Middleware para Redes de Sensores Sem Fio*

Antonio A.F. Loureiro, Linnyer Beatrys Ruiz, Fernanda Paixão Franciscani, Rainer Ronnie Pereira Couto, José Marcos S. Nogueira

Departamento de Ciência da Computação
Universidade Federal de Minas Gerais
Belo Horizonte, Minas Gerais

{loureiro, linnyer, fepaixao, rainerpc, jmarcos}@dcc.ufmg.br

Resumo

Na última década, houve um grande avanço tecnológico nas áreas de sensores, circuitos integrados e comunicação sem fio, que levou a criação das redes de sensores sem fio. As novas tecnologias de fabricação e integração reduziram o custo e o tamanho dos nodos sensores que formam estas redes.

As redes de sensores definem novos conceitos e problemas. Alguns, tais como gerenciamento de localização e energia, são fundamentais, independentes da aplicação. Muitas características das redes de sensores, tais como auto-organização, localização, mecanismos de endereçamento e serviços de *binding*, coleta de dados envolvendo problemas de cobertura de área e exposição, topologia dinâmica, fusão de dados, arquitetura da aplicação, mecanismos de segurança e tráfego são desafios em relação aos sistemas distribuídos tradicionais, mas também representam novas oportunidades de pesquisa.

Um *middleware* para RSSFs precisam ser leves, eficientes em energia, prover comunicação assíncrona entre os componentes, prover informações sobre o contexto, permitir escalabilidade, robustez, tratar do tempo e da localização.

Neste tutorial, apresentamos o objetivo, a funcionalidade e as características de um *middleware* construído para redes de sensores sem fio. Também abordamos o uso de mecanismos de tolerância a falhas neste tipo de rede. Com este intuito, na primeira parte do trabalho fazemos uma revisão dos tipos de sistemas distribuídos e os requisitos para construção de um *middleware* para cada um deles. Na segunda parte, apresentamos as redes de sensores sem fio e discutimos o uso de *middleware* e mecanismos de tolerância falhas em tais redes.

*Este trabalho foi financiado parcialmente pelo CNPq.

1 Introdução

Um *middleware* é uma camada de software entre o sistema operacional – incluindo os protocolos de comunicação –, e aplicações distribuídas que interagem entre si através de um suporte à comunicação. *Middleware* é um termo amplamente utilizado para denotar um conjunto de serviços genéricos. De uma forma geral, um *middleware* é uma classe de tecnologias projetada para ajudar a gerenciar a complexidade e heterogeneidade inerentes aos sistemas distribuídos e redes. Pode-se dizer que um *middleware* provê uma abstração de programação em um sistema distribuído. Ao fazer isto, um *middleware* provê aos programadores componentes básicos de mais alto nível que soquetes e outras APIs (*Application Programming Interfaces*), providos pelo sistema operacional. Ou seja, um *middleware* tem como objetivo facilitar a interação entre módulos de software distribuídos. Isto reduz significativamente a carga sobre os programadores de aplicações além de diminuir a chance de erros de programação. Algumas vezes, um *middleware* é informalmente chamado de "encanamento" porque conecta aplicações distribuídas através de canais de comunicação para troca de dados. Dependendo do tipo de aplicação, diferentes componentes e funcionalidades estão presentes num *middleware*.

Um *middleware* pode ser visto como o software que torna um sistema distribuído mais facilmente programável. Assim como é mais difícil programar um computador sem um sistema operacional, também é mais difícil programar um sistema distribuído sem um *middleware*, especialmente quando uma operação que envolve sistemas heterogêneos é necessária [1].

Um *middleware* é projetado para tornar transparente para as aplicações diferentes tipos de heterogeneidade que programadores de sistemas distribuídos devem lidar, como, por exemplo, a heterogeneidade das plataformas computacionais (arquiteturas de computadores, sistemas operacionais, e linguagens de programação) e tecnologias de comunicação de dados. O projeto de um *middleware* também inclui aspectos relacionados com o gerenciamento da qualidade de serviço (QoS), segurança da informação e abstrações de programação para prover transparência com relação à distribuição e heterogeneidade do dado e processamento em relação a aspectos como concorrência, replicação, falhas, mobilidade e localização de entidades usuárias e provedoras de serviços.

Como qualquer outro sistema, o projeto de um *middleware* deve considerar as características do sistema distribuído para o qual está sendo desenvolvido. Neste tutorial, tratamos de questões relacionadas de um *middleware* para RSSFs – Redes de Sensores Sem Fio. Uma RSSF é formada por um conjunto de dispositivos compactos e autônomos, chamados nodos sensores. Os nodos da rede de sensores são distribuídos por uma área, despertam, realizam auto-teste, descobrem sua localização, se organizam, comunicam entre si formando uma rede sem fio *ad hoc* e interagem com o ambiente com o objetivo de coletar dados. Uma rede *ad hoc* é uma rede onde os nodos comunicam diretamente entre si, não dependendo de uma infra-estrutura fixa de comunicação. Os nodos de uma RSSF podem processar localmente os dados coletados e/ou enviar esses dados a um ou mais pontos de acesso da RSSF com uma outra rede, como a Internet. É possível também haver nodos atuadores na rede para controlar variáveis do ambiente.

As redes de sensores sem fio são um tipo especial de sistema distribuído *ad hoc* onde, geralmente, os nodos executam uma tarefa colaborativa e, todo o projeto da rede é guiado fundamentalmente pela aplicação de sensoriamento. As RSSFs podem ser formadas por centenas a milhares de nodos sensores e projetadas para executar suas tarefas sem intervenção humana, isto é, após a deposição dos nodos no ambiente de interesse, a rede não é assistida por operadores.

As redes de sensores possuem outras características que as diferenciam das demais redes *ad hoc*. Em geral, os nodos são descartáveis e a fonte de energia não é recarregável [2]. As falhas não são exceções mas acontecimentos freqüentes já que a energia é limitada e nodos podem ser destruídos. A conectividade também pode não ser permanente em razão das falhas nos nodos, da natureza *ad hoc* da rede e das condições ambientais. Para que o sistema continue operando, mesmo de forma degradada, é necessário empregar mecanismos de tolerância a falhas.

Neste tutorial apresentamos uma revisão sobre sistemas distribuídos e *middleware* e discutimos o objetivo, a funcionalidade e as características de um *middleware* para uma RSSF. Também discutimos a implementação de mecanismos de tolerância a falhas neste tipo de rede. O texto está organizado como segue. A seção 2 trata dos tipos de sistemas distribuídos e seus requisitos não funcionais. Na seção 3 apresentamos alguns projetos de *middleware* desenvolvidos para sistemas distribuídos tradicionais e na seção 4 para ambientes móveis. Na seção 5 apresentamos uma visão geral de rede de sensores e como um *middleware* pode auxiliar na sua configuração, manutenção e comunicação em tais redes. A seção 6 discute esquemas de tolerância a falhas para RSSFs. Finalmente, a seção 7 apresenta algumas conclusões sobre este tutorial.

2 Sistemas Distribuídos

Um sistema distribuído é composto por uma coleção de componentes distribuídos sobre um conjunto de computadores (denominados *hosts*), os quais estão conectados por uma rede de comunicação de dados. Os componentes interagem entre si com o objetivo de trocar informação e/ou utilizar serviços disponíveis no sistema. O objetivo primordial de um *middleware* é facilitar essa interação.

Algumas das diferenças entre um sistema distribuído para ambientes fixos e móveis são os dispositivos, conexões e contexto de execução.

- ***Tipos de dispositivos***: dispositivos para uma rede tradicional são normalmente fixos, ou seja, suas localizações não variam ao longo do tempo. Alguns dos dispositivos usados nessas redes são computadores pessoais (PCs), estações de trabalho e *mainframes*. Dispositivos típicos na rede móvel, como PDAs – Personal Digital Assistants, celulares e *smartcards*, normalmente são pequenos, leves, portáteis, usam energia de bateria e possuem uma interface de comunicação sem fio. Todas estas características permitem a mobilidade desses dispositivos. Outra diferença importante é o poder computacional típico dos dispositivos computacionais das redes fixa e móvel: os primeiros possuem uma capacidade de processamento, memória, armazenamento, e periféricos bem maior que os segundos, além da restrição da energia dos dispositivos móveis.

- ***Tipos de conexões***: apesar da conexão contínua (permanente) não ser uma realidade em nenhum dos dois ambientes, as quebras de conexões são mais freqüentes no ambiente móvel. No ambiente fixo as eventuais quebras de conexão são ocasionadas por falhas imprevistas ou por razões administrativas. No ambiente móvel, entretanto, quebras de conexão não são exceções; ao contrário, fazem parte do funcionamento normal do sistema. Um usuário móvel pode, a qualquer instante, sair do alcance de comunicação ou experimentar uma queda repentina na largura de banda. Assim, conexões em ambiente móveis são ditas *intermitentes*.

- ***Tipos de contexto de execução***: denomina-se contexto o conjunto de fatores que podem influenciar o comportamento da aplicação, o que inclui tanto características internas do dispositivo quanto do ambiente externo. Em ambiente fixos diz-se que o contexto é estático, já que alterações como mudanças de localização e variação de largura de banda são infreqüentes. No extremo oposto, temos o contexto dinâmico dos ambiente móveis: dispositivos são desconectados freqüentemente, localizações não são constantes e tanto a largura de banda de transmissão quanto a qualidade do sinal podem sofrer alterações abruptas.

Os sistemas distribuídos podem ser divididos em tradicionais, textslad hoc e nômades, cada um com suas necessidades próprias. A seguir, são apresentados a definição e os requisitos básicos para a construção de tais sistemas distribuídos.

2.1 Sistemas Distribuídos Tradicionais

Sistemas distribuídos tradicionais, como mostrados na figura 1, são conjuntos de *hosts* fixos, conectados permanentemente através de canais de comunicação estáticos. Além de uma rede fixa com uma alta largura de banda e disponibilidade (quebras de conexões são eventos raros), assume-se que os *hosts* possuem processadores rápidos e uma grande quantidade de memória.

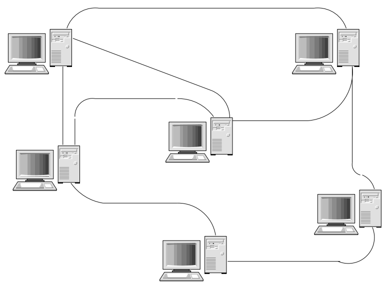

Figura 1: Exemplo de um sistema distribuído tradicional.

Ao projetar aplicações para este tipo de sistema, o projetista deve lidar com os seguintes requisitos não-funcionais:

- ***Escalabilidade***, ou a habilidade do sistema suportar uma carga crescente de trabalho. Em sistemas centralizados (cliente-servidor) a escalabiliade do sistema é ligada diretamente à carga suportada pelo servidor. Sistemas distribuídos podem obter melhor escalabilidade através da distribuição de carga entre diversos *hosts* ou através da replicação de dados. Nas duas técnicas, a escalabilidade é obtida através da disponibilização de cópias de serviços ou de dados. Nesse caso, o desafio é atingir o grau de transparência necessário para que alterações na arquitetura não exijam mudanças radicais no código e nos componentes do sistema.

- ***Extensibilidade***, ou seja, facilidade de inserção de novos módulos ou propriedades no sistema. Em geral, um sistema distribuído sofre alterações durante o seu tempo de vida, seja através da inserção de plataformas mais novas e rápidas ou através da remoção de componentes obsoletos, o que demanda uma forma simples e eficiente de se alterar o sistema.

- ***Heterogeneidade***. Sistemas distribuídos devem lidar com uma diferentes sistemas operacionais, hardware e linguagens de programação. Homogeneidade em sistemas distribuídos pode resultar em uma maior susceptibilidade a falhas e ataques. Um sistema heterogêneo atenua esse fato, além de proporcionar uma melhor escalabilidade, pois tem-se a liberdade de utilizar componentes diferentes para serviços críticos.

- ***Compartilhamento de recursos***. Alguns dispositivos do sistema distribuído são compartilhados entre diversos usuários e aplicações. Alguma forma de gerenciamente deve ser estabelecida para garantir o acesso seguro a esses elementos.

- ***Tolerância a falhas***. Um sistema distribuído deve ser capaz de tratar falhas isoladas sem que isto comprometa o seu funcionamento global. Existem três tipos de falhas: hardware, software e sistema – cada uma exige um tratamento adequado. Falhas de sistema assumem uma importância maior em sistemas distribuídos devido à multiplicidade de componentes e *hosts*. Este assunto será discutido na seção 6.

2.2 Sistemas Distribuídos *Ad hoc*

Um sistema distribuído *ad hoc* é composto por um conjunto de elementos computacionais móveis, conectados através de uma rede sem fio intermitente, cujos canais de comunicação apresentam qualidade váriavel (figura 2). Em redes móveis *ad hoc*, ou MANETs – Mobile Ad hoc Networks –, a comunicação é ponto-a-ponto, o que significa dizer que os elementos computacionais atuam como clientes, servidores e roteadores de mensagens. A diferença básica para um ambiente tradicional é a falta de uma infra-estrutura fixa, ou seja, os *hosts* possuem total liberdade de movimentação. Essa dinâmica permite entrada e saída de nodos na rede de forma simples, porém exige a criação de algoritmos de roteamentos complexos e caros.

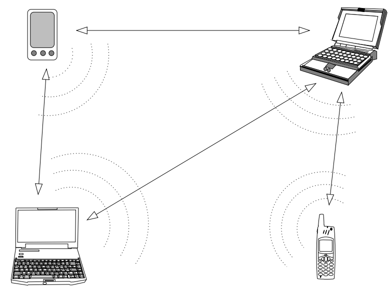

Figura 2: Exemplo de um sistema distribuído *ad hoc*.

Os requisitos não-funcionais para sistemas distribuídos tradicionais ainda são válidos no ambiente móvel *ad hoc*. A escalabilidade se torna um fator crítico: o canal de comunicação é compartilhado entre vários *hosts* e não é confiável. Além disso, os avanços da tecnologia irão permitir uma presença massiva de elementos computacionais móveis, o que agrava ainda mais o problema da escalabilidade. Em ambientes móveis *ad hoc*, a variedade de dispositivos e modos de comunicação é ainda maior que em sistemas tradicionais. A tolerância a falhas é outro fator importante, uma vez que as desconexões e erros na comunicação são eventos comuns em tais redes. O compartilhamento de recursos exige uma atenção especial, pois os canais sem fio são inerentemente inseguros.

Um tipo especial de sistema distribuído *ad hoc* é a rede de sensores sem fio. Uma RSSF tende a ser autônoma e requer um alto grau de cooperação para executar as suas tarefas. Os nodos combinam tecnologia de micro-sensores, computação de baixa potência e comunicação sem fio em um sistema compacto [3]. Um nodo sensor é composto de memória, bateria, processador, transceptor (inclui todo o sistema de transmissão e recepção, óptico ou RF – rádio freqüência) e dispositivo de sensoriamento.

As redes *ad hoc* têm como função básica prover um suporte à comunicação entre elementos computacionais, que individualmente, podem estar executando tarefas distintas. As redes de sensores sem fio tendem a executar uma função colaborativa onde os nodos sensores provêem dados, que são processados por pontos de acesso chamados de nodos sorvedouros (*sink nodes*) ou nodos de monitoração (*monitoring nodes*) ou por estações base. As redes de sensores sem fio compartilham muitos dos desafios encontrados em sistemas distribuídos sem fio *ad hoc* tradicionais. Isto inclui a limitação de energia para cada nodo, a restrição de largura de banda e a utilização de canais, que geralmente, apresentam uma elevada taxa de erros. Contudo, as redes de sensores são formadas por uma quantidade expressiva de nodos que são projetados para operações tipicamente desacompanhadas, apresentam limitação de todos os seus recursos (processador, memória, transceptor, bateria), a fonte de energia não é recarregável e não apresentam uma comunicação tipicamente fim-a-fim, isto é, em geral os nodos não têm identidade (endereço individual). Os dados são nomeados pelos atributos e as aplicações requisitam dados correspondentes a certos atributos, o que torna a arquitetura das redes de sensores sem fio diferente das redes *ad hoc*. Como mencionado, as falhas em rede de sensores sem fio não são exceções, pois um nodo pode deixar a rede por diferentes problemas, principalmente problemas de energia.

Assim, a topologia da rede é dinâmica mesmo sendo os nodos estacionários. As redes de sensores sem fio serão discutidas nas seção 5.

Pelos argumentos apresentados acima, pode-se concluir que um *middleware* projetado para um ambiente tradicional não atende aos requisitos do ambiente móvel *ad hoc*. Logo, é necessário projetar um *middleware* que considere essas características.

2.3 Sistemas Distribuídos Nômades

Dá-se o nome nômade aos sistemas distribuídos que possuem uma infra-estrutura fixa (por exemplo, roteadores e *hosts*) e com elementos terminais dotados de mobilidade. Estes últimos se conectam à infra-estrutura através de estações-base dotadas de interfaces para redes fixas e sem fio. Conceitualmente, são sistemas intermediários entre as redes fixas tradicionais e redes móveis *ad hoc*, e exigem tratamento diferenciado das soluções apresentadas para esses ambientes.

3 *Middleware* para Sistemas Distribuídos Fixos

Um *middleware* para um sistema distribuído pode ser classificado em três categorias principais, de acordo com Capra *et al.* [4]: *middleware* orientado a objetos, *middleware* orientado a mensagens e *middleware* orientado a transações.

3.1 *Middleware* Orientado a Objetos e Baseado em Componentes

Um *middleware* orientado a objetos provê comunicação entre objetos distribuídos, ou seja, um objeto-cliente requisita a execução de uma operação a um objeto-servidor que pode residir em outro *host*. Esta classe de *middleware* evoluiu da Chamada Remota de Procedimento (*RPC –Remote Procedure Calls* [5]), onde a forma básica de interação é síncrona, ou seja, o objeto-cliente que faz o pedido é bloqueado até que o objeto-servidor tenha retornado a resposta. Esta abordagem tem a vantagem de se assemelhar bastante à semântica da chamada local de procedimentos, mas é especialmente sensível a problemas de alta latência na rede. Produtos nesta categoria incluem implementações do OMG CORBA [6], como o IONA Orbix [7] e o Bordland VisiBroker [8], o Modelo de Componentes CORBA (CCM – *CORBA Component Model* [9]), o Microsoft COM [10], Java/RMI [11] e o *Enterprise JavaBeans* [12].

3.2 *Middleware* Orientado a Mensagens

Um *middleware* orientado a mensagens provê a comunicação entre componentes distribuídos através do intercâmbio de mensagens. Um componente-cliente envia uma mensagem através da rede contendo a requisição da execução de um serviço e seus parâmetros a um componente-servidor, que pode responder com uma mensagem-resposta contendo o resultado da execução do serviço. Um *middleware* orientado a mensagens provê comunicação assíncrona de uma forma bem natural, alcançando desvinculação entre cliente e servidor, como é requisitado por sistemas móveis: o cliente está apto a continuar seu processamento assim que o *middleware* tenha aceitado sua mensagem; o servidor acaba por enviar uma mensagem de resposta e o cliente estará apto a coletá-la em um momento conveniente. A aplicação não tem conhecimento do gerenciamento das mensagens, que é feito de forma totalmente transparente pelo *middleware*.

3.3 *Middleware* Orientado a Transações

Um *middleware* orientado a transações é usado principalmente em arquiteturas em que os componentes são aplicações de banco de dados. Este *middleware* provê transações envolvendo componentes que executam em *hosts* distribuídos. Neste caso, um componente-cliente agrupa várias operações em uma transação que o *middleware* então transporta através da rede para cada componente-servidor de forma transparente tanto para os clientes quanto para os servidores. Este *middleware* provê tanto comunicação síncrona quanto assíncrona entre *hosts* heterogêneos e oferece alta confiabilidade, garantindo a atomicidade das transações desde que os servidores participantes implementem o protocolo *two-phase-commit*. Entretanto, esta solução causa uma sobrecarga se não há a necessidade de se usar transações, o que acontece em vários cenários em ambientes móveis.

4 *Middleware* para Sistemas Móveis Distribuídos

Um *middleware* para sistemas móveis, tanto *ad hoc* quanto nômade, precisa ser leve, prover comunicação assíncrona entre componentes e permitir que os projetistas de aplicações tenham conhecimento sobre o ambiente de execução, como explicado a seguir:

- ***Carga computacional baixa***. Aplicações móveis são executadas em dispositivos com recursos escassos como pouca memória, baixa velocidade do processador e energia limitada. Desta forma, torna-se necessário definir um compromisso entre a carga computacional e os requisitos não-funcionais alcançados pelo *middleware*, pois não é factível executar um *middleware* pesado em tais dispositivos.

- ***Prover comunicação assíncrona***. Os dispositivos móveis tipicamente apresentam conexões intermitentes com a rede. Tipicamente os usuários destes dispositivos se conectam à rede por curtos períodos de tempo, e ainda há os casos da perda de conexão devido à entrada em uma área sem cobertura. Assim, é freqüente o cenário onde o cliente pede um serviço e o servidor provê o serviço mas ambos não estejam conectados ao mesmo tempo, tornando necessária uma forma de comunicação assíncrona.

- ***Prover informações sobre o contexto***. Diferentemente dos sistemas fixos, os sistemas móveis executam em contextos muito dinâmicos. Esta dinamicidade torna inviável que o projetista da aplicação antecipe todos os contextos possíveis de execução e instrua (a priori) o *middleware* sobre como agir em cada situação. Uma vez que não existe um conhecimento completo que possa ser explorado, o *middleware* não pode tomar decisões de forma transparente para a aplicação e, ao mesmo tempo, garantir a melhor qualidade de serviço. Ao contrário, o *middleware* deve interagir com a aplicação, fazendo-a ciente das mudanças no contexto de execução e ajustando dinamicamente seu próprio comportamento ao usar a informação que a aplicação lhe retorna. Projetar a aplicação ciente do contexto de execução torna mais complexa a tarefa dos programadores de aplicações. Assim é importante a definição de interfaces simples para que as aplicações possam interagir dinamicamente com o *middleware*.

Na seção 3 vimos que um *middleware* para um sistema fixo pode ser classificado em três categorias principais. De acordo com os requisitos apresentados acima, esses tipos apresentam inadequações com relação aos ambientes móveis, como discutido a seguir.

O *middleware* orientado a objetos tem aplicabilidade limitada em ambientes móveis devido à alta carga computacional requerida pela execução destes sistemas, pela forma síncrona de requisição de objetos e pelo princípio de transparência que guiou seu projeto, que impede qualquer tipo de conhecimento sobre o contexto.

O *middleware* orientado a mensagens, por sua vez, requer dispositivos com abundância de recursos, especialmente em termos de quantidade de memória para armazenar as filas de mensagens recebidas mas não processadas ainda. Esta é uma limitação que impede a utilização de produtos como Sun Java Message Queue [13] e IBM MQSeries [14] em dispositivos portáteis. Além disso, não é provida nenhuma forma de conhecimento sobre o ambiente, isto é, as mensagens são manipuladas transparentemente pelo *middleware*, sem nenhum conhecimento por parte da aplicação.

Por fim, tem-se o *middleware* orientado a transações. Nele o protocolo *two-phase-commit*, utilizado para garantir a atomicidade das transações, representa uma sobrecarga se não for necessária a utilização de transações, como é o caso típico de aplicações para ambientes móveis. Novamente, a carga computacional e a transparência deste tipo de *middleware* nos impede de usar produtos como o IBM CICS [15] e o BEA Tuxedo [16] em ambientes móveis, apesar do sucesso de seu uso em ambientes fixos.

Assim, verifica-se que um *middleware* desenvolvido para um sistema distribuído tradicional não pode ser adotado com sucesso em ambientes móveis, uma vez que eles geram uma carga computacional alta, e/ou provêem um paradigma de comunicação síncrona, e/ou foram projetados com base no princípio da transparência. Além disso, devido à grande diferença entre os requisitos dos sistemas fixos e dos sistemas móveis, nem mesmo a adaptação de um *middleware* de sistemas fixos para ambientes móveis se mostra como uma boa solução. Portanto, novos projetos de *middleware* devem ocorrer tendo em vista, os desafios impostos pelas redes móveis (*ad hoc*, redes de sensores sem fio e nômades.

De acordo com Capra *et al.* [4], as soluções existentes de *middleware* para sistemas móveis podem ser agrupadas em três categorias principais: *middleware* reflexivo, *middleware* baseado em espaço de tuplas e *middleware* sensível ao contexto. Aqui propomos uma pequena modificação ao esquema proposto por Capra *et al.* [4]. Assumimos que o terceiro conjunto, *middleware* sensível ao contexto, também envolve o *middleware* desenvolvido para aplicações ou serviços específicos.

A seguir são apresentados alguns exemplos de *middleware* desenvolvidos para ambientes móveis, exemplificando cada uma das vertente descritas acima.

4.1 *Middleware* Reflexivo Orientado a Objetos

O conceito de reflexão foi inicialmente introduzido por Smith [17] como um princípio que permite a um programa ter acesso a sua interpretação, refletir sobre ela e alterá-la. Este conceito é uma ruptura com a filosofia da "caixa preta", em que os detalhes de implementação são escondidos do usuário da plataforma.

A reflexão é, portanto, o meio através do qual um sistema é capaz de inspecionar e alterar seu comportamento durante a execução e de acordo com regras estabelecidas. Basicamente, um sistema reflexivo é capaz de realizar tanto a auto-inspeção quanto a auto-adaptação. Para conseguir isto, um sistema reflexivo possui uma representação de si mesmo. Esta representação é conectada de forma causal ao objeto representado, o domínio. Assim, qualquer mudança no domínio tem um efeito no sistema e vice-versa [18]. Esta conexão causal entre o sistema e sua representação é implementada pela reificação e absorção.

A reificação é a ação de expor a representação interna de um sistema em termos de entidades programáveis que podem ser manipuladas em tempo de execução. Já a absorção mostra-se como um processo inverso, que consiste em refletir as mudanças realizadas nas entidades reificadas no sistema real [19].

Um sistema reflexivo pode ser divido em duas partes: o nível-base e o meta-nível. O primeiro está relacionado com *o que* o sistema faz, ou seja, suas funcionalidades, enquanto que o segundo trata *como* o sistema faz, lidando com o processamento reflexivo.

As entidades presentes no meta-nível são chamadas de meta-objetos e a interface do meta-nível, isto é, o protocolo de interação com os meta-objetos é normalmente chamado de Protocolo de Meta-Objeto (MOP – *Meta-Object Protocol*). O MOP é que define o estilo de reflexão provido pelo sistema.

A seguir, estão descritos alguns projetos de sistemas desta categoria:

- ***OpenCorba*** [20]: é um *broker* aberto e reflexivo, baseado no modelo CORBA (*Common Object Request Broker Architecture*), que permite ao usuário adaptar dinamicamente a representação e as políticas de execução da aplicação. Através do uso de reflexão, OpenCorba distingue o que um objeto faz (classes) de como o objeto faz (meta-classes). Um protocolo específico permite a alteração, em tempo de execução, de propriedades das classes e meta-classes.

  A utilização de uma linguagem reflexiva e o uso de meta-classes facilita a reusabilidade e provê a flexibilidade necessária para ambientes dinâmicos. Entretanto, os aspectos reflexivos desse sistema têm impacto negativo no seu desempenho já que o custo do tratamento de mensagens entre classes e meta-classes e o controle da reflexão tornam o OpenCorba inviável para ambientes móveis, onde recursos como memória e processamento são escassos.

- ***Open ORB*** [21]: é uma plataforma de *middleware* altamente configurável e dinâmica. Porém, ao contrário do OpenCorba, Open ORB já foi projetado para atender requisitos de aplicações que envolvem multimídia e mobilidade. A arquitetura Open ORB permite uma configuração personalizada da plataforma, uma vez que os componentes possuem interfaces bem definidas e são estruturados hierarquicamente (componenentes podem ser compostos por outros componentes). Isso é uma vantagem em ambientes móveis, pois um núcleo reduzido do *middleware* pode ser definido eliminando-se elementos desncessários do sistema, atendendo às restrições físicas de memória dos dispositivos portáteis.

- ***DynamicTAO*** [22]: é uma extensão do ORB Corba TAO e que permite a configuração dinâmica da plataforma ORB e das aplicações que são executadas sobre esse ele. Essa reconfiguração engloba componentes que implementam o controle de concorrência, segurança e monitoramento. DynamicTAO também permite a configuração personalizada do *middleware*, possibilitanto a carga e liberação de módulos da memória em tempo de execução. Para permitir a configuração de uma grande coleção de ORBs distribuídos, DynamicTAO também exporta uma meta-interface destinada à criação de agentes móveis de reconfiguração.

4.2 *Middleware* Baseado em Espaço de Tuplas

Como desconexões são eventos comuns em ambientes móveis, a comunicação necessita, intrisicamente, de certo assincronismo e desvinculação, no sentido de que a computação pode ocorrer mesmo que as partes envolvidas estejam desconectadas. O sincronismo implícito de certas soluções de textslmiddleware dificulta, ou mesmo inviabiliza, seu uso em ambientes móveis.

O paradigma da comunicação baseado em espaços de tuplas foi projetado para prover um assincronismo entre processos comunicantes e pode ser aplicado a ambientes móveis e dinâmicos, como *ad hoc*, mesmo que esse não seja o ambiente para o qual foi projetado. Em Linda, um dos precurssores desse modelo de comunicação em ambientes fixos, há uma memória compartilhada globalmente que atua como um repositório de estruturas denominadas *tuplas*. Tuplas são os elementos básicos de comunicação entre processos e são manipuladas através de primititas como **write**, **read** e **take**. O assincronismo da comunicação é caracterizado pela inserção e leitura de tuplas em momentos diferentes. Isto é possível pois tuplas possuem tempo de vida independente dos processos que as geram, assim escrita e leitura estão desvinculadas.

Possuir conhecimento do contexto é uma necessidade comum em aplicações para ambientes móveis. Sistemas de espaços de tuplas dificultam este processo pois suas primitivas e tipos de dados são extremamente básicos. Outra desvantagem desses sistemas é a sincrozinação: uma tupla replicada pode sofrer alterações quando suas unidades retentoras estão desconectadas, o que torna o processo de reconciliação uma tarefa árdua.

A seguir são descritas duas tecnologias implementadas segundo este paradigma:

- ***Lime*** [23]: divide o espaço de tuplas em diversos sub-espaços, definindo regras baseadas em conectividade para o compartilhamento de tuplas. Cada sub-espaço é associado unicamento a uma unidade móvel, que pode tanto ser um agente móvel (mobilidade lógica) ou um *host* móvel (mobilildade física). O modelo Lime retira os detalhes pertinentes à mobilidade do projeto de aplicações, o que é uma vantagem para os desenvolvedores de software, porém uma desvantagem em sistemas que desejam ter uma visão parcial ou completa do contexto. Para lidar com este problema, Lime define um espaço comum e transiente onde estão disponíveis informação sobre o contexto global e insere dados de localização nas tuplas, o que torna o contexto parcialmente visível. Além disso Lime permite inserir ações em resposta a mudanças de contexto no espaço de tuplas. Apesar disso, Lime não permite a adaptação do comportamento do próprio *middleware*.

- ***Tspaces*** [24]: é um *middleware* desenvolvido pela IBM para prover a comunicação, computação e gerenciamento de dados em dispositivos portáteis. TSpaces é uma união entre espaços de tuplas e banco de dados: enquanto o primeiro provê flexibilidade no modelo de comunicação, o segundo lida com a estabilidade, durabilidade e armazenamento e consulta de dados. Sua implementação em Java possui a vantagem da portabilidade. Em TSpaces há uma distinção entre clientes e servidores: os espaços de tuplas existem somente em servidores. Uma vez criado um espaço de tuplas, o cliente realiza a interação através das funções básicas de leitura e escrita.

  TSpaces se diferencia dos outros sistemas por permitir modificações dinâmicas do comportamento do *middleware* e permitir alteração nos tipos de dados e suas operações. Apesar de adequado ao ambiente nômade, TSpaces não é aplicável às redes *ad hoc* devido à sua diferenciação entre clientes e servidores.

4.3 *Middleware* para Aplicações/Serviços Específicos

Alguns projetos de *middleware* foram criados para aplicações específicas e, geralmente, lidam com um problema único e determinado referente à mobilidade. Em geral, tais projetos possuem uma implementação rígida e são adequados para poucos contextos, ou seja, a falta de flexibilidade é compensada por um ganho na eficiência.

A seguir, estão descritos alguns projetos de sistemas desta categoria:

- ***Nexus*** [25]: provê informações de localização às aplicações que necessitam desses dados. O *middleware* trabalha com GPS para ambientes externos e com sensores para ambientes internos. A localização é utilizada pelas aplicações para fornecer dados personalizados para cada usuário.

- ***Odyssey*** [26]: é um *middleware* desenvolvido para dar suporte às aplicações de acesso a informação em ambientes móveis. O modelo assume que as aplicações são executadas em dispositivos móveis e os dados estão armazenados em servidores remotos, o que o torna ideal para o cenário nômade.

Odissey utiliza um modelo colaborativo de adaptação entre o sistema operacional (elemento que gerencia os recursos) e a aplicação (elemento que determina as políticas de adaptação). Cada aplicação deve fazer um registro no *middleware* para um ou mais recursos monitorados. Assim que uma alteração ocorre em um desses recursos, as aplicações são notificadas e as devidas ações são tomadas.

5 Middleware para Redes de Sensores Sem Fio

Na última década, houve um grande avanço tecnológico nas áreas de sensores, circuitos integrados e comunicação sem fio, que levou a criação das redes de sensores sem fio. As novas tecnologias de fabricação e integração reduziram o custo e o tamanho dos nodos sensores que formam estas redes. A figura 3 apresenta alguns tipos de micro-sensores sem fio resultantes de pesquisas em diversas instituições, como o Smart Dust [27] da *University of California, Berkeley*, WINS (*Wireless Integrated Network Sensors*) [28] da *University of California, Los Angeles* e JPL Sensor Webs [29] do *Jet Propulsion Lab* da NASA.

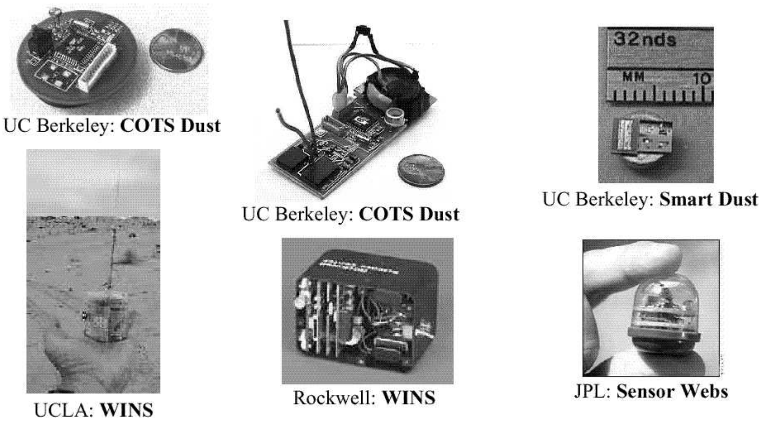

Figura 3: Projetos acadêmicos de nodos sensores.

A redução do tamanho dos nodos sensores restringe a capacidade de seus compontentes (bateria, processador, memória, transceptor, e dispositivo sensor). Já a redução do custo permite que as RSSFs sejam formadas por uma grande quantidade de nodos (centenas a milhares) e utilizadas em diversas aplicações incluindo monitoração ambiental, *smart spaces*, aplicações médicas, monitoração de tráfego, segurança, aplicações militares, monitoração industrial, agricultura de precisão e outras [30, 31, 32, 33, 34]. Esses sistemas prometem revolucionar as aplicações de monitoração, provendo granularidade de dados não realizável por outros meios [2].

Os nodos de uma RSSF são depositados (lançados) em um ambiente, caem, realizam auto-teste, descobrem sua localização, e se organizam formando uma rede *ad hoc*. Em função do custo reduzido dos componentes e do ambiente onde as redes de sensores são depositadas, os nodos não são facilmente acessíveis para a recarga de suas baterias, intervenção humana e até mesmo para recuperação, isto é, os nodos são considerados descartáveis. Neste tipo de rede, falhas não são exceções [35].

Em sistemas distribuídos tradicionais, as conseqüências das falhas, ou seja o colapso do sistema, a interrupção no fornecimento do serviço e a perda de dados, podem ser evitados pelo uso de técnicas de tolerância a falhas. Essas técnicas devem garantir o funcionamento correto do sistema mesmo na ocorrência de falhas. A implementação de mecanismos de tolerância a falhas em qualquer tipo de sistema sugere a redundância de hardware e/ou software com um custo associado. Contudo, sistemas construídos com componentes autônomos, operando em um ambiente remoto e sem intervenção de técnicos e operadores, não conseguem ser confiáveis pela simples aplicação de tolerância a falhas. Este é o caso das redes de sensores sem fio. Embora não se consiga garantir o funcionamento correto deste tipo de rede, os mecanismos de tolerância a falhas podem ser utilizados para que o sistema continue operando dentro de um determinado nível de qualidade de serviço.

Muitas das funções associadas às RSSFs, inclusive funções de tolerância a falhas, podem ser implementadas em um *middleware*. Nas próximas seções, abordamos estes três assuntos: redes de sensores sem fio, *middleware* para RSSFs e a implementação de mecanismos de tolerância a falhas.

5.1 Rede de Sensores Sem Fio

Um nodo sensor é um elemento computacional com capacidade de processamento, memória, interface de comunicação sem fio, além do dispositivo sensor, como mostrado na figura 4. Os nodos são distribuídos, têm restrições de energia, e devem possuir mecanismos para auto-configuração e adaptação devido a problemas como falhas de comunicação, de energia e variações nas condições ambientais. Cada nodo de uma RSSF pode ser equipado com um ou mais dispositivos sensores, tais como acústico, sísmico, infravermelho, vídeo-câmera, estresse mecânico, calor, temperatura, radiação e pressão.

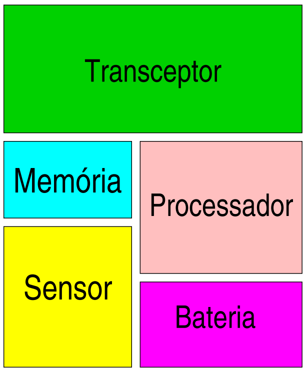

Figura 4: Hardware básico de um nodo sensor

O fluxo de informação em uma RSSF é geralmente unidirecional, isto é, os nodos disseminam suas informações na rede sempre em direção ao ponto de acesso. O esquema de comunicação entre o nodo fonte e o ponto de acesso é normalmente *multi-hop*, como mostrado na figura 5, a fim de economizar energia.

O ponto de acesso representa o elemento computacional que processa e armazena a informação coletada pela rede. Este é o ponto onde usuários e aplicações podem ter acesso aos dados coletados pelos nodos sensores.

As RSSFs podem ser planas ou hierárquicas, dentre vários outros critérios de classificação. Nas RSSFs planas não existe agrupamento de nodos e nas redes hierárquicas os nodos se organizam em grupos e elegem um líder que recebe os dados coletados pelos nodos membros e os envia ao ponto de acesso usando comunicação *single hop*, em redes homogêneas, ou *multi hop* em redes heterogêneas.

Nas redes plans, os pontos de acesso são, em geral, nodos com maior capacidade de hardware chamados de nodos sorvedouros (*sink nodes*) ou nodos de monitoração (*monitoring nodes*). No caso das redes hierárquicas, o ponto de acesso é uma estação base [36]. Ao contrário do nodo sorvedouro, assume-se que a estação base não têm limitação de energia e capacidade computacional. Os pontos de acesso, sejam eles estações base ou nodos sorvedouros, enviam as informações coletadas pela rede aos observadores podendo utilizar nodos *gateway* para esta finalidade [37, 38, 39].

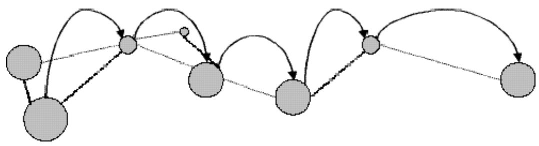

Figura 5: Transmissão Multi hop em RSSFs

O observador é uma entidade (por exemplo, usuário, outra rede ou aplicação) que tem interesse nos dados coletados e informações disseminadas pelo conjunto de sensores que compõe a rede. De acordo com o tipo da aplicação, o observador pode enviar solicitações (consultas), informando seus interesses, e receber respostas geradas por um conjunto total ou parcial dos nodos. Quando apenas alguns nodos participam da resolução de uma consulta, diz-se fidelidade.

O serviço de coleta de dados (monitoração) pode ser contínuo, periódico, realizado a partir da ocorrência de eventos ou a partir de consultas. Depois da coleta, os nodos podem processar os dados e disseminá-los ou não na rede. O serviço de disseminação de dados pode ser diferente do serviço de coleta, isto é, os nodos podem coletar dados continuamente e disseminá-los em intervalos programados. Portanto, uma RSSF é projetada a partir dos objetivos da aplicação para a qual se destina. A partir da definição das características da aplicação e do tipo de ambiente onde a RSSF irá atuar é que se define o tipo de composição (homogênea ou heterogênea), organização (plana ou hierárquica), tipo de coleta (periódica, contínua, reativa, tempo real), tipo de disseminação da informação (contínua, programada, dirigida a eventos, sob demanda) e funções de manutenção da rede [36].

O modo básico de operação de uma RSSFs é significativamente diferente das redes de computadores e sistemas distribuídos tradicionais e *ad hoc*, devido a sua integração com o mundo físico e também porque a rede opera sem intervenção humana direta e na maioria das aplicações, os nodos realizam a mesma tarefa e colaboram entre si com um objetivo comum.

Uma RSSF tende a ser autônoma e requer um alto grau de cooperação para executar as tarefas definidas para a rede. Isto significa que mecanismos de tolerância a falhas e algoritmos distribuídos tradicionais, como protocolos de comunicação e eleição de líder, devem ser revistos para esse tipo de ambiente antes de serem usados diretamente. O mesmo acontece para o projeto do *middleware*. Na próxima seção, discutimos o projeto do *middleware* para RSSFs.

5.2 O Uso de *Middleware* em Redes de Sensores Sem Fio

O objetivo de uma RSSF é monitorar e, eventualmente controlar um ambiente. O projeto de uma RSSF é dependente da aplicação e das características do ambiente onde essas redes irão trabalhar.

Atualmente, grande parte das pesquisas sobre RSSFs tratam do desenvolvimento de algoritmos e funções que são projetados especificamente para um tipo de rede. Em sua maioria, essas funções e algoritmos são testados usando simuladores e executam diretamente sobre o hardware. A identificação e implementação de primitivas de sistema operacional para este tipo de rede ainda é uma área inicial de pesquisa. Um sistema operacional utilizado na família de nodos sensores Motes é o TinyOS [40]. Trata-se de um sistema operacional (escalonador de eventos) bastante simples. Seu objetivo é reduzir a memória ocupada e o custo do sistema. Suas principais características são: manipulação direta do hardware; não existência de gerenciamento de processos, um processo sendo executado por vez; e inexistência de suporte para memória virtual e alocação dinâmica de memória.

A figura 6 apresenta a arquitetura do TinyOS. Um *middleware* poderia ser construído tanto entre a aplicação e o TinyOS quanto entre a aplicação e o próprio hardware.

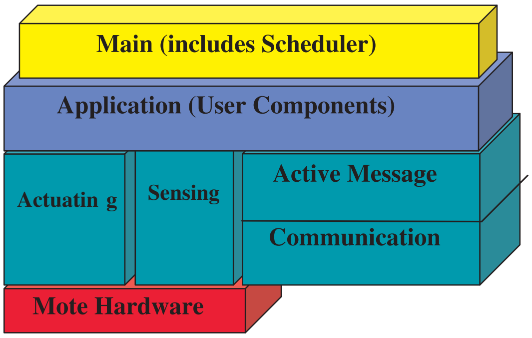

Figura 6: Arquitetura simplificada do TinyOS.

Todos os mecanismos providos por um *middleware* devem considerar os requisitos de sistemas móveis (ver seção 4) e respeitar as características das RSSFs quais sejam, eficiência em energia, robustez e escalabilidade.

Outra propriedade de um *middleware* para RSSFs é imposta pelo princípio do conhecimento da aplicação nos nodos. O *middleware* tradicional é projetado para acomodar uma grande variedade de aplicações sem necessariamente precisar de conhecimento de aplicação. Porém, o *middleware* para uma RSSF deve prover mecanismos para incorporar o conhecimento de aplicação na infra-estrutura e na RSSF [41].

***Middleware* para Configuração dos Elementos de Rede.** Alguns nodos sensores permitem configurar seus componentes (por exemplo, o transcepetor). A configuração realizada diretamente sobre o hardware pode ser uma tarefa complexa e difícil. Um *middleware* construído sob a plataforma dos nodos pode permitir a configuração em função dos requisitos da aplicação até mesmo em tempo de execução. Alguns exemplos de tarefas de configuração executadas com auxílio do *middleware* são: configuração da potência de transmissão do transceptor, configuração do tipo de coleta, calibração do dispositivo sensor, configuração do tipo de disseminação dos dados coletados.

***Middleware* para RSSFs Heterogêneas.** Nas RSSFs heterogêneas é comum a coexistência de tecnologias e plataformas diferentes de hardware, sitemas operacionais, protocolos de rede, linguagens de consulta e mesmo aplicações. Um *middleware* para RSSFs heterogêneas pode se encarregar de compatibilizar os diversos elementos heterogêneos que compõem a rede de maneira a uniformizar métodos de acesso, configuração, formato de dados, tipos de disseminação, tipos de coleta.

***Middleware* para Interação.** Os componentes de uma RSSF (homogênea ou heterogênea) interagem entre si com o objetivo de trocar informação e/ou utilizar serviços uns dos outros. O objetivo primordial de um *middleware* é facilitar essa interação. Um *middleware* pode prover interoperabilidade entre diferentes nodos sensores na mesma rede, assim como entre os observadores e a rede, diferentes redes de sensores e entre uma aplicação de gerenciamento e a RSSF. Um *middleware* pode ser disponibilizado nos elementos de rede (nodos comuns, atuadores, nodos líderes de grupo e nodos *gateways*) ou construídos nos pontos de acesso para tornar transparente ao observador a configuração e a manutenção da rede, assim como a resolução de uma consulta.

***Middleware* para Manutenção.** Assim como a configuração, as funções relacionadas à manutenção também podem ser programadas em um *middleware* com características de reflexão (ver seção 4.1). A partir do conhecimento do estado da rede sob diferentes aspectos (energia, topologia, conectividade, densidade) ou em função de variações nas condições ambientais, o *middleware* altera o comportamento do sistema durante a execução. A capacidade de configurar flexivelmente uma plataforma de *middleware* tem o potencial de tornar mais eficiente o suporte oferecido às aplicações. Um suporte dinâmico de *middleware* permite a manutenção, em tempo de execução do serviço fornecido. O uso da adaptação dinâmica permite que a configuração de uma plataforma de *middleware* seja otimizada como resposta imediata a variações no estado da rede, variações nas condições ambientais e nos requisitos das consultas dos observadores [41].

***Middleware* para Aplicação Específica.** Um *middleware* pode ser utilizado para dar o suporte à comunicação distribuída no contexto da disseminação de dados, configuração e manutenção dos nodos, configuração e manutenção da rede e dos serviços, assim como podem ser construídos para o contexto da aplicação como, por exemplo, *middleware* para fusão de dados.

Algumas das funções de RSSFs que podem ser implementadas em um *middleware* são:

- ***Recuperação de Informação***. As RSSFs são tipicamente *data centric*, isto é, a consulta não é realizada para um nodo através do seu identificador ou endereço. A consulta é feita para a rede em função de um atributo. Por exemplo, "existe uma temperatura superior a 100 graus?". Este paradigma de comunicação se assemelha a sistemas de recuperação de informação com base no conteúdo, o que é diferente da comunicação tradicional de estilo RPC. A comunicação baseada em evento é mais adequada às características de RSSFs que esquemas de pedido-resposta tradicionais. As consultas também podem utilizar um critério de fidelidade, isto é, selecionar nodos, parâmetros ou algoritmos com os quais resolvem um certo problema de acordo com determinadas restrições de recurso.

- ***Negociação de Contexto***. Um *middleware* pode garantir que as entidades envolvidas em uma rede de sensores sem fio (observadores, nodos de monitoração, nodos comuns e atuadores) trabalhem de forma coordenada, transparente e sincronizada para prover serviços ao usuário final. Assim, conforme o estado da rede se altera em função de um determinado parâmetro, o *middleware* tem a responsabilidade de realizar a manutenção com base no contexto.

- ***Gerenciamento***. Em sistemas tradicionais, cada dispositivo de computação pertence a alguém que é responsável pela configuração, manutenção, e manipulação de erro. Em contraste, os nodos de uma RSSF têm que operar desacompanhados, o que significa que o *middleware* para RSSFs tem que prover novos níveis de apoio para configuração automática, manutenção e manipulação de erro.

- ***Tempo Real***. Desde que as RSSFs processam dados do mundo real, os conceitos de tempo físico e localização têm um papel muito mais importante que em sistemas tradicional. Tempo e localização de eventos do mundo real são elementos chaves por fundir leituras de sensor individuais e obter um alto-nível do sensoriamento resultante. Algumas áreas de aplicação podem exigir características de tempo real. Então, o apoio para tempo e gerenciamento de localização deve ser firmemente integrado em uma infra-estrutura de *middleware* para RSSFs.

- ***Manutenção da Área de Cobertura***. Em muitas aplicações a área de cobertura de uma RSSF pode variar devido a vários fatores, entre eles a mobilidade dos nodos e a presença de fontes de energia limitadas e não recarregáveis. A mobilidade pode ampliar ou modificar os limites da área de cobertura inicialmente definida. No caso da fonte de energia limitada, nodos que tornam-se indisponíveis e não são substituídos reduzem a área de cobertura inicial e podem influenciar na precisão do resultado. A cobertura é considerada uma medida de qualidade de serviço [34]. Na literatura, a área cobertura é definida sobre diferentes pontos de vista, entre eles o melhor e o pior caso. No pior caso, tentativas são realizadas para quantificar a qualidade de serviço procurando áreas de baixa observabilidade dos nodos e detectando regiões de monitoração descobertas. No melhor caso, o objetivo é encontrar áreas de alta observabilidade dos nodos e identificar as melhores regiões de monitoração. A cobertura também pode ser representada por definições estocásticas como, por exemplo por funções de distribuição uniforme, distribuição Gaussiana e distribuição de Poisson.

- ***Tolerância a Falhas***. Nas RSSFs os nodos podem falhar, principalmente, devido a variações nas condições do ambiente físico e podem perder sua funcionalidade devido à falta de energia. Considerando que em muitas aplicações a troca ou reposição de nodos é difícil, a RSSF deve implementar mecanismos de tolerância a falhas que possam garantir o funcionamento da rede. Na próxima seção discutimos os mecanismos de tolerância a falhas aplicados a RSSFs.

6 Tolerância a Falhas em RSSFs

Segundo Weber [42], o objetivo de tolerância a falhas é alcançar dependabilidade. O termo dependabilidade[1] indica a qualidade do serviço fornecido por um dado sistema e a confiança depositada no serviço fornecido. Os principais atributos de dependabilidade são confiabilidade, disponibilidade, segurança de funcionamento (*safety*), segurança (*security*), mantenabilidade, testabilidade e comprometimento do desempenho (*performability*).

A seguir, discutimos cada um destes atributos com relação às RSSFs.

6.1 Confiabilidade em RSSFs

As RSSFs são projetadas para atenderem a uma série de requisitos. Um dos requisitos importantes é que elas sejam confiáveis tanto quanto possível e apesar do meio em que estão inseridas. A confiabilidade é definida como sendo a capacidade que um sistema tem em responder a uma dada especificação dentro de condições definidas e durante um certo tempo de funcionamento. A confiabilidade é uma característica importante para as RSSFs porque o reparo é praticamente impossível neste tipo de rede.

[1] Este termo é uma tradução literal do termo em inglês *dependability*.

6.2 Disponibilidade em RSSFs

Os nodos de uma RSSFs são depositados (muitas vezes lançados sobre uma área) em *sleep mode* para economizar energia, despertam, realizam auto-teste, descobrem sua localização (utilizando alguma técnica como GPS – *Global Position System* ou em colaboração com os vizinhos) e se auto-organizam formando uma rede *ad hoc*. Durante a deposição, alguns nodos podem ser destruídos (total ou parcialmente). Também durante a descoberta de localização e formação da rede, alguns nodos podem estar isolados dos demais. Dependendo da distribuição e do estado dos nodos, podem surgir áreas descobertas de sensoriamento e comunicação. Após o período de formação da rede, os nodos iniciam as atividades de sensoriamento, processamento local e disseminação da informação. Estas atividades consomem energia. No tempo, os nodos deixam a rede (temporariamente ou definitivamente) por diferentes motivos mas principalmente por problemas de energia. O nível de energia de um nodo define seu aspecto operacional. Já o aspecto operacional da rede é definido em termos de seu mapa de energia, isto é, se todos os nodos que compõem a rede estão operacionais em um determinado instante. Portanto, a disponibilidade de uma RSSF é a probabilidade do sistema estar operacional num determinado instante de tempo.

6.3 Segurança em RSSFs

Uma RSSF está sujeita a diferentes tipos de ameaças: internas, externas, acidentais ou maliciosas. Quando falamos de ameaças internas e acidentais, dizemos que a rede pode apresentar outros problemas além daqueles relacionados com o nível de energia. Por exemplo, os sensores dos nodos podem estar descalibrados e produzindo dados que corrompem o serviço de coleta. Um exemplo para ameaças externas e acidentais diz respeito a variações nas condições ambientais (por exemplo, ruído) que causam interferências nas comunicações ocasionando perdas de pacotes. Nos casos de ameaças internas e externas acidentais, os nodos devem descontinuar suas funções para evitar problemas e deseperdício de energia. Então, segurança de funcionamento (*safety*) é a probabilidade do sistema estar operacional, e executar sua função corretamente, ou descontinuar suas funções de forma a não provocar dano a outros sistemas que dele dependam. Quando tratamos de ameaças maliciosas, definimos segurança (*security*) como proteção contra impedimento de serviço (DoS – *Denial of Service*), falhas maliciosas, visando privacidade, autenticidade, integridade e irrepudiabilidade dos dados.

6.4 Comprometimento do Desempenho em RSSFs

O comprometimento do desempenho está relacionado à queda de desempenho provocado por falhas, onde o sistema continua a operar, mas degradado em desempenho. Em certas aplicações de RSSFs, um atraso na entrega de uma informação pode torná-la inválida ou inconsistente. Em outras aplicações, a qualidade do serviço exige que todos os nodos da rede participem da coleta de dados. Se houverem regiões descobertas em função da falhas de alguns nodos, haverá comprometimento do desempenho da rede.

6.5 Mantenabilidade em RSSFs

A mantenabilidade significa a facilidade de realizar a manutenção do sistema, ou seja, a probabilidade que um sistema com defeitos seja restaurado a um estado operacional dentro de um período determinado. Restauração envolve a localização do problema, o reparo físico e a colocação em operação. No caso das RSSFs, o reparo físico pode ser difícil ou mesmo impossível de ser realizado. A manutenção pode utilizar nodos redundantes já existentes na rede ou determinar uma deposição incremental de nodos que venham a substituir o nodo com defeito. Um serviço de manutenção de área de cobertura para RSSFs foi definido em [35]. A partir do mapa de topologia da rede, o serviço de manutenção verifica qual o conjunto mínimo de nodos que cobrem a rede. Então, o serviço coloca os nodos que não fazem parte do conjunto mínimo em "sleep mode". Quando um nodo do conjunto mínimo apresenta nível de energia mínimo, o serviço de manutenção ativa o nodo redundante para cobrir a área. Este é um esquema de tolerância a falhas que exige nodos incrementais e consumo de energia para realizar suas funções.

6.6 Testabilidade em RSSFs

A testabilidade é a capacidade de testar certos atributos internos ao sistema ou facilidade de realizar certos testes. Quanto maior a testabilidade, melhor a mantenabilidade, e por conseqüência menor o tempo que o sistema não estará disponível devido a reparos. A testabilidade em RSSFs é um assunto polêmico porque qualquer operação realizada sobre a rede consome energia.

6.7 Projetando Dependabilidade

A construção de um sistema *dependable* requer esforços em todas as fases de desenvolvimento do sistema. Existem passos a serem realizados na fase de projeto, na fase de implementação e em tempo de execução, assim como durante a manutenção e aprimoramento. Na fase de de projeto, pode-se aumentar a dependabilidade de um sistema através de técnicas de prevenção de falhas. Na fase de implementação, a dependabilidade pode ser aumentada através de técnicas de remoção de falhas. Finalmente, em tempo de execução são necessárias técnicas de tolerância a falhas e de evasão de falhas.

Uma RSSF construída com a capacidade de tolerar falhas conseguirá manter-se operando, talvez de uma forma degradada, na presença de falhas. Para que um sistema seja tolerante a falhas, ele deve ser capaz de detectar, diagnosticar e confinar as falhas, além de compensá-las e de recuperar-se delas.

O termo "evasão de falhas" diz respeito a atitude de um sistema que, quando se desvia do comportamento normal, mesmo se o comportamento continua atendendo as especificações do sistema, se reconfigura para reduzir a demanda sobre um componente com grande potencial de falhar. Este é o caso quando nas RSSFs nodos em situação crítica de energia estão participando de algum serviço.

O grau de tolerância a falhas que um sistema requer pode ser especificado quantitativamente ou qualitativamente. Quando a meta de dependabilidade é quantitativa, ela é normalmente expressa como a taxa de falhas máxima permitida (por exemplo, 10% dos sensores indisponíveis por sub-área). O problema desta forma de expressão é a dificuldade em se saber quando ela foi atingida.

Alternativamente, pode-se usar a especificação qualitativa dos requisitos de dependabilidade. Seguem algumas especificações típicas:

- ***Fail-safe***: projete o sistema de forma que, quando ele sofrer um certo número de falhas, ele falhe da forma segura. Por exemplo, quando um certo número de nodos falham e deixam uma área de monitoração descoberta, o observador deve ser informado de que os dados não correspondem a área de cobertura total.

- ***Fail-op***: projete o sistema de forma que, quando ele sofrer um certo número de falhas, ele ainda forneça um subconjunto do comportamento especificado para ele. Por exemplo, ainda que existam áreas descobertas, a rede é capaz de fornecer o serviço com relação às áreas ainda operacionais.

- ***No single point of failure***: projete o sistema de forma que a falha de um único componente, seja ele qual for, não faça o sistema falhar. Normalmente, em tais sistemas, o componente falho pode ser substituído ou consertado antes que outra falha ocorra. No caso das RSSFs os nodos não podem ser consertados visto que são descartáveis. Pode-se, então, utilizar esquemas de nodos *back-up* para substituir os nodos que falharam.

- ***Consistência***: projete o sistema de forma que toda a informação entregue pelo sistema seja equivalente à informação que seria entregue por uma instância de um sistema não falho. Este atributo é difícil de ser construído em RSSFs devido a sua integração com o mundo físico e também porque a rede opera sem intervenção humana direta e na maioria das aplicações, os nodos realizam a mesma tarefa e colaboram entre si com um objetivo comum.

Como visto, a aplicação dos atributos de dependabilidade em RSSFS representa novos desafios e grande oportunidade de pesquisa. A seguir discutimos os níveis de tolerância a falhas em RSSFs.

6.8 Níveis de Tolerância a Falhas em RSSFs

As técnicas de tolerância a falhas devem garantir o funcionamento correto do sistema mesmo na ocorrência de falhas. Todos os mecanismos de tolerância a falhas são baseadas em redundância, exigindo componentes adicionais ou algoritmos especiais.

Heimerdinger e Weinstock [43] definem três níveis em que a tolerância a falhas pode ser aplicada quais sejam, hardware, software, e sistemas. A seguir, discutimos estas definições considerando as RSSFs.

Tolerância a Falhas de Hardware. Tradicionalmente, a tolerância a falhas tem sido usada para compensar falhas nos recursos computacionais (hardware). Ao gerenciar recursos extras de hardware, o subsistema do elemento computacional aumenta sua habilidade de continuar sua operação. Medidas de tolerância a falhas de hardware incluem replicação de processadores, memória adicional, fontes de energia redundantes, e comunicação redundante. A comunicação redundante pode ser conseguida com certo custo em RSSFs mas a replicação de processadores, memória adicional e fonte de energia no mesmo elemento computacional é impossível para a atual tecnologia das RSSFs. A tolerância a falha do hardware para RSSFs pode ser realizada utilizando redundância de nodos. Além da tolerância a falhas, o uso da redundância em RSSFs pode ser motivado por um outro fator: precisão. De qualquer forma, a redundância sugere uma alta densidade de nodos, isto é, aumenta o número de sensores por área. Se a densidade for alta podem surgir áreas de interseção de coleta, informações redundantes e interferência nas comunicações, além do desperdício de energia. No caso de uma alta densidade, alguns nodos podem ser retirados administrativamente de serviço e transformados em nodos *backup*. Quando os nodos ativos começarem a deixar a rede por problemas de energia, os nodos *backup* são ativados. No caso de uma baixa densidade, a área de cobertura da rede pode estar comprometida, exigindo a ativação ou outro lançamento de nodos. A alta densidade causa a redundância que pode servir como mecanismo de precisão e tolerância a falhas mas que representa um maior consumo de energia.

Tolerância a Falhas de Software. Um segundo nível de tolerância a falhas reconhece que um hardware tolerante a falhas não consegue, por si só, garantir alta disponibilidade. Assim, também é importante estruturar o software dos elementos computacionais para compensar falhas decorrentes de erros transitórios, entrada de dados não prevista ou de erros de projeto. Esta é a tolerância a falhas de software. Uma RSSF é um sistema distribuído reativo, já que interage continuamente com o ambiente[2]. A parte de controle é normalmente modelada por uma máquina de estados finitos estendida e comunicante (CEFSM – *Communicating Extended Finite State Machine*). A máquina é estendida no sentido que predicados podem ser associados a transições e comunicante porque interage com outras máquinas. Este tipo de sistema é difícil de validar, seja através de verificação formal, simulação ou teste. Sendo assim, atenção especial deve ser dada ao projeto desses sistemas, que além da correção deve se preocupar com a eficiência da solução proposta [2].

Tolerância a Falhas do Sistema. Um terceiro nível reconhece que disponibilizar hardware e software tolerante a falhas em RSSFs não garante a alta disponibilidade. As RSSFs operam em diferentes ambientes e estão sujeitas a todo tipo de ameaça (interna, externa, acidental e maliciosa). É necessário prover funções que compensem falhas que não são baseadas no elemento computacional. Esta é a tolerância a falhas do sistema. Por exemplo,

[2] De forma genérica, um ambiente é tudo que se encontra fora do espaço de endereçamento de um processo, incluindo aí o sistema operacional, outros processos no próprio nodo, canal de comunicação e ambiente físico que gera os "eventos" que devem ser processados.

uma RSSF sofrendo de interferência nas comunicações ou no sensoriamento em função das condições ambientais.

Finalmente e importante que as medidas de tolerância a falhas sejam compatíveis em todos os níves.

7 Conclusão

Os requisitos não-funcionais como, escalabilidade, extensibilidade, heterogeneidade, compartilhamento de recursos, tolerância a falhas para sistemas móveis *ad hoc* distribuídos são válidos para RSSFs. Contudo, em RSSFs estes requisitos adquirem outras características em função dos componentes, tipo de operação e ambiente onde tais redes trabalham.

Um *middleware* para RSSFs precisam ser leves, eficientes em energia, prover comunicação assíncrona entre os componentes, prover informações sobre o contexto, permitir escalabilidade, robustez, tratar do tempo e da localização.

As redes de sensores definem novos conceitos e problemas. Alguns, tais como gerenciamento de localização e energia são assuntos fundamentais, em que muitas aplicações confiam para obter a informação necessária [34]. Muitas características das redes de sensores, tais como auto-organização, localização, mecanismos de endereçamento e serviços de *binding*, coleta de dados envolvendo problemas de cobertura de área e exposição, topologia dinâmica, fusão de dados, arquitetura da aplicação, mecanismos de segurança e tráfego são desafios em relação aos sistemas distribuídos tradicionais, mas também representam novas oportunidades de pesquisa.

Referências

[1] D. Bakken. Middleware. *Encyclopedia of Distributed Computing*, 2002.

[2] A. A. Loureiro, Linnyer B. Ruiz, Raquel A. Mini, and José Marcos S. Nogueira. Rede de sensores sem fio. Minicurso do Simpósio Brasileiro de Redes de Computadores. Natal, RN, Brasil, 2003.

[3] Praveen Rentala, Ravi Musunuri, Shashidhar Gandham, and Udit Saxena. Survey on sensor network. Submitted as per the requirements of Mobile Computing (CS6392) Course, Disponível via <http://www.citeseer.nj.nec.com/479874.html> Acessado em 25 de julho de 2002.

[4] L. Capra, W. Emmerich, and C. Mascolo. Middleware for Mobile Computing (A Survey). In *Advanced Lectures on Networking - Networking 2002 Tutorials*, pages 20–58, Pisa, Itália, Maio 2002. Springer Verlag.

[5] W. Stevens. *UNIX Network Programming*. Prentice Hall, 1997.

[6] A. Pope. *The Corba Reference Guide: Understanding de Common Object Request Broker Architecture*. Addison-Wesley, 1998.

[7] S. Baker. *Corba Distributed Objects: Using Orbix*. Addison-Wesley, 1997.

[8] V. Natarajan, S. Reich, and B. Vasudevan. *Programming With Visiobroker: A Developer's Guide to Visiobroker for Java*. John Wiley & Sons, 2000.

[9] OMG. CORBA Componet Model. http://www.omg.org/cgi-bin/doc?orbos/97-06-12.

[10] D. Rogerson. *Inside COM*. Microsft Press, 1997.

[11] E. Pitt and K. McNiff. *Java RMI: The Remote Method Invocation Guide*. Addison-Wesley, 2001.

[12] R. Monson-Haefel. *Enterprise Javabeans*. O'Reilly & Associates, 2000.

[13] R. Monson-Haefel, D. Chappell, and M. Loukides. *Java Message Service*. O'Reilly & Associates, 2000.

[14] IBM Redbooks. *MQSeries Version 5.1 Administration and Programming Examples*. IBM Corporation, 1999.

[15] E. Hudders. *CICS A Guide to Internal Structure*. Wiley, 1994.

[16] C. Hall. *Building Client/Server Applications Using TUXEDO*. Wiley, 1996.

[17] B. Smith. *Reflection and Semantics in a Procedural Programming Language*. PhD thesis, MIT, 1982.

[18] H. Duran and G. Blair. A Resource Management Framework for Adaptive Middleware. In *3th IEEE International Symposium on Object-oriented Real-time Distributed Computing (ISORC'2k)*, Newport Beach, California, EUA, Março 2000.

[19] F. Costa and F. Kon. Novas Tecnologias de Middleware: Rumo à Flexbilização e ao Dinamismo. In *Anais do 20o Simpósio Brasileiro de Redes de Computadores*, 2002.

[20] T. Ledoux. OpenCorba: A Reflective Open Broker. *Reflection'99*, Springer-Verlag, LNCS 1616:197–214, Julho 1999.

[21] G. Blair, G. Coulson, A. Andersen, M. Clarke, F. Costa, H. Duran, R. Moreira, N. Parlavantzas, and K. Saikoski. The Design and Implementation of Open ORB version 2. *IEEE Distributed System Online Journal*, 2(6), 2001.

[22] F. Kon, B. Gill, M. Anand, R. Campbell, and M. Mickunas. Secure Dynamic Reconfiguration of Scalable CORBA System with Mobile Agents. In *Proceedings of the IEEE Joint Symposium on Agent Systems and Applications/Mobile Agents (ASA/MA'2000)*, Setembro 2000.

[23] G. Picco, A. Murphy, and G. Roman. LIME: Linda Meets Mobility. In *Proceedings of the 21st international conference on Software engineering*, pages 368–377. IEEE Computer Society Press, 1999.

[24] P. Wyckoff, S. McLaughry, T. Lehman, and D. Ford. Tspaces. *IBM Systems Journal*, Agosto 1998.

[25] D. Fritsch, D. Klinec, and S. Volz. NEXUS Positioning and Data Management Concepts for Loacation Aware Applications. In *Proceedings of the 2nd International Symposium on Telegeoprocessing*, pages 171–184, Nice-Sophia-Antipolis, France, 2000.

[26] M. Satyanarayanan. Mobile Information Access. *IEEE Personal Communications*, 3(1), 1996.

[27] Smart dust. autonomous sensing and communication in a cubic millimeter. *Disponível via <http://robotics.eecs.berkeley.edu/Epister/SmartDust>. Acessado em 11 de abril de 2002.*

[28] WINS Wireless integrated network sensors. Disponível via <http://www.janet.ucla.edu/wins/>. Acessado em 12 de março de 2002.

[29] JPL Sensor Webs. http://sensorwebs.jpl.nasa.gov/, 2002.

[30] D. Estrin, R. Govindan, and J. Heidemann. Special issue on embedding the internet. *Communications of ACM*, 5(43), May 2000.

[31] Stephanie Lindsey, Cauligi Raghavendra, and Krishna Sivalingam. Data gathering in sensor networks using the energy delay metric. Disponível via <http://www.citeseer.nj.nec.com/461425.html>. Acessado em 12 de maio de 2002.

[32] Deborah Estrin, Ramesh Govindan, John S. Heidemann, and Satish Kumar. Next century challenges: Scalable coordination in sensor networks. Proceedings of the fifth annual ACM/IEEE International Conference on Mobile computing and networking. pages 263-270, August 1999, Seattle, WA USA.

[33] B. R. Badrinath, M. Srivastava, K. Mills, J. Scholtz, and K. Sollins. Special issue on smart spaces and environments. *IEEE Personal Communications*, Oct 2000.

[34] Seapahn Meguerdichian, Farinaz Koushanfar, Miodrag Potkonjak, and Mani B. Srivastava. Coverage problems in wireless ad hoc sensor networks. In *INFOCOM*, pages 1380–1387, 2001.

[35] Linnyer Beatrys Ruiz, José M. S. Nogueira, and Antonio A. Loureiro. Manna: A management architecture for wireless sensor networks. *IEEE Communications Magazine*, 41(2):116–125, February 2003.

[36] Linnyer Beatrys Ruiz, José Marcos S. Nogueira, and Antonio A. Loureiro. Functional and information models for wireless sensors networks. In *Actes GRES03 - Colloque Francophonic sur La Gestion de Reseaux et Service*, 2003.

[37] Loren Schwiebert, Sandeep K. S. Gupta, and Jennifer Weinmann. Research challenges in wireless networks of biomedical sensors. In *Mobile Computing and Networking*, pages 151–165, 2001.

[38] Mani B. Srivastava, Richard R. Muntz, and Miodrag Potkonjak. Smart kindergarten: sensor-based wireless networks for smart developmental problem-solving enviroments. pages 132–138. Mobile Computing and Networking, 2001.

[39] A. Cerpa, J. Elson, D. Estrin, L. Girod, M. Hamilton, and J. Zhao. 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, April 2001.

[40] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, and Kristofer Pister. System architecture directions for network sensors. In *ASPLOS*, 2000. http://webs.cs.berkeley.edu/tos/.

[41] Kay Römer, Oliver Kasten, and Friedmann Mattern. Middleware challenges for wireless sensor networks. *Mobile Computing and Communications Review*, 6(2), 2002.

[42] T. Weber, I. Jansch-Pôrto, and R. Weber. Um roteiro para exploração dos conceitos básicos de tolerância a falhas. http://www.inf.ufrgs.br/ taisy/, 2003.

[43] W. Heimerdinger and C. Weinstock. A conceptual framework for system fault tolerance. Technical Report CMU/SEI-92-TR-33, Carnegie Mellon University, Pittsburgh, PA, 1992.