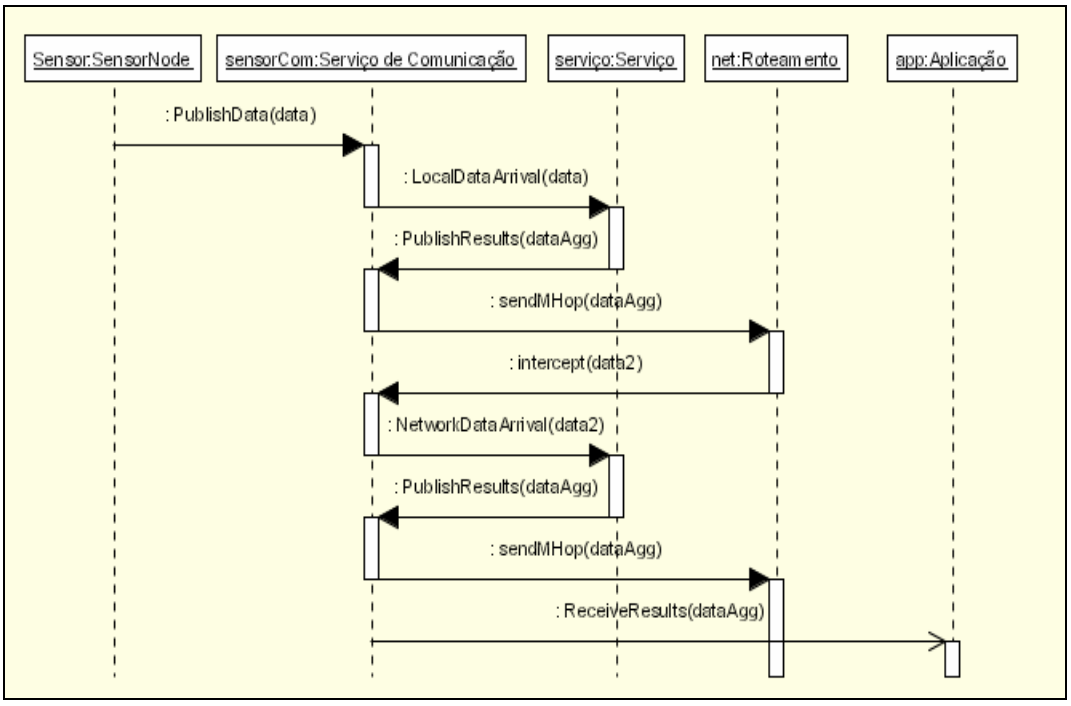


Figura 22: Diagrama de Seqüência da fase de publicação de dados

Como exemplo, considere-se o serviço de agregação de dados. O manipulador `Matching_Aggregation` verifica se os atributos do dado combinam com os atributos especificados por alguma solicitação de função de agregação. Caso o resultado da comparação seja positivo, os dados são entregues para o respectivo serviço de agregação. Estão previstos componentes para executar quatro tipos básicos de função de agregação: COUNT, MAX, MIN e AVERAGE. A função COUNT computa o número de ocorrências de um dado fenômeno monitorado. As funções MAX e MIN computam os valores máximo e mínimo, respectivamente, do dado monitorado. A função AVERAGE computa a média aritmética dos valores de dados sensoriados. Novos serviços de agregação podem ser implementados para representar funções mais complexas. Ao expirar o tempo definido pelo atraso de agregação, definido na solicitação de agregação, a função é aplicada e os dados agregados resultantes são passados para o módulo de comunicação e dali para o protocolo de disseminação, como uma nova mensagem XML de anúncio de dados a ser enviada na rede. Caso o protocolo em execução utilize um *driver* do tipo 1, o dado é convertido de volta para a representação adotada pelo protocolo antes de ser entregue para a sub-camada de comunicação. Caso seja utilizado um *driver* do tipo 2, a mensagem XML é diretamente encapsulada no pacote de dados a ser enviado.

Ao receber um pacote proveniente de outro sensor, o protocolo de disseminação verifica, através de um campo do seu cabeçalho, se o pacote contém uma mensagem da aplicação (dado ou interesse) ou de infra-estrutura (mensagem de controle). Caso seja um pacote de controle, ele é tratado pelo próprio protocolo de disseminação de dados. Caso seja um pacote da aplicação, ele é passado para o módulo de comunicação. Quando mensagens XML são encapsuladas diretamente na carga útil do pacote de dados do protocolo de disseminação, o manipulador responsável pela análise XML verifica o tipo da mensagem (interesse ou dado) para tomar a ação apropriada. Caso seja uma mensagem de interesse, ela é despachada para o manipulador `Parse_Interest`. Esse manipulador verifica se o interesse recebido combina com as características do nó (local e tipo de sensor). Em caso afirmativo, armazena o interesse. Caso contrário, passa-o de volta ao protocolo de disseminação, que irá tomar as decisões quanto a re-encaminhar ou não o interesse. Caso seja uma mensagem de dado, ela é inicialmente encaminhada para o manipulador `Matching_Data`, que verifica se o dado combina com algum interesse previamente recebido pelo nó. Se combina, a mensagem é entregue para os manipuladores específicos, encarregados de notificar sua chegada para os serviços solicitados e despachá-la para os mesmos. A seguir, ela recebe o mesmo tratamento de dados gerados localmente.

Os manipuladores `Parse_Interest` e `Matching_Data` são exclusivamente utilizados quando uma representação proprietária para dados e interesses não é fornecida pelo protocolo de rede. Se o protocolo adota um formato proprietário de representação, as funções de ambos os manipuladores são realizadas pelo próprio protocolo.

Ao atingir o nó sorvedouro, as mensagens de dados são convertidas em uma mensagem SOAP de anúncio de dados, denominada `Publish_Data` (Figura 23) e enviadas para a aplicação.

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:m0="empty">
  <SOAP-ENV:Body>
    <m:PublishData xmlns:m="http://namespace.example.com">
      <parameter ID="NODE_MAC_ADDRESS">
        <m0:DataValue>Elephant</m0:DataValue>
        <m0:Location unit="LatLong">
          <m0:x>35.00</m0:x>
          <m0:y>-23.00</m0:y>
        </m0:Location>
        <m0:Intensity>0.6</m0:Intensity>
        <m0:Confidence>0.85</m0:Confidence>
        <m0:Energy>0.9</m0:Energy>
```

```
        <m0:TimeStamp>08:16:40</m0:TimeStamp>
      </parameter></m:PublishData>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figura 23: Mensagem SOAP de anúncio de dados (`Publish_Data`)

6.6 ETAPA DE INSPEÇÃO E ADAPTAÇÃO DO SISTEMA

O middleware proposto permite que as aplicações acessem dinamicamente seus perfis para consulta (inspeção) e, se necessário, para alteração (adaptação) (Figura 24). Através do documento WSDL, o middleware expõe para as aplicações as partes de seu comportamento ou do contexto que podem ser inspecionadas e/ou alteradas (processo de reificação [73]). A solicitação por uma inspeção do estado do sistema é feita enviando uma mensagem SOAP `Request_Inspection_In` (Anexo 8), a qual por sua vez invoca a operação `Request_Inspection` no Serviço Web da rede. A partir da análise das informações fornecidas como resultado da operação `Request_Inspection` (Anexo 9), a aplicação pode decidir modificar o comportamento do sistema, alterando algum parâmetro de QoS ou alguma política de execução previamente registrados. Para solicitar alterações, a aplicação envia uma mensagem SOAP `Request_Adaptation` (Anexo 10). Através de sua capacidade adaptativa, o middleware atende aos requisitos da aplicação, comunicando-se com os protocolos e dispositivos necessários para realizar a política de adaptação requerida.

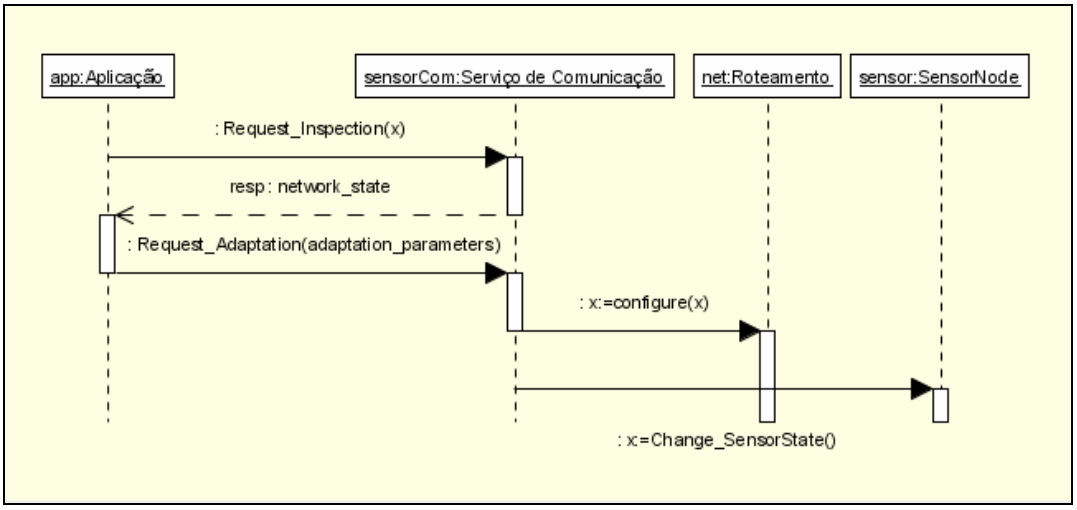


Figura 24: Diagrama de Sequência das fases de inspeção e adaptação

Para manter a informação de contexto atualizada, o middleware realiza o monitoramento dos estados da rede e da aplicação. São previstas duas possíveis

estratégias de monitoramento: contínua ou baseada em limiar. No caso de monitoramento contínuo, cada nó sensor envia periodicamente, a um intervalo pré-definido (provavelmente coincidente com o intervalo de envio de dados), as informações que se desejam monitorar. Exemplos dessas informações são o nível residual de energia do nó e o atraso nos seus enlaces com os nós vizinhos. No segundo caso, os sensores mantêm as informações de estado armazenadas localmente até que um limiar pré-estabelecido no valor de algum parâmetro seja atingido, e só então as enviam. Informações de monitoramento podem ser agregadas em nós líderes, segundo alguma função definida, antes de serem enviadas para nós sorvedouros, a fim de minimizar o número de mensagens na rede. Tanto as informações que se desejam monitorar quando os limiares de envio de dados são definidos nas mensagens de interesses da aplicação. As informações de estado obtidas são armazenadas em uma base de dados do estado da aplicação e em uma base de dados do estado da rede. Os valores correntes dos parâmetros monitorados são enviados em mensagens de dados para o gerente de políticas do módulo de adaptação. O manipulador `Monitor_State` verifica, através de um *flag* na mensagem de dados, que ela contém informação de monitoramento, e a encaminha para o serviço de adaptação no gerente. O gerente, por sua vez, verifica os perfis de aplicações armazenados. Se os parâmetros de QoS oferecidos pelo sistema não combinam com os requisitos definidos em algum perfil para o contexto de execução vigente, o middleware tenta adaptar o comportamento do sistema, executando alguma política de adaptação, a qual, por sua vez, irá requerer a alteração de parâmetros dos dispositivos e protocolos (Figura 25).

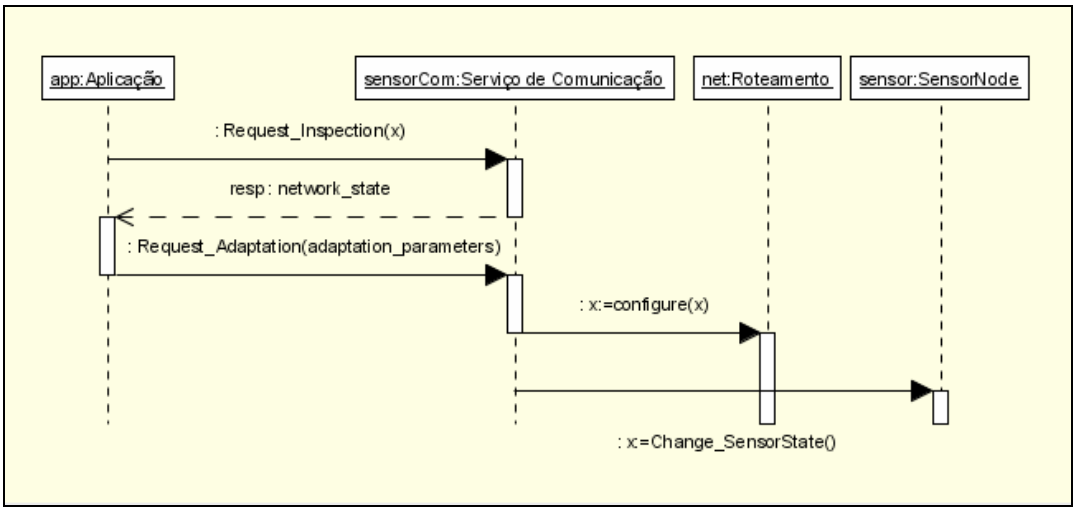


Figura 25: Diagrama de Sequência da fase de adaptação detalhada

6.6.1 Políticas de Adaptação

Políticas de adaptação são previamente registradas no sistema como conjuntos de ações a serem realizadas quando os requisitos de QoS estabelecidos por uma aplicação não estão sendo atendidos, para um dado contexto de execução. Há políticas agressivas e conservadoras, e seu principal objetivo é balancear os requisitos de QoS da aplicação com o estado da rede, visando estender o tempo de vida global da RSSF. Exemplos de políticas de adaptação criadas para o sistema proposto são: aumentar a confiabilidade dos dados (dada pelo valor de acurácia média fornecida); diminuir a latência; diminuir o consumo de energia; e aumentar a largura de banda disponível.

A política para aumentar a confiabilidade dos dados pode ser implementada por uma ação que envolve a alteração de parâmetros usados pelos sensores ou por ações que envolvem a alteração de parâmetros usados pelo middleware. Para ativar tal política é necessário, inicialmente, saber se a atual confiabilidade, dada pela acurácia dos dados, é satisfatória ou não para a aplicação. Somente a aplicação possui informações sobre os valores reais esperados. Ela é, portanto, responsável por notificar o sorvedouro se a acurácia dos dados fornecidos pela rede atende ou não seus requisitos. A acurácia pode ser obtida por comparação com valores conhecidos, reportados por outras técnicas de medição. Caso a aplicação informe que a acurácia dos dados está abaixo do limiar desejado, o sorvedouro pode disparar a política de adaptação. Uma ação possível consiste em solicitar que os sensores aumentem suas taxas de envio de dados. Outras duas ações possíveis são: (i) solicitar que o serviço de agregação diminua o grau de agregação atualmente utilizado; ou (ii) solicitar ao serviço de seleção de nós ativos uma nova execução do algoritmo de seleção, aumentando-se o número de nós ativos. Com um número maior de sensores ativos a acurácia fornecida pela rede provavelmente será maior (apesar de haver exceções a esse comportamento, conforme será descrito na Seção 7.3).

As políticas para diminuir o consumo de energia ou para aumentar a banda disponível na rede podem ser implementadas por duas ações: diminuir a taxa de envio de dados e desligar alguns sensores. A primeira ação requer que uma mensagem de controle seja enviada para os nós sensores a fim de que estes alterem a taxa de envio de dados correntemente usada. Já a segunda ação requer uma nova rodada do algoritmo de seleção de nós ativos, nesse caso diminuindo-se o número de nós ativos. O middleware,

entretanto, tem que verificar se a ação a ser realizada não irá prejudicar o nível mínimo de QoS solicitado pelas aplicações em execução.

A política para diminuir a latência de dados pode ser implementada por duas ações: diminuir o atraso de agregação e/ou diminuir o intervalo entre o envio de dados.

O sorvedouro é responsável por definir a mudança no número de nós ativos, ou qualquer alteração de parâmetro necessária (como mudar a taxa de aquisição).

Duas diferentes abordagens para a adaptação podem ser adotadas: reativas ou proativas. Na abordagem reativa, as decisões são tomadas **após** se verificar que a rede se encontra em um dado contexto de execução e com determinados parâmetros de QoS. Na abordagem proativa, tendências são calculadas, para tentar impedir uma violação de QoS antes que ela ocorra. Para isso, séries históricas dos dados monitorados devem ser mantidas, exigindo mais capacidade de processamento e armazenamento dos nós, consequentemente consumindo mais energia da rede. Caso seja adotada uma abordagem proativa, a variação do dado ao longo do tempo é registrada e uma curva descrevendo seu comportamento é construída. Técnicas de inferência ou lógica nebulosa podem ser usadas para antecipar um estado não desejável, e medidas preventivas podem ser adotadas. Por exemplo, um sensor pode construir uma curva de decaimento de sua energia e pode ser programado para adaptar seu comportamento de acordo com os valores exibidos na curva. Se o sensor percebe, através da curva, que ele irá exceder um limiar previamente estabelecido e ficar sem energia antes do período de tempo especificado para a tarefa atribuída à rede, ele pode agir proativamente para evitar tal situação. O sensor pode diminuir sua taxa de envio de dados ou aumentar o tempo em que fica em um modo de baixo consumo de energia, independente dos requisitos ditados pela aplicação. A decisão de mudar o modo de operação de um sensor ou alterar sua taxa de dados pode ser tomada individualmente, ou pode ser gerenciada pelos líderes de *clusters*.

7 Os Serviços de Decisão e de Seleção de Nós Ativos

Como visto, o sistema de middleware proposto oferece um serviço de decisão automática para a configuração da rede de sensores, abrangendo a escolha da topologia lógica da rede e do protocolo de disseminação de dados a serem adotados. O sistema oferece também um serviço de gerência de recursos na rede, que consiste basicamente em um módulo de seleção dos nós ativos, o qual é responsável pela escolha dos sensores que deverão permanecer ativos para executar uma tarefa de sensoriamento solicitada pela aplicação. Todas essas escolhas são feitas de forma transparente para os usuários da rede e são baseadas nos requisitos e interesses ditados pela aplicação. As seções a seguir descrevem detalhadamente os serviços de decisão e de seleção de nós ativos oferecidos pelo middleware e apresentam avaliações de seu desempenho.

7.1 SERVIÇO DE DECISÃO: ESTABELECIMENTO DA CONFIGURAÇÃO DA REDE

O núcleo do serviço de decisão é formado por um algoritmo de decisão. As entradas do algoritmo são informações obtidas dos perfis das aplicações e das bases de dados de sensores. Como visto na Seção 6.3, as saídas do algoritmo consistem no modelo de entrega de dados, no protocolo de disseminação de dados, e na topologia lógica da rede a serem adotados.

Para o algoritmo de decisão, um interesse síncrono (ver Seção 6.2) enquadra uma aplicação no modelo de entrega iniciado pelo emissor. Uma consulta de longa duração é melhor atendida pelo modelo de entrega contínuo, enquanto consultas sobre eventos específicos indicam um modelo de entrega baseada em eventos. Muitas vezes, um interesse assíncrono especifica a necessidade de monitoramento contínuo e, ao mesmo tempo, a necessidade de ser notificado sobre algum evento específico. Por exemplo, uma aplicação pode estar interessada em monitorar continuamente, com uma taxa baixa de dados, os valores de temperatura de uma floresta a fim de construir gráficos mensais de tais valores. Porém, ao mesmo tempo, tal aplicação deseja ser notificada imediatamente sobre temperaturas acima de um limiar estabelecido, que indique a possibildiade da ocorrência de um incêndio. Tais requisitos apontam para o uso de um modelo de entrega híbrido. O tipo da consulta, a área geográfica alvo (em relação à área total coberta pela

rede) e o número estimado de nós fontes (geradores de dados) são utilizados para decidir entre cinco estratégias de disseminação de dados/topologias: baseados em difusão direcionada no modo *pull* [56]; baseados em difusão direcionada modo *push* [56]; comunicação direta com o sorvedouro; topologia hierárquica de 1 nível; e topologia hierárquica de 2 níveis. A versão atual do algoritmo de decisão adotado é mostrada na Figura 26.

```
Se interesse é Síncrono (consulta simples)
  Se a área alvo é próxima do Sorvedouro
     Se há muitos sensores na área alvo
        Usar difusão direcionada
    Senão
        Usar unicast com comunicação direta com o sorvedouro
  Senão
    Usar difusão direcionada com encaminhamento geográfico (GEAR)
Se interesse é Assíncrono
  Se é consulta de longa duração (envio de dados periódico)
     Se área alvo possui grande quantidade de nós
      Usar protocolo baseado em clusters
      Determinar número de clusters e número máximo de nós de um cluster
      Se todos os líderes de clusters estão ao alcance do sorvedouro
        Usar protocolo de hierarquia de um nível
      Senão
        Usar protocolo com 2 ou mais níveis de hierarquia
     Senão
      Usar protocolo com topologia plana baseado em algoritmo guloso
  Se é detecção de eventos
     Se alta freqüência de envio de dados
      Se área alvo possui grande quantidade de nós
        Usar protocolo baseado em clusters
      Senão
        Usar difusão modo pull com encaminhamento geográfico
     Senão
        Usar difusão modo push
```

Figura 26: Algoritmo de Decisão do Middleware

Os trechos do algoritmo em negrito representam valores que devem ser obtidos através de exaustivas simulações. Alguns parâmetros utilizados na decisão podem ser explicitamente definidos pelo usuário ao submeter seus interesses. Por exemplo, o número de sorvedouros a ser adotado pode ser definido pela aplicação. Como o algoritmo de decisão tem muitos sentenças que envolvem parâmetros subjetivos, como "se há **muitas** fontes de dados" ou "se as fontes de dados são **próximas** ao sorvedouro", um modelo baseado em lógica nebulosa pode ser empregado para aprimorar os resultados do processo de decisão. O desenvolvimento desse modelo é assunto para trabalhos futuros.

7.2 AVALIAÇÃO DO SERVIÇO DE DECISÃO

Simulações foram realizadas utilizando-se uma parte do algoritmo de decisão apresentado na Figura 26, com o intuito de mostrar que a configuração da rede de

sensores de acordo com os requisitos específicos das aplicações pode aumentar o desempenho global da rede, em termos de consumo de energia. Adotou-se como métrica de desempenho da rede a **energia média dissipada** [66], que é uma medida definida como a razão da energia total consumida por nó na rede sobre o número de eventos distintos entregues aos nós sorvedouros, e é diretamente relacionado ao tempo de vida da rede.

Nas simulações adotou-se o protocolo de disseminação de dados Difusão Direcionada (DD [66]) e foram simulados os diferentes modos de operação do protocolo (*pull* e *push*), de acordo com diferentes requisitos da aplicação. Foi implementado um *driver* XML do tipo 1, que converte mensagens XML para o formato adotado pelo DD, e vice-versa. O formato adotado pelo protocolo DD para representar dados e interesses consiste em vetores de atributos escritos na linguagem C++. Implementou-se uma parte do algoritmo de decisão usado pelo middleware, responsável por decidir entre dois modos de operação do DD: *two phase pull* ou *one phase push* [56]. As simulações foram realizadas no simulador de redes NS-2, na versão 2.27 [97]. O NS é um simulador de redes orientado a eventos discretos amplamente utilizado pela comunidade acadêmica.

Como resultado das avaliações, foram obtidos ganhos de até 25% da energia média dissipada quando selecionando o modo de operação do DD mais apropriado segundo a aplicação, de uma forma transparente para os desenvolvedores da aplicação. A seguir são descritas as simulações realizadas.

7.2.1 Descrição das Simulações

Os fatores que influenciam a escolha entre os modos *pull* e *push* são o número de sorvedouros, o número de nós gerando dados (fontes) e a taxa de envio de dados desejada pela aplicação. Esses parâmetros foram variados nas simulações e os resultados obtidos foram utilizados para validar e refinar o algoritmo de decisão utilizado pelo middleware. Um campo sensor com área retangular de 200m X 200m foi simulado (Figura 27). Foram gerados cenários com 50 e 100 nós, aleatoriamente distribuídos no campo. Nós fontes foram selecionados aleatoriamente dentre os nós sensores localizados em um retângulo de 70m X 70m dentro do campo sensor (área alvo). No modelo de energia adotado, a potência dissipada foi configurada para 35mW em modo ocioso, 395mW em modo de recepção e 660mW em modo de transmissão [66]. Esse modelo é o mesmo reportado nos

artigos descrevendo o protocolo DD [42,66]. O alcance do rádio foi configurado para 40m.

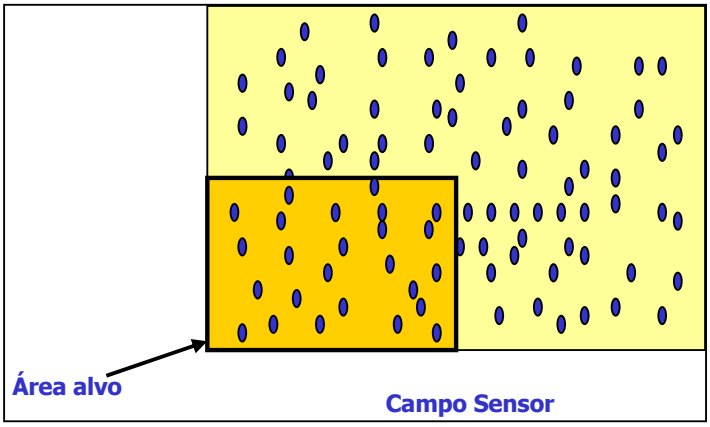


Figura 27: Esquema do cenário simulado

O primeiro parâmetro analisado foi o número de nós sorvedouros, que foi variado de 1 a 10. Como pode ser observado no gráfico da Figura 28, a variação do número de nós sorvedouros não mostrou diferença significativa entre os modos *pull* e *push*. As barras de erro mostradas nas figuras representam o intervalo de confiança de 95% dos resultados. Esse resultado pode ser atribuído ao fato de que o *overhead* de mensagens de controle gerado pelo modo *pull* não é ainda significativo para até 10 sorvedouros. Como em cenários típicos de instalação em geral há apenas um pequeno número de sorvedouros, não se considerou necessário simular um número maior desses nós.

O segundo parâmetro avaliado foi a variação do número de nós fontes. Para números de fontes variando de 1 a 7, o modo *push* dissipou menos energia. A partir de 7 fontes, o modo *pull* passou a apresentar os menores valores de energia dissipada (Figura 29). Esses resultados comprovam o melhor desempenho do modo *pull* para cenários com muitos nós gerando dados e indicam que o número de fontes é um fator importante na decisão da configuração da rede.

Para avaliar as duas variações do DD em função da taxa de dados, a taxa de envio das mensagens de interesse e de dados exploratórios foram mantidas conforme a implementação original do protocolo, respectivamente 30 e 60 segundos [66]. O tamanho das mensagens de dados e de interesses é, respectivamente, 64Bytes e 36Bytes. Nessas condições, os modos *pull* e *push* dissipam a mesma quantidade de energia em taxas altas de dados (1 dado a cada 5 seg). Com taxas mais baixas, o *pull* dissipa cerca de 25% a

mais do que o *push*, ou seja, a rede trabalha mais para entregar informação útil para a aplicação. Esse *overhead* deve-se ao fato do número de mensagens de controle no modo *pull* equiparar-se ao número de mensagens de dados, quando taxas muito baixas são usadas.

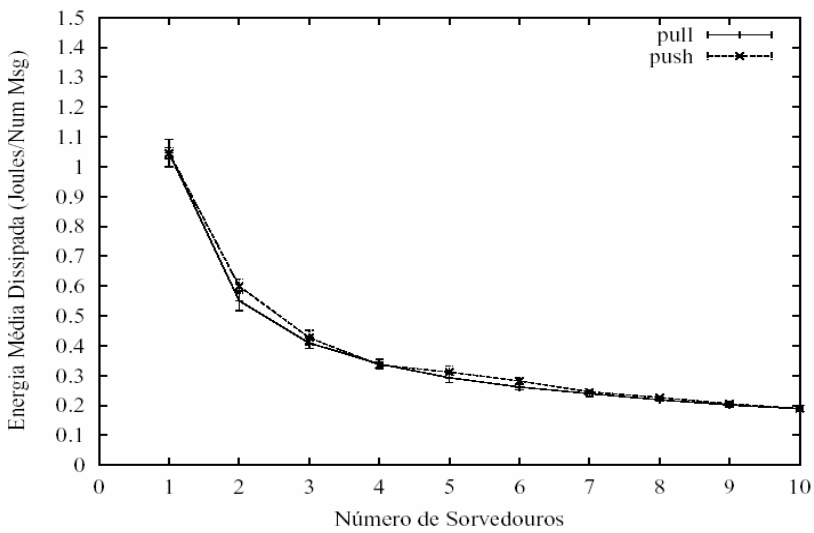


Figura 28: Variação do número de nós sorvedouros em cenários com 50 nós

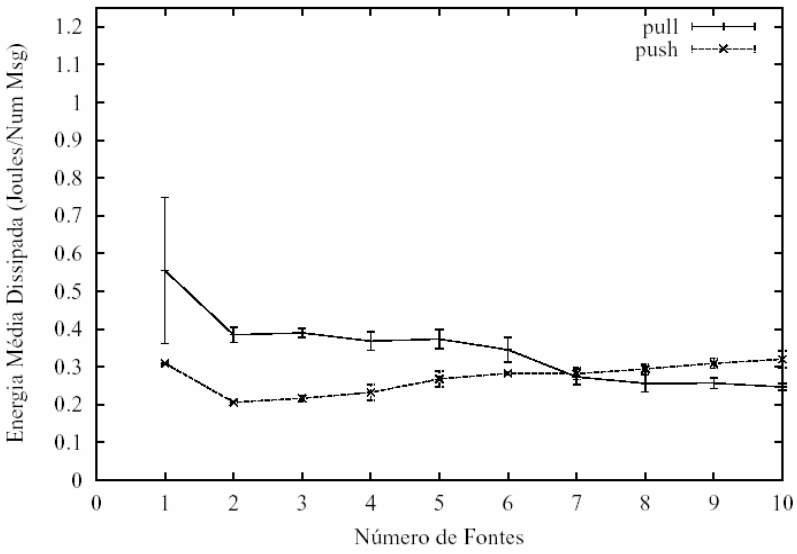


Figura 29: Variação do número de nós fontes em cenários com 50 nós

Como conclusão, o algoritmo de decisão do *middleware* deve basear sua escolha entre *pull* e *push* em dois fatores principais: (i) se há grande quantidade de nós capazes de gerar dados de interesse para a aplicação, usar o modo *pull*; (ii) se a taxa de dados desejada pela aplicação for muito baixa, usar o modo *push*, ou ajustar as taxas de envio das mensagens de controle de acordo com a taxa de dados.

É importante ressaltar que as principais vantagens do serviço de decisão oferecido pelo middleware são obtidas quando mais de uma opção de protocolo estão disponíveis e a opção mais adequada é transparentemente escolhida, com base nas informações extraídas dos interesses submetidos pelo usuário. A avaliação de tais benefícios, combinando cenários com diferentes características com o uso de diferentes topologias e protocolos, é tema de trabalhos futuros .

7.3 SERVIÇO DE SELEÇÃO DE NÓS ATIVOS

A seleção dos nós ativos é uma tarefa de gerenciamento da rede, que tem por objetivo aproveitar suas características intrínsecas, como a alta densidade de nós e a conseqüente redundância da informação sensoriada, para otimizar seu funcionamento e prolongar seu tempo de vida.

O esquema utilizado pelo middleware para a seleção de nós visa maximizar o tempo de vida da rede enquanto garante a QoS solicitada pela aplicação [35]. O algoritmo adotado pelo serviço de seleção busca selecionar o melhor conjunto de nós para a execução da tarefa de sensoriamento recebida.

Duas estratégias podem ser utilizadas para prolongar o tempo de vida de uma RSSF: (i) minimizar o consumo de energia da rede escolhendo o menor número possível de nós capazes de fornecer a qualidade desejada pela aplicação; e (ii) maximizar a energia residual dos nós escolhidos, ou seja, consumir energia de modo uniforme entre os sensores ao longo de tempo, evitando assim, a morte prematura de nós excessivamente utilizados. Ambas as estratégias são utilizadas no algoritmo proposto. Além disso, o algoritmo leva em consideração a relevância potencial, do ponto de vista da aplicação, dos sensores individuais.

O esquema para seleção de nós foi modelado como um problema da mochila, com algumas restrições adicionais. As próximas seções descrevem a formalização do problema de seleção dos nós ativos em uma rede de sensores como um Problema da Mochila, sua solução através de um algoritmo guloso e os resultados da avaliação desse algoritmo.

7.3.1 Modelos da rede e da aplicação

Antes de apresentar a formalização do problema de seleção de nós ativos adotada no presente trabalho, é necessário descrever os modelos adotados, com as respectivas restrições assumidas.

No modelo de rede adotado assume-se que todos os nós da rede conhecem suas coordenadas geográficas e as coordenadas de seus vizinhos. As coordenadas do nó podem ser obtidas através do uso de dispositivos GPS ou através de algoritmos de triangulação [93]. As coordenadas dos vizinhos podem ser transmitidas (i) em mensagens iniciais de configuração, durante a inicialização da rede; (ii) como mensagens de HELLO solicitadas; ou (iii) podem ser enviadas em mensagens de dados.

Dois diferentes canais são considerados: um canal de comunicação e um canal de *paging* de baixa potência. O canal de *paging* implementa um mecanismo para "acordar" os sensores que estão em modo inativo. Esquemas de sincronização temporal (por exemplo, [40]) estão disponíveis nos nós e suas mensagens de sincronização são enviados pelo canal de *paging*.

Uma política de escalonamento de atividades de comunicação é empregada em cada nó e em cada canal de comunicação. É importante ressaltar que, a cada instante, um nó pode receber ou enviar dados usando no máximo um canal. Assume-se que os protocolos de rede e enlace subjacentes são capazes de escalonar a atividade de comunicação em cada canal, sem perda de dados causada por colisão ou interferência.

Os rádios de todos os nós possuem o mesmo alcance de transmissão. Além disso, os rádios possuem controle de potência, transmitindo sempre na potência mínima necessária para alcançar o destino (próximo salto).

Quanto à comunicação dos dados, essa é feita através de múltiplos saltos, desde a origem do dado até o nó sorvedouro, com sensores intermediários realizando agregação dos dados, sempre que solicitado pela aplicação. A área que cada sensor é capaz de monitorar (raio de cobertura) é definida como a área circular em torno do sensor com raio igual ao alcance do sensor.

Para o correto funcionamento do serviço de seleção proposto, é necessário construir um modelo de operação da rede, quanto ao consumo de energia de seus nós. No modelo

adotado, considera-se que todos os sensores na rede são capazes de operar em um modo *sleep* ou inativo ou de acordo com *K* modos ativos pré-definidos. Modos ativos considerados no presente trabalho referem-se ao papel que um sensor exerce para uma dada tarefa de sensoriamento. Dois principais papéis são considerados: (i) nó fonte, para sensores localizados dentro da área alvo; (ii) nó roteador, para sensores dentro ou fora da área alvo, responsáveis pelo encaminhamento dos dados de seus vizinhos. Um sensor pode desempenhar ambos os papéis simultaneamente. Em cada modo, um nó sensor dissipa uma quantidade diferente de energia [43]. O consumo total de energia em um nó pode ser definido como a soma dos valores de energia gastos com sensoriamento, transmissão, recepção e processamento. Conhecendo-se a taxa de transmissão de dados, as distâncias entre os nós no caminho desde a origem de dados até o sorvedouro e a duração do monitoramento, pode-se derivar o gasto de energia dos sensores para realizar uma tarefa de sensoriamento. Um sensor no modo inativo consome uma quantidade de energia que pode ser considerada desprezível. A energia gasta para ligar e desligar os dispositivos de hardware do sensor também é considerada desprezível.

Os diferentes modos de operação de um sensor e seus papéis são representados na Tabela 1.

Tabela 1: Modos de Operação e Papéis dos Sensores.

Modo	Papel	Dispositivo de Sensoriamento	Processador	Transmissor	Receptor
Inativo	---	*Off*	*Off*	*Off*	*Off*
Ativo	Fonte	*On*	*On*	*On*	*Off*
Ativo	Roteador	*Off*	*On*	*On*	*On*
Ativo	Fonte/Roteador	*On*	*On*	*On*	*On*

Com relação à aplicação, dentre as diversas classes de aplicações para redes de sensores, uma aplicação de monitoramento ambiental interessada em receber medições contínuas sobre o fenômeno monitorado foi escolhida como modelo no presente trabalho. A aplicação define uma *taxa de envio de dados*, *uma região geográfica de interesse*, *o tempo total de monitoração* e, opcionalmente, uma ou mais *funções de agregação a serem aplicadas nos dados brutos*. Um exemplo de tarefa de sensoriamento submetida por uma aplicação desse tipo seria: "reportar a temperatura média, a máxima e a mínima de uma região, nas próximas 24horas, amostrando a região a cada 60 segundos". Esse tipo de aplicação não possui requisitos rígidos de latência, porém, requer um certo nível de

acurácia dos dados gerados e tempos de vida longos da rede, a fim de capturar variações com longo ciclos nos fenômenos monitorados.

7.3.2 Problema da Mochila na Seleção dos Sensores Ativos em uma RSSF

O Problema da Mochila (*Knapsack Problem* [26]) pode ser enunciado da seguinte forma: dados um número $M \geq 0$, um inteiro positivo N e, para cada i em $\{1;.... ; N\}$, um número $v_i \geq 0$ e um número $w_i \geq 0$, encontrar um subconjunto S de $\{1;.... ; N\}$ que maximize $v(S)$ sujeito à restrição $w(S) \leq M$, onde $v(S)$ denota a soma $\Sigma_{i \in S} v_i$ e, analogamente, $w(S)$ denota a soma $\Sigma_{i \in S} w_i$. Os números v_i e wi podem ser interpretados respectivamente como utilidade (ou valor) e peso de um objeto i. O número M pode ser interpretado como a capacidade de uma mochila, ou seja, o peso máximo que a mochila comporta. O objetivo do problema consiste, então, em encontrar uma coleção de objetos, a mais valiosa possível, que respeite a capacidade da mochila. Ou seja, o algoritmo para resolver o Problema da Mochila busca maximizar a utilidade dos objetos colocados numa mochila de capacidade limitada.

Seja T uma tarefa de sensoriamento solicitada por uma aplicação. No algoritmo utilizado no presente trabalho, as seguintes assunções são feitas:

- o tempo é dividido em ciclos ou *rounds j*, de duração p, durante os quais o sub-conjunto de nós selecionados e o papel de cada nó (sensor ou roteador) permanecem constantes;

- uma tarefa T lançada no início de um *round j* pode durar um intervalo de tempo igual a um número inteiro múltiplo de p. O primeiro ciclo de uma tarefa submetida pela aplicação começa logo após a execução do algoritmo de seleção.

Para a formalização do problema de seleção dos sensores ativos como um Problema da Mochila, considera-se que:

M = orçamento de energia

N = número total de sensores existentes na rede

v_i = utilidade do nó para a tarefa T

w_i = consumo de energia (peso) do nó i

Neste trabalho, os objetos considerados no problema são os nós sensores, com seus respectivos pesos e utilidades. O orçamento de energia é definido como a quantidade máxima de energia que se aceita consumir na rede para a execução da tarefa solicitada. Quando todos os nós são escolhidos para participar da tarefa, o orçamento (100%) é a soma das energias residuais de todos os nós. Como a capacidade da mochila é definida em termos da quantidade de energia para realizar uma tarefa, o peso do nó, w_i, também é dado em termos de energia. Portanto, w_i é o custo de energia (sensoriamento e comunicação) do nó sensor *i*, caso seja escolhido para participar da tarefa de sensoriamento *T*. Tal custo, por sua vez, depende de:

- tipo do dispositivo de sensoriamento (temperatura, movimento, imagem);
- modo de operação do sensor *i* durante a execução da tarefa *T;*
- tempo durante o qual o sensor opera;
- taxa de aquisição dos dados (taxas de sensoriamento e de transmissão);
- quantidade de dados que o sensor encaminha.

Para o valor v_i, ou utilidade do nó *i*, adotou-se uma abordagem que prioriza nós com valores maiores de energia residual U_i e relevância R_i. O valor U_i significa a quantidade corrente de energia do nó *i*. O valor R_i refere-se à relevância potencial do nó *i* para a aplicação, e determina o quanto tal nó pode contribuir para fornecer informação relevante para a aplicação. Assim, a relevância de um nó depende das suas características físicas e topológicas descritas a seguir:

PN_i = precisão nominal do sensor i

F_i = parâmetro associado ao ruído ambiental da medição realizada pelo sensor i

N_i = parâmetro associado aos sensores vizinhos de i, em termos de sensoriamento

A_i = parâmetro que representa a proximidade da área alvo definida pela aplicação

Com a aplicação do Problema da Mochila na seleção de nós ativos, a soma da utilidade dos nós colocados na mochila é otimizada sob a restrição do orçamento de

energia considerado. O algoritmo busca maximizar R_i e a energia residual final dos sensores escolhidos. A função objetivo do problema é dada a seguir:

$$\text{Max } \Sigma\, x_i\, (\propto R_i + \beta\,(U_i - w_i)) \quad \text{sujeito a } \Sigma\, x_i\, w_i \leq M \qquad (1)$$

onde $x_i \in \{0,1\}$.

Um valor 0 na variável x_i indica que o sensor i não está selecionado para participar da tarefa T. O termo $U_i - w_i$ corresponde à energia final do sensor i, caso ele seja escolhido para a tarefa (energia residual inicial, U_i, menos o gasto do sensor na tarefa, w_i). Os coeficientes $\propto$ e β são usados para balancear as prioridades dadas a cada termo da equação, e dependem dos requisitos de QoS da aplicação. Como valores *default*, são usados $\propto = \beta = 1$.

Conforme descrito anteriormente, o termo R_i é função de diferentes parâmetros. Cada parâmetro contribui com um peso diferente para o cálculo de R_i. O valor de PN_i é uma característica física de cada sensor, que depende apenas de seus coeficientes e de sua curva de calibração. Como tal parâmetro varia muito pouco de sensor para sensor e seu valor é baixo [84], assume-se que PN_i possui o menor peso dentre todos os termos para o cálculo de R_i.

O parâmetro F_i corresponde ao ruído ambiental presente nas medições realizadas pelo sensor i. Ele é influenciado principalmente pelas características físicas do local onde i foi instalado. Os ruídos presentes nas medições de um sensor limitam a acurácia que pode ser obtida no dado entregue para a aplicação. O parâmetro F_i é, na realidade, um valor normalizado que depende do nível real de ruído ambiental, S_i, o qual varia de 0 a 100 [84]. Quanto maior o ruído S_i menor a acurácia fornecida. Como F_i pode alcançar um valor relativamente alto e varia bastante de sensor para sensor, atribuiu-se a F_i um peso maior do que o de PN_i, cujo valor é dado pela equação:

$$F_i = 1 - \frac{\text{Si}}{100} \qquad (2)$$

Os maiores pesos foram atribuídos aos parâmetros A_i, que representa a proximidade da área alvo definida pela aplicação, e N_i, que representa o número de sensores vizinhos de i, em termos de sensoriamento**.** Os valores desses dois parâmetros são altamente correlacionados. Por exemplo, um sensor muito próximo da área alvo e que possua um

número muito pequeno de vizinhos, tem uma relevância muito alta para a tarefa de sensoriamento. Por outro lado, um sensor distante da área alvo e com elevado número de vizinhos tem uma relevância baixa.

O valor de N_i é inversamente proporcional à quantidade de vizinhos do sensor. A importância do valor medido por um nó em um local *X,Y* é proporcional à fração com que ele contribui para o seu sensoriamento desse local. Quanto mais medições são realizadas no mesmo local, menor a fração com que cada nó contribui para o seu sensoriamento. O número máximo de vizinhos de um nó pode variar de zero até *N- 1* (função da densidade da rede). Então o valor N_i pode variar de 1 (valor máximo, quando o número de vizinhos é 0), a até $1/N-1$.

Quanto ao valor de A_i, sensores com distâncias d_i da área alvo maiores do que o alcance do rádio R_r são automaticamente excluídos da seleção. Para sensores localizados a distâncias menores do que o alcance do rádio, atribui-se a A_i um valor normalizado. Tal valor é obtido dividindo-se inicialmente a distância pelo alcance de rádio. Como se deseja atribuir um valor de relevância menor a sensores localizados a maiores distâncias da área alvo, aplica-se a fórmula:

$$A_i = 1 - \frac{d_i}{R_r} \qquad (3)$$

A partir da correlação observada entre A_i e N_i, e dos diferentes pesos com que cada parâmetro contribui no cálculo da relevância de um sensor para a aplicação, a seguinte equação é utilizada para obter R_i:

$$R_i = \delta PN_i + \phi F_i + \gamma\left(\frac{1}{A_i N_i}\right) \qquad (4)$$

onde ϕ, δ e γ são coeficientes que representam os pesos de cada parâmetro, e $\delta < \phi < \gamma$.

7.3.2.1 Inclusão de Informações quanto à Prioridade de QoS

Considerando a classificação dos perfis da aplicação quanto à prioridade dada aos diferentes requisitos de QoS, apresentada na Seção 6.2, a função objetivo original (1) é

alterada para incluir pesos diferentes conforme a prioridade definida. Para perfis baseados no desempenho, atribuem-se valores maiores para o coeficiente α. Para perfis baseados no tempo de vida, atribuem-se valores maiores para o coeficiente β. Finalmente, para perfis baseados na razão são atribuídos valores iguais para os dois coeficientes.

7.3.3 Algoritmo de Programação Dinâmica para o Problema da Mochila

O Problema da Mochila pode ser resolvido de forma ótima através de um algoritmo baseado na técnica de Programação Dinâmica. A técnica de Programação Dinâmica consiste em dividir um problema em subproblemas menores. As soluções dos subproblemas menores são computadas e armazenadas. A partir dessas soluções, são computadas soluções para subproblemas maiores, até chegar à solução final. A vantagem dessa técnica está em nunca recalcular a solução para um subproblema que já foi resolvido. Uma discussão mais profunda sobre Programação Dinâmica pode ser encontrada em [26]. As complexidades espacial e temporal do algoritmo baseado em Programação Dinâmica para resolver o Problema da Mochila são ambas de ordem $O(NM)$.

Devido à limitação de recursos computacionais nas RSSFs, soluções com alto grau de complexidade não são adequadas. Como alternativa para a solução do Problema da Mochila através de Programação Dinâmica, estudou-se uma abordagem heurística, de menor ordem de complexidade, descrita a seguir.

7.3.4 Heurística Gulosa para o Problema da Mochila

Pode-se adotar uma heurística gulosa para o Problema da Mochila, a qual consiste em um algoritmo de dois passos: (i) primeiro, os ítens são ordenados segundo a razão entre suas utilidades e seus pesos, isto é, $\frac{vi}{w1} \leq \frac{v2}{w2} \leq .... \leq \frac{vn}{wn}$; (ii) segundo, os ítens são inseridos na mochila de forma inversa à ordem dada por (i), até que um item não "caiba" na mochila (peso do item maior que a capacidade disponível na mochila).

O algoritmo é guloso porque toma decisões locais que parecem ser as mais promissoras no momento (isto é, insere o primeiro item da lista) e, uma vez tomada a decisão, ela jamais é reconsiderada. A Figura 30 apresenta a descrição detalhada do algoritmo.

Entrada: Conjunto Cn = {1, 2, ..., n},

```
  Para cada item i ∈ Cn um peso wi ∈ Z+ e um valor vi ∈ Z+
  M ∈ Z+
Saída: S ⊆ Cn  sujeito a condição Σ i∈ S  wi ≤ M
1: Ordena os itens em Cn segundo a razão vi  / wi
2: disponível := M
3: i := n
4: S := 0
5: enquanto disponível > 0 ∧ i  ≥ 1 faça
6:        se wi ≤ disponível então
7:    S := S ∪ {i }
8: fim se
9:        disponível := disponível – wi
10: i := i – 1
11: fim enquanto
```

Figura 30: Algoritmo Guloso para o Problema da Mochila

Esse algoritmo não produz a solução ótima, porém tem menor complexidade do que a solução dada por programação dinâmica. O algoritmo tem um laço do tipo *enquanto* que executa $O(N)$ vezes. A operação de inserir um item no conjunto S (linha 7) é $O(1)$, portanto a complexidade total do laço é $O(N)$. Se for implementada através do algoritmo *Quicksort*, a ordenação na linha 1 tem custo $O(NlogN)$, no melhor caso, e $O(N^2)$, no pior caso. Portanto, pode-se considerar que a complexidade do algoritmo é da ordem $O(N) + O(NlogN) = O(NlogN)$. Tal algoritmo é computacionalmente leve o suficiente para ser executado de forma *on-line*, dentro da rede de sensores.

7.3.5 Restrições

A escolha dos nós ativos em uma RSSF está sujeita a uma série de restrições, que devem ser levadas em conta por qualquer esquema de seleção adotado. A seguir enumeram-se as restrições consideradas no presente trabalho e os procedimentos utilizados para atendê-las.

7.3.5.1 Restrições de energia

Como RSSFs são altamente restritas em energia, a primeira restrição a ser considerada, **R1**, é justamente a quantidade de energia finita da rede. A cada round j, a energia consumida pelo conjunto de sensores selecionados não pode ser maior do que o orçamento de energia da rede naquele *round*:

$$\sum_{i=1}^{N} x_i w_i T_j \leq M_j \quad \forall j \qquad \textbf{(R1)}$$

N = número total de sensores na rede

x_i = {0,1}, indicando se o sensor *i* foi selecionado (1) ou não (0) para ficar ativo no *round* j

M_j = orçamento de energia para o *round* j

T_j = duração do round j

w_i = consumo de energia do sensor *i* durante o *round j*

A restrição **R1** já é considerada pelo algoritmo de seleção baseado na mochila, visto que a capacidade da mochila é o orçamento total da rede em cada *round* considerado.

Uma segunda restrição, **R2**, também relacionada com a energia, leva em conta que um sensor só é elegível para permanecer ativo em um round *j* se possuir energia suficiente para permanecer vivo até o final do *round*. A fim de atender essa restrição, criou-se um limiar mínimo de energia, *L*, que um nó deve possuir para ser elegível para seleção. Nesse limiar é considerado que se o nó está na área alvo, ele deve ter no mínimo energia suficiente para sensoriar na taxa definida e transmitir seus dados. Caso contrário, ele deve ter no mínimo energia para encaminhar os dados de seus vizinhos. A quantidade de dados gerada a cada *round* é facilmente inferida a partir da taxa de aquisição definida pela aplicação, do número de nós fontes considerados e da duração do *round*. A restrição **R2** pode ser então formulada como segue:

$$x_i \leq \frac{\mathrm{Ui}}{\mathrm{L}} \qquad \begin{cases} x_i = 0 : \textit{excluído} \\ x_i = 1 : \textit{elegível} \end{cases} \qquad (\mathbf{R2})$$

Como x_i é uma variável binária, pode-se observar que, se a energia residual U_i do sensor *i* é menor do que o limiar *L*, x_i é configurada para 0 (o sensor não pode ser selecionado). Caso contrário, se $U_i \geq L$, então a variável x_i pode ou não ser configurada para 1 (isto é, o sensor *i* é elegível).

A restrição **R2** pode ser incluída na mochila, acrescentando-se um *if* adicional ao algoritmo, ou pode ser resolvida através de um procedimento anterior à mochila. Nesse último caso, antes de rodar o algoritmo de seleção dos nós ativos, um procedimento é executado para excluir do algoritmo os nós que estejam abaixo do limiar estabelecido. É importante destacar que, ao rodar tal procedimento sem conhecimento prévio das rotas selecionadas entre as fontes e o sorvedouro, os valores utilizados para o cálculo do limiar são bastante "pessimistas", já que se supõe que todo nó não fonte é um roteador. Por outro

lado, ao rodar tal procedimento após a rede estar ativa e as rotas formadas (depois da seleção inicial), pode-se levar em conta quanto de energia o nó irá consumir, considerando seu real papel na rede (fonte, roteador), para todas as aplicações em execução.

7.3.5.2 Restrição de Cobertura e Conectividade

Garantir ***cobertura de sensoriamento*** suficiente para atender a aplicação e ***conectividade da rede*** entre os nós ativos são requisitos críticos em qualquer RSSF. Como o principal propósito de RSSFs é monitorar o ambiente, uma rede tem que manter cobertura de sensoriamento suficiente na região de interesse, mesmo quando opera em um modo de baixo consumo de energia. A cobertura de sensoriamento é um dos fatores que caracteriza a qualidade do monitoramento fornecida por uma RSSF. Diferentes aplicações requerem diferentes graus de cobertura. Algumas aplicações podem requerer apenas que cada local em uma região seja monitorado por um nó, enquanto outras aplicações requerem graus de cobertura significativamente mais altos [144]. O requisito de cobertura pode ser dinamicamente alterado após uma rede ter sido instalada, devido a mudanças nos requisitos da aplicação ou nas condições ambientais.

Além de garantir a cobertura de sensoriamento, a fim de operar com sucesso, uma rede de sensores deve fornecer conectividade satisfatória, de modo que todos os nós ativos possam se comunicar para realizar a fusão de dados e reportar seus resultados para os nós sorvedouros.

A combinação das garantias de cobertura e conectividade constitui um requisito especial introduzido por RSSFs. Sem cobertura de sensoriamento satisfatória, a rede pode monitorar o ambiente sem obter acurácia suficiente ou pode até mesmo apresentar "sombras de sensoriamento" (locais onde não ocorre nenhum sensoriamento). Sem conectividade suficiente, os nós podem não ser capazes de se coordenar eficientemente ou transmitir seus dados para os sorvedouros [144].

Garantindo a cobertura e a conectividade: **O Algoritmo de Cobertura por Discos**. No presente trabalho, considera-se que um ponto p é coberto (monitorado) por um nó i se a distância Euclidiana entre eles é menor do que o alcance de sensoriamento de i, denominado R_s, i.e., $d(p,i) < R_s$. Considera-se também que a região de cobertura RC do sensor i é a área circular com centro em X,Y, onde X,Y são as coordenadas geográficas de

i, e cujo raio é igual a R_s. Uma região convexa *A* é definida como tendo um grau de cobertura *K* (ou seja, *A* é *K*-coberta) se todo ponto *p* dentro de *A* é coberto por no mínimo *K* nós [144]. Além disso, supõe-se que quaisquer dois nós *i* e *j* podem se comunicar diretamente um com o outro se sua distância Euclidiana é menor do que o alcance de comunicação dos nós, R_r, i.e., d(i,j) < R_r.

A restrição de cobertura e conectividade R3 pode ser formulada como se segue. Dada uma região convexa *A* e um grau de cobertura *K* especificado pela aplicação (antes ou após a instalação da rede), deve-se maximizar o número de nós inativos sob a restrição de que (i) os nós ativos garantam que *A* é no mínimo *K*-coberta e (ii) todos os nós ativos são conectados. Ou seja, para todo ponto *p* de *A*:

$\sum_{i \in A} x_i \geq K$ (grau de cobertura desejado pela aplicação) (**R3**)

Para atender a essa restrição, adotou-se um procedimento baseado no algoritmo de cobertura por discos [99], a ser executado antes da execução do algoritmo de seleção baseado na mochila. Esse procedimento consiste em duas etapas, a primeira para garantir a cobertura da região alvo e a segunda para garantir a conectividade da rede.

Na primeira etapa do procedimento, a região alvo, definida como a área retangular cujos limites são fornecidos pela aplicação, é totalmente coberta por discos cujo diâmetro é definido como a precisão espacial desejada pela aplicação. Em seguida, o procedimento seleciona heuristicamente *K* nós que devem ficar ativos dentro de cada disco. Essa seleção pode ser totalmente aleatória ou pode levar em conta a energia residual dos nós.

Já na segunda etapa, o campo sensor é totalmente coberto por discos de raio igual ao alcance do rádio, R_r. Para garantir a conectividade da rede, o procedimento deve garantir que em cada disco há pelo menos um nó ativo. Caso já haja um nó selecionado na primeira etapa do procedimento, tal nó exercerá ambos os papéis, de sensor e de roteador.

7.4 AVALIAÇÃO DO SERVIÇO DE SELEÇÃO DE NÓS ATIVOS

Os benefícios de se utilizar o serviço de seleção de nós ativos do middleware foram verificados através de simulações no JIST (Java *in Simulation Time* [71]), um simulador de eventos discretos escrito em Java. O algoritmo guloso do Problema da Mochila foi implementado. Neste trabalho, o algoritmo de seleção é executado no nó sorvedouro e assume-se que todas as informações necessárias para sua execução são transmitidas pelos

nós sensores em mensagens de dados e de configuração. Em trabalhos futuros, será simulada a execução do algoritmo em nós líderes de *clusters*, o que permitirá uma maior escalabilidade da solução.

Uma aplicação de monitoramento ambiental foi simulada. A tarefa de sensoriamento solicitada consistia em monitorar valores de temperatura de uma região alvo durante determinado período de tempo. A aplicação estava interessada nos dados brutos de temperatura (sem função de agregação), com os seguintes requisitos: (i) uma resolução espacial mínima de $40m^2$com um grau de cobertura igual a 1 (no mínimo 1 sensor a cada $40m^2$); (ii) uma taxa de aquisição de 1 dado a cada 10 segundos, e (iii) uma acurácia de dados acima de um limiar definido. A acurácia foi dada pelo valor do erro médio quadrático (MSE). O MSE foi calculado como a diferença entre um conjunto de valores considerados como "reais" e o conjunto de valores gerados pelos sensores. Quanto ao tempo de vida da rede, devia ser longo o suficiente para garantir que os dados fossem coletados durante o período de tempo solicitado pela aplicação, respeitando os requisitos de QoS.

A fim de simular medições reais efetuadas pelos sensores, o fenômeno físico a ser monitorado pela rede (no caso, os valores de temperatura) precisa ser definido. Para não restringir o algoritmo a cenários específicos, adotou-se um modelo genérico de processos físicos. No modelo, se há uma fonte de dado em um ponto *p*, seu valor é difundido no ambiente segundo uma potência da distância [13]. Então, os valores efetivamente reportados pelos sensores aos nós sorvedouros são uma função de sua precisão nominal, da distância a cada uma das fontes de dados e do ruído ambiental associado à medição. Adotou-se a equação abaixo:

$$V(p) = \frac{\sum_{numfontes} kdist(f) + 1^{-a} V(p(f)) PN_i + F_i}{numfontes} \qquad (5)$$

Na equação, $V(p)$ é o valor sensoriado no ponto *p*, $dist(f)$ é a distância entre o ponto *p* e a fonte *i*, $V(p(f))$ é o valor de dado "real" gerado pela fonte *f*, PN_i é a precisão nominal do sensor *i*, F_i o ruído associado à sua medição, e os valores *k* e *a* são coeficientes. Nas simulações foram utilizados os valores $a = 1$ e $k = 0.25$ [13].

Na versão atual do modelo considera-se apenas a variação espacial do fenômeno, eliminando-se a sua variação temporal. Portanto, os dados "reais" simulados fornecem uma visão instantânea do fenômeno físico monitorado. O fator tempo pode ser facilmente incluído no modelo e será assunto para trabalhos futuros.

Dados "reais" foram gerados a partir de uma distribuição gaussiana com média 100 e variância 10. Para a precisão nominal de cada nó atribuiu-se um valor gerado aleatoriamente entre 95 e 100. Tal faixa de valores foi escolhida a partir dos estudos reportados em [84] sobre a variação da precisão de diferentes tipos de sensores, sujeitos a condições ambientais reais. O ruído ambiental de cada medição foi gerado a partir de uma distribuição gaussiana com média zero e variância 1.

Um campo de sensores foi criado com 300 nós aleatoriamente distribuídos em uma área retangular de 200m x 200m (Figura 31). Cada nó tinha um alcance de rádio de 40m e um raio de sensoriamento de 20m. A dissipação de energia com sensoriamento e comunicação foi similar ao modelo dos nós Sensoria WINS 3.0, descrito em [43]. Nesse modelo, a dissipação de energia no modo *sleep* é cerca de 416.3mW; a dissipação no modo ocioso (*idle*) é 727.5mW; no modo de recepção é 751.6mW; e no modo de transmissão é 986.0mW. Ainda segundo o modelo, a energia consumida com processamento pode ser considerada desprezível.

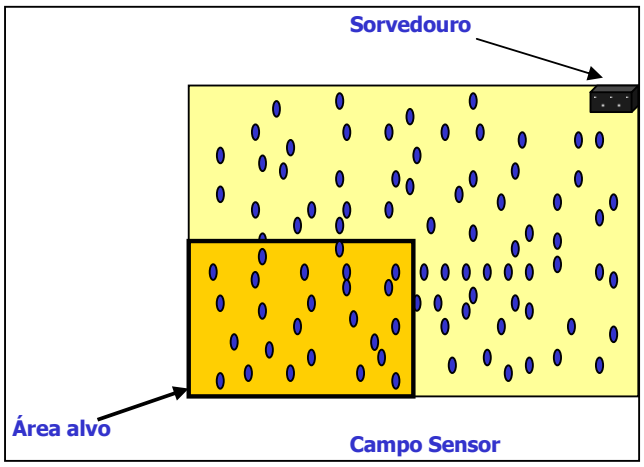


Figura 31: Cenário simulado

Sensores geradores de dados (fontes) foram selecionados aleatoriamente a partir de nós localizados em um retângulo de 100m X 100m (área alvo) dentro do campo sensor. Um único nó sorvedouro foi colocado na extremidade superior direita do campo sensor. Como nessa fase não há interesse em simular qualquer protocolo de comunicação

específico, assumiu-se a existência de protocolos hipotéticos, que entregam dados das fontes para o sorvedouro através do caminho mais curto (em termos de distância geográfica), sem perda de dados. Cada simulação roda por 1000 segundos, divididos em 10 ou mais ciclos, ao final dos quais a energia residual da rede e o MSE eram calculados. As barras de erro mostradas em todos os gráficos representam o intervalo de confiança de 95% dos resultados.

7.4.1 Analisando o Serviço de Seleção Proposto

Na primeira fase de simulações, foram comparados os resultados de se selecionarem diferentes porcentagens de nós ativos, em termos de energia residual da rede e acurácia de dados. Como há uma única aplicação usando a rede, o conjunto inteiro de nós poderia ser escalonado a fim de fornecer a melhor QoS possível. Entretanto, deseja-se mostrar que, ativando apenas um sub-conjunto de nós, pode-se satisfazer a QoS solicitada pela aplicação, economizando recursos da rede para novas tarefas e aplicações.

O orçamento de energia da rede (capacidade da mochila) é especificado como a porcentagem de nós ativos, a qual foi variada de 30% a 100%. Um valor de orçamento dado em porcentagem significa que a capacidade da mochila é configurada para a soma dos pesos das respectivas porcentagens de nós. Na abordagem gulosa, assume-se que os pesos de todos os nós são iguais e correspondem à sua energia inicial. Todos os sensores possuem um valor de energia inicial escolhido aleatoriamente entre 15 e 20 Joules. Antes de executar o procedimento de seleção, os nós são ordenados de acordo com sua relevância e energia residual, de modo que o procedimento possa dar prioridade a nós com maiores valores para esses parâmetros. Após executar o procedimento, as rotas das origens de dados (nós fontes) até o sorvedouro são estabelecidas e mantidas inalteradas até o final dos ciclos. Os nós não alocados para permanecer ativos são completamente desligados. O custo de energia para desligar e reiniciar o rádio é considerado desprezível.

O tempo de monitoramento solicitado pela aplicação corresponde a 10 ciclos e o MSE máximo tolerado é 0.3. Gráficos da energia residual da rede e do MSE ao final de cada ciclo, para todos os orçamentos considerados, são mostrados nas Figuras 32 e 33.

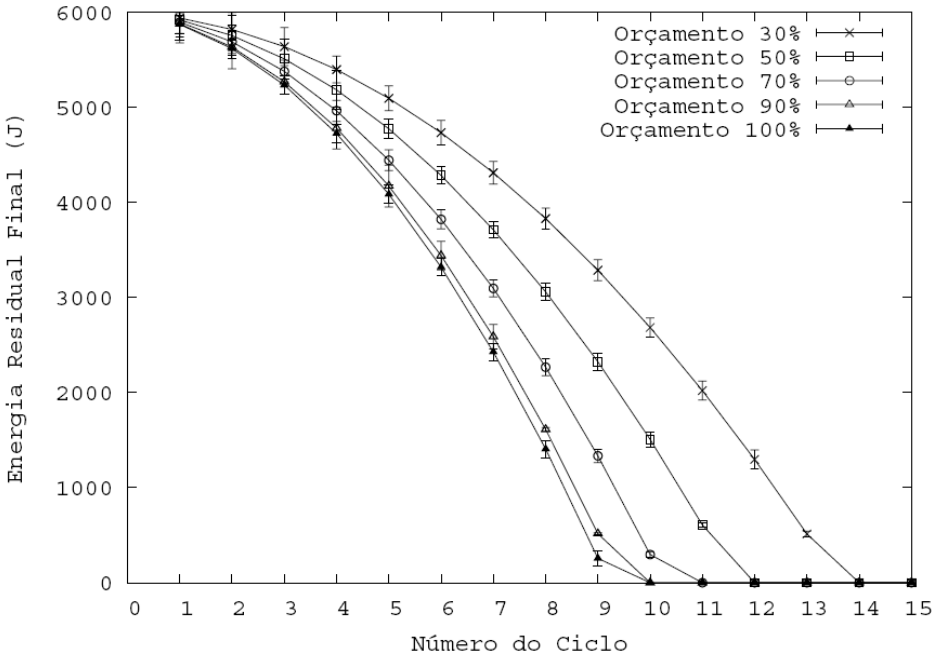


Figura 32: Energia residual da rede a cada ciclo, para os diferentes orçamentos.

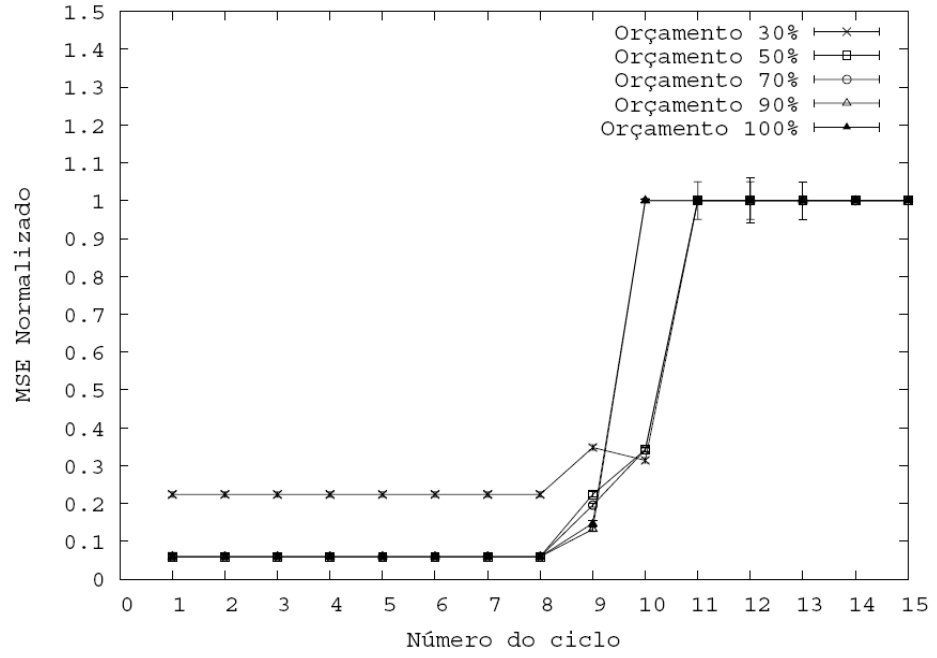


Figura 33: MSE normalizado a cada ciclo, para os diferentes orçamentos.

Um ganho de mais de 1000% de energia ao final do 10º ciclo é obtido quando apenas 30% dos nós são ativados, em contraste com a ativação de 100% dos nós. Nos resultados observou-se que o MSE para orçamentos de 40 a 100% comportou-se de forma bastante similar até o 8º ciclo. Para o orçamento de 30%, o MSE foi maior desde o primeiro ciclo. A partir do 8º ciclo, o MSE começou a aumentar, para todos os

orçamentos. Tal resultado deveu-se ao fato de, a partir desse ciclo, um número maior de sensores ter sua energia esgotada. A expiração do tempo de vida dos nós fontes ou de nós localizados no caminho entre fontes e sorvedouros impede a entrega de dados, aumentando o erro gerado. Embora o MSE aumente, até o nono ciclo ele permanece ainda abaixo do ponto tolerado pela aplicação, para todos os orçamentos, exceto o de 30%. No décimo ciclo, o MSE está abaixo do limiar tolerado (0.3) apenas para orçamentos de 80 a 100%, significando que a QoS solicitada não está mais sendo atendida pelos demais orçamentos. Como o tempo de monitoramento solicitado foi de 10 ciclos, os resultados mostram que com 80% de nós a QoS da aplicação pôde ser satisfeita, com uma economia de energia de mais de 200%. Economia de energia adicional foi obtida para orçamentos menores, porém às custas de não satisfazer os requisitos de acurácia da aplicação.

Para orçamentos menores do que 30%, a densidade da rede passou a ser muito baixa, ocasionando problemas de falta de cobertura e/ou conectividade, detectados no momento da execução do algoritmo dos discos. Portanto, tais resultados não foram apresentados nos gráficos.

É importante observar que há uma correlação entre a taxa de aquisição de dados e as porcentagens de nós que podem ser selecionados, a qual não foi explorada nesta versão do trabalho. Nas simulações realizadas, a aplicação solicita que dados sejam reportados a cada 10 segundos, uma taxa relativamente alta de aquisição. Com taxas de dados altas, ao se ativar uma porcentagem de nós muito baixa, os nós ativos terão uma carga alta de trabalho, esgotando sua energia rapidamente. Ainda que uma porção grande da rede permaneça viva, e possa ser ativada em futuros *rounds*, a morte prematura de nós pode fazer com que porções do campo sensor no caminho entre fontes e sorvedouro fiquem com uma baixa densidade de nós. Embora o algoritmo de cobertura por discos garanta a conectividade da rede, a densidade muito baixa em algumas regiões faz com que a chance dos mesmos nós serem sempre selecionados seja grande, novamente sobrecarregando tais nós. Então podem ocorrer rupturas na rede e a qualidade desejada pela aplicação pode não ser fornecida pelo tempo suficiente, apesar de ainda haver nós com energia. Ou seja, apesar da solução proposta para o escalonamento de nós tentar evitar isso, a rede pode acabar consumindo sua energia de maneira heterogênea entre os nós. Em trabalhos futuros a taxa de aquisição de dados será um parâmetro variado a fim de se analisar sua correlação com diferentes orçamentos e tempos de vida da rede.

A seguir, variou-se o número de nós sensores enquanto se mantinha o tamanho do campo sensor, para analisar o efeito da densidade da rede. A economia de energia obtida ativando-se 80% do nós para cenários com 400 nós foi similar a cenários com 300 nós. Para 200 nós, uma economia de energia menor, embora ainda significativa, de 300% foi obtida. Nos dois casos o comportamento do MSE seguiu o mesmo padrão observado para 300 nós. Esses resultados indicam que o esquema de escalonamento de nós é mais eficiente quanto mais densa for a RSSF.

7.4.2 Política de Adaptação

Evidências de que, para orçamentos inferiores a 80%, o erro obtido ultrapassava o limiar estipulado pela aplicação, apesar de ainda haver uma considerável quantidade de nós vivos na rede, levantaram as seguintes considerações: (i) executar o algoritmo de seleção apenas no primeiro ciclo, onde a energia residual dos nós é muito semelhante (considerando a rede recém instalada e não utilizada por outras aplicações), pode não ser uma estratégia eficiente; e (ii) adotar um orçamento fixo para a rede em todos os ciclos de uma tarefa também pode levar ao uso ineficiente da RSSF ou ao não atendimento dos requisitos da aplicação, apesar de haver recursos disponíveis.

Se um valor de erro acima do limiar desejado é detectado, um aumento do orçamento inicialmente estipulado poderia levar a rede a se recuperar da quebra de QoS e, dessa forma, a atender a aplicação por ciclos adicionais. Por outro lado, em casos onde o erro já apresenta valores abaixo do limiar logo nos primeiros ciclos, economia de energia adicional poderia ser obtida diminuindo-se o orçamento inicial.

Foram, então, simuladas estratégias de adaptação fornecidas pelo middleware a fim de avaliar o impacto dessas estratégias sobre o desempenho da rede. Em uma primeira etapa, avaliou-se apenas o efeito de diminuir o orçamento adotado pela rede, ao se observar que o valor do erro estava sendo mantido dentro do limiar. Nas simulações realizadas, o orçamento de energia era inicialmente configurado para 100% e a energia residual da rede e o MSE eram monitorados ao final de cada ciclo. Foram estabelecidos quatro limiares de consumo de energia, onde se levou em conta o número de nós na rede e suas energias iniciais. Caso algum limiar fosse ultrapassado, uma política de adaptação proativa era aplicada, cuja meta era reduzir o consumo de energia na rede, dessa forma prolongando seu tempo de vida. A ação implementando tal política consistiu em reduzir a

porcentagem de nós ativos na rede. Tal ação foi executada como uma nova execução do algoritmo de seleção, diminuindo-se o orçamento da rede, ou seja, a capacidade da mochila. Sendo assim, estabeleceu-se que, quando o primeiro limiar fosse ultrapassado, o orçamento seria diminuído para 90%, depois para 80%, e assim sucessivamente, enquanto o valor de acurácia dos dados gerados era monitorado para verificar se permanecia abaixo do valor solicitado.

As Figuras 34 e 35 apresentam gráficos da energia residual e do MSE ao final de cada ciclo, com e sem a utilização da estratégia de adaptação.

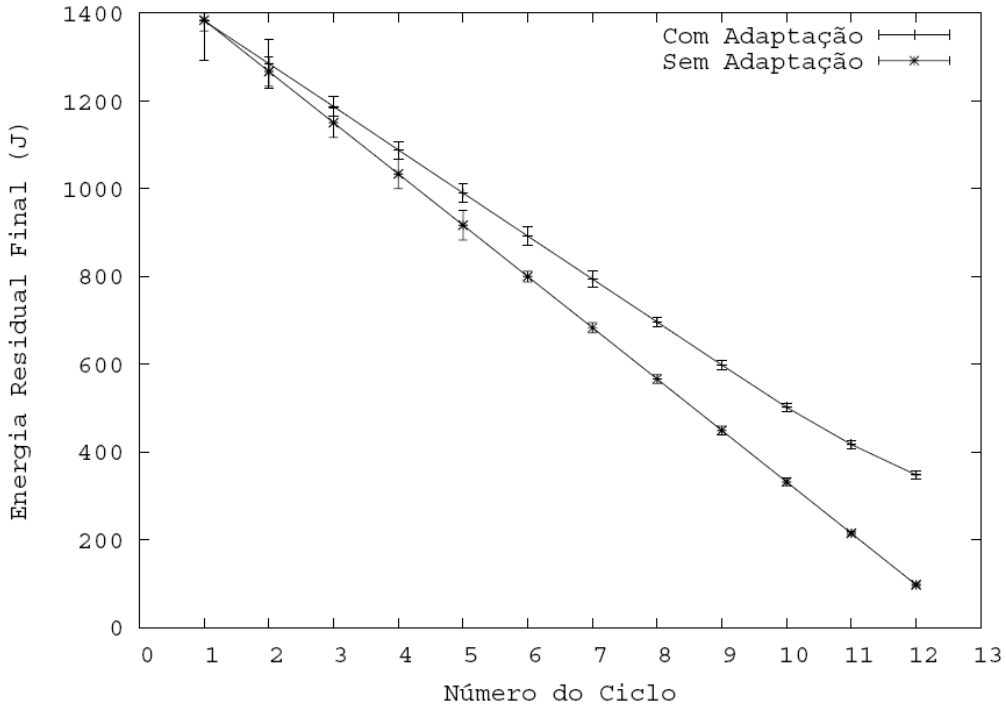


Figura 34: Energia residual da rede para cada ciclo, com e sem adaptação.

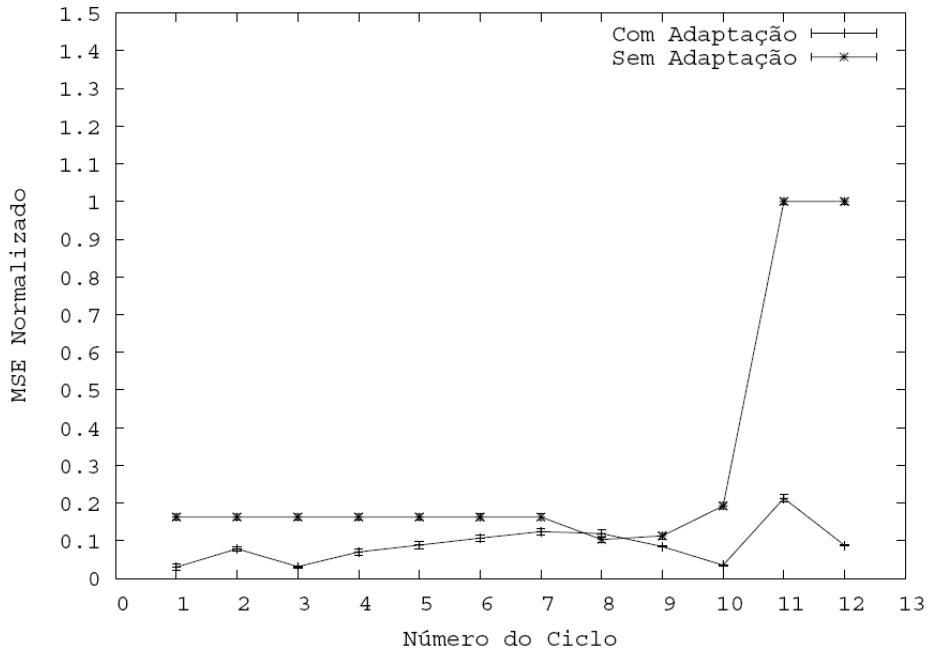


Figura 35: MSE Normalizado para cada ciclo, com e sem adaptação.

Os cenários utilizados são os mesmos descritos na Seção 7.4.1. O gráfico na Figura 34 mostra que, sem adaptação, a RSSF teve sua energia totalmente esgotada ao final do décimo terceiro ciclo. O valor de MSE ficou abaixo do limiar desejado apenas até o décimo ciclo. A principal razão para esse resultado foi a morte dos nós fontes devido à falta de energia. Adotando-se a política de adaptação, a rede ainda manteve cerca de 20% de energia residual ao final do décimo terceiro ciclo, com o erro sendo mantido dentro do limiar (Figura 35). Portanto, com a política adotada o tempo de vida da rede em termos de energia foi estendido para ciclos adicionais.

Uma limitação da estratégia simulada é que ela apenas busca economizar energia, através da diminuição do orçamento da rede sempre que possível. Entretanto, não é tratado o caso em que o erro ultrapassa o limiar desejado em ciclos iniciais, apesar de haver recursos disponíveis na rede.

Em uma segunda etapa de simulação, estabeleceu-se uma outra estratégia adaptativa na qual, a partir de um orçamento inicial (o qual nesse caso foi variado de 30 a 100%), em cada ciclo, ao se constatar que o valor do erro estava abaixo do limiar considerado, uma nova execução do algoritmo de seleção era realizada, com o orçamento sendo diminuído 10% a cada ciclo. Por outro lado, se o erro ultrapassava o limiar desejado, o orçamento

era aumentado, segundo a expressão: M = M + *a* M. A fim de determinar o valor mais apropriado para o coeficiente *a*, ele foi variado nas simulações de 0.1 a 0.5, com os melhores resultados (menos oscilações e mais melhoria no desempenho final do sistema) foram obtidos para *a* = 0.2.

Ao se realizar simulações com essa estratégia, observou-se que, diminuindo-se o orçamento quando o erro estava abaixo do limiar, economias de até 200% na energia final da rede foram obtidas para orçamentos de 100% a 70%, em comparação com a estratégia de manter fixo o orçamento. Para orçamentos menores do que 70% não houve variação significativa com o uso da estratégia adaptativa. Tal comportamento se deve ao fato de, com orçamentos baixos, o erro já estar próximo ao limiar logo nos primeiros ciclos, impedindo que o orçamento seja significativamente diminuído em ciclos posteriores.

Um comportamento não esperado com relação aos valores de erro foi que, quando se aumentava o orçamento para tentar resolver a quebra de QoS (valores de erro acima do limiar) nem sempre o erro diminuía. Observou-se que isso ocorreu devido ao fato de que alguns nós eram selecionados por possuírem alta energia residual, apesar de baixa acurácia. Como tentativa de se minimizar o problema, decidiu-se aumentar o peso atribuído ao valor de acurácia em relação a energia residual, ao selecionar os nós ativos. Para esse fim, alterou-se o coeficiente α da equação (1) para ter um peso maior que o do coeficiente β. É importante lembrar que, no esquema de seleção proposto, a utilidade de um nó é dada tanto por sua energia residual quanto pela sua relevância potencial para a aplicação. Portanto, no modelo de aplicação considerado, a relevância é uma medida diretamente relacionada a acurácia. Também foram aumentados os pesos atribuídos ao termo que representa a precisão nominal dos sensores e ao termo que representa o ruído ambiental (termos PN_i e F_i da equação (4)).

Os resultados mostraram que, adotando-se as melhorias descritas no processo de seleção, juntamente com a estratégia adaptativa, o tempo de vida da rede pôde ser estendido para mais de 20 ciclos (atendendo o requisito de QoS), sendo que a partir do 21º ciclo o requisito de cobertura da rede não foi mais atendido (Figura 36 e Figura 37). Ainda assim, houve nós com baixa acurácia sendo selecionados, pela sua alta energia residual. Observou-se que tais nós seriam boas escolhas como roteadores, porém não como fontes de dados. Entretanto, o algoritmo de seleção não distingue entre os papéis que um nó irá assumir ao ficar ativo. Como possível solução para esse problema, foi

implementada uma variação do algoritmo de seleção na qual os papéis de sensor e roteador são separados durante a escolha dos nós ativos. Nessa variação, o algoritmo passou a ser executado em duas etapas, onde se consideravam critérios diferentes para a seleção, ou seja, foram criadas duas "mochilas" diferentes.

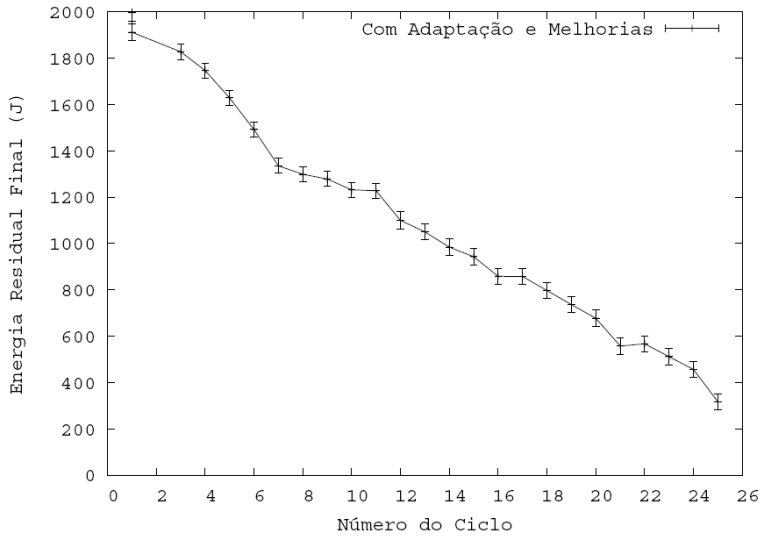


Figura 36: Energia residual ao final de cada ciclo, com adaptação e melhorias.

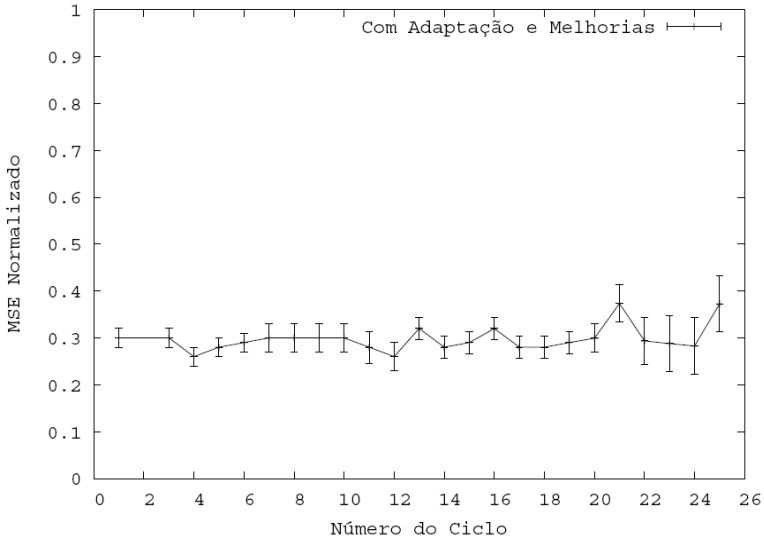


Figura 37: MSE ao final de cada ciclo com adaptação e melhorias.

7.4.3 Utilizando Duas Mochilas no Processo de Seleção

Nessa etapa de simulação, procurou-se explorar o fato de que os papéis de fonte e roteador requerem diferentes características dos nós. Nós com grande potencial para desempenhar o papel de roteador devem possuir maior número de vizinhos (considerando

o alcance do rádio) e uma alta energia residual. Já nós com grande potencial para atuar como fontes devem possuir um menor número de vizinhos (considerando o raio de sensoriamento) e uma alta acurácia de dados. Portanto, o algoritmo de seleção foi dividido em duas etapas, onde foram usadas duas funções objetivo diferentes. Na primeira, são selecionados apenas nós para atuar como fontes. A função objetivo utilizada foi alterada para considerar apenas as características mais relevantes para esse papel. Na segunda etapa, são selecionados os nós para atuar como roteadores. Da mesma forma, a função objetivo foi alterada para considerar as características relevantes para o papel de roteador.

Foram, então, executadas simulações utilizando essa variação do algoritmo de seleção e o mecanismo adaptativo descrito na Seção 7.4.2. Como resultado, o tempo de vida da rede foi estendido para mais de 30 ciclos, sendo que até o 28º ciclo os requisitos de QoS e cobertura da rede foram atendidos (Figura 38 e Figura 39). Um comportamento interessante observado foi que, ao se usar a variação com duas mochilas, quando se diminuía o orçamento da rede, como estratégia de adaptação, na maior parte das simulações o MSE também diminuía, em vez de aumentar, como foi observado com o uso de apenas uma mochila. Isso pode ser explicado pelo fato de que, ao priorizar exclusivamente os critérios relevantes para um nó atuar como fonte, com orçamentos menores os melhores nós fontes já eram selecionados. Ao se inserir novos nós, muitas vezes esses novos nós possuíam valores piores de acurácia, elevando o valor do erro.

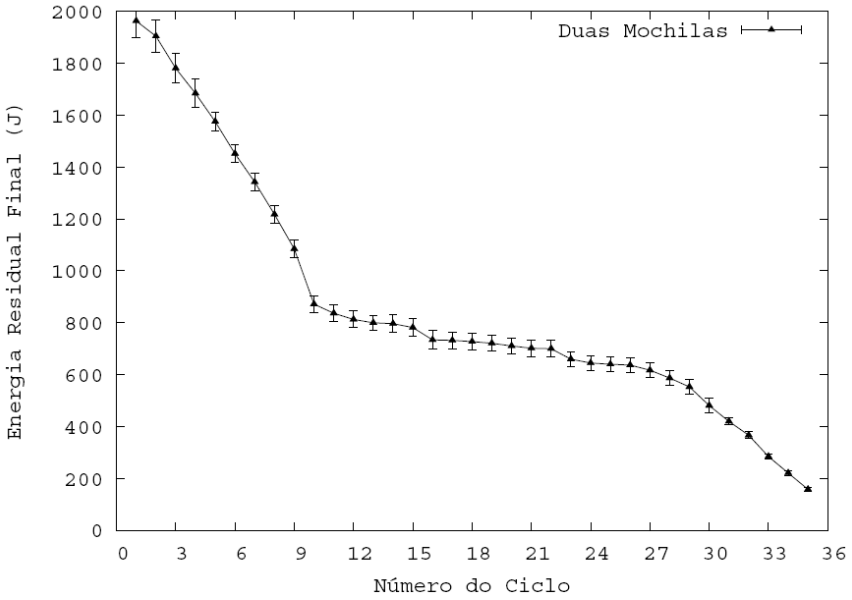


Figura 38: Energia residual ao final de cada ciclo, utilizando “duas mochilas”.

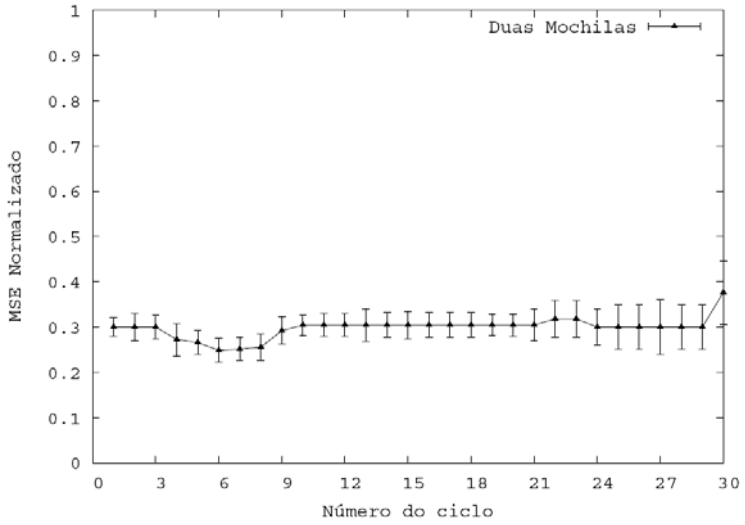


Figura 39: MSE ao final de cada ciclo, utilizando “duas mochilas”.

7.4.4 Considerando Agregação de Dados

Com freqüência, as aplicações estão interessadas em valores agregados (de acordo com alguma função), em vez de nos valores de dados individuais reportados pelos sensores. Com o intuito de avaliar o efeito da agregação sobre o esquema de seleção de nós proposto, simulou-se uma aplicação interessada em consultar os valores médios de temperatura na área alvo. Os mesmos cenários criados para as simulações anteriores foram utilizados, alterando-se apenas a descrição do interesse da aplicação.

As Figuras 40 e 41 apresentam os gráficos da energia residual e do MSE ao final do décimo ciclo de simulação, para cada orçamento, considerando dados brutos ou dados agregados. A Figura 40 mostra que os valores de energia residual não variaram significativamente para dados brutos ou agregados. Isso ocorreu porque simulou-se o serviço de agregação sendo implementado apenas no nó sorvedouro, após todos os dados sensoriados terem sido reportados. Idealmente, a agregação deve ser realizada em nós intermediários, dessa forma economizando energia de transmissão. A Figura 41 mostra que, para orçamentos de 30% e 40%, o MSE para dados brutos aumentou, enquanto permaneceu praticamente o mesmo para dados com agregação. Duas principais razões contribuíram para esse resultado. Primeiro, observou-se que, com um pequeno número de nós ativos, as fontes eram sobrecarregadas com a tarefa de rotear dados de vários nós vizinhos, dessa forma consumindo muita energia. Esse fato conduz à segunda razão: eventualmente nós fontes não são selecionados para ficarem ativos em um ciclo (por estarem com energia muito baixa). Quando a aplicação está interessada em valores

médios, a supressão de um pequeno número de fontes não tem um impacto significativo na acurácia de dados final, o mesmo não ocorrendo quando a aplicação deseja os dados brutos.

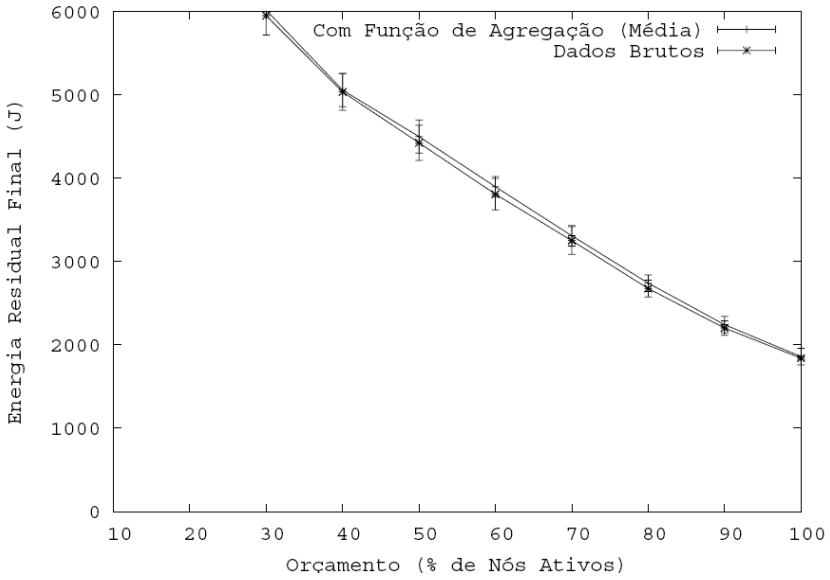


Figura 40: Energia residual da rede para os diferentes orçamentos, considerando dados brutos e agregados

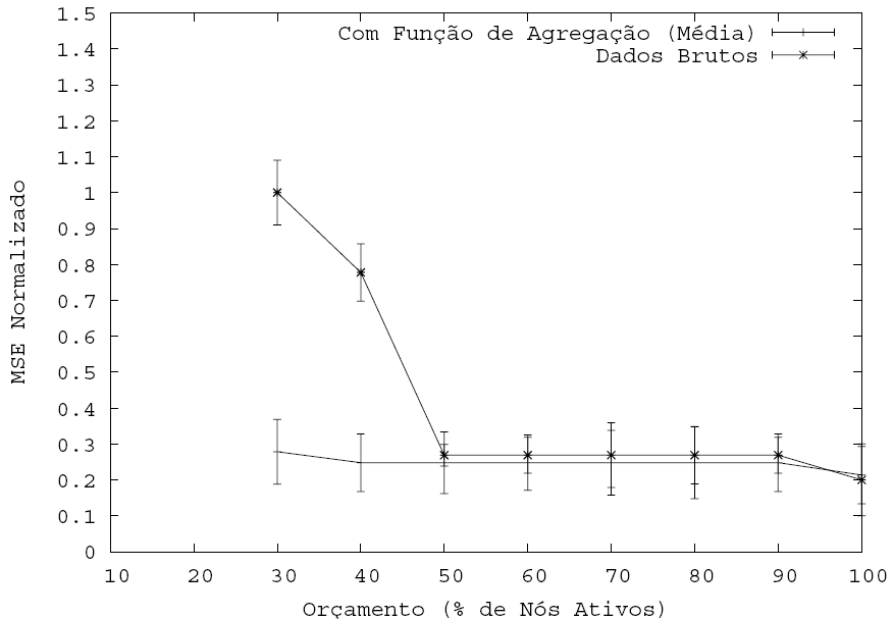


Figura 41: MSE Normalizado para os diferentes orçamentos, considerando dados brutos e agregados

Outro fator que tem impacto nos resultados quando se considera a aplicação de funções de agregação de dados é o grau de variabilidade espacial nos valores do fenômeno monitorado. Se o valor do fenômeno varia muito de um ponto para outro da

área alvo, um número maior de nós precisarão estar ativos para gerar dados agregados mais precisos, ou seja, que melhor reflitam os valores reais observados. Tal impacto é principalmente relevante se a função de agregação for do tipo MAX ou MIN. Nesses casos, a supressão de um único valor pode gerar um erro elevado, se tal valor for o maior (ou menor) naquela região. Em trabalhos futuros será avaliado o impacto da variação espacial do fenômeno monitorado sobre as diferentes funções de agregação. Nesses trabalhos, será incorporado ao modelo de geração de dados a distribuição temporal dos valores gerados.

7.4.5 Perfis de Aplicação quanto a Prioridade de QoS

As simulações anteriores (exceto no caso da segunda política de adaptação) assumiram um perfil de aplicação baseado na melhor razão custo/benefício entre QoS e consumo de energia. A seguir será avaliado o efeito de se usarem os diferentes perfis considerados no trabalho. Para o perfil baseado em desempenho, o valor do coeficiente $\propto$ foi configurado para 50, enquanto β foi configurado para 1. Para o perfil baseado em tempo de vida, o valor do coeficiente $\propto$ foi configurado para 1 e β para 50. O valor 50 foi escolhido por tentativa e erro em exaustivas simulações. Em trabalhos futuros pretende-se adotar uma metodologia para a análise mais cuidadosa do comportamento dos valores dos coeficientes $\propto$ e β.

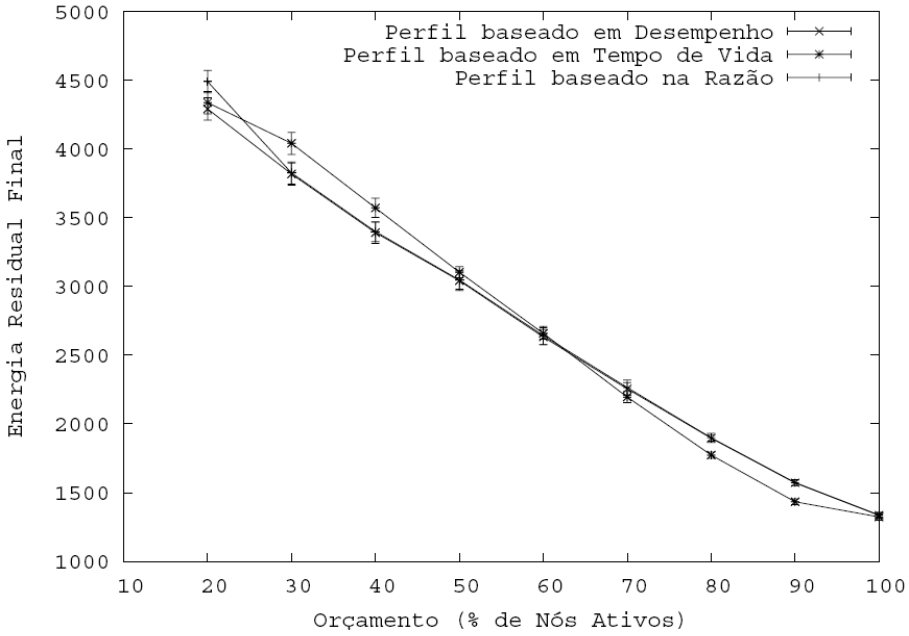


Figura 42: Energia residual da rede no 10º *round*, para os diferentes orçamentos, considerando os três perfis de QoS

As mesmas configurações de cenários descritas na Seção 7.4.1 foram utilizadas. A Figura 42 e a Figura 44 apresentam respectivamente gráficos dos valores de energia residual e MSE ao final do décimo ciclo de simulação, para cada orçamento e cada tipo de perfil. Os resultados mostram que a energia final não mudou significativamente entre os diferentes perfis. Pode-se atribuir esse resultado ao fato de que o algoritmo de seleção roda antes do primeiro ciclo de simulação, quando a energia residual de todos os nós é muito similar. Para verificar o comportamento dos perfis em diferentes cenários, executou-se um novo conjunto de simulações onde os valores iniciais de energia dos nós foram escolhidos aleatoriamente entre 0 e 20 J (simulando-se, assim, a execução do algoritmo após a rede estar em execução há algum tempo, com consumos heterogêneos entre os nós). Como pode ser visto na Figura 43, com esses parâmetros houve uma diferença mais significativa na quantidade final de energia na rede para cada perfil. Para o perfil baseado em tempo de vida houve uma economia de cerca de 50% no consumo de energia.

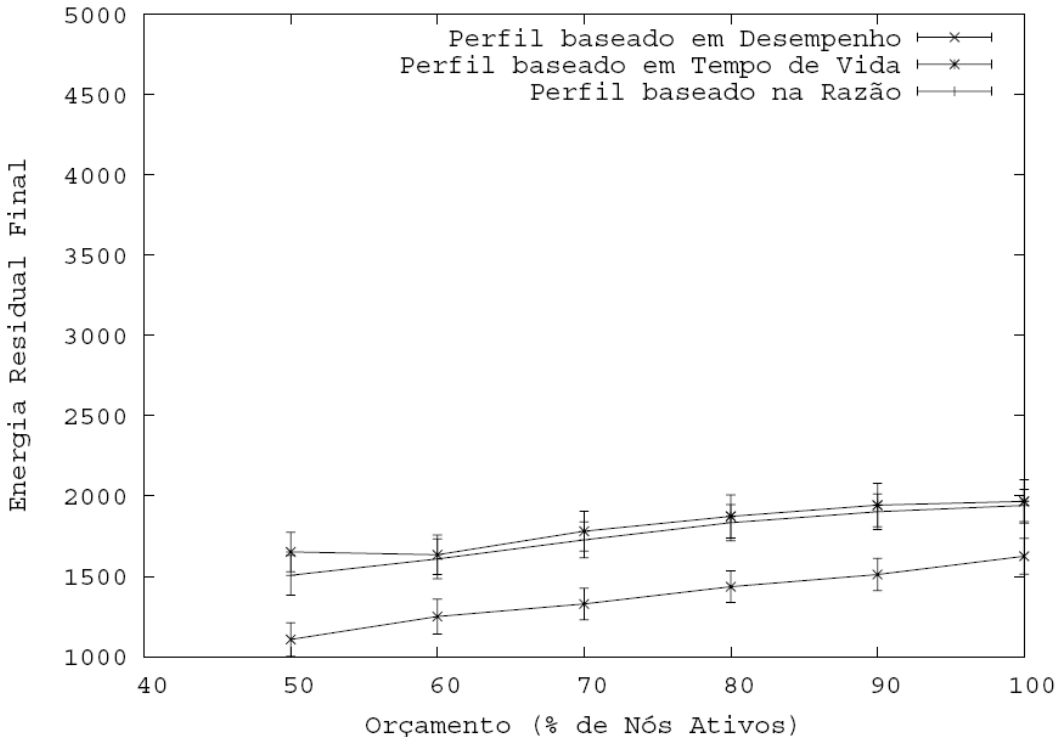


Figura 43: Energia residual da rede no 10º *round*, para os diferentes orçamentos e perfis de QoS, com energia inicial dos nós de 0 a 20J

Por outro lado, os valores de relevância variaram bastante entre diferentes nós. Os resultados mostrados na Figura 44 corroboram tal fato. Quando a aplicação decide dar

prioridade à relevância, no caso do modelo de aplicação adotado, à acurácia dos dados (perfil baseado em desempenho), o valor final do erro foi de fato significativamente menor do que quando o tempo de vida é priorizado.

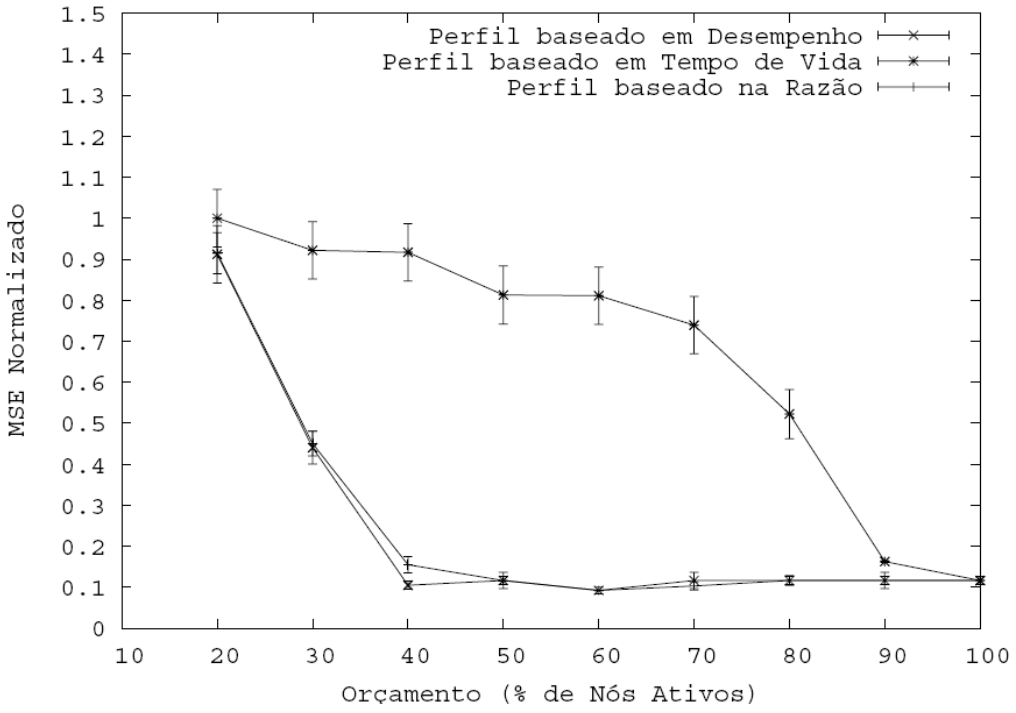


Figura 44: MSE Normalizado no 10º *round*, para os diferentes orçamentos, considerando os três perfis de QoS

7.4.6 Comparação com a Seleção Aleatória de Nós

Para fins de comparação com a estratégia de seleção de nós proposta nesta tese, um esquema de seleção aleatória dos nós foi simulado. O intuito da comparação é avaliar os benefícios da estratégia de maximizar a energia residual dos nós e sua relevância para a aplicação, adotada no procedimento de seleção proposto, com um esquema simplista, que não leva em conta tais fatores. Nesse esquema, para cada orçamento é alocado de forma aleatória o número de nós correspondente àquela porcentagem, em relação ao número total de nós na rede. Leva-se em conta apenas a área alvo solicitada pela aplicação, garantindo que a porcentagem correspondente de nós fontes é selecionada. Porém, para a escolha dos demais nós não se consideram os critérios de energia residual, nem outros relacionados à sua relevância para a aplicação, como o nível de ruído nas medições. Além disso, não são executados procedimentos para garantir a cobertura de sensoriamento nem a conectividade da rede.

As Figuras 45 e 46 mostram a energia residual e o MSE normalizado, ao final de 10 ciclos, para o esquema de seleção usado no middleware e a para abordagem de seleção aleatória. Observa-se que a energia residual final da rede foi maior, em todos os orçamentos, quando se adotou o esquema proposto na tese. Já o MSE foi sempre menor com a abordagem proposta do que com a seleção aleatória dos nós. Nas simulações com o esquema aleatório, observou-se que, por não haver garantia alguma de conectividade da rede, em muitos casos, principalmente com orçamentos menores, as fontes não conseguiam entregar seus dados ao sorvedouro por não conseguirem estabelecer uma rota até ele. A seleção sem levar em conta a energia residual dos nós também mostrou-se ineficiente, já que houve um menor balanceamento do consumo de energia na rede, diminuindo seu tempo de vida.

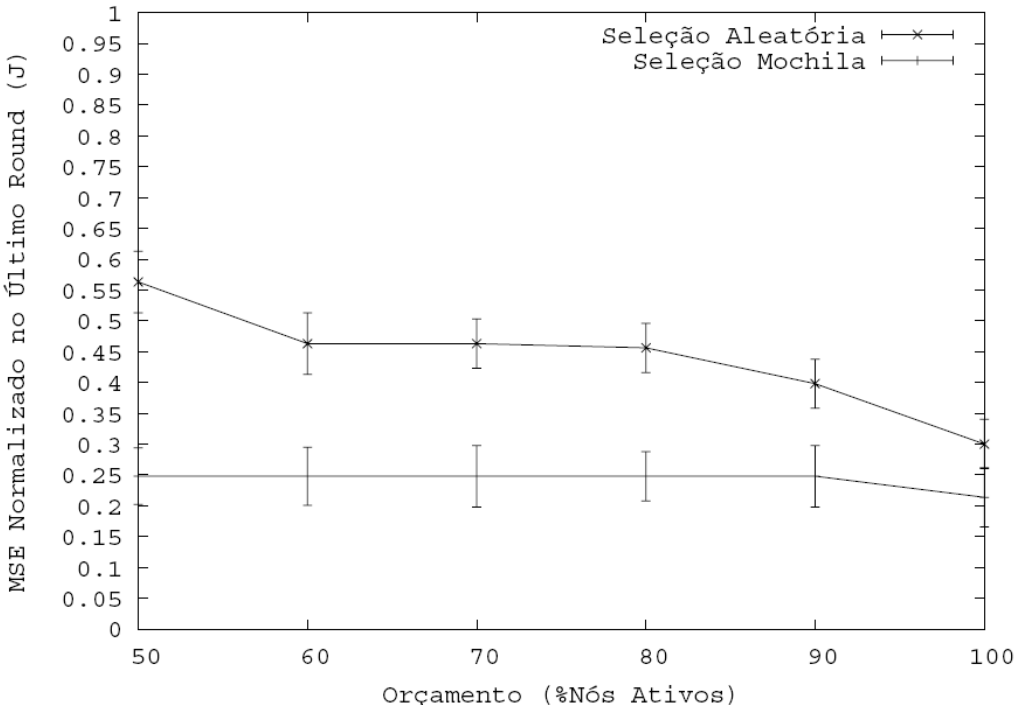


Figura 45: MSE Normalizado no 10º *round*, para o esquema de alocação proposto e um esquema de seleção aleatório de nós

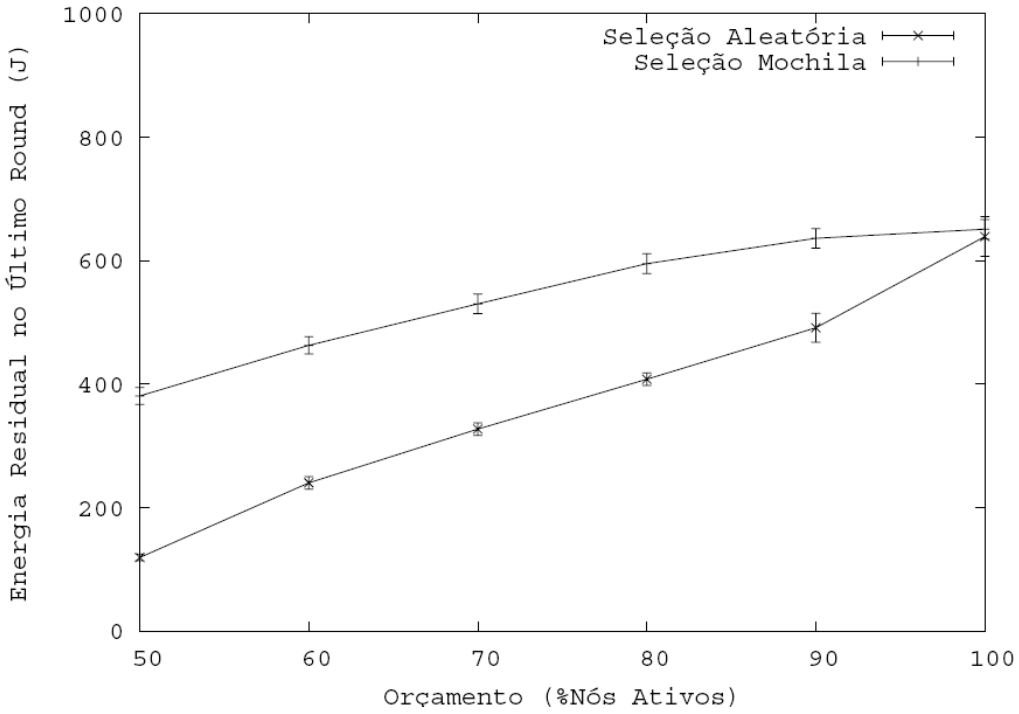


Figura 46: Energia residual da rede no 10º *round*, para o esquema de alocação proposto e um esquema de seleção aleatório de nós

8 Desenvolvimento do Protótipo do Sistema

Como prova de conceito para a proposta de sistema de middleware apresentada nesta tese, foi construído um protótipo com algumas de suas principais funcionalidades. Os objetivos da construção do protótipo são validar as interações entre aplicações e a RSSF e fornecer uma base para análises dos requisitos do sistema, em termos de memória e processamento, a fim de avaliar a viabilidade de sua instalação em dispositivos de hardware atualmente existentes para sensores sem fio.

Para a construção do protótipo adotou-se o paradigma de orientação a objetos e um processo de desenvolvimento iterativo e incremental, onde a cada etapa novas funcionalidades foram adicionadas ao sistema em construção. O protótipo foi implementado usando a linguagem de programação Java. Como as características de hardware e, conseqüentemente, os componentes de software a serem instalados, diferem enormemente entre nós sorvedouros e sensores, diferentes plataformas de desenvolvimento foram utilizadas para implementar cada tipo de nó.

Cada etapa do desenvolvimento do protótipo consistiu em construir os modelos lógico e físico de um grupo de funcionalidades e a seguir implementar o que foi modelado. Os modelos lógico e físico do sistema foram construídos utilizando a linguagem de modelagem unificada UML, versão 1.5 [134].

A arquitetura do sistema de middleware proposto consiste em vários módulos, onde cada módulo implementa um conjunto de funcionalidades do middleware e corresponde a um sub-sistema, ou componente. Os sub-sistemas, por sua vez, são compostos por conjuntos de classes a serem especificadas e implementadas a cada iteração do desenvolvimento do protótipo.

Além das etapas de desenvolvimento seguidas na construção do protótipo, sua implementação foi adicionalmente separada em duas fases. O objetivo da primeira fase consistiu em modelar e implementar o módulo de comunicação nos nós sorvedouros. A implementação desse módulo permitiu validar a interação de alto nível entre aplicações e

a rede de sensores e testar as invocações das operações fornecidas pelo Serviço Web da rede.

O objetivo da segunda fase da implementação foi analisar a carga computacional do módulo de comunicação nos nós sensores. Nessa fase, foi avaliado o uso de mensagens no formato XML e no formato XML binário, WBXML, a fim de comparar o seu desempenho, em termos de tráfego na rede e gasto de memória dos dispositivos. Foi também implementado um serviço simples de agregação para avaliar o gasto de memória relativo ao tratamento das mensagens XML, seu encaminhamento através dos diferentes manipuladores e seu despacho para o módulo que implementa o serviço solicitado.

As seções a seguir descrevem os ambientes de desenvolvimento utilizados na construção do protótipo, as classes implementadas e as medições efetuadas.

8.1 PROTÓTIPO DO NÓ SORVEDOURO

O nó sorvedouro onde foi executado o protótipo consistiu em um Pentium 4, 1.8 GHZ, com 1.5 GBytes de memória RAM e HD de 40 GBytes. As classes representando as funcionalidades necessárias para o sorvedouro, bem como o Serviço Web da rede foram implementados utilizando o ambiente Apache Axis para desenvolvimento de Serviços Web [4] e a plataforma J2SE (Java Standard Edition) versão 1.4.2_01. O Axis fornece uma implementação da máquina SOAP e todas as classes necessárias para a análise e composição de mensagens SOAP, bem como a infra-estrutura necessária para aplicações clientes invocarem operações dos Serviços Web. Além disso, ele inclui suporte completo para a linguagem WSDL. Axis fornece suporte à linguagem WSDL de três maneiras:

- quando um serviço é instalado no Axis, os usuários podem acessar a URL do serviço com um navegador Web padrão e, anexando "?WSDL" ao final da URL, eles obtêm um documento WSDL gerado automaticamente o qual descreve o respectivo serviço;

- uma ferramenta chamada "WSDL2Java" é fornecida, a qual constrói classes Java a partir da descrição de serviços em documentos WSDL;

- uma ferramenta chamada "Java2WSDL" é fornecida, a qual constrói documentos WSDL a partir de classes Java.

As seções a seguir descrevem as principais classes de domínio especificadas para o protótipo e a implementação das classes relacionadas ao Serviço Web da rede de sensores através do uso da plataforma Axis.

8.1.1 Descrição das Principais Classes de Domínio

As principais classes de domínio criadas para o nó sorvedouro podem ser vistas nas Figuras 47 a 49. Para melhor legibilidade, foram construídos três diagramas, cada um representando um sub-conjunto de classes com maior grau de relacionamento entre si. Nas figuras, os retângulos compartimentados correspondem às classes. O primeiro compartimento exibe o nome da classe e os outros dois exibem os seus atributos e métodos. As linhas entre os retângulos representam relações entre as instâncias de objetos pertencentes a classes. A classe denominada `WSN_Service` corresponde à especificação lógica do Serviço Web da rede. As classes de implementação correspondentes foram geradas de acordo com os pacotes fornecidos pelo Axis, e serão descritas na Seção 8.1.2.

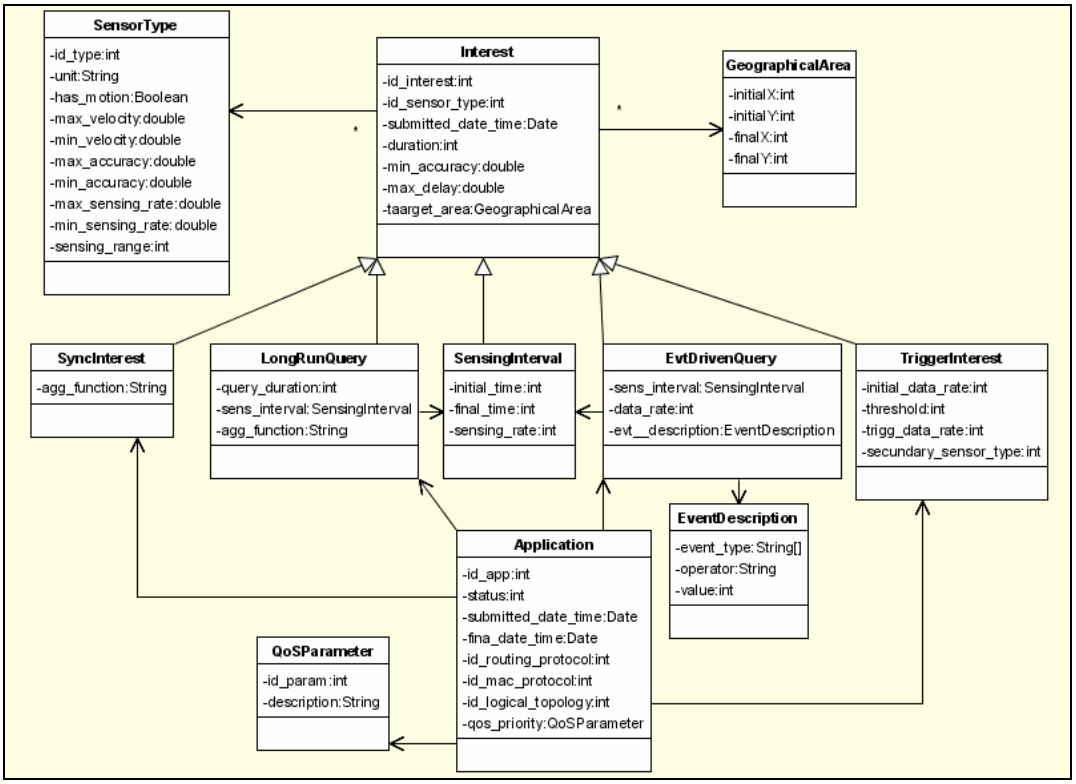


Figura 47: Diagramas com as principais classes de Domínio relacionadas aos interesses da aplicação

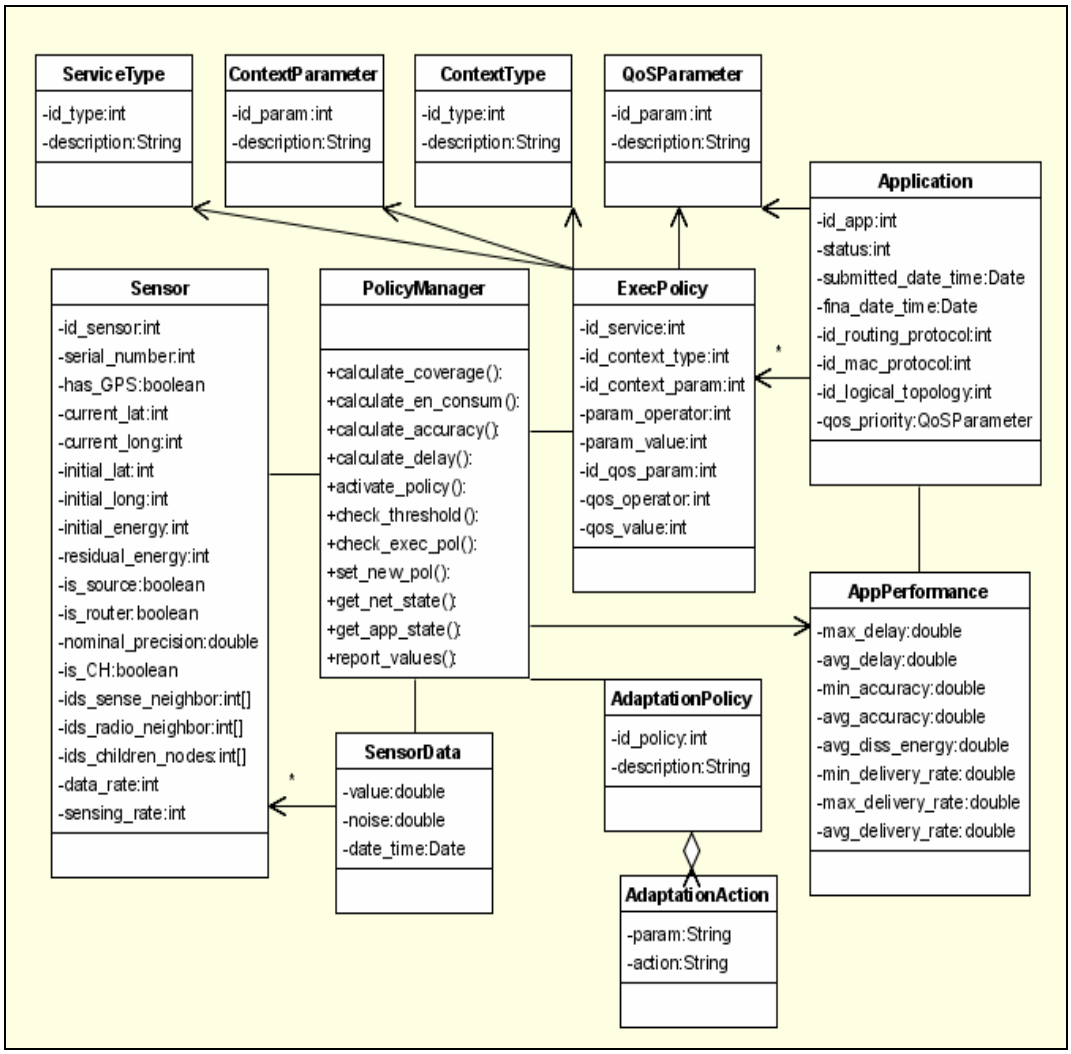


Figura 48: Principais classes de domínio relacionadas ao gerenciamento da execução das tarefas

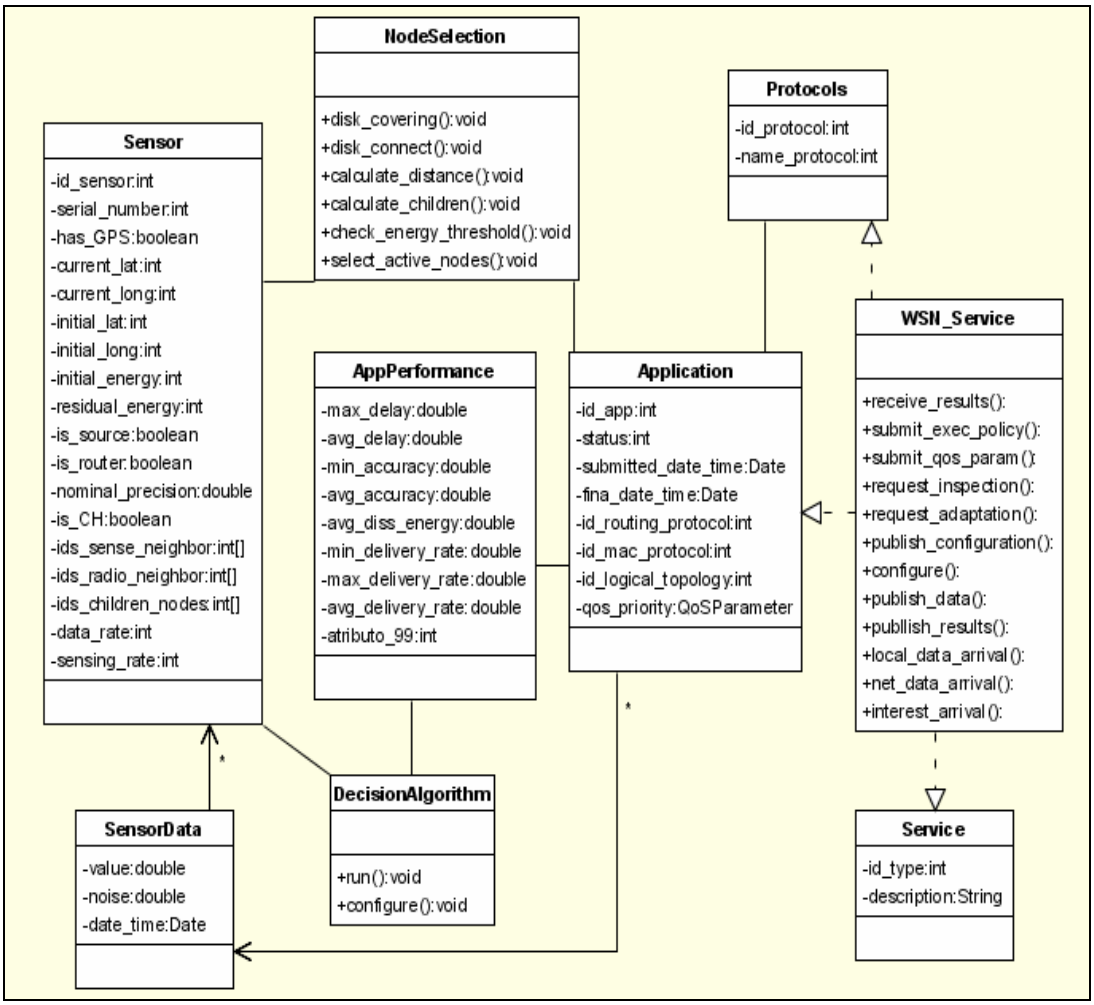


Figura 49: Principais classes de Domínio relacionadas aos módulos de decisão e de seleção

8.1.2 Descrição do Ambiente de Desenvolvimento e Execução

O Serviço Web representando a rede de sensores foi instalado no servidor de aplicações TomCat [5] e o documento descrevendo os serviços disponíveis na rede foi escrito na linguagem WSDL. As seguintes operações foram oferecidas pelo Serviço Web implementado:

- **`Subscribe_Interest`** – usada pela aplicação para submeter uma consulta representando um interesse assíncrono.

- **`Publish_Content`** – usada por um nó sensor para informar sua descrição de serviços.

- **`Publish_Data`** - usada por um nó sensor para comunicar seus dados.

As operações são invocadas por mensagens SOAP de mesmo nome. Como visto no Capítulo 5, para o acesso à rede pelas aplicações, são utilizados *proxies*, que convertem as chamadas de métodos na linguagem da aplicação em chamadas de operações SOAP. No protótipo desenvolvido, foi implementada uma aplicação cliente na linguagem Java. Portanto, para a geração do *proxy* Java para o acesso à rede pela aplicação foi usada a ferramenta WSDL2Java.

A ferramenta WSDL2Java consiste em uma classe Java a qual, como visto anteriormente, é disponibilizada junto com o pacote da plataforma Axis. Ela recebe como entrada um documento WSDL representando um Serviço Web e gera chamadas de métodos em Java os quais correspondem à invocação das operações disponibilizadas pelo serviço. Ela pode ser usada para gerar as classes necessárias tanto para o lado cliente quanto servidor. No lado cliente, para cada entrada na seção "*type*" do documento WSDL serão gerados uma classe Java e um objeto *holder* se esse tipo for usado como um parâmetro de entrada/saída. Para cada elemento *portType* do documento será gerada uma interface Java. Para cada elemento *binding*, uma classe *Stub* e para cada elemento *service* serão gerados uma interface de serviço e uma implementação do serviço (o localizador ou *locator*). Objetos *holder* podem ser usados como parâmetros de entrada e saída ou de saída. Java não implementa o conceito desse tipo de parâmetro. Para obter esse comportamento, a API Java JAX-RPC [69] especifica o uso de classes *holder*. Uma classe *holder* é simplesmente uma classe que contém uma instância de seu próprio tipo.

A Interface de Definição de Serviço (*Service Definition Interface* – SDI) é a interface derivada de um *portType* de um documento WSDL. Ela é a interface utilizada para acessar as operações do serviço. Uma classe *Stub* implementa a SDI. O nome da classe é composto pelo nome do *binding* (no WSDL) acrescido do sufixo "*Stub*". Ela contém o código que converte as chamadas de métodos em mensagens SOAP, usando os objetos *Service* e *Call* fornecidos pela plataforma Axis. Ela funciona como um *proxy* para o serviço remoto, permitindo que o mesmo seja chamado exatamente como se fosse um objeto local. Em outras palavras, não é necessário lidar com *endpoints* URL, *namespaces*, ou vetores de parâmetros envolvidos na invocação dinâmica via os objetos *Service* e *Call*. A classe *Stub* esconde todos esses detalhes do usuário.

A Figura 50 apresenta a SDI criada pelo protótipo, com a definição dos principais métodos a serem implementados.

```
public interface SinkPortType extends java.rmi.Remote {

public java.lang.String[] querySensors(String param) throws
java.rmi.RemoteException;
public java.lang.String publishContent(String IDSOurce, String typeSOurce, long
longSource, long latSource, long energySource, long confidenceMaxSource, long
confidenceMinSource, long dataRateMaxSource, long dataRateMinSource) throws
java.rmi.RemoteException;
public void publishData(String nodeID, long intensity, long timeStamp, String
type, String dataValue, long latitude, long longitude) throws
java.rmi.RemoteException;
public void subscribeInterest(String appId, String type, long dataRate, long
duration, long latPointA, long longPointA, long latPointB, long longPointB,
float threshold) throws java.rmi.RemoteException;
public empty.InterestDescType receiveInterest(String nodeId) throws
java.rmi.RemoteException;
public empty.DataDescType[] getData(String appId) throws
java.rmi.RemoteException;
        ....
}
```

Figura 50: SDI do protótipo

Normalmente, um programa cliente não instancia uma classe *Stub* diretamente. Em vez disso, ele irá instanciar uma classe localizadora de serviço (*service locator*) e chamar um método *get* dessa classe, o qual retorna um objeto *Stub*. A classe localizadora é derivada da cláusula *service* no documento WSDL. WSDL2Java gera dois objetos a partir da uma cláusula *service*. Por exemplo, dado o trecho abaixo do WSDL da rede:

```
<service name="SinkService">
    <port name="SinkSOAPPort" binding="y:SinkSOAPBiding">
      <soap:address location="http://bocaiuva/SinkSOAPBinding/"/>
    </port>
  </service>
```

WSDL2Java irá gerar a interface de serviço representada na Figura 51.

```
public interface SinkService extends javax.xml.rpc.Service {

public String getSinkSOAPPortAddress();

public com.example.namespace.SinkPortType getSinkSOAPPort() throws
javax.xml.rpc.ServiceException;
public    com.example.namespace.SinkPortType    getSinkSOAPPort    (java.net.URL
portAddress) throws javax.xml.rpc.ServiceException;
```

Figura 51: Interface de serviço do protótipo

WSDL2Java irá também gerar a classe localizadora que implementa essa interface (Figura 52).

```
public class SinkServiceLocator extends org.apache.axis.client.Service
implements com.example.namespace.SinkService {  ...
}
```

Figura 52: Classe localizadora do protótipo, gerada automaticamente

A interface de serviço `SinkService` define um método *get* para cada porta listada no elemento *service* do WSDL. A classe `SinkServiceLocator` é a implementação dessa

interface. Ela implementa os métodos *get* e serve como um localizador para obter instâncias Stub. A classe *Service* do pacote Axis por *default* criará um *Stub* que aponta para a *endpoint* URL descrita no documento WSDL, mas pode-se também especificar uma URL diferente ao se solicitar um *portType*.

Exatamente como uma classe *Stub* representa o lado cliente de um Serviço Web implementado em Java, um *skeleton* é uma classe Java para o lado servidor. Para criar classes *skeleton*, deve-se especificar a opção "--server-side --skeletonDeploy true" para a ferramenta WSDL2Java. A ferramenta WSDL2Java irá gerar todas as classes geradas anteriormente (para o lado cliente), e gera adicionalmente os arquivos listados na Tabela 2.

Tabela 2: Classes Java geradas para o lado cliente pela ferramenta WSDL2Java.

Cláusula WSDL	**Classes Java geradas**
Para cada elemento *binding*	Uma classe *skeleton*
	Um *template* para implementação da classe
Para todos os serviços	Um arquivo deploy.wsdd
	Um arquivo undeploy.wsdd

Se a opção "--skeletonDeploy true" não for especificada, não será gerada uma classe *skeleton*. Em vez disso, o arquivo de configuração de instalação (gerado automaticamente, e chamado deploy.wsdd) irá indicar que a classe de implementação será diretamente instalada. O arquivo de configuração irá conter metadados extras descrevendo as operações e parâmetros da classe de implementação. Nesse caso, os arquivos gerados no lado servidor são listados na Tabela 3.

Tabela 3: Classes adicionais geradas para o lado servidor pela WSDL2Java.

Cláusula WSDL	**Classes Java geradas**
Para cada elemento *binding*	Um *template* para implementação da classe
Para todos os serviços	Um arquivo deploy.wsdd com metadados descrevendo as operações
	Um arquivo undeploy.wsdd

A classe *skeleton* é a classe que se situa entre o engenho Axis e a implementação real do Serviço Web. Seu nome é o nome do *binding* acrescido do sufixo "Skeleton".

Na implementação do protótipo no sorvedouro, optou-se por não gerar uma classe *skeleton*. A ferramenta WSDL2Java gerou, então, a partir do *binding*, a classe *template* `SinkSOAPBidingImpl`. A classe *template* gerada não contém código algum, seu corpo é vazio. É função do desenvolvedor do serviço preencher os seus métodos de acordo com as especificações do serviço. A Figura 53 apresenta a classe gerada preenchida com os principais métodos que foram implementados para o protótipo.

```
public class SinkSOAPBidingImpl implements com.example.namespace.SinkPortType{

public SinkSOAPBidingImpl () {}
public empty.SensorDescType publishContent(empty.SensorDescType sourceNode)
throws java.rmi.RemoteException {
        .......
    }
public void publishData(empty.DataDescType data) throws java.rmi.RemoteException
{....  }
public void subscribeInterest(java.math.BigInteger appId, empty.InterestDescType
interest) throws java.rmi.RemoteException
  {.....}
public empty.DataDescType[] getData(java.math.BigInteger appId) throws
java.rmi.RemoteException
   {.....}

....
}//fim da classe
```

Figura 53: Classe de implementação do Serviço Web para o protótipo

A ferramenta WSDL2Java cria ainda dois arquivos: "deploy.wsdd" e "undeploy.wsdd", para cada serviço, os quais podem ser usados para instalar o serviço no Axis uma vez que se tenham preenchido os métodos da classe de implementação, compilado o código e tornado as classes disponíveis para o engenho Axis.

Para validar a comunicação entre aplicações clientes e a RSSF, uma aplicação baseada em eventos foi implementada como um aplicativo Java que emite consultas para a rede e recebe os resultados. A aplicação implementada, tendo previamente obtido a URL do nós sorvedouro através de UDDI, obtém uma referência para o Serviço Web da rede e invoca a operação de submissão de um interesse assíncrono, representando uma consulta de longa duração. A aplicação foi implementada como uma thread que recebe continuamente os dados gerados pela rede, de acordo com os parâmetros definidos em seu interesse. A Figura 54 apresenta os trechos de código da aplicação que fazem a comunicação com o Serviço Web da rede, invocando suas operações através da chamada de métodos.

```
public class Aplicacao extends Thread
{......
  try {
  SinkService ss = new SinkServiceLocator();
  URL sink_serviceURL = new URL(ss.getSinkSOAPPortAddress());
```

```
  SinkPortType sink_port = null;
        if (sink_serviceURL == null) {
            sink_port = ss.getSinkSOAPPort();
        }
     else {
            sink_port = ss.getSinkSOAPPort(sink_serviceURL);
        }
.....
sink_port.subscribeInterest(new java.math.BigInteger (app), desc_interesse);
  while (data == null)
  { data = sink_port.getData(new java.math.BigInteger (app));
    wait ();}
}//fim do try
catch (Exception e) {}
}
}
```

Figura 54: Trechos da classe da Aplicação implementada

8.2 PROTÓTIPO DOS NÓS SENSORES

A configuração de hardware para os sensores emulada no presente trabalho é dos sensores Sensoria WINS NG 2.0 [119], dotados de processadores SH-4 com 167 MHz e de 32 MBytes de memória RAM.

O protótipo nos nós sensores foi emulado utilizando a plataforma J2ME (Java Micro-edition) [68], com a configuração CLDC (*Connected Limited Device Configuration*) e o perfil MIDP (*Mobile Information Device Profile*) versão 2.0. A configuração CLDC é voltada para dispositivos de pequeno porte, com limitados recursos computacionais e de energia, conectados a algum tipo de rede, em geral sem fio. Ela é a menor das configurações disponíveis na plataforma J2ME, contemplando dispositivos com no mínimo 160kBytes de memória RAM e processadores de 16 bits. O perfil MIDP foi especificamente projetado para telefones móveis e celulares e oferece as funcionalidades básicas requeridas por aplicações móveis, como conexões com rede, interface de usuário, armazenamento local e persistência de dados, entre outras.

Para a construção do protótipo dos nós sensores utilizou-se o *Wireless Toolkit* [68], o qual consiste em um conjunto de ferramentas que fornecem ambientes de emulação para dispositivos compatíveis com o perfil MIDP e a configuração CLDC. Um aplicativo construído no *Wireless Toolkit* consiste em uma classe Java derivada da classe MIDLet. Para o protótipo foi implementada uma *Midlet* que instancia a classe representando um sensor e uma *Midlet* que instancia a classe representando a aplicação (Figura 55).

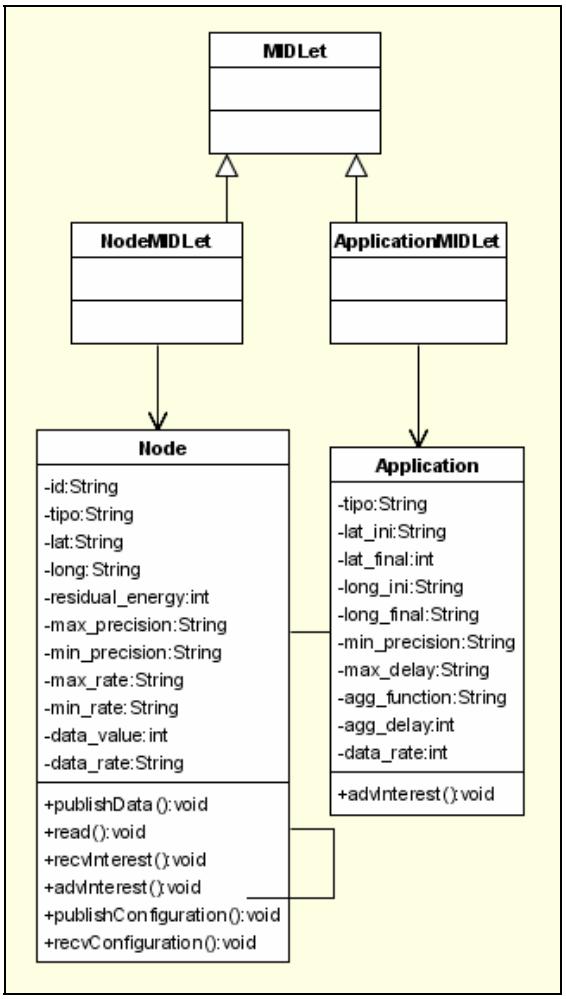


Figura 55: Midlets criadas para representar a aplicação e os nós sensores

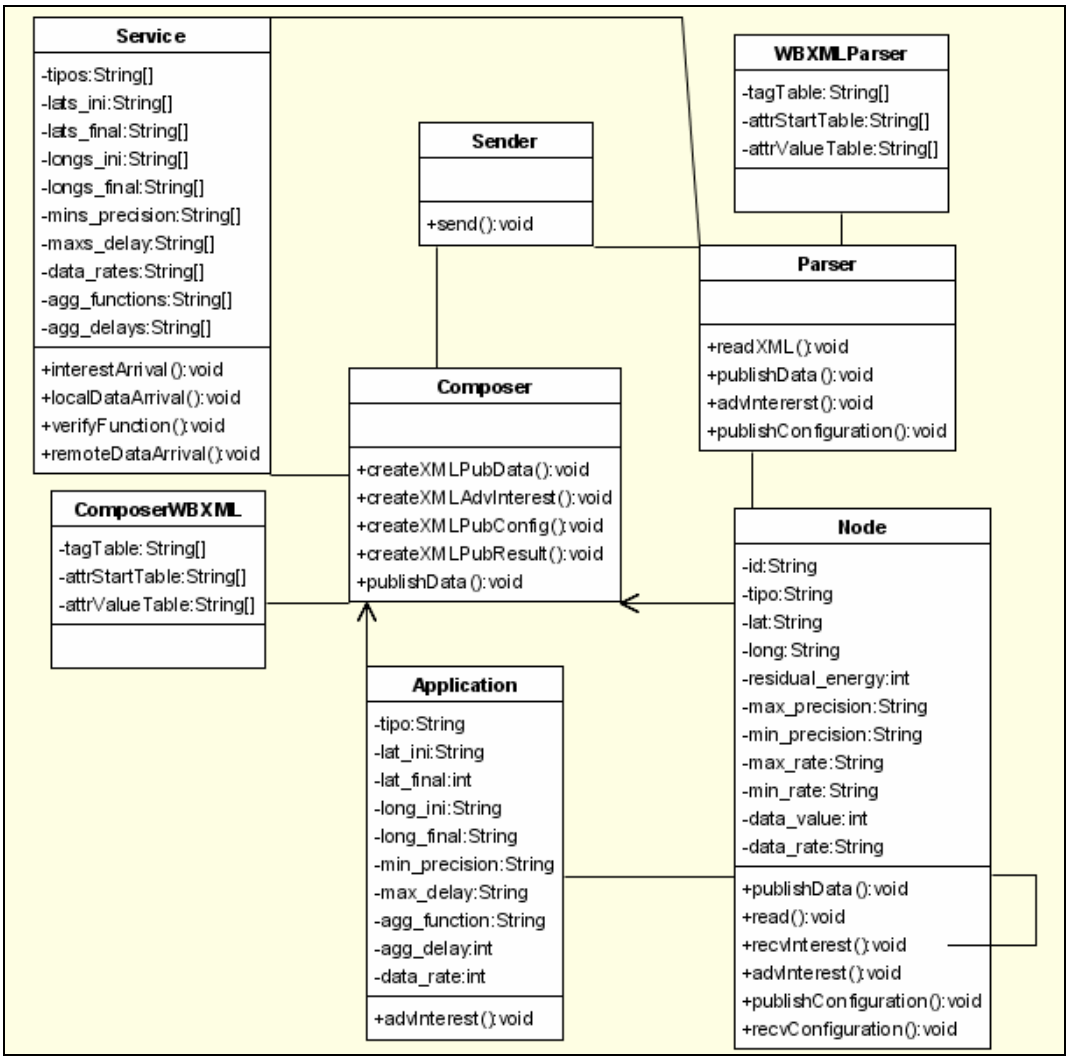


Figura 56: Principais classes do Protótipo do Nó Sensor

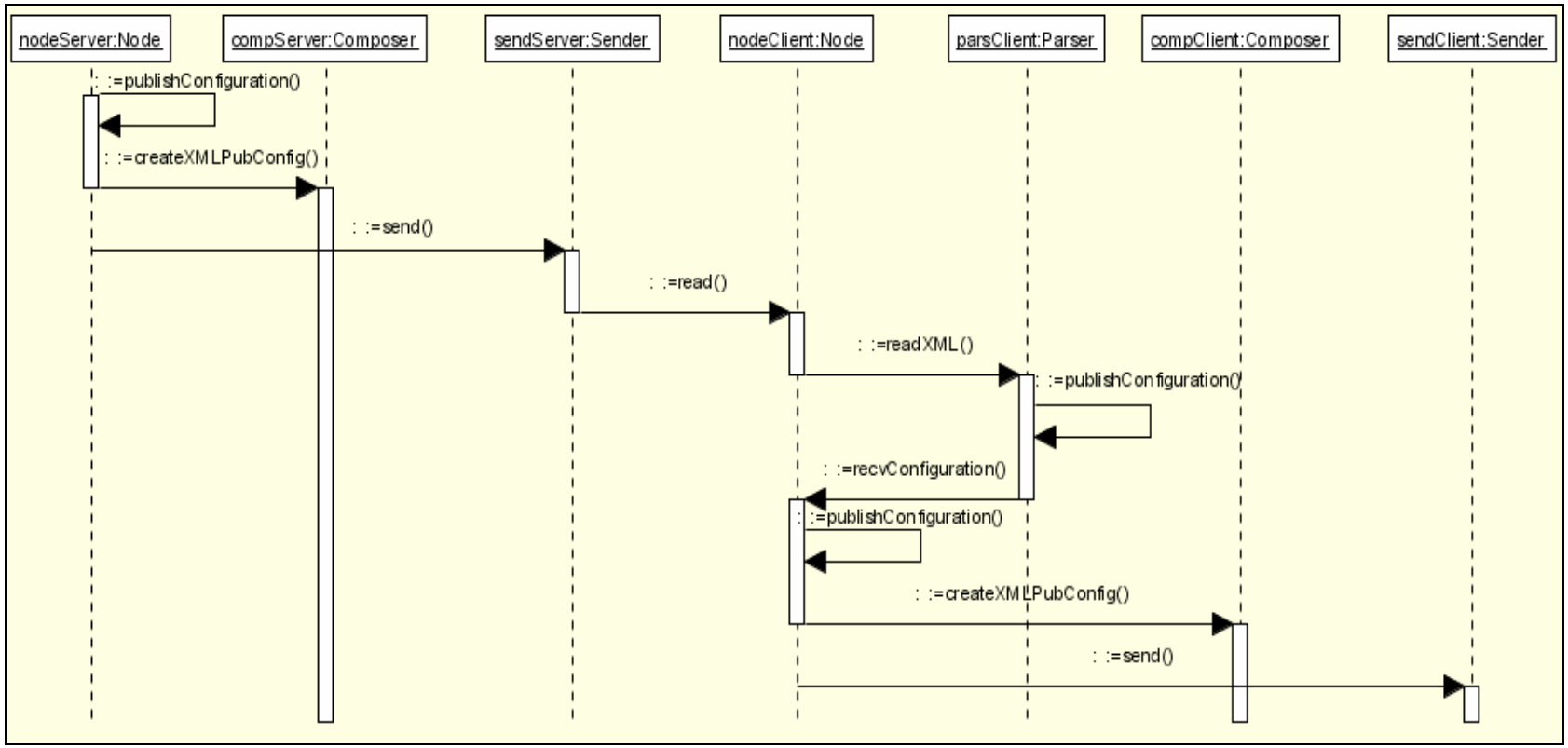


Figura 57: Diagrama de Seqüência da fase de Descoberta Interna de Serviços da Rede

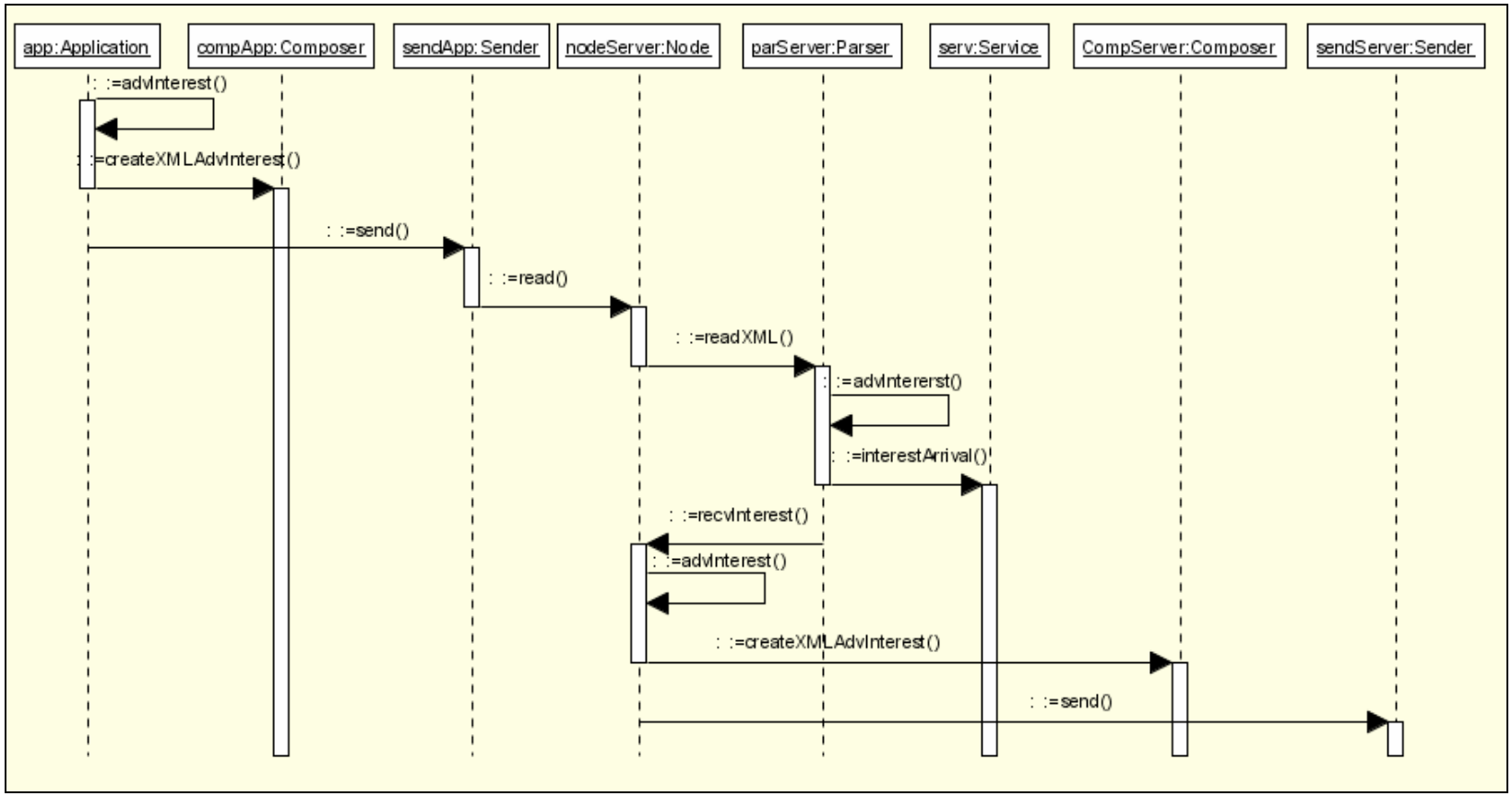


Figura 58: Diagrama de Seqüência da fase de Submissão de Interesses

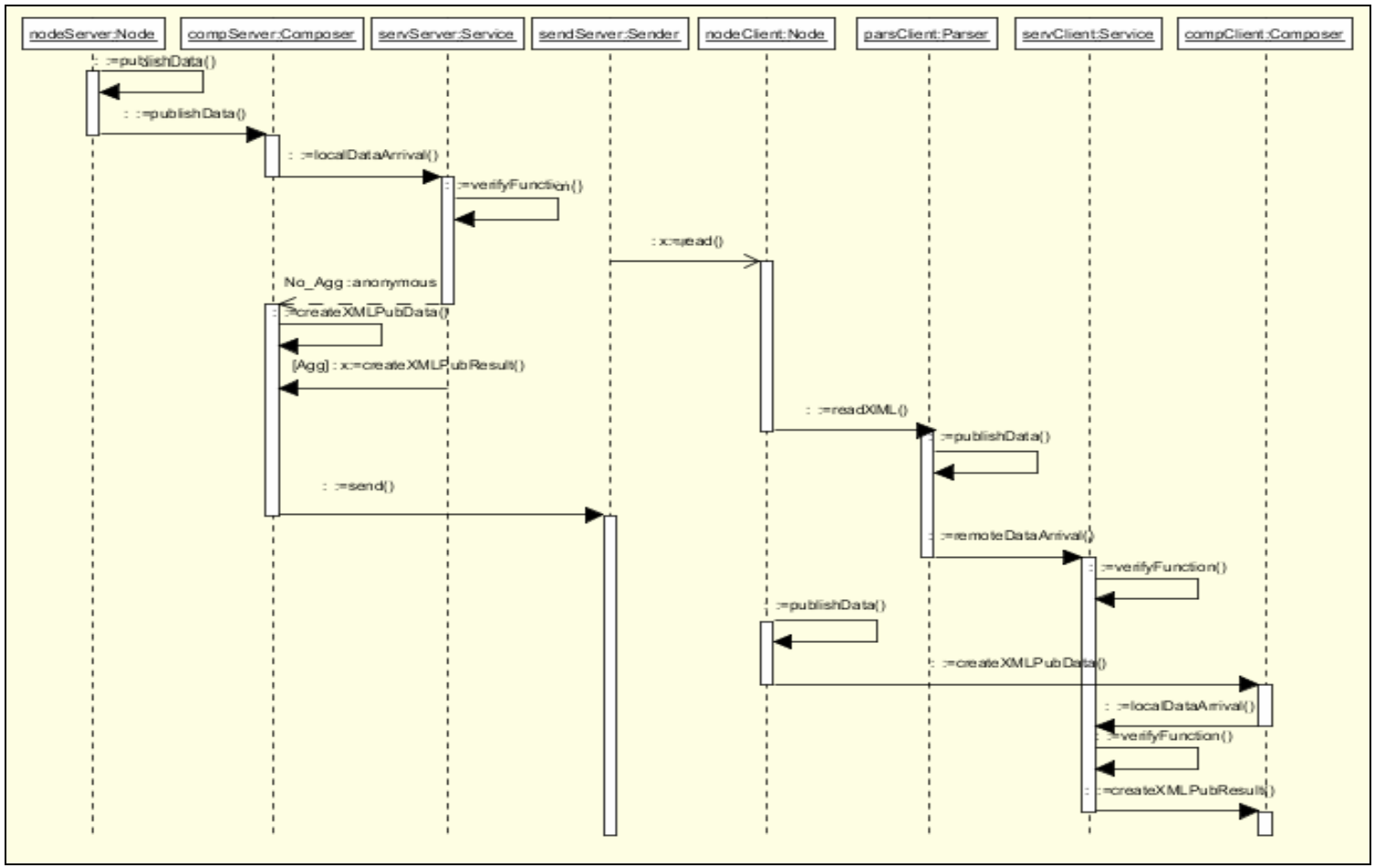


Figura 59: Diagrama de Seqüência da fase de Publicação de Dados

Para a implementação do módulo de comunicação nos sensores foi usado o pacote kXML [76]. O kXML fornece um analisador e um escritor XML baseados no modo *pull* [76], que executam em todas as plataformas Java, incluindo a J2ME. Devido a seus pequenos requisitos de memória e processamento, kXML é especialmente adequado para *applets* e aplicativos Java rodando em dispositivos móveis com recursos restritos, como Palms e celulares. A Figura 56 apresenta o diagrama de classes do protótipo nos nós sensores. As Figuras 57 a 59 representam os diagramas de seqüência para cada etapa de funcionamento da rede implementada. O analisador e o escritor XML estão representados na Figura 56 como as classes `Parser` e `Composer`, respectivamente. A classe `Sender` é utilizada para o envio de mensagens através da rede. A comunicação na rede é baseada em soquetes UDP, implementados pela classe *DatagramConnection*, disponível na biblioteca da plataforma J2ME. A classe `Service` representa um módulo de serviço do middleware. Foi implementado o serviço de agregação, que oferece métodos para realizar funções MAX, MIN e AVERAGE. O tamanho do arquivo ".jar" (formato de instalação Java) composto por todas as classes necessárias para um sensor é de 90kBytes, incluindo todas as bibliotecas necessárias. Esse tamanho é perfeitamente compatível com os recursos de memória disponíveis nos dispositivos sensores considerados como alvo deste trabalho.

Embora a linguagem XML seja uma opção útil para a troca de dados entre aplicações, devido a seu alto grau de portabilidade e extensibilidade, ela possui a desvantagem de ser muito verbosa. Um documento XML é um documento baseado em texto, possível de ser lido por pessoas. Portanto, foi concebido desde seu projeto para ser verboso. Devido às baixas capacidades de transmissão das RSSFs atuais, enviar extensos documentos XML entre os sensores pode não ser uma opção eficiente. Além disso, analisar e compor documentos grandes pode consumir muito do limitado poder de processamento dos dispositivos da rede. Para lidar com essa limitação, existem codificações binárias da linguagem XML. Um dos formatos binários mais utilizados é o WBXML [138], desenvolvido para o uso conjunto com o protocolo WAP [150] em telefones celulares. O formato WBXML foi projetado para reduzir o tamanho de transmissão de documentos XML, permitindo o uso mais eficiente de dados XML em canais de comunicação de banda estreita. Tal formato permite a transmissão compacta sem perda de funcionalidade ou de informação semântica. O WBXML codifica a

estrutura e o conteúdo das entidades do documento XML. Ele trabalha substituindo nomes e valores de *tags* e atributos por "*tokens*". Meta-informações, incluindo Esquema XML e seções condicionais, são removidas quando o documento é convertido para o formato binário.

O pacote kxml possui suporte para analisar e compor mensagens em WBXML. Utilizar WBXML com o analisador disponibilizado pelo kxml é apenas uma questão de substituir a classe `Parser` pela `WbxmlParser` e a classe `Composer` pela `WbxmlComposer`.

Obviamente, não se pode usar o formato binário sem que todos os nós comunicantes forneçam suporte para o mesmo, incluindo o sorvedouro, já que este deve ser capaz de traduzir requisições recebidas das aplicações clientes para o formato binário usado pelos sensores e vice-versa. Então, nas transmissões dentro da rede os nós podem usar apenas o formato XML binário. Porém, para fazer a comunicação com módulos do middleware ou com a aplicação, é preciso fazer a conversão entre o formato binário e XML puro. Originalmente, a implementação do formato WBXML fornece métodos para fazer a conversão para documentos XML descritos através de DTDs (Data *Type Definition*) [135], porém não há suporte para Esquemas XML. Entretanto, os autores em [24] adaptaram o formato WBXML para operar com Esquemas XML, e obtiveram ótimos resultados de desempenho ao comparar o uso de WBXML com outras técnicas de compressão de mensagens XML.

No protótipo desenvolvido foram implementados os dois formatos (XML e WBXML), a fim de comparar seu desempenho em termos de número de bytes transmitidos na rede e consumo de memória dos dispositivos. Os resultados obtidos nas medições são descritos a seguir.

8.2.1.1 Medidas Realizadas com o Protótipo

O J2ME *Wireless Toolkit* permite ao desenvolvedor examinar vários aspectos das aplicações criadas, com o intuito de otimizar as mesmas, aumentando sempre que possível sua eficiência e velocidade de execução. Para isso, são incluídas as seguintes ferramentas, que foram utilizadas para realizar as medições com o protótipo desenvolvido:

- Perfilador (*Profiler*) - Permite examinar o tempo de execução e a freqüência de uso dos métodos da aplicação.

- Monitor de Memória - Permite examinar a utilização da memória da aplicação.

- Monitor de Rede - Permite monitorar as transmissões entre cada dispositivo e a rede. Podem-se monitorar transmissões de vários tipos de protocolos de redes, tanto orientado a pacotes quanto a fluxos de dados. Há suporte para TCP, UDP, HTTP, HTTPS, além de outros protocolos.

Na avaliação de desempenho dos diferentes formatos de mensagens usados, os métodos foram agrupados segundo a etapa de funcionamento da rede a que se referem. Nas tabelas, `PubContent` refere-se à etapa de descoberta de serviços interna, na qual sensores trocam mensagens descrevendo seus serviços. `AdvInterest` refere-se à etapa de submissão de interesses pela aplicação. `PubData` refere-se à etapa de envio de dados pelos sensores. As tabelas abaixo mostram os resultados obtidos com os monitores de memória e de rede. Os resultados representam as medições de um Nó Sensor, ou seja, seu consumo individual de memória e o tráfego gerado pelo mesmo na rede, ao realizar cada uma das operações listadas.

Tabela 4: Medições Realizadas usado o formato XML original.

Operação	Memória (bytes)	Tráfego na Rede (bytes)
PubContent (envio)	10748	139
PubContent (recepção e repasse)	10344	139
AdvInterest (recepção e repasse)	33216	178
PubData (envio)	13136	42
PubData (recepção e repasse)	6136	42

Tabela 5: Medições Realizadas usado o formato WBXML.

Operação	Memória (bytes)	Tráfego na Rede (bytes)
PubContent (envio)	8152	47
PubContent (recepção e repasse)	5964	47
AdvInterest (recepção e repasse)	21140	57
PubData (envio)	9396	24
PubData (recepção e repasse)	4016	24

Como observado, o formato WBXML diminuiu não só o consumo de memória nos dispositivos, como reduziu em aproximadamente 70% o tráfego na rede, quando comparado com o formato XML. Portanto, ele representa uma melhor escolha para a representação das mensagens no interior da rede de sensores. É importante acrescentar que os tamanhos das mensagens geradas no formato WBXML são inferiores até mesmo

aos tamanhos utilizados em protocolos de RSSFs que adotam formatos binários proprietários. Por exemplo, o protocolo LEACH [59,60] adota mensagens de dados de 250 e 500Bytes. O protocolo Difusão Direcionada [66] trabalha com mensagens de dados de 64Bytes e mensagens de interesses de 34Bytes.

9 Conclusões e Trabalhos Futuros

Esta tese propôs um sistema de middleware baseado em serviços, especificamente projetado para redes de sensores sem fio. As características e funcionalidades de tal middleware permitem-no atender ao amplo espectro de requisitos das aplicações de RSSFs, fornecendo QoS e ciência de contexto, bem como satisfazendo a necessidade de obter alta eficiência da rede em termos de consumo de energia.

Este capítulo apresenta as principais contribuições do sistema de middleware proposto, bem como possíveis direções futuras de pesquisa a serem seguidas. Na Seção 9.1 são descritas as inovações trazidas pela proposta e são ressaltados seus principais benefícios. Na Seção 9.2 são apresentados os trabalhos futuros.

9.1 PRINCIPAIS CONTRIBUIÇÕES

As principais contribuições do sistema proposto podem ser sintetizadas sob dois aspectos. Do ponto de vista de projeto, a proposta oferece um novo paradigma para a arquitetura de redes de sensores sem fio, baseada na área de Serviços Web, e trazendo todos os benefícios de flexibilidade e interoperabilidade inerentes a essa abordagem. Do ponto de vista de um sistema de middleware, a proposta procura incorporar todas as características e funcionalidades necessárias, tanto para as aplicações clientes, como capacidades de inspeção e ciência de contexto, como para os desenvolvedores dessas aplicações, livrando-os da tarefa de lidar com decisões de infra-estrutura e assim facilitando o seu trabalho. Tais contribuições são detalhadas a seguir.

1. **Proposta de um Modelo de Programação para RSSFs baseado em Serviços**. A adoção de uma abordagem baseada em serviços na especificação do middleware proposto fornece um modelo de programação que pode ser utilizado para o projeto e construção de RSSFs flexíveis. A principal vantagem na utilização desse modelo é que ele provê uma visão abstrata da RSSF como fornecedora de serviços, sob o ponto de vista das aplicações clientes. Além de propor a adoção desse modelo, foi proposta a adoção de padrões ubíquos de protocolo e linguagem para a construção de uma interface flexível e de alto nível para o acesso à RSSF. Portanto, a primeira

contribuição desta tese pode ser desmembrada em dois sub-ítens detalhados a seguir.

- **Fornecimento de uma visão abstrata da rede como fornecedora de serviços**. Essa visão é independente da infra-estrutura e dos protocolos subjacentes, e permite que diferentes aplicações utilizem a mesma rede, com diferentes configurações. O fato das tarefas a serem realizadas pelas RSSF serem bastante específicas das aplicações tem levado a sistemas projetados com arquiteturas estáticas e programadas para tarefas específicas. Com o projeto de RSSFs atuais, não é possível para um usuário utilizar os recursos do sistema de qualquer forma que não seja a pré-estabelecida pela aplicação específica para a qual a rede foi inicialmente projetada. Essa abordagem de projeto gera sistemas essencialmente fechados para usuários externos e para outros sistemas que desejam acessar a rede de uma forma diferente da que foi previamente especificada. Tal inflexibilidade de operação e interação consiste em um obstáculo quando se considera que sistemas de RSSFs em geral possuem longos ciclos de desenvolvimentos e freqüentemente atendem a diferentes usuários com necessidades variáveis. Com o modelo de programação proposto, a rede de sensores pode ser vista como uma entidade que fornece serviços para vários usuários com necessidades diferentes e dinâmicas.O modelo prevê a incorporação de novas funcionalidades sob a forma de novos serviços que podem ser implementados por terceiros e facilmente adicionados ao sistema.

- **Proposta de uso da linguagem XML e do protocolo SOAP**, ambos reconhecidos como padrões Internet, como mecanismos para representar toda a comunicação da aplicação. O acesso ao sistema através de uma interface padrão de alto nível, baseada na linguagem XML e no protocolo SOAP, permite uma interação do tipo aplicação-aplicação (*business-to-business*), a qual é muito mais flexível para as aplicações clientes do que o acesso através de interfaces gráficas pré-definidas. Tal interface também pode ser utilizada por desenvolvedores de serviços para incorporar seus serviços à infra-estrutura do middleware. O fato de não se basear em protocolos proprietários, e sim em padrões ubíqüos da Web, como SOAP e XML, fornece ao middleware

grandes capacidades de interoperabilidade e extensibilidade. Aplicações podem ser escritas em diferentes linguagens de programação e interagir com a rede através do suporte de ferramentas para a geração automática de mensagens SOAP a partir de documentos WSDL. Desenvolvedores de protocolos de comunicação para a rede ou de novos serviços para o middleware podem interagir com a rede através dos Esquemas XML e do documento WSDL fornecidos.

2. **Incorporação de Características e Funcionalidades Necessárias a RSSFs.** O middleware proposto busca satisfazer os requisitos e incorporar as funcionalidades necessárias para atender à ampla gama de aplicações de RSSFs. Tais funcionalidades são providas sob a forma de componentes de serviços do middleware. Dentre os requisitos atendidos e aos serviços fornecidos pelo sistema de middleware proposto, podem-se destacar os listados a seguir.

   - **Fornecimento de capacidades de ciência de contexto e adaptação**, essenciais para garantir o funcionamento eficiente da rede e o atendimento das necessidades das aplicações, no ambiente dinâmico no qual operam as RSSFs. Tais capacidades são fornecidas através da implementação de mecanismos baseados no princípio da reflexão.

   - **Fonecimento de um serviço de decisão automatizado** para estabelecer a melhor configuração da rede de acordo com a tarefa solicitada pela aplicação. Esse serviço tem o objetivo principal de tomar decisões sobre mecanismos de baixo nível que freqüentemente estão fora do escopo do desenvolvedor de aplicações. As decisões tomadas visam fornecer uma utilização eficiente da rede em termos de consumo de energia. Uma versão inicial do algoritmo de decisão adotado pelo middleware foi implementada a partir da literatura existente e está sendo continuamente refinado a partir dos resultados de trabalhos de simulação. Através do serviço de decisão, o middleware permite que uma RSSF seja otimizada de acordo com as necessidades das aplicações.

   - **Fonecimento de um serviço de seleção de nós ativos,** responsável por selecionar os nós da rede que irão participar de uma tarefa de sensoriamento recebida. O gerenciamento inteligente das tarefas de uma RSSF, alocando diferentes

porcentagens de nós para atender às aplicações, permite criar configurações de rede que fornecem desde uma grande cobertura/alta qualidade com um curto tempo de vida/grande consumo de energia até pouca cobertura/baixa qualidade e longo tempo de vida/baixo consumo, dependendo de que nível de QoS é desejado pela aplicação. A partir dos resultados de simulações podem-se construir curvas de compromisso entre diferentes requisitos, auxiliando os gerentes da rede ou os desenvolvedores das aplicações em suas decisões quanto à configuração. Tais curvas podem fornecer, por exemplo, a maior acurácia que pode ser obtida, dado um orçamento (porcentagem de nós ativos). Por outro lado, pode-se desejar fixar o requisito de acurácia e, nesse caso, as curvas podem informar o tempo de vida máximo que a rede pode atingir.

Os benefícios dos serviços oferecidos pelo middleware foram avaliados através da realização de simulações. As interações do sistema com aplicações clientes, bem como a viabilidade da implantação do middleware nos limitados dispositivos sensores foram avaliadas através da implementação de um protótipo em Java.

Quanto ao serviço de decisão, uma sub-parte da versão atual do algoritmo foi simulada e observou-se uma melhoria de até 25% na energia média dissipada na rede quando o protocolo de comunicação e seus parâmetros são selecionados de acordo com os requisitos da aplicação. É importante notar que o módulo de decisão do middleware executa tal seleção integralmente, sem qualquer interferência da aplicação.

Com relação ao serviço de seleção de nós ativos, ganhos de até 1000% na energia residual final da rede foram obtidos quando apenas uma porcentagem de 30% dos nós era selecionada para realizar uma tarefa, em contraste com a ativação de todos os nós. Porém, com tal orçamento os requisitos da aplicação não puderam ser atendidos até o final do tempo de monitoramento solicitado. Ganhos menores, porém ainda significativos, foram obtidos com a ativação de um maior número de sensores, respeitando-se integralmente os requisitos de QoS solicitados pela aplicação. Variações do algoritmo de seleção foram avaliadas, com resultados bastante promissores. A adoção da diferenciação dos papéis de sensor e roteador na escolha dos nós ativos levou a tempos de vida da rede muito mais longos do que quando não se diferenciavam tais papéis, mostrando que os recursos da rede puderam ser aproveitados de forma muito mais eficiente, enquanto os requisitos da aplicação mantinham-se respeitados.

Quanto ao serviço de adaptação, foram simuladas duas diferentes políticas, nas quais o consumo de energia da rede e os valores de QoS fornecidos para a aplicação eram monitorados a cada ciclo e ações eram disparadas para alterar o comportamento da rede quando necessário. As ações implementadas consistiram em novas execuções do algoritmo de seleção de nós ativos, alterando seus parâmetros. Observou-se que o tempo de vida da rede pôde ser estendido ao se aplicar tais políticas, enquanto os parâmetros de QoS da aplicação continuaram a ser atendidos.

Uma estratégia simplista de seleção dos nós foi simulada para comparar com a estratégia adotada pelo middleware. A estratégia do middleware teve melhor desempenho tanto com relação ao consumo de energia quanto à QoS fornecida para a aplicação.

Quanto ao protótipo do sistema, sua construção permitiu validar a interação de alto nível entre a RSSF e aplicações clientes, realizada através dos nós sorvedouros, mostrando como tal interação pode ser implementada sem esforço de programação pelos desenvolvedores, com o auxílio de ferramentas existentes. Medidas foram realizadas com o protótipo dos nós sensores, comprovando que o tamanho em Bytes do middleware, bem como o consumo de memória exigido pelo módulo de comunicação e por um módulo de serviço implementado, permitem sua instalação em dispositivos de hardware atualmente existentes para sensores. Adicionalmente, um protocolo XML binário foi implementado para comparar seu desempenho com o do formato XML original. Resultados mostraram que o protocolo binário exige menos memória e gera menos tráfego na rede, enquanto mantém a semântica original das mensagens XML, sendo, portanto, mais adequado para a comunicação dentro da rede.

9.2 TRABALHOS FUTUROS

Os trabalhos futuros a serem realizados serão conduzidos ao longo de diferentes etapas. A seguir são detalhadas as principais atividades a serem realizadas em cada etapa e suas metas.

9.2.1 Extensão do Protótipo do Sistema

O protótipo do sistema, já parcialmente implementado na plataforma Java, será incrementado, adicionando-se as funcionalidades dos serviços de reflexão e adaptação

oferecidos pelo middleware. A principal meta dessa extensão do protótipo é analisar o desempenho dos serviços oferecidos.

Para a implementação do serviço de reflexão, localizado no sorvedouro, será utilizada a API de reflexão Java. Essa API é composta pelo pacote `java.lang.reflect` e pela classe `java.lang.Class`. A API fornece suporte para aplicações que requerem acesso a membros públicos de um objeto alvo (baseado na classe do objeto em tempo de execução) ou a membros declarados por uma dada classe. A partir de mensagens SOAP enviadas pela aplicação, o sistema irá inspecionar as variáveis solicitadas e retornar o resultado da inspeção para a aplicação.

O serviço de adaptação, por sua vez, será implementado por um módulo de software que, a partir de mensagens SOAP de solicitação enviadas pela aplicação, irá alterar parâmetros ou variáveis no perfil da aplicação solicitante. As alterações realizadas irão afetar o comportamento do sistema, podendo levar a novas execuções do algoritmo de seleção ou a mudanças na configuração da rede. O impacto, em termos de processamento, de receber as solicitações de adaptação e efetuar as mudanças solicitadas será avaliado utilizando-se as ferramentas disponível na plataforma J2ME. Já o impacto em termos do tempo de resposta da rede para se reconfigurar será avaliado com a realização de simulações.

9.2.2 Refinamento e Aprimoramento do Módulo de Seleção de Nós Ativos

O principal objetivo dessa etapa é estender o domínio de utilização do algoritmo de seleção de nós ativos, tornando-o suficientemente genérico para atender os requisitos dos vários tipos de aplicações de RSSFs. Portanto, o módulo de seleção de nós ativos terá suas funcionalidades ampliadas, com duas principais melhorias previstas:

- Considerar sensores heterogêneos quanto às suas capacidades de sensoriamento.
- Adaptar o algoritmo de seleção para o tratamento de aplicações sensíveis no tempo, ou seja, onde a latência é um requisito de QoS essencial, em vez da energia residual ou do tempo de vida da rede. Deve-se verificar se o algoritmo baseado no Problema da Mochila pode continuar a ser utilizado ou se alguma outra solução baseada em programação linear deve ser investigada

9.2.3 Refinamento e Avaliação do Algoritmo de Decisão

Finalmente, novas simulações devem ser executadas a fim de medir os parâmetros de desempenho do middleware cuja avaliação depende da análise do comportamento do sistema em redes de grande escala. O principal objetivo das futuras simulações é validar o algoritmo de decisão. Devem ser simulados cenários usando os diferentes requisitos das aplicações, representados através dos diversos tipos de mensagens de interesses, e adotando as diferentes soluções de topologia/protocolo existentes na versão atual do algoritmo. O desempenho da rede para cada combinação de requisitos da aplicação *versus* configuração da rede deve ser avaliado a fim de comprovar os ganhos do serviço de decisão do middleware. Essa etapa já vem sendo desenvolvida por alunos de Mestrado em Informática do NCE/IM.

Referências Bibliográficas

1 "With Glacier Park in Its Path, Fire Spreads to 40,000Acres", *New York Times*, v. 150, n. 51864, p. 24, 9/2/2001.

2 AGRE, J., CLARE, L. "An integrated architecture for cooperative sensing networks". *IEEE Computer Magazine* v. 33, n. 5, pp. 106-108, May 2000.

3 AKYILDIZ, I. et al. "Wireless sensor networks: a survey". *Computer Networks* v. 38, n. 4, pp. 393–422, March 2002.

4 Apache Axis [Online]. Disponível em: http://ws.apache.org/axis/. Último acesso: 23/05/2005.

5 Apache Jakarta Tomcat Project [Online]. Disponível em: http://jakarta.apache.org/tomcat/. Último acesso: 23/05/2005.

6 AUSTER, B. "Waging War", *ASEE Prism Online Magazine* v.11, n. 6, pp. 19-23, Feb. 2002 .

7 BANGOLAE, S. L., JAYASUMANA, A. P., CHANDRASEKAR, V. "Gigabit Networking: Digitized Radar Data Transfer and Beyond". In: *Proceedings of the IEEE International Conference on Communications (ICC '03),* USA, May 2003.

8 BELLWOOD, T., et al. UDDI V. 3.0 Specification [Online]. Disponível em: http://uddi.org/pubs/uddi-v3.00-published-20020719.htm. Último acesso: junho/2003.

9 BENINI, L., DEMICHELI, G. *Dynamic Power Management: Design Techniques and CAD Tools*. Kluwer, Norwell, MA, 1997.

10 BHARDWAJ, M., GARNETT, T., CHANDRAKASAN, A. P. "Upper bounds on the lifetime of sensor networks". In: *Proceedings of the IEEE International Conference on Communications (ICC'01),* Finland, June 2001.

11 BONNET, P., GEHRKE, J. E., SESHADRI, P. "Querying the physical world", *IEEE Personal Communications, Special Issue on Smart Spaces and Environments* v. 7, n.5, pp.10-15, Oct. 2000.

12 BONNET, P., GEHRKE, J. E., SESHADRI, P. "Towards Sensor Database Systems". In: *Proceedings of the Second International Conference on Mobile Data Management,* Hong Kong, Jan. 2001.

13 BOULIS, A., GANERIWALD, S., SRIVASTAVA, M. B., "Aggregation in Sensor Networks: An Energy-Accuracy Trade-off", *Elsevier Science Ad Hoc Networks, Special Issue on Sensor Network Protocols and Applications*, v. 1, n. 2-3, pp. 317-331, Sep. 2003.

14 BOULIS, A., SRIVASTAVA, M. B., "Node-level Energy Management for Sensor Networks in the Presence of Multiple Applications", *ACM Baltzer Wireless Networks, The Journal of Mobile Communications, Computation and Information (WINET), Special Issue on Pervasive Computing*, v. 10, n. 6, pp. 737 – 746, Nov. 2004.

15 BRAGINSKY, D., ESTRIN, D. “Rumor routing algorithm for sensor networks”. In: *Proceedings of the First ACM International Workshop on Wireless SensorNetworks and Applications (WSNA 2002),* USA, Sep. 2002.

16 BULUSU, N., ESTRIN, D., GIROD, L., et al. “Scalable coordination for wireless sensor networks: self-configuring localization systems”. In: *Proceedings of the International Symposium on Communication Theory and Applications (ISCTA 2001)*, UK, July 2001.

17 CAMPBELL, A. T., “Mobiware: QOS aware middleware for mobile multimedia communications”. In: *Proceedings of the 7th IFIP International Conference on High Performance Networking (HPN),* EUA, April 1997.

18 CAPRA, L., EMMERICH, W., MASCOLO, C. “CARISMA: Context-Aware Reflective mIddleware System for Mobile Applications”. *IEEE Transactions on Software Engineering* v. 29, n.10, pp. 929-945, 2003.

19 CAPRA, L., EMMERICH, W., MASCOLO, C. “Reflective Middleware Solutions for Context-Aware Applications”. In: *Proceedings of the Reflection 2001*, pp. 126-133, Lecture Notes in Computer Science 2192, Springer Verlag, Japan, 2001.

20 CERPA, A., ELSON, J., HAMILTON, et al. “Habitat monitoring: application driver for wireless communications technology”. In: *Proceedings of the 2001 ACM SIGCOMM Workshop on Data Communications in Latin America and the Caribbean*, Costa Rica, April 2001.

21 CHAKRABARTY, K., et al. “Grid coverage for surveillance and target location in distributed sensor networks”, *IEEE Transactions on Computers* v. 51, n.12, pp. 1448-1453, 2002.

22 CHEN, B. et al. “Span: An Energy-Efficient Coordination Algorithm for Topology Maintenance”, In: *Proceedings of the 2001 ACM/IEEE Ad Hoc Wireless Networks*, Italy, July 2001.

23 CHU, M., HAUSSECKER, H., ZHAO, F. “Scalable information-driven sensor querying and routing for ad hoc heterogeneous sensor networks”, *International Journal of High Performance Computing Applications* v. 16, n. 3, pp. 90 –110, 2002.

24 COKUS, M., WINKOWSKI, D. “XML Sizing and Compression Study For Military Wireless Data”. *XML Conference and Exposition 2002*, Baltimore, MD, USA Dec. 8-13, 2002.

25 COLERI, S., PURI, A., VARAIYA, P. “Power Efficient System for Sensor Networks”. In: *Proceedings of the Eighth IEEE International Symposium on Computers and Communication (ISCC 2003),* Turkey, July 2003.

26 CORMEN, T. H., et al, *Introduction to Algorithms*, MIT Press, 2001.

27 COSTA, F., DURAN, H., PARLAVANTZAS, N., et al. “The Role of Reflective Middleware in Supporting the Engineering of Dynamic Applications”, *Reflection and Software Engineering, Springer-Verlag, LNCS* v. 1826, pp 79-98, 2000.

28 Cougar Project. [Online]. Disponível em: http://www.cs.cornell.edu/database/cougar. Último acesso: 23/05/2005.

29 COULSON, G. “What is reflective middleware?”, *IEEE Distributed Systems Online* v. 2, n. 8, Dec. 2001. Disponível em: computer.org/dsonline/middleware/ RMarticle1.htm. Último acesso: 23/05/2005.

30 COULSON, G., BLAIR, G.S., CLARK, M., et al. “The Design of a Configurable and Reconfigurable Middleware Platform”, *ACM Distributed Computing Journal* v. 15, n. 2, pp 109-126, April 2002.

31 COULSON, G., et al., "OpenCOM v2: A Component Model for Building Systems Software", In: *Proceedings of IASTED Software Engineering and Applications (SEA'04)*, Cambridge, MA, ESA, Nov 2004.

32 DAM, T. V., LANGENDOEN, K. “An Adaptive Energy-Efficient MAC Protocol for Wireless Sensor Networks”. In: *Proceedings of the First ACM Conference on Embedded Networked Sensor Systems (Sensys'03),* USA, Nov. 2003.

33 DAVIES, N., WADE, S., FRIDAY, A., et al. “Limbo: A tuple space based platform for adaptive mobile applications”. In: *Proceedings of the International Conference on Open Distributed Processing/Distributed Platforms (ICODP/ICDP '97),* Canada, May 1997.

34 DELICATO, F. C., et al. “A Flexible Middleware System for Wireless Sensor Networks”. In: *Proceedings of the ACM/IFIP/USENIX International Middleware Conference (Middleware2003),* pp. 474-492, Rio de Janeiro, July 2003.

35 DELICATO, F. C., et al. “Application-Driven Node Management in Multihop Wireless Sensor Networks”. In: *Proceedings of the 4th IEEE International Conference on Networking (ICN2005),* Reunion Island, April 2005.

36 DELICATO, F. C., et al. “Middleware Orientado a Serviços para Redes de Sensores sem Fio”. In: *Anais do 22º Simpósio Brasileiro de Redes de Computadores (SBRC2004),* Rio Grande do Sul, Brasil, Maio 2004.

37 DELICATO, F. C., et al. “Reflective Middleware for Wireless Sensor Networks”. In: *Proceedings of the 20th ACM Symposium on Applied Computing (SAC2005),* pp. 1155-1159, USA, March 2005.

38 EDWARDS, K. *Core JINI*. Prentice Hall, 1999.

39 ELSON, J., ESTRIN, D. “An address-free architecture for dynamic sensor networks”. Technical Report 00-724, Computer Science Department, USC, USA, 2000.

40 ELSON, J., GIROD, L., ESTRIN, D. “Fine-grained network time synchronization using reference broadcasts”. In: *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI 2002),* USA, Dec. 2002.

41 ESTRIN, D., GOVINDAN, R., HEIDEMANN, J. “Embedding the Internet”, *Communications of the ACM* v. 43, n. 5, pp. 38-41, May 2000.

42 ESTRIN, D., GOVINDAN, R., HEIDEMANN, J., KUMAR, S. “Next century challenges: scalable coordination in sensor networks”. In: *Proceedings of the* ACM MobiCom’99, pp. 263–270, USA, Aug. 1999.

43 ESTRIN, D., SAYEED, A., SRIVASTAVA, M."Wireless Sensor Networks". Tutorial apresentado em: *The Eighth ACM International Conference on Mobile Computing and Networking*, USA, Sep. 2002.

44 FIELDING, R., et al. RFC 2616. Hypertext Transfer Protocol - HTTP/1.1. June, 1999. Disponível em: ftp://ftp.rfc-editor.org/in-notes/rfc2616.txt. Último acesso: junho/2003.

45 FLINN, J., NARAYANAN, D., SATYANARAYANAN, M. "Self-tuned remote execution for pervasive computing". In: *Proceedings of the Eighth IEEE HotOs Conference*, Germany, May 2001.

46 FREED, N., BORENSTEIN, N. "Multipurpose Internet Mail Extensions (MIME) part One: Format of Internet Message Bodies". IETF RFC 2045. Disponível em: http://ietf.org/rfc/rfc2045.txt, Nov. 1996. Último acesso: junho/ 2003.

47 FROLIK, J. "QoS Control for Random Access Wireless Sensor Networks". In: *Proceedings of the 2004 Wireless Communications and Networking Conference (WCNC04),* USA, March 2004.

48 GAO, Q., et al. "Energy Efficiency Design Challenge in Sensor Networks". In: *Proceedings of the London Communications Symposium (LCS 2002),* UK, Sep. 2002.

49 GARLAN, D., SIEWIOREK, D., SMAILAGIC, A., et al. "Project Aura: Toward distraction-free pervasive computing". *IEEE Pervasive Computing, Special Issue on "Integrated Pervasive Computing Environments",* v. 21, n. 2, pp. 22-31, April-June, 2002.

50 GELERNTER, D. "Generative Communication in Linda". *ACM Transactions on Programming Languages and Systems* v.7, n. 1, pp. 80-112, Jan.1985.

51 GOTTSCHALK, K., et al. "Introduction to web services architecture". *IBM Systems Journal Online* v. 41, n. 2, 2002. Disponível em: http://www.research.ibm.com/journal/sj/412/gottschalk.pdf. Último acesso: 23/05/2005.

52 GOVINDAN, R., et al. "The Sensor Network as a Database". USC Technical Report No. 02-771, Sep. 2002.

53 GRAHAM, S., et al. *Building Web Services with Java: Making Sense of XML, SOAP, WSDL, and UDDI*. Sams Publishing, 2002.

54 HAN, Q., VENKATASUBRAMANIAN, N. "Autosec: An integrated middleware framework for dynamic service brokering". *IEEE Distributed Systems Online* v. 2, n.7, 2001.

55 HE, T., et al. "AIDA: Adaptive Application Independent Data Aggregation in Wireless Sensor Networks", *ACM Transactions on Embedded Computing System, Special issue on Dynamically Adaptable Embedded Systems* v. 3, n.2, pp. 426 – 457, May 2004.

56 HEIDEMAN, J., SILVA, F., ESTRIN, D. "Matching Data Dissemination Algorithms to Application Requirements". In: *Proceedings of the ACM SenSys Conference*, pp. 218-229, USA, Nov. 2003.

57 HEIDEMANN, J., et al. “Building Efficient Wireless Sensor Networks with Low-Level Naming”. In: *Proceedings of the 2001 ACM Symposium on Operating Systems Principles,* pp. 146-159, Canada, Oct. 2001.

58 HEIDEMANN, J., et al. “Diffusion filters as a flexible architecture for event notification in wireless sensor networks”. Technical Report ISI-TR-556, USC/Information Sciences Institute, April 2002.

59 HEINZELMAN, W., CHANDRAKASAN, A., BALAKRISHNAN, H. “An Application-Specific Protocol Architecture for Wireless Microsensor Networks*”. IEEE Transactions on Wireless Communications* v.1, n. 4, pp. 660-670, Oct. 2002.

60 HEINZELMAN, W., CHANDRAKASAN, A., BALAKRISHNAN. “Energy-Efficient Communication Protocol for Wireless Microsensor Networks”. In: *Proceedings of the 33rd Hawaii International Conference on System Sciences (HICSS '00),* USA, Jan. 2000.

61 HEINZELMAN, W., MURPHY, A., CARVALHO, H., et al. “Middleware to Support Sensor Network Applications”. *IEEE Network Magazine Special Issue* v. 18, n. 1, pp. 6-14, Jan. 2004.

62 HERRING, C., KAPLAN, S. “Component-based software systems for smart environments”, *IEEE Personal Communications* v.7, n.5, pp. 60-61, Oct. 2000.

63 HILL, J., et al. “System architecture directions for networked sensors”, *ACM SIGOPS Operating Systems Review* v. 34, n. 5, pp. 93-104, Dec. 2000.

64 HOLDER, O. BEN-SHAUL, I., GAZIT, H. “System Support for Dynamic Layout of Distributed Applications”. In: *Proceedings of the 19th International Conference on Distributed Computing Systems*, pp. 403-411, USA, May 1999.

65 IEEE Standard 802.11 [Online]. Disponível em: http://grouper.ieee.org/groups/802/11/. Último acesso: 24/05/2005.

66 INTANAGONWIWAT, C., GOVINDAN, R., ESTRIN, D. “Directed diffusion: a scalable and robust communication paradigm for sensor networks”. In: *Proceedings of the ACM/IEEE MobiCom 2000*, pp. 56-67, USA, Aug. 2000.

67 IYER, R., KLEINROCK, L. “QoS Control for Sensor Networks”. In: *Proceedings of the IEEE International Conference on Communications (ICC '03),* v. 26, n. 1, pp. 517 - 521, USA, May 2003.

68 J2ME Wireless Toolkit [Online]. Disponível em: http://java.sun.com/products/j2mewtoolkit/. Último acesso: 23/05/2005.

69 Java API for XML-Based RPC (JAX-RPC) [Online]. Disponível em: http://java.sun.com/xml/jaxrpc/index.jsp/. Último acesso: 23/05/2005.

70 JIANG, Q., MANIVANNAN, D. “Routing Protocols for Sensor Networks”. In: *Proceedings of the IEEE Consumer Communications and Networking Conference (CCNC 2004),* USA, Jan. 2004.

71 JIST – Java in Simulation Time. Disponível em: http://jist.ece.cornell.edu/. Último acesso: 23/05/2005.

72 KANNAN, R., KALIDINDI, R., IYENGAR, S. S., et al. “Energy and Rate based MAC Protocol for Wireless Sensor Networks”. *ACM SIGMOD Record, Special*

Issue: Special section on sensor network technology and sensor data managment v.32, n. 4, pp. 60 - 65, Dec. 2003.

73 KON, F., COSTA, F., BLAIR, G.S., CAMPBELL, R. "The Case for Reflective Middleware". *Communications of the ACM* v. 45, n. 6, June 2002.

74 KRISHNAMACHARI, B., ESTRIN, D., WICKER, S. "Modelling Data-Centric Routing in Wireless Sensor Networks". In: *Proceedings of the 2002 IEEE Infocom*, EUA, June 2002.

75 KULIK, J., RABINER, W., BALAKRISHNAN, H. "Adaptive Protocols for Information Dissemination in Wireless Sensor Networks". In: *Proceedings of the 5th ACM/IEEE Mobicom Conference*, USA, Aug. 1999.

76 KXML Project [Online]. Disponível em http://kxml.objectweb.org/. Último acesso: 15/03/2005.

77 LETTIERI, P., FRAGOULI, C., SRIVASTAVA, M. "Low power error control for wireless links". In: *Proceedings of the Third Annual ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom*), pp.139-150, Hungary, Sep. 1997.

78 LI, S., SON, S., STANKOVIC, J. "Event detection services using data service middleware in distributed sensor networks". In: *Proceedings of the 2nd International Workshop on Information Processing in Sensor Networks*, USA, April 2003.

79 LINDSEY, S., RAGHAVENDRA, C. S. "PEGASIS: Power-efficient Gathering in Sensor Information Systems". In: *Proceedings of the Aerospace Conference*, pp. 1125-1130, 2002.

80 LIU, J., et al. "A dual-space approach to tracking and sensor management in wireless sensor networks". In: *Proceedings of the First ACM International Workshop on Wireless SensorNetworks and Applications (WSNA 2002),* USA, Sep. 2002.

81 LIU, T., MARTONOSI, M. "Impala: A Middleware System for Managing Autonomic Parallel Sensor Systems". In: *Proceedings of the ninth ACM Symposium on Principles and Practice of Parallel Programming (SIGPLAN),* pp.107 – 118, USA, June 2003.

82 LIU, T., SADLER, C. M., ZHANG, P., et al.. "Implementing Software on Resource-Constrained Mobile Sensors: Experiences with Impala and ZebraNet". In: *Proceedings of the Second International Conference on Mobile Systems, Applications, and Services (MobiSYS 2004)*, USA, June 2004.

83 LU, G., KRISHNAMACHARI, B., RAGHAVENDRA, C. "An Adaptive Energy-Efficient and Low-Latency MAC for Data Gathering in Sensor Networks". In: *Proceedings of the 4th International Workshop on Algorithms for Wireless, Mobile, Ad Hoc and Sensor Networks*, USA, April 2004.

84 MAINWARING, A., et al. "Wireless sensor network for habitat monitoring". In: *Proceedings of the 2002 ACM International Workshop on Wireless Sensor Networks and Applications*, Atlanta, USA, Sep. 2002.

85 MASCOLO, C., CAPRA, L., EMMERICH, W. “Middleware for Mobile Computing (A Survey)”. Invited Paper In: *Advanced Lectures in Networking*, LNCS 2497, 2002.

86 MEGUERDICHIAN, S, et al. “Localized algorithms in wireless ad-hoc networks: location discovery and sensor exposure”. In: *Proceedings of the 2001 ACM Symposium on Mobile Ad Hoc Networking & Computing (MobiHOC2001),* pp. 106 - 116 , USA, Oct. 2001.

87 MEGUERDICHIAN, S., POTKONJAK, M. “Low Power 0/1 Coverage and Scheduling Techniques in Sensor Networks”, UCLA Technical Reports 030001, Jan. 2003.

88 Microsoft Corp. and Digital Equipment Corp. [Online]. "The Component Object Model Specification”. Disponível em: http://www.graphcomp.com/info/specs/com/comch01.htm. Último acesso: 23/5/2005.

89 Microsoft Corporation. “.Net Framework”. Disponível em: http://msdn.microsoft.com/netframework/default.asp. Último acesso: junho/2004.

90 Microsoft Corporation. “Distributed Component Object Model Protocol-DCOM/1.0,” draft, Nov. 1996 [Online]. Disponível em: http://www.microsoft.com/Com/resources/comdocs.asp. Último acesso: junho/2004.

91 MURPHY, A. L., PICCO, G. P., ROMAN, G.-C. “Lime: A Middleware for Physical and Logical Mobility”. In: *Proceedings of the 21st International Conference on Distributed Computing Systems*, pp. 524-533, EUA, April 2001.

92 NAHRSTEDT, K., XU, D. “QoS-aware middleware for ubiquitous and heterogeneous environments”, *IEEE Communications Magazine* v. 39, n.11, pp. 140–148, 2001.

93 NASIPURI, A., LI, K. “A Directionality based Location Discovery Scheme for Wireless Sensor Networks”. In: *Proceedings of the First ACM International Workshop on Wireless Sensor Networks and Applications (WSNA'02),* USA, Sep. 2002.

94 NEWCOMER, E., *Understanding web services: XML, WSDL, SOAP and UDDI*. Addison-Wesley, 2002.

95 NOBLE, B. D., SATYANARAYANAN, M. “Experience with Adaptive Mobile Applications in Odyssey”. Mobile Networks and Applications v. 4, n. 4, pp. 245-254, 1999.

96 NOURY, N., et al. “Monitoring behavior in-home using a smart fall sensor”. In: *Proceedings of the IEEE-EMBS Special Topic Conference on Microtechnologies in Medicine and Biology*, pp. 607–610, Oct. 2000.

97 NS-2 (The Network Simulator version 2) [Online]. Disponível em: http://www.isi.edu/nsnam/ns/. Último acesso: 24/05/ 2005.

98 OMG Common Object Request Broker Architecture: Core Specification. V. 3.0.3. [Online] Disponível em:

http://www.omg.org/technology/documents/formal/corba_2.htm. Último acesso: março/2004.

99 PACH, J., AGARWAL, P. K. *Combinatorial Geometry*, Wiley Pubs., New York, 1995.

100 PEI, G, CHIEN C. "Low power TDMA in Large Wireless Sensor Networks". In: *Proceedings of the Military Communications Conference (MILCOM'01),* pp. 28-31, USA, Oct. 2001.

101 PERILLO, M., HEINZELMAN, W. "Providing Application QoS Through Intelligent Sensor Management". In: *Proceedings of the 1st IEEE International Workshop on Sensor Network Protocols and Applications*, USA, May 2003.

102 PERILLO, M., HEINZELMAN, W. "Optimal sensor management under energy and reliability constraints". In: *Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC '03),* USA, March 2003.

103 PERILLO, M., HEINZELMAN, W. "Sensor Management Policies to Provide Application QoS", *Elsevier AdHoc Networks Journal, Special Issue on Sensor Network Applications and Protocols* v. 1, n. 2-3, pp.235-246, 2003.

104 PERING, T. A., BURD, T. D., BRODERSEN, R. W. "The simulation and evaluation of dynamic voltage scaling algorithms". In: *Proceedings of the 1998 International Symposium on Low Power Electronics and Design (ISLPED),* pp. 76 - 81, USA, Aug. 1998.

105 PERRIG, A., et al. "SPINS: Security protocols for sensor networks". In: *Proceedings of the Seventh Annual International Conference on Mobile Computing and Networking*, Italy, July 2001.

106 PETRIU, E. M., et al. "Sensor-based information appliances", *IEEE Instrumentation and Measurement Magazine* v. 3, n. 4, pp. 31-35, Dec. 2000.

107 PINTO, A. J. G., 2004, *Mecanismo de Agregação de Dados baseado em Técnicas Paramétricas aplicado em Redes de Sensores*. Dissertação de M.Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil.

108 PRABHAKAR, B., UYSAL-BIYIKOGLU, E., GAMAL, A. E. "Energy-efficient transmission over a wireless link via lazy packet scheduling". In: *Proceedings of the 2001 IEEE Infocom*, EUA, April 2001.

109 PRIYANTHA, N., CHAKRABORTY, A., BALAKRISHNAN, H. "The cricket location-support system". In: *Proceedings of the ACM MobiCom'00*, pp. 32–43, USA, Aug. 2000.

110 QI, H., KURUGANTI, P. T., XU, Y. "The Development of Localized Algorithms in Wireless Sensor Networks", Invited Paper, *Sensors 2002* v. 2, pp. 286-293, 2002.

111 QIU, Q., PEDRAM, M. "Dynamic power management based on continuous-time Markov decision processes". In: *Proceedings of the Design Automation Conference (DAC),* pp. 555–561, USA, June 1999.

112 RABAEY, J., AMMER, J., SILVA, J. L., et al. "Pico-Radio: ad-hoc wireless networking of ubiquitous low energy sensor/monitor nodes". In: *Proceedings of*

the IEEE Computer Society Annual Workshop on VLSI (WVLSI'00), pp. 9-12, USA, April 2000.

113 RAGHUNATHAN, V., SCHURGERS, C., PARK, S., et al. "Energy Aware Wireless Microsensor Networks". IEEE Signal Processing Magazine v.19, n. 2, pp. 40-50, March 2002.

114 RAGHUNATHAN, V., SPANOS, P., SRIVASTAVA, M. "Adaptive power-fidelity in energy aware wireless embedded systems". In: *Proceedings of the 22nd IEEE Real-Time Systems Symposium (RTSS'01)*, UK, Dec. 2001.

115 RAJENDRAN, V., OBRACZKA, K., GARCIA-LUNA-ACEVES, J.J.. "Energy-Efficient, Collision-Free Medium Access Control for Wireless Sensor Networks". In: *Proceedings of the First ACM Conference on Embedded Networked Sensor Systems (Sensys'03),* pp. 181 – 192, USA. Nov. 2003.

116 RÖMER, K., KASTEN, O., MATTERN, F. "Middleware Challenges for Wireless Sensor Networks". *ACM Mobile Computing and Communications Review* v. 6, n. 2, 2002.

117 SANKARASUBRAMANIAM, Y., AKAN, O. B., AKYILDIZ, I. F. "ESRT: Event-to-Sink Reliable Transport in Wireless Sensor Networks". In: *Proceedings of the 4th ACM international symposium on Mobile ad hoc networking & computing (MobiHoc'03)*, pp. 177 - 188, USA, June 2003.

118 SCHURGERS, C., ABERTHORNE, O., SRIVASTAVA, M. "Modulation scaling for energy aware communication systems". In: *Proceedings of the 2001 International Symposium on Low power electronics and design (ISLPED),* pp. 96-99, Aug. 2001.

119 Sensoria WINS 2.0 [Online]. Disponível em: http://www.sensoria.com/. Último acesso: 24/05/2005.

120 Sensoria WINS 3.0 Specification Sheet [Online]. Disponível em: http://www.sensoria.com/products-wins30.htm. Último acesso: 24/05/2005.

121 SHAH, R. C., RABAEY, J. "Energy aware routing for low energy ad hoc sensor networks". In: *Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC2002),* USA, March 2002.

122 SHEN, C., SRISATHAPORNPHAT, C., JAIKAEO, C. "Sensor Information Networking Architecture and Applications", *IEEE Personal Communications* v. 8, n. 4, pp. 52–59, Aug. 2001.

123 SHIH, E. et al. "Physical Layer Driven Protocol and Algorithm Design for Energy-Efficient Wireless Sensor Networks". In: *Proceedings of the 7th annual international conference on Mobile computing and networking (ACM MOBICOM 01),* pp. 272–286, Italy, July 2001.

124 SINGH, M., PRASANNA, V. K. "A hierarchical model for distributed collaborative computation in wireless sensor networks". In: *Proceedings of the 5th Workshopon Advances in Parallel and Distributed Computational Models*, France, April 2003.

125 SINHA, A., WANG, A., CHANDRAKASAN, A. P. "Algorithmic transforms for efficient energy scalable computation". In: *Proceedings of the 2000*

International Symposium on Low Power Electronics and Design, pp. 31-36, Italy, July 2000.

126 SMITH, B. C., 1982, *Reflection and Semantics in a Procedural Programming Language*. Ph.D. thesis, MIT, USA.

127 SOHRABI, K., GAO, J., AILAWADHI, V., et al. "Protocols for self-organization of a wireless sensor network". *IEEE Personal Communication Magazine* v.7, pp. 16-27, Oct. 2000.

128 SRINIVASAN, V., NUGGEHALLI, P., RAO, R. "Design of optimal energy aware protocols for wireless sensor networks". In: *Proceedings of the IEEE Vehicular Technology Conference (VTC'01)*, pp. 6-9, Greece, May 2001.

129 SUN Microsystems, "Enterprise JavaBeans Specification 2.0." [Online]. Disponível em: http://java.sun.com/products/ejb/docs.html. Último acesso: junho/2003.

130 SUN Microsystems, "Implementing Services On Demand and the Sun Open Net Environment (Sun ONE)." [Online]. Disponível em: http://www.sun.com/software/sunone/wpimplement/wp-implement.pdf. Último acesso: junho/2003.

131 TILAK, S., ABU-GHAZALEH, N. B., HEINZELMAN, W. "A taxonomy of wireless micro-sensor network models". *ACM SIGMOBILE Mobile Computing and Communications Review* v. 6, n. 2, pp. 28-36, April 2002.

132 TILAK, S., ABU-GHAZALEH, N. B., HEINZELMAN, W. "Infrastructure tradeoffs for sensor networks". In: *Proceedings of the First ACM International Workshop on Wireless Sensor Networks and Applications*, USA, Sep. 2002.

133 ULMER, C. "Organization Techniques in Wireless In-situ Sensor Networks". Technical Report, Georgia Tech. Disponível em: http://www.craigulmer.com/research/. Último Acesso: 23/05/2005.

134 UML Specification [Online]. Disponível em: http://www.uml.org/. Último acesso: 24/05/2005.

135 W3C (World Wide Web Consortium) Recommendation 04 February 2004 [Online], "Extensible Markup Language (XML) 1.0 (Third Edition)". Disponível em: http://www.w3.org/TR/REC-xml. Último acesso: 24/05/2005.

136 W3C (World Wide Web Consortium) Recommendation 24 June 2003 [Online], "SOAP Version 1.2 Part 0: Primer". Disponível em: http://www.w3.org/TR/2003/REC-soap12-part0-20030624/. Último acesso: 24/05/2005.

137 W3C (World Wide Web Consortium) Recommendation 28 October 2004 [Online], "XML Schema Part 0: Primer Second Edition". Disponível em: http://www.w3.org/TR/xmlschema-0/. Último acesso: 24/05/2005.

138 W3C Note 24 June 1999 [Online], "WAP Binary XML Content Format". Disponível em: http://www.w3.org/TR/wbxml/. Último acesso: 24/05/2005.

139 W3C Recommendation [Online]. "SOAP version 1.2. Part 0: Primer", 24 June 2003. Disponível em: http://www.w3.org/TR/soap12-part0/. Último acesso: 24/05/2005.

140 W3C Recommendation 24 June 2003 [Online], "Simple Object Access Protocol (SOAP) Version 1.2 Part 1". Disponível em: http://www.w3.org/TR/soap12-part1/. Último acesso: 24/05/2005.

141 W3C Working Draft [Online]. "Web services description language (WSDL) Version 2.0 Part 1: Core Language". Disponível em: http://www.w3.org/TR/wsdl12/. Último acesso: 24/05/2005.

142 WAN, C., CAMPBELL, A., KRISHNAMURTHY, L. "PSFQ: A Reliable Transport Protocol For Wireless Sensor Networks". In: *Proceedings of the First Workshop on Sensor Networks and Applications (WSNA),* USA, Sep. 2002.

143 WAN, C., EISENMAN, S. B., CAMPBELL, A. T. "CODA: Congestion Detection and Avoidance in Sensor Networks". In: *Proceedings of the ACM SenSys 2003*, EUA, Nov. 2003.

144 WANG, X., et al. "Integrated Coverage and Connectivity Configuration in Wireless Sensor Networks". In: *Proceedings of the 1st international Conference on Embedded Networked Sensor Systems*, pp. 28 - 39, USA, Nov. 2003.

145 WARNEKE, B., LIEBOWITZ, B., PISTER, K. S. J, "Smart dust: communicating with a cubic-millimeter computer", *IEEE Computer* v.34, n.1, pp. 44-51, Jan. 2001.

146 Web Services Project Apache [Online]. Disponível em http://ws.apache.org/. Último acesso: 24/05/2005.

147 Web Services Toolkit [Online]. Disponível em: http://www.alphaworks.ibm.com/tech/webservicestoolkit. Último acesso: 23/05/2005.

148 WEISER, M., WELCH, B., DEMERS, A., et al. "Scheduling for reduced CPU energy". In: *Proceedings of the USENIX Symposium on Operating SystemsDesign and Implementation*, pp. 13–23, USA, Nov. 1994.

149 WELCH, G., BISHOP, G. "An Introduction to the Kalman Filter". Technical Report number 95-041, University of North Carolina at Chapel Hill, DCS. Disponível em: http://www.cs.unc.edu/~welch/kalman/kalmanIntro.html.

150 Wireless Application Protocol (WAP) Specification. [Online]. Disponível em http://www.wapforum.org/what/technical.htm/. Último acesso: 23/05/2005.

151 XU, Y., BIEN, S., MORI, Y., et al. "Topology Control Protocols to Conserve Energy in Wireless Ad Hoc Networks". CENS Technical Report 0006, UCLA, USA, Jan. 2003.

152 XU, Y., HEIDEMANN, J., ESTRIN, D. "Adaptive Energy-Conserving Routing for Multihop Ad Hoc Networks", Research Report 527, USC/Information Sciences Institute, USA, Oct. 2000.

153 XU, Y., HEIDEMANN, J., ESTRIN, D. "Geography-informed Energy Conservation for Ad Hoc Routing". In: *Proceedings of the 7th Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom2001)*, pp. 70 – 84, Italy, July 2001.

154 YAO, F., DEMERS, A., SHENKER, S. “A scheduling model for reduced CPU energy”. In: *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, pp. 374-382, 1995.

155 YAO, Y., GEHRKE, J. E. “The Cougar Approach to In-Network Query Processing in Sensor Networks”. *ACM Sigmod Record* v. 31, n. 3, pp. 9 – 18, Sep. 2002.

156 YE, F. et al. “A Two-tier Data Dissemination Model for Large-scale Wireless Sensor Networks”. In: *Proceedings of the 8th Annual International Conference on Mobile Computing and Networking*, pp.148 – 159, USA, Sep. 2002.

157 YE, W., HEIDEMAN, J., ESTRIN, D. “An energy-efficient MAC protocol for wireless sensor networks”. In: *Proceedings of the 21st Annual Joint Conference of the IEEE Computer Communications Societies (INFOCOM 2002),* USA, June 2002.

158 YOUNIS, M., YOUSSEF, M., ARISHA, K. “Energy-aware routing in cluster-based sensor networks”. In: *Proceedings of the International Symposium on Modeling, Anaysis and Simulation of Computer and Telecommunication Systems*, USA, Oct. 2002.

159 YU, Y., GOVINDAN, R., ESTRIN, D. “Geographical and Energy Aware Routing: A Recursive Data Dissemination Protocol for Wireless Sensor Networks”. Technical Report-01-0023, UCLA Computer Science Department, May 2001.

160 YU, Y., KRISHNAMACHARI, B., PRASANA, V. K. “Energy-Latency Tradeoffs for Data Gathering in Wireless Sensor Networks”. In: *Proceedings of the IEEE Annual Joint Conference of Computer Communications Societies (INFOCOM 2004)*, Hong Kong, March 2004.

161 YU, Y., KRISHNAMACHARI, B., PRASANA, V. K. “Issues in Designing Middleware for Wireless Sensor Networks”. *IEEE Network Magazine, Special Issue on Middleware Technologies for Future Communication Networks* v.18, n.1, pp.15-21, Jan. 2004.

Anexo 1: Esquema XML para a Mensagem `PublishSensorDescription`.

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Body">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="PublishSensorDesc" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="Confidence">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Max" />
        <xs:element ref="Min" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="DataDomain">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Value" maxOccurs="unbounded" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="DataRate">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Max" />
        <xs:element ref="Min" />
      </xs:sequence>
      <xs:attribute name="unit" type="xs:NMTOKEN" use="required" />
    </xs:complexType>
  </xs:element>
  <xs:element name="Energy">
    <xs:complexType mixed="true">
      <xs:attribute name="unit" type="xs:NMTOKEN" use="required" />
    </xs:complexType>
  </xs:element>
  <xs:element name="Env">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Header" />
        <xs:element ref="Body" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="GeographicLocation">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="x" />
        <xs:element ref="y" />
      </xs:sequence>
      <xs:attribute name="unit" type="xs:NMTOKEN" use="required" />
    </xs:complexType>
  </xs:element>
  <xs:element name="Header" type="xs:string" />
  <xs:element name="Max">
    <xs:complexType mixed="true" />
  </xs:element>
  <xs:element name="Min">
    <xs:complexType mixed="true" />
  </xs:element>
```

```
  <xs:element name="parameter">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="TTL" />
        <xs:element ref="Type" />
        <xs:element ref="DataDomain" />
        <xs:element ref="GeographicLocation" />
        <xs:element ref="Energy" />
        <xs:element ref="Confidence" />
        <xs:element ref="DataRate" />
      </xs:sequence>
      <xs:attribute name="ID" type="xs:NMTOKEN" use="required" />
      <xs:attribute name="NetworkID" type="xs:NMTOKEN" use="required" />
    </xs:complexType>
  </xs:element>
  <xs:element name="PublishSensorDesc">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="parameter" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="TTL">
    <xs:complexType mixed="true">
      <xs:attribute name="unit" type="xs:NMTOKEN" use="required" />
    </xs:complexType>
  </xs:element>
  <xs:element name="Type">
    <xs:complexType mixed="true" />
  </xs:element>
  <xs:element name="Value">
    <xs:complexType mixed="true" />
  </xs:element>
  <xs:element name="x">
    <xs:complexType mixed="true" />
  </xs:element>
  <xs:element name="y">
    <xs:complexType mixed="true" />
  </xs:element>
</xs:schema>
```

Anexo 2 : Documento WSDL da rede para acesso pelas aplicações clientes.

```
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/" xmlns:y="http://namespace.example.com" xmlns:ns="empty"
targetNamespace="http://namespace.example.com">
    <types>
        <xsd:schema targetNamespace="empty" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:ns1="empty" elementFormDefault="qualified">
        <xsd:complexType name="SensorDescType">
            <xsd:sequence>
                <xsd:element name="TTL" type="xsd:long"/>
                <xsd:element name="Type" type="xsd:string"/>
                <xsd:element name="DataDomain" type="ns1:DataDomainType"/>
                <xsd:element name="GeographicLocation" type="ns1:GeographicLocationType"/>
                <xsd:element name="Energy" type="xsd:long"/>
                <xsd:element name="Confidence" type="ns1:IntervalType"/>
                <xsd:element name="DataRate" type="ns1:IntervalType"/>
            </xsd:sequence>
            <xsd:attribute name="ID" type="xsd:ID" use="required"/>
            <xsd:attribute name="NetworkID" type="xsd:ID" use="optional"/>
        </xsd:complexType>
        <xsd:complexType name="GeographicLocationType">
            <xsd:sequence>
                <xsd:element name="x" type="xsd:long"/>
                <xsd:element name="y" type="xsd:long"/>
```

```
            </xsd:sequence>
            <xsd:attribute name="unit" type="xsd:string"/>
        </xsd:complexType>
        <xsd:complexType name="IntervalType">
            <xsd:sequence>
                <xsd:element name="Max" type="xsd:long"/>
                <xsd:element name="Min" type="xsd:long"/>
            </xsd:sequence>
            <xsd:attribute name="unit" type="xsd:string"/>
        </xsd:complexType>
        <xsd:complexType name="DataDomainType">
            <xsd:sequence>
                <xsd:element name="Value" type="xsd:string" maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:complexType>
        <xsd:complexType name="FilterType">
            <xsd:sequence>
                <xsd:element name="Decription" type="xsd:string"/>
                <xsd:element name="Elements" maxOccurs="unbounded">
                    <xsd:complexType/>
                </xsd:element>
            </xsd:sequence>
            <xsd:attribute name="ID" type="xsd:ID" use="required"/>
        </xsd:complexType>
        <xsd:complexType name="DataDescType">
            <xsd:sequence>
                <xsd:element name="DataValue" type="xsd:string"/>
                <xsd:element name="Location" type="ns1:GeographicLocationType"/>
                <xsd:element name="Intensity" type="xsd:long"/>
                <xsd:element name="Confidence" type="xsd:long"/>
                <xsd:element name="Energy" type="xsd:long"/>
                <xsd:element name="TimeStamp" type="xsd:long"/>
            </xsd:sequence>
            <xsd:attribute name="ID" type="xsd:ID"/>
        </xsd:complexType>
        <xsd:complexType name="InterestDescType">
            <xsd:sequence>
                <xsd:element name="SensorType" type="xsd:string"/>
                <xsd:element name="DataType" type="xsd:string"/>
                <xsd:element name="DataRate" type="xsd:long"/>
                <xsd:element name="Duration" type="xsd:long"/>
                <xsd:element name="AggregationFunction" type="xsd:string"/>
                <xsd:element name="AggregationDelay" type="xsd:float"/>
                <xsd:element name="Area">
                    <xsd:complexType>
                        <xsd:sequence>
                            <xsd:element name="PointA" type="ns1:GeographicLocationType"/>
                            <xsd:element name="PointB" type="ns1:GeographicLocationType"/>
                        </xsd:sequence>
                    </xsd:complexType>
                </xsd:element>
                <xsd:element name="Threshold" type="xsd:float"/>
            </xsd:sequence>
        </xsd:complexType>
        <xsd:complexType name="InjectedFilterType">
            <xsd:sequence>
                <xsd:element name="Interface" type="xsd:string"/>
                <xsd:element name="Program" type="xsd:base64Binary"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:schema>
</types>
<message name="PublishContentMsg">
    <part name="parameter" type="ns:SensorDescType"/>
</message>
<message name="PublishDataMsg">
    <part name="parameter" type="ns:DataDescType"/>
</message>
<message name="SubscribeInterestMsg">
    <part name="parameter" type="ns:InterestDescType"/>
</message>
```

```
    <message name="SubscribeFilterMsg">
        <part name="parameter" type="ns:InjectedFilterType"/>
    </message>
    <portType name="NodePortType">
        <operation name="PublishContent">
         <output message="y:PublishContentMsg"/>
        </operation>
        <operation name="PublishData">
         <output message="y:PublishDataMsg"/>
        </operation>
    </portType>
    <portType name="SinkPortType">
        <operation name="SubscribeInterest">
         <output message="y:SubscribeInterestMsg"/>
        </operation>
        <operation name="SubscribeFilter">
         <output message="y:SubscribeFilterMsg"/>
        </operation>
    </portType>
    <binding name="NodeSOAPBinding" type="y:NodePortType">
        <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
        <operation name="PublishContent">
         <soap:operation soapAction="urn:#PublishContent"/>
         <output>
              <soap:body use="literal"/>
         </output>
        </operation>
        <operation name="PublishData">
         <soap:operation soapAction="urn:#PublishData"/>
         <output>
              <soap:body use="literal"/>
         </output>
        </operation>
    </binding>
    <binding name="SinkSOAPBiding" type="y:SinkPortType">
        <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
        <operation name="SubscribeInterest">
         <soap:operation soapAction="urn:#SubscribeInterest"/>
         <output>
              <soap:body use="literal"/>
         </output>
        </operation>
        <operation name="SubscribeFilter">
         <soap:operation soapAction="urn:#SubscribeFilter"/>
         <output>
              <soap:body use="literal"/>
         </output>
        </operation>
    </binding>
    <service name="NodeService">
        <port name="NodeSOAPPort" binding="y:NodeSOAPBinding">
         <soap:address location="http://bocaiuva/NodeSOAPBinding/"/>
        </port>
    </service>
    <service name="SinkService">
        <port name="SinkSOAPPort" binding="y:SinkSOAPBiding">
         <soap:address location="http://bocaiuva/SinkSOAPBinding/"/>
        </port>
    </service>
</definitions>
```

Anexo 3: Documento WSDL para desenvolvimento de novos serviços.

```
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/" xmlns:y="http://namespace.example.com" xmlns:ns="empty"
targetNamespace="http://namespace.example.com">
    <types>
        <xsd:schema targetNamespace="empty" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
```

```
      xmlns:ns1="empty" elementFormDefault="qualified">
      <xsd:complexType name="ResultDataDescType">
          <xsd:sequence>
              <xsd:element name="DataValue" type="xsd:string"/>
              <xsd:element name="Confidence" type="xsd:long"/>
              <xsd:element name="TimeStamp" type="xsd:long"/>
          </xsd:sequence>
          <xsd:attribute name="IDService" type="xsd:ID"/>
      </xsd:complexType>
      <xsd:complexType name="DataDescType">
          <xsd:sequence>
              <xsd:element name="DataValue" type="xsd:string"/>
              <xsd:element name="Location" type="ns1:GeographicLocationType"/>
              <xsd:element name="Intensity" type="xsd:long"/>
              <xsd:element name="Confidence" type="xsd:long"/>
              <xsd:element name="Energy" type="xsd:long"/>
              <xsd:element name="TimeStamp" type="xsd:long"/>
          </xsd:sequence>
          <xsd:attribute name="ID" type="xsd:ID"/>
      </xsd:complexType>
      <xsd:complexType name="InterestDescType">
          <xsd:sequence>
              <xsd:element name="SensorType" type="xsd:string"/>
              <xsd:element name="DataType" type="xsd:string"/>
              <xsd:element name="AggregationFunction" type="xsd:string"/>
              <xsd:element name="AggregationDelay" type="xsd:float"/>
          </xsd:sequence>
      </xsd:complexType>
      <xsd:complexType name=" ServiceConfigureType ">
          <xsd:sequence>
              <xsd:element name="SensorType" type="xsd:string"/>
              <xsd:element name="DataType" type="xsd:string"/>
                  <xsd:element name="Parameter" type="xsd:string"/>


          </xsd:sequence>
          <xsd:attribute name="IDService" type="xsd:ID"/>
      </xsd:complexType>
      </xsd:schema>
</types>
<message name=" PublishResultMsg ">
    <part name="parameter" type="ns:ResultDataDescType"/>
</message>
<message name="DataArrivalMsg ">
    <part name="parameter" type="ns:DataDescType"/>
</message>
<message name="SubscribeInterestMsg">
    <part name="parameter" type="ns:InterestDescType"/>
</message>
<message name=" ConfigureMsg ">
    <part name="parameter" type="ns:ServiceConfigureType"/>
</message>

<portType name="SinkPortType">
    <operation name="PublishResult">
        <output message="y:PublishResultMsg"/>
    </operation>
     <operation name="localDataArrival">
        <output message="y:DataArrivalMsg"/>
    </operation>
    <operation name="networkDataArrival">
        <output message="y:DataArrivalMsg"/>
    </operation>
    <operation name="InterestArrival">
        <output message="y:SubscribeInterestMsg"/>
    </operation>
    <operation name=Configure">
        <output message="y:ConfigureMsg"/>
    </operation>
</portType>
```

```
    <binding name="SinkSOAPBiding" type="y:SinkPortType">
        <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
        <operation name="PublishResult">
         <soap:operation soapAction="urn:#PublishResult "/>
         <output>
              <soap:body use="literal"/>
         </output>
        </operation>
        <operation name="localDataArrival ">
         <soap:operation soapAction="urn:#localDataArrival"/>
         <output>
              <soap:body use="literal"/>
         </output>
        </operation>
        <operation name="networkDataArrival ">
         <soap:operation soapAction="urn:#networkDataArrival "/>
         <output>
              <soap:body use="literal"/>
         </output>
        </operation>

        <operation name="InterestArrival">
         <soap:operation soapAction="urn:#InterestArrival"/>
         <output>
              <soap:body use="literal"/>
         </output>
        </operation>
        <operation name="Configure">
         <soap:operation soapAction="urn:#Configure"/>
         <output>
              <soap:body use="literal"/>
         </output>
        </operation>

    </binding>
    <service name="SinkService">
        <port name="SinkSOAPPort" binding="y:SinkSOAPBiding">
         <soap:address location="http://bocaiuva/SinkSOAPBinding/"/>
        </port>
    </service>
</definitions>
```

Anexo 4: Mensagem SOAP anunciando um interesse síncrono com requisitos de QoS (atraso e acurácia).

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" >
        <SOAP-ENV:Body>
          <m:SubscribeSynchInterest xmlns:m="http://namespace.example.com">
              <parameter>
                  <m:SensorType>Temperature</m:SensorType>
                  <m:Area>
                      <m:PointA unit="LatLong">
                          <m:x>35.00</m:x> <m:y>-23.00</m:y>
                      </m:PointA>
                      <m:PointB unit="LatLong">
                          <m:x>35.02</m:x> <m:y>-23.03</m:y>
                      </m:PointB>
                  </m:Area>
                  <m:Accuracy>0.1</m:Accuracy>
                  <m:MaxDelay unit="mSeconds">0.5</m:MaxDelay>
                  <m:Constraint>
                      <m:value>35.00</m:value>
                      <m:operation>GT</m:operation>
                  </m:Constraint>
              </parameter>
          </m:SubscribeSynchInterest>
        </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Anexo 5: Mensagem SOAP anunciando um interesse assíncrono do tipo "consulta de longa duração".

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" >
       <SOAP-ENV:Body>
         <m:SubscribeLongRunningQuery xmlns:m="http://namespace.example.com">
            <parameter>
               <m:SensorType>Temperature</m:SensorType>
               <m:DataRate unit="mSeconds">20</m:DataRate>
               <m:Duration unit="Seconds">20</m:Duration>
               <m:Accuracy>0.1</m:Accuracy>
               <m:Area>
                  <m:PointA unit="LatLong">
                     <m:x>35.00</m:x> <m:y>-23.00</m:y>
                  </m:PointA>
                  <m:PointB unit="LatLong">
                     <m:x>35.02</m:x> <m:y>-23.03</m:y>
                  </m:PointB>
               </m:Area>
               <m:Constraint>
                  <m:value>35.00</m:value>
                  <m:operation>GT</m:operation>
               </m:Constraint>
            </parameter>
         </m:SubscribeLongRunningQuery>
       </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Anexo 6: Mensagem SOAP anunciando um interesse assíncrono do tipo "detecção de evento".

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" >
       <SOAP-ENV:Body>
         <m: SubscribeEventdrivenInterest xmlns:m="http://namespace.example.com">
            <parameter>
               <m:SensorType>Temperature</m:SensorType>
               <m:EventType>Four-legged animal</m: EventType>
               <m:Accuracy>0.1</m:Accuracy>
               <m:Area>
                  <m:PointA unit="LatLong">
                     <m:x>35.00</m:x> <m:y>-23.00</m:y>
                  </m:PointA>
                  <m:PointB unit="LatLong">
                     <m:x>35.02</m:x> <m:y>-23.03</m:y>
                  </m:PointB>
               </m:Area>
            </parameter>
         </m: SubscribeEventdrivenInterest >
       </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Anexo 7: Mensagem SOAP anunciando um interesse de ativação de tarefa (operação `Subscribe_Triggering_Interest`).

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" >
       <SOAP-ENV:Body>
         <m:InterestTrig xmlns:m="http://namespace.example.com">
            <parameter>
               <m:Area>
                  <m:PointA unit="LatLong">
                     <m:x>35.00</m:x> <m:y>-23.00</m:y>
                  </m:PointA>
                  <m:PointB unit="LatLong">
                     <m:x>35.02</m:x> <m:y>-23.03</m:y>
                  </m:PointB>
               </m:Area>
               <m:mainSensor>
                  <m:SensorType>Motion</m:SensorType>
                  <m:DataRate unit="mSeconds">40</m:DataRate>
```

```
                </m:mainSensor>
                <m:DataType>Four Legged Animal</m:DataRate>
                <m:secundarySensor>
                    <m:SensorType>Imagery</m:SensorType>
                </m:secundarySensor>
            </parameter>
        </m:InterestTrig>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Anexo 8: Mensagem SOAP `Request_Inspection_In` (mensagem de entrada para a operação `Request_Inspection`).

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:m0="empty">
    <SOAP-ENV:Body>
        <m: Request_Inspection_In xmlns:m="http://namespace.example.com">
            <m0:parameter>max_delay</m0:parameter>
            <m0:parameter>min_accuracy</m0:parameter>
            <m0:parameter>residual_energy</m0:parameter>
            <m0:parameter> priorityQoS </m0:parameter>
            <m0:parameter> execPolicy</m0:parameter>
     </m: Request_Inspection_In>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Anexo 9: Mensagem SOAP `Request_Inspection_Out` (mensagem de saída para a operação `Request_Inspection`).

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:m0="empty">
    <SOAP-ENV:Body>
        <m: Request_Inspection_Out xmlns:m="http://namespace.example.com">
            <m0:max_delay>0.5</m0:max_delay>
            <m0:min_accuracy>0.8</m0:min_accuracy>
            <m0:residual_energy>2300</m0:residual_energy>
            <m0:priorityQoS>delay</m0:priorityQoS>
            </m0:execPolicy id ="1">
                <context id ="1">
                    <network>
                        <param type="average energy" operator="gt" value="5"/>
                        <param type="bandwidth" operator="gt" value="100"/>
                    </network>
                    <application>
                        <param type="temperature" operator="lt" value="30"/>
                    </application>
                </context>
                <qos>
                    <param type="delay" operator="lt" value="5"/>
                    <param type="data rate" operator="gt" value="10"/>
                </qos>
            </m0:execPolicy>
     </m: Request_Inspection_Out>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Anexo 10: Mensagem SOAP solicitando ativação da política de adaptação para aumentar a confiabilidade.

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:m0="empty">
    <SOAP-ENV:Body>
        <m:Request_Adaptation xmlns:m="http://namespace.example.com">
            <m0:increase_reliability>
              <m0:parameter>min_accuracy</m0:parameter>
              <m0:value>95</m0:parameter>
            </m0:increase_reliability>
     </m:Request_Adaptation>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Anexo 11: Mensagem SOAP solicitando ativação da política de adaptação para diminuir a latência.

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:m0="empty">
    <SOAP-ENV:Body>
        <m:Request_Adaptation xmlns:m="http://namespace.example.com">
            <m0:decrease_delay>
            <m0:parameter>max_delay</m0:parameter>
            <m0:value>0.2</m0:parameter>
            </m0:decrease_delay>
     </m:Request_Adaptation>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```