Um modelo de programação para RSSF com suporte a reconfiguração dinâmica de aplicações *

Adriano Branco[1], **Noemi Rodriguez**[1], **Silvana Rossetto**[2]

[1]Dep. de Informática – Pontifícia Universidade Católica do Rio de Janeiro (PUC-RJ)
R. Marques de S.Vicente, 225 – Gávea – CEP:22453-900 – Rio de Janeiro, RJ–Brasil

[2]Dep. de Ciência da Computação – Universidade Federal do Rio de Janeiro (UFRJ)
Caixa-Postal:68530 – CEP:21941-590 – Rio de Janeiro, RJ–Brasil

{abranco,noemi}@inf.puc-rio.br, silvana@dcc.ufrj.br

Abstract. *Programming and maintaining WSN applications is typically a tedious and error-prone task. We discuss a model to minimize the difficulties of development and reconfiguration of such applications. This model is based on a set of parametrized components and on a Finite State Machine, and allows the remote configuration of different applications over the same set of installed components. To evaluate this idea, we built a component library based on the requirements of different WSN applications. In this paper, we describe this library and, using a typical WSN application, evaluate its impact on the development process, and the ease of applying modifications to a running application. We also measure the additional impact of remote configuration on network activity.*

Resumo. *Neste trabalho descrevemos um modelo de programação para redes de sensores sem fio que pretende simplificar as tarefas de criação e reconfiguração de aplicações. O modelo se baseia no uso conjunto de componentes parametrizáveis e de máquinas de estados finitos, e permite a implementação de diferentes tipos de aplicações para redes de sensores sem fio e a configuração remota dessas aplicações. Para avaliá-lo, criamos uma biblioteca de componentes a partir de um levantamento de requisitos de diferentes classes de aplicações. Realizamos alguns testes para avaliar o quanto essa biblioteca de componentes pode facilitar o desenvolvimento de novas aplicações, o quanto é fácil aplicar novas alterações sobre as aplicações em execução, e o impacto na quantidade de mensagens na rede por conta do uso da configuração remota.*

1. Introdução

No cenário de Redes de Sensores Sem Fio (RSSF), vêm ganhando especial atenção os dispositivos com tamanhos e recursos limitados (motes). Esses dispositivos podem ser utilizados em diversos tipos de aplicações, principalmente em aplicações que necessitam de uma distribuição em grandes áreas ou mesmo em áreas de difícil acesso.

O trabalho de desenvolvimento de aplicações para RSSF é impactado pelos mesmos desafios encontrados no desenvolvimento de aplicações para sistemas distribuídos

*Este trabalho foi parcialmente financiado pelo Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) sob os processos 135882/2009-5 e 308192/2007-9.

que utilizam plataformas tradicionais. Porém alguns desses desafios são aumentados pela escassez de recursos computacionais dos motes, por um modelo de programação fortemente orientado a eventos e pela tecnologia de comunicação sem fio com características de redes ad-hoc. Além disso, a programação feita na ótica de cada nó sensor dificulta o desenvolvimento nos casos em que as aplicações utilizam os dispositivos em grande escala e onde o desenvolvedor precisa ter uma visão da rede como um todo.

Associado aos desafios típicos da computação distribuída, uma demanda comum no cenário de RSSF é a necessidade de realizar ajustes na aplicação após a sua implantação. Esses ajustes podem ser uma simples alteração no período de monitoração, uma correção no fluxo de processamento ou até a ativação ou desativação de pequenas funcionalidades. A plataforma de software de uso mais comum para o desenvolvimento de aplicações em RSSF não disponibiliza, de forma nativa, opções para reconfiguração remota após a implantação da aplicação e, na maioria dos casos, é inviável recuperar todos os motes para aplicar novas alterações. As alternativas mais comuns para reprogramação dos nós sensores envolvem a carga remota e substituição de componentes inteiros, dessa forma, indo muito além de uma simples reconfiguração e podendo consumir mais recursos da rede do que o necessário.

Uma solução intermediária seria disponibilizar para o desenvolvedor um conjunto de componentes parametrizáveis e que possam ter a sua execução definida através de um controle de fluxo simplificado. Aliado à opção para reconfiguração remota, tanto do fluxo de processamento quanto dos parâmetros dos componentes, esse esquema é suficiente em muitos cenários e envolve menos gastos de energia.

O objetivo principal do nosso trabalho é investigar um modelo de programação que minimize as dificuldades de desenvolvimento e reconfiguração dinâmica das aplicações em RSSF. Um caminho para alcançar essas duas metas é permitir o uso das mesmas técnicas de programação, tanto no processo de desenvolvimento de uma nova aplicação, como na etapa seguinte de reconfiguração em tempo de execução.

Para facilitar a criação de novas aplicações, identificamos um conjunto de componentes de alto nível que se combinam para formar diferentes aplicações. Através de um controle de fluxo baseado em máquinas de estados finitos (*Finite State Machine*) (FSM), o desenvolvedor indica quais componentes irá utilizar e qual será a sequência de operação dos mesmos. Tanto o fluxo da FSM quanto o comportamento funcional dos componentes são definidos através de parâmetros. O modelo de controle de fluxo proposto permite a criação de várias máquinas de estado dentro de uma mesma aplicação e ainda possibilita a dependência entre estados de diferentes máquinas, facilitando a construção de aplicações mais complexas, ou a visão de uma mesma RSSF usada para diferentes finalidades.

A possibilidade de configuração dinâmica é alcançada simplesmente permitindo que esses parâmetros sejam alterados remotamente. Como exemplo, em uma aplicação de coleta periódica de temperatura, pode-se alterar o período de coleta ou alterar o fluxo operacional para calcular a média da temperatura dos nós do mesmo grupo. Também é possível adicionar, em tempo de execução, um novo fluxo para esta aplicação, como um alarme de temperatura alta, mantendo o fluxo anterior inalterado.

Este texto está organizado da seguinte forma: na próxima seção (2), apresentamos os trabalhos relacionados; na seção 3, apresentamos o nosso sistema, abordando os

componentes propostos e o modelo de transição para a FSM; na seção 4, apresentamos alguns cenários de teste para nossa análise experimental; finalizamos com a seção 5, na qual apresentamos nossas considerações e abordamos as melhorias e os trabalhos futuros.

2. Trabalhos Relacionados

Um dos primeiros trabalhos a propor simplificação para programação de plataformas no estilo motes foi o TinyDB [Madden et al. 2005]. Apesar de não utilizar o conceito de biblioteca de componentes, o TinyDB disponibiliza um conjunto de funções para agregações simples e permite reconfigurar a coleta de dados da rede através de comandos remotos baseados numa linguagem simplificada do estilo SQL. O TinyDB limita-se às aplicações com características de coleta de dados, com possibilidade de agregações em uma topologia hierárquica de roteamento e sem interação local entre os nós da rede.

Um conjunto de trabalhos mais recentes, com foco em modelos de programação mais adequados para RSSF, está ligado ao conceito de *macroprogramação*. O argumento principal para a macroprogramação em RSSF é que toda aplicação para essas redes requer naturalmente a interação entre os nós individuais, então, ao invés de projetar as aplicações a partir do código de cada nó, propõe-se o inverso, que a aplicação seja projetada considerando a rede como um todo e que, em uma etapa seguinte, o código necessário seja instanciado em cada nó individual. Dentre os representantes dessa linha, destacamos Regiment [Newton et al. 2007], Pleiades [Kothari et al. 2007], Cosmos [Awan et al. 2007], WADL [Cervantes et al. 2008] e ATaG [Bakshi et al. 2005]. Uma dificuldade encontrada nessas soluções é que embora a visão de programação da rede como um todo seja mais adequada, em geral, os componentes básicos que devem ser executados nos nós individuais ainda precisam ser implementados diretamente pelo desenvolvedor, desviando a atenção para os nós individuais em uma etapa seguinte ao projeto da aplicação.

Dentre os trabalhos que abordam a reprogramação remota, temos os que utilizam um modelo de máquina virtual simplificado, como Maté [Levis and Culler 2002], ou os trabalhos baseados no DELUGE [Hui and Culler 2004], que carregam o programa original na linguagem nativa. A linguagem utilizada por Maté é bem simples e permite acesso apenas aos recursos básicos do mote, não disponibilizando uma biblioteca de componentes de alto nível que facilite a construção de novas aplicações. Os trabalhos que utilizam o DELUGE normalmente precisam fazer a carga completa da aplicação em conjunto com o sistema operacional. Abordagens mais recentes possibilitam a recarga em separado de componentes originais da aplicação [Munawar et al. 2010], permitindo reduzir o custo da reprogramação remota.

No trabalho de Kasten e Romer é proposto um modelo de programação baseado em máquinas de estados para RSSF, chamado OSM [Kasten and Römer 2005], cujo objetivo é simplificar a programação dos motes. OSM utiliza um processo de compilação para gerar código na linguagem nativa dos nós sensores e não aplica o conceito de reprogramação. Como outros trabalhos, OSM também considera que o desenvolvedor deve construir os componentes mais básicos na linguagem de programação do nós sensores.

O modelo de programação para RSSF que propomos neste trabalho integra um mecanismo para configuração do fluxo de processamento da aplicação com uma biblioteca de componentes parametrizáveis, a partir do qual, na maioria dos casos, o desen-

volvedor não precisará construir nenhum componente adicional. Comparando com as opções de carga de software remota, temos um modelo de programação de mais alto nível e simplificado do que as soluções que utilizam máquina virtual ou carga de código nativo. Comparando com o TinyDB, o nosso modelo disponibiliza mais funcionalidades e ainda permite interações locais. Aproveitamos a ideia do modelo de FSM utilizado por OSM e implementamos um modelo de programação que permite a reconfiguração remota de aplicações.

3. Sistema para reconfiguração dinâmica de aplicações baseado em FSM e componentes parametrizáveis

Propomos um sistema que permite a configuração remota de aplicações por meio da definição dos parâmetros de alguns componentes e de um ou mais fluxos de execução. O controle da aplicação é definido por uma máquina de estados finitos (FSM) na qual o desenvolvedor define o fluxo de processamento dos componentes através da combinação de estados, eventos e ações. O comportamento funcional dos componentes é definido através de um conjunto de parâmetros. As possíveis combinações entre o controle de fluxo e as parametrizações dos componentes permitem a construção de diferentes tipos de aplicações para RSSF. Nesta seção, descrevemos os elementos básicos do sistema proposto.

3.1. Biblioteca de componentes parametrizáveis

Para a identificação dos componentes da nossa biblioteca, utilizamos três abordagens. Primeiro, identificamos os diferentes comportamentos de comunicação numa RSSF, considerando a necessidade de interação entre os nós e o roteamento de/para a estação base. Em seguida, a partir de casos levantados na literatura, especificamos três diferentes aplicações para RSSF e identificamos as diferentes ações que cada aplicação utilizou. Essas aplicações estão detalhadas na tabela 1. Finalmente, revisitamos alguns modelos de programação e listamos os diferentes tipos de elementos que cada modelo propôs. Selecionamos alguns modelos de programação específicos para macroprogramação em RSSF ou que trabalhassem com abstrações de mais alto nível de uma rede de sensores, os modelos avaliados foram: Regiment [Newton et al. 2007], Pleiades [Kothari et al. 2007], Cosmos [Awan et al. 2007], WADL [Cervantes et al. 2008], TinyDB [Madden et al. 2005] e ATaG [Bakshi et al. 2005].

A partir dessa avaliação geral, identificamos as similaridades e oportunidades de parametrizações e, com isso, obtivemos uma lista de componentes parametrizáveis, muitos dos quais são utilizados em mais de uma aplicação. Na ótica da aplicação, nossa biblioteca fornece funcionalidades para acesso local ao mote e também para operações em grupos de motes. Assim, algumas das dificuldades típicas de programação em RSSF ficaram transparentes para a camada de aplicação. Como exemplos, podemos citar as operações para agregação de valores, a eleição de líder de grupo e o roteamento de mensagens pela rede.

Os parâmetros da nossa biblioteca estão divididos em Parâmetros Gerais e num conjunto de parâmetros que definem até três Operações de Coleta distintas. Os Parâmetros Gerais definem os valores utilizados para identificação dos grupos de motes, a definição do tempo dos temporizadores genéricos disponibilizados para a aplicação e o período para

Tabela 1. Características das aplicações de referência

Aplicação 1 - Alarme de incêndio florestal	
Descrição: Quando um nó identifica uma situação de alarme, ele consulta seus vizinhos para confirmar o alarme e envia uma mensagem para a estação servidora.	
Trabalho origem	Regiment [Newton et al. 2007]
Tipo de Agrupamento	Comunicação com os vizinhos imediatos. Agrupamento por alcance do rádio.
Tipo de Comunicação	Propagação 1 *hop* com retorno. Roteamento para a estação servidora.
Aplicação 2 - Monitor de temperatura e Alarme de incêndio predial	
Descrição Essa aplicação combina monitoração de temperatura e alarme de incêndio para um prédio. Para monitoração de cada ambiente, um nó centralizador do ambiente calcula periodicamente a temperatura média entre os nós do ambiente e envia a informação para a estação servidora. Para o alarme, o nó que identificar a situação irregular deve enviar uma mensagem para estação servidora.	
Trabalho origem	WADL [Cervantes et al. 2008]
Tipo de Agrupamento	Comunicação com os vizinhos *n* saltos. Agrupamento por posicionamento do nó (andar e sala).
Tipo de Comunicação	Propagação *n hops* com retorno seletivo. Roteamento para a estação servidora.
Aplicação 3 - Estacionamento urbano e Monitoração de vagas	
Descrição Essa aplicação tem um processo de reserva de vagas de estacionamento combinado com uma monitoração das vagas por região. Para monitoração, os nós centralizadores de cada região sumarizam periodicamente a situação das vagas e enviam esses dados para a estação servidora. No processo de reserva, um nó móvel solicita aos nós vizinhos uma vaga disponível e em seguida confirma a reserva de uma das vagas.	
Trabalho origem	Pleiades [Kothari et al. 2007]
Tipo de Agrupamento	Comunicação com os vizinhos *n* saltos. Agrupamento por região do nó.
Tipo de Comunicação	Propagação *n hops* com retorno seletivo. Roteamento para a estação servidora.

reeleição do nó coordenador. Para as Operações de Coleta, pode-se definir o período da coleta, as regras para formação de grupos e a função de coleta. A operação de coleta pode ser uma simples leitura de um sensor local ou uma agregação de valor entre os motes de um mesmo grupo, incluindo também a possibilidade de operações comparativas de resultados contra valores de referência também parametrizados.

Criamos uma arquitetura em camadas em que o desenvolvedor pode controlar o fluxo dos componentes do nível mais alto e pode parametrizar os componentes dessa e de outras camadas. Na Figura 1, apresentamos uma visão das camadas funcionais da nossa arquitetura.

Nas camadas superiores, temos os componentes que podem ser acessados diretamente pelo controle de fluxo, por exemplo, temos as operações locais no mote e as operações de agregação entre motes do mesmo grupo. As camadas intermediárias e inferiores estão isoladas do controle de fluxo, mas os principais componentes podem ser parametrizados conforme a necessidade da aplicação. Essa separação é obtida utilizando

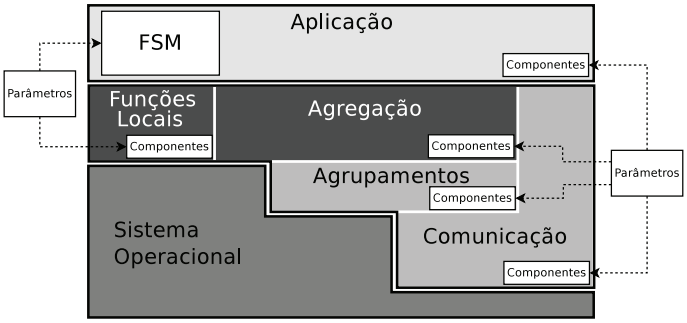

Figura 1. Camadas da arquitetura de execução

o conceito de grupo de motes, no qual a aplicação opera o objeto "Grupo" enquanto que outros componentes trabalham, de forma transparente à aplicação, para efetivar essas operações. A identificação dos grupos é feita através de parâmetros registrados em cada mote. Um determinado grupo é definido pelos motes que contêm os mesmos valores para determinados parâmetros.

O módulo de agregação contém os principais componentes disponibilizados diretamente para o desenvolvedor, como as funções para agregação que incluem funções básicas como SUM (soma dos valores coletados) e MAX (valor máximo coletado), e funções mais complexas, como Reserva de Recurso e Sumarização de dados. O módulo de Funções Locais disponibiliza acesso aos sensores do dispositivo. O módulo Agrupamento é responsável pela validação dos grupos, tanto para as solicitações provenientes da aplicação do mote, como para as solicitações externas de outros motes. O módulo de comunicação controla os protocolos de comunicação disparando operações ou reagindo a solicitações externas. Esse módulo implementa três protocolos básicos: propagação e retorno de valores de forma radial a partir de um nó da rede (NHops), roteamento de mensagens de/para a estação base e difusão dos dados da FSM pela rede. Essa camada também processa o recebimento dos dados de configuração enviados pela estação base, possibilitando a reconfiguração dinâmica da aplicação.

A partir desse conjunto inicial de componentes básicos foi possível implementar as três aplicações de referência da tabela 1. O resultado está disponível no link: `http://www.inf.puc-rio.br/~abranco/files/SBRC2011`

3.2. Controle do fluxo de processamento baseado em FSM

Para permitir combinações diferentes de um dado conjunto de componentes, implementamos um controle de fluxo de operação baseado no modelo de FSM. Esse modelo é facilmente adaptável ao modelo típico de eventos utilizado em RSSF.

Nossa implementação considera que a FSM será definida por uma tabela de transições. De forma simplificada, cada transição contém um estado de entrada, um evento válido para esse estado, uma ação de saída e o estado de saída da transição. O Controle da FSM, ao receber um novo evento, verifica se existe uma transição válida para o estado corrente. Se sim, muda o estado corrente para o estado de saída e dispara a ação de saída. Na Figura 2.a, apresentamos um diagrama simplificado da arquitetura do controle de FSM e, na Figura 2.b, temos uma exemplo de configuração da tabela de transições.

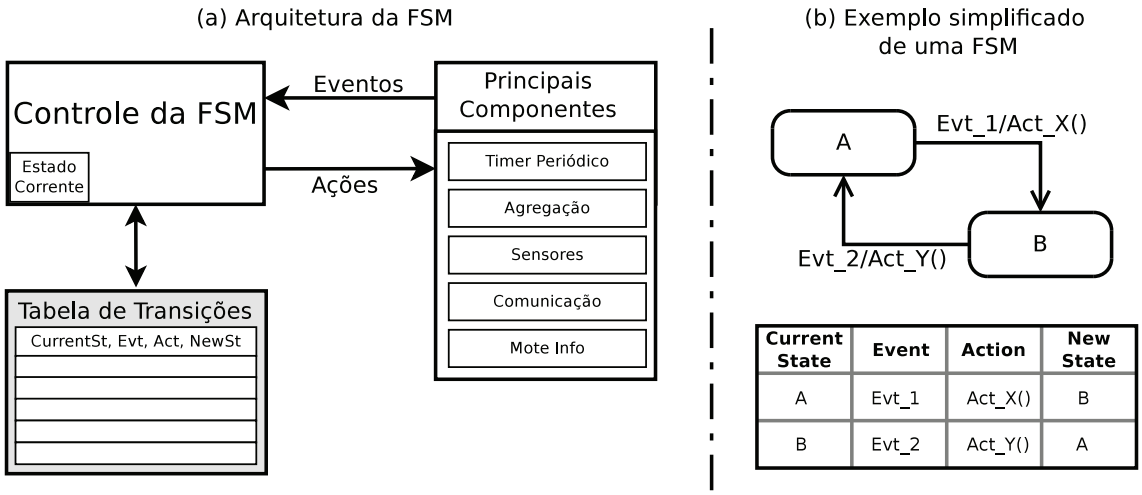

Figura 2. Arquitetura de controle da FSM e exemplo de tabela de transições

A definição dos estados é livre e de responsabilidade do desenvolvedor, mas as possíveis ações e eventos dependem dos componentes disponibilizados no sistema. Por exemplo, a solicitação de leitura de um sensor é uma ação, enquanto que a resposta da leitura é um evento. Outro exemplo de evento é o disparo do timer periódico de coleta. Na tabela 2, apresentamos um resumo das principais ações e eventos disponibilizados na versão atual da nossa implementação. Esse conjunto pequeno de ações, combinado com as diversas opções de parametrizações, permite o desenvolvimento de uma grande gama de aplicações. Como exemplo, na subseção 4.2, apresentaremos a implementação da aplicação esboçada na Figura 4.

Tabela 2. Principais ações e eventos disponíveis

Componente	Ação	Evento
Timer Periódico	–	Disparo para uma nova Coleta (TimerFired)
Sensor Local	Inicia leitura (readSensor())	Resultado da leitura (sensorDone)
	Comparar resultado com parâmetro (testValue())	Resultado da comparação (testValueDone)
Mote Infos	Verifica se é coordenador (testCoord())	Resultado da verificação (testCoordDone)
Agregação	Disparar a agregação indicada nos parâmetros (startAggreg())	Resultado da agregação (AggregDone)
	Comparar resultado com parâmetro (testAggreg())	Resultado da comparação (testAggregDone)
Timer customizado	Iniciar contagem (startTimerX())	Timer finalizado (timerXFired)
Comunicação	Enviar comando para os motes do mesmo grupo (sendComm())	Recebimento de comando de outro mote (recComm)
	Enviar dados para o servidor (sendBS())	Confirmação do envio (sendDone)

3.3. Implementação

A nossa implementação foi realizada usando a linguagem nesC [Gay et al. 2003] e o sistema operacional TinyOS [Levis et al. 2004]. Para os testes reais, usamos a plataforma de hardware MICAz [Crossbow 2004]. Os testes mais exaustivos e monitorados foram executados no simulador TOSSIM [Levis et al. 2003], que acompanha o TinyOS.

O nosso protocolo de formação de grupo de motes (NHops) foi implementado através da difusão de uma mensagem que se propaga por uma quantidade pré-definida de saltos e retorna o valor solicitado, caso o nó faça parte do mesmo grupo do nó originador. A difusão dos dados da FSM pela rede foi implementada utilizando o protocolo DIP [Lin and Levis 2008]. O roteamento de/para a estação base foi implementado utilizando uma customização baseada no protocolo CTP [Fonseca et al. 2007]. Utilizamos o DIP e o CTP por já fazerem parte do TinyOS, o que simplificou a nossa implementação.

4. Avaliação

Avaliamos o nosso sistema sob dois aspectos: (i) primeiro observamos as facilidades de configuração e reconfiguração de uma nova aplicação; (ii) depois avaliamos o impacto adicional em termos de comunicação e quantidade de bytes requeridas para a reconfiguração de uma aplicação em execução.

No primeiro caso, as métricas utilizadas foram a quantidade de linhas na tabela de transições para a FSM e o número de parâmetros necessários para a implementação da aplicação. No segundo caso, a métrica utilizada foi a quantidade de mensagens enviadas na rede. Conforme [Shnayder et al. 2004] podemos considerar o envio de uma mensagem como a operação de maior custo em relação ao consumo de bateria e, consequentemente, a operação que mais afeta o tempo de vida útil de um nó. Para podermos contabilizar os envios, executamos a aplicação no simulador TOSSIM habilitando os logs das funções de comunicação.

Apresentamos a seguir o modelo de rede utilizado como referência para nossos testes. Em seguida, abordamos cada teste e apresentamos os resultados específicos.

4.1. Modelo de rede para os testes

Definimos uma aplicação e uma rede de referência para podermos simular os nossos testes em um ambiente controlado. Vamos assumir uma aplicação de monitoração predial, em que o motes estão distribuídos de forma equidistante em três ambientes, sendo que a estação base se comunica somente com o mote central. Em cada ambiente há um grupo de nove motes. Cada mote só consegue se comunicar com os vizinhos no raio de alcance do rádio, independentemente de estarem ou não no mesmo grupo. Para conseguir formar os grupos tivemos que configurar o parâmetro de salto máximo (*n Hops Max*) igual a dois. A estação servidora (fonte de reconfiguração da aplicação) está conectada ao nó central (25) através da estação base (EB). Essa configuração requer, propositalmente, que os protocolos utilizados precisem rotear a maioria das mensagens.

A Figura 3 apresenta a nossa rede de referência. As setas cinzas indicam o alcance da comunicação entre os motes.

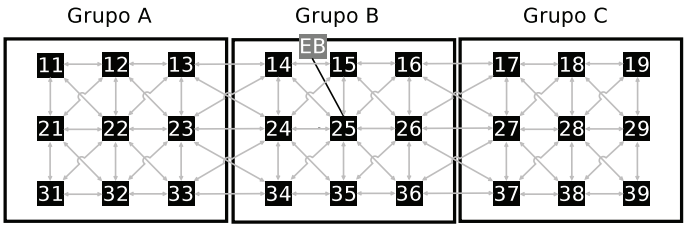

Figura 3. Rede de referência

4.2. Primeiro Teste: Desenvolvendo uma nova aplicação

O primeiro teste teve o objetivo de exercitar a criação de uma nova aplicação no nosso sistema. Para isso, definimos uma aplicação de "Coleta periódica de dados". A nossa aplicação de teste calcula periodicamente a temperatura média de cada grupo de nós da nossa rede de referência e, em seguida, envia esse valor para a estação servidora. O nó centralizador de cada grupo é definido num processo de eleição de líder em que o nó com a maior disponibilidade de bateria é eleito. Com o objetivo de distribuir o consumo, periodicamente é executado um processo para reeleição do líder.

Como citado na seção 3, a configuração da aplicação utiliza uma estrutura de parâmetros e uma tabela de transições como definição do fluxo de funcionamento da aplicação. Para essa aplicação foi preciso definir apenas seis parâmetros, que incluem o período da coleta, o tipo de operação de agregação e a definição do grupo. Já em relação à FSM, foi necessário definir 15 transições, sendo oito para a inicialização obrigatória do mote e sete para o controle da aplicação de coleta. Na tabela 3, apresentamos um resumo da quantidade de parâmetros utilizados e o respectivo tamanho em bytes. A Figura 4.a apresenta a FSM para controle da coleta e a Figura 4.b mostra a estrutura de dados com os parâmetros usados.

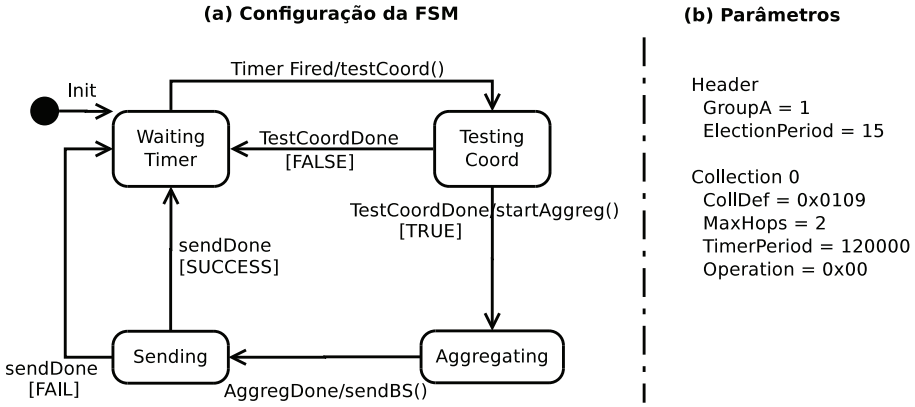

Figura 4. (a) FSM para controle de uma aplicação de coleta e (b) Definição dos respectivos parâmetros.

Executamos essa aplicação no simulador do TinyOS para contabilizar a quantidade de envios em cada etapa de funcionamento da aplicação. No caso da Coleta, os valores correspondem a uma operação de coleta. Na tabela 4, apresentamos a quantidade de mensagens enviadas em cada etapa, com o cuidado de separar os valores para o CTP. Esses valores foram obtidos numa execução equivalente a 20 minutos, com coleta de dois

Tabela 3. Primeiro Teste: Quantidade de bytes utilizados.

	Parâmetros	Transições de FSM
Quantidade de itens	6	15
Tamanho em bytes	13	60

em dois minutos e uma reeleição de coordenador aos 15 minutos.

Tabela 4. Quantidade de envios para cada tipo de operação

Etapa	CTP Inic	Eleição	Reeleição	Coleta
Aplicação	–	855	150	152
CTP	511	14	8	18

O CTP tem características dinâmicas que deixam um resíduo de mensagens independente da aplicação. Encontramos esse resíduo durante as etapas de Eleição e Reeleição, que não utilizam esse protocolo

4.3. Segundo Teste: Modificando uma aplicação

Para essa parte do teste, vamos supor que a coleta periódica não seja suficiente para identificar rapidamente alguma variação de temperatura. Nesse caso, precisamos incluir uma monitoração do tipo alarme que envie para a estação servidora uma mensagem quando a temperatura ultrapassar determinado valor de referência. Com isso, podemos aumentar o período de coleta original, diminuindo o consumo de energia.

Essa modificação pode ser vista como a inclusão de mais uma aplicação na rede, ativando uma segunda operação de coleta e adicionando uma nova máquina de estado. Em nossa implementação, a aplicação necessitou definir cinco novos valores, entre os quais estão o período de leitura do sensor local, o valor de referência e a operação de comparação para gerar o alarme. Também modificamos o período de coleta da configuração anterior. A nova máquina de estados requer sete transições, alterando de 15 para 22 o número total de transições de estado na rede. Na tabela 5, apresentamos um resumo da quantidade de parâmetros utilizados e o respectivo tamanho em bytes. A Figura 5.a apresenta a segunda FSM para controle do alarme e a Figura 5.b mostra a estrutura de dados com os respectivos parâmetros atualizados.

Tabela 5. Segundo Teste: Quantidade de Bytes utilizados na configuração final

	Parâmetros	Transições de FSM
Quantidade de itens	11	22
Tamanho em bytes	24	88

Executamos essa aplicação no simulador e contabilizamos a quantidade de envios adicionais por conta da reconfiguração remota. Apresentamos esses valores na tabela 6.

O DIP tem características dinâmicas que deixam um resíduo de mensagens independente da aplicação. Encontramos esse resíduo durante a etapa Inicialização dos Parâmetros, que não utiliza esse protocolo.

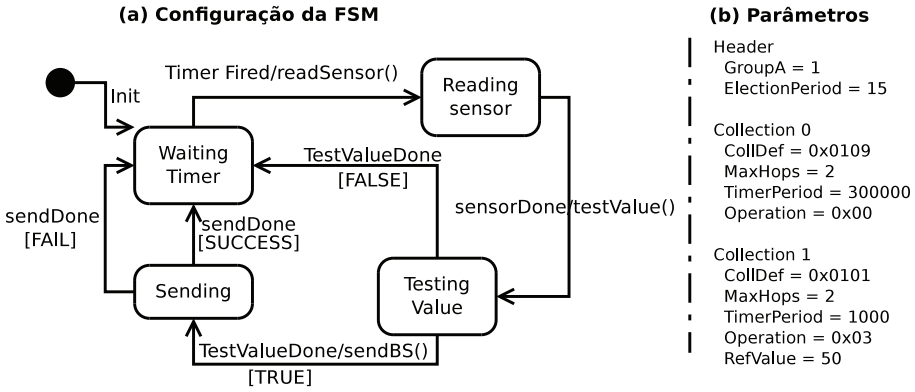

Figura 5. (a) FSM para controle de uma aplicação de alarme e (b) Definição dos respectivos parâmetros

Tabela 6. Quantidade de envios para a operação de reconfiguração remota

Etapa	DIP Inic	Param Inic
Aplicação	–	863
CTP	–	404
DIP	499	49

A partir dos valores experimentais podemos projetar uma operação de longo prazo e calcular o custo de reconfiguração em função da quantidade de coletas. Vamos considerar nossa implementação para a aplicação de coleta periódica executando na configuração de rede de referência com os três grupos de coleta. A partir dos valores das tabelas 4 e 6 podemos dizer que uma reconfiguração equivale a aproximadamente 10,7 coletas $((DIPInic + ParamInic)/(Coleta))$. Supondo uma aplicação funcionando por três meses com coleta a cada 10 minutos, o custo de uma reconfiguração remota equivale 107 minutos de operação ou 0,083% do período de três meses.

É importante ressaltar que essa projeção considera o comportamento da rede como um todo. Quando olhamos a quantidade de envios para cada nó, identificamos valores maiores para nós roteadores e nós coordenadores. A nossa projeção não se altera muito, pois a diferença para os nós roteadores é equivalente tanto na reconfiguração remota como na operação normal. Já para os nós coordenadores, o processo de reeleição garante o revezamento em função da carga da bateria.

4.4. Avaliação dos resultados

Para contrapor a nossa solução com uma solução convencional, vamos supor uma codificação em NesC para a mesma aplicação do primeiro teste. Aplicando a alteração do segundo teste, podemos afirmar que seriam necessárias as seguintes intervenções no código original: adicionar mais um fluxo de controle no programa principal, configurar mais um temporizador e definir mais uma mensagem de dados.

Na nossa experiência, uma solução parametrizável reduz o esforço necessário para reconfigurar uma aplicação. A quantidade de elementos alterados é pequena perto de uma suposta intervenção em um programa NesC. O desenvolvimento em NesC também é mais susceptível a erros de programação do que a configuração de módulos já prontos.

Em relação ao procedimento para reconfigurar a aplicação criando uma nova versão, podemos considerar duas alternativas: a primeira, seria o nosso procedimento de atualização remota, transmitindo somente a configuração da nova aplicação; a segunda seria a geração e carga de um novo programa. Na segunda alternativa, poderíamos considerar ainda as opções de carga remota ou de recuperação física do mote para carga local. Assumindo que a carga local muitas vezes é inviável, vamos comparar aqui as duas opções de carga remota.

Em nosso sistema, a quantidade de dados enviados na reconfiguração depende da quantidade de transições da FSM e da configuração dos parâmetros. Uma FSM de 40 transições terá 160 bytes e para os parâmetros estamos utilizando uma estrutura fixa de 90 bytes, totalizando 250 bytes.

Para o NesC/TinyOS, existem soluções que carregam o código completo do mote [Hui and Culler 2004], e outras que permitem a carga parcial de código [Munawar et al. 2010]. Uma carga de aplicação em mote no TinyOS pode variar de poucas dezenas até mais de uma centena de Kilobytes, e ainda requer que o mote tenha espaço de memória adicional (*flash memory*) para essa tarefa.

Assumindo uma aplicação em NesC/TinyOS com controle de fluxo e protocolos de comunicação, necessitando carregar 10K bytes, seria necessário enviar 40 vezes mais bytes do que a nossa solução.

Complementando a nossa avaliação, implantamos a aplicação exemplo em sensores MICAz e executamos alguns testes simples em uma rede com três motes desse tipo. Os resultados observados foram satisfatórios. O mote MICAz utiliza o processador ATmega128L com 128K bytes de memória para programa (ROM) e 4K bytes de memória para dados (RAM). A aplicação exemplo ocupou 3.666 bytes de memória RAM e 50.436 bytes de memória ROM. O objetivo principal dessa avaliação foi confirmar que o modelo de programação proposto se adequa às limitações de memória dos dispositivos usados nas RSSF.

5. Considerações Finais

Neste trabalho, identificamos um conjunto de componentes parametrizáveis que implementam as tarefas básicas de uma RSSF e um subconjunto de ações e eventos que permitem ligar esses componentes e assim configurar diferentes tipos de aplicações para essas redes. A utilização do conceito de FSM para a programação do fluxo de processamento permitiu uma maior flexibilidade e melhor aproveitamento do conceito de componentes reutilizáveis, além de se adequar bem ao modelo de operação baseado em eventos, típico das aplicações para RSSF.

Com o conjunto de componentes definido, foi possível implementar facilmente três aplicações com diferentes comportamentos (aplicações de referência) e ainda outras aplicações utilizadas nos cenários de testes. Mostramos também que associando o modelo de programação baseado em máquinas de estados com um conjunto de componentes parametrizáveis é possível alcançar duas metas importantes em termos de reconfiguração de aplicações em RSSF: (i) realizar a reconfiguração remota de aplicações em *ponto pequeno*, i.e., sem a necessidade de substituir componentes inteiros da aplicação e com impacto no tempo de vida da rede aceitável; e (ii) permitir que em uma mesma RSSF possam

coexistir vários fluxos de execução independentes e, ao mesmo tempo, com possibilidade de troca de informações entre eles.

A implementação atual possibilita a execução em todas plataformas disponibilizadas no TinyOS e pode ser encontrada no link: `http://www.inf.puc-rio.br/~abranco/files/SBRC2011`

Nossa implementação pode ser utilizada como suporte para o desenvolvimento de novas linguagens de macroprogramação, uma vez que já temos implementadas as principais abstrações de comunicação e interação entre nós utilizadas em aplicações de RSSF. Com isso, retiramos o ônus do desenvolvedor de implementar os componentes elementares, requisitando apenas a configuração dos seus parâmetros.

A continuidade deste trabalho prevê o desenvolvimento de uma linguagem para especificação de aplicações para RSSF e um compilador para essa linguagem que permita gerar a descrição das aplicações no modelo de sistema proposto neste trabalho, definindo a tabela de transições e os parâmetros de configuração dos componentes. Além disso, queremos avaliar a necessidade de estender o conjunto inicial de componentes parametrizáveis, assim como o conjunto de ações e eventos permitidos, para atender aos requisitos de aplicações mais específicas.

Como trabalho futuro, pretendemos investigar com mais detalhe as vantagens da reconfiguração dinâmica em ponto pequeno, oferecida pelo modelo de programação proposto. Também queremos explorar as possibilidades de intercepção entre os fluxos de execução independentes dentro de uma mesma rede para tratar demandas específicas de determinados cenários de aplicação das RSSF.

Referências

[Awan et al. 2007] Awan, A., Jagannathan, S., and Grama, A. (2007). Macroprogramming heterogeneous sensor networks using cosmos. *EuroSys '07: Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, pages 159–172.

[Bakshi et al. 2005] Bakshi, A., Prasanna, V. K., Reich, J., and Larner, D. (2005). The abstract task graph: a methodology for architecture-independent programming of networked sensor systems. *EESR '05: Proceedings of the 2005 workshop on End-to-end, sense-and-respond systems, applications and services*, pages 19–24.

[Cervantes et al. 2008] Cervantes, H., Donsez, D., and Touseau, L. (2008). An architecture description language for dynamic sensor-based applications. In *Consumer Communications and Networking Conference, 2008. CCNC 2008. 5th IEEE*, pages 147–151.

[Crossbow 2004] Crossbow (2004). Micaz datasheet. Product folder.

[Fonseca et al. 2007] Fonseca, R., Gnawali, O., Jamieson, K., Kim, S., Levis, P., and Woo, A. (2007). *The Collection Tree Protocol (CTP) - TEP 123*. TinyOS 2.1, http://www.tinyos.net/tinyos-2.1.0/doc/html/tep123.html.

[Gay et al. 2003] Gay, D., Levis, P., von Behren, R., Welsh, M., Brewer, E., and Culler, D. (2003). The nesc language: A holistic approach to networked embedded systems. pages 1–11.

[Hui and Culler 2004] Hui, J. W. and Culler, D. (2004). The dynamic behavior of a data dissemination protocol for network programming at scale. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, SenSys '04, pages 81–94, New York, NY, USA. ACM.

[Kasten and Römer 2005] Kasten, O. and Römer, K. (2005). Beyond event handlers: programming wireless sensors with attributed state machines. In *IPSN '05: Proceedings of the 4th international symposium on Information processing in sensor networks*, page 7, Piscataway, NJ, USA. IEEE Press.

[Kothari et al. 2007] Kothari, N., Gummadi, R., Millstein, T., and Govindan, R. (2007). Reliable and efficient programming abstractions for wireless sensor networks. *PLDI '07: Proceedings of the 2007 ACM SIGPLAN conference on Programming language design and implementation*, pages 200–210.

[Levis and Culler 2002] Levis, P. and Culler, D. (2002). Maté: a tiny virtual machine for sensor networks. In *ASPLOS-X: Proceedings of the 10th international conference on Architectural support for programming languages and operating systems*, pages 85–95, New York, NY, USA. ACM.

[Levis et al. 2003] Levis, P., Lee, N., Welsh, M., and Culler, D. (2003). Tossim: accurate and scalable simulation of entire tinyos applications. In *Proceedings of the 1st international conference on Embedded networked sensor systems*, SenSys '03, pages 126–137, New York, NY, USA. ACM.

[Levis et al. 2004] Levis, P., Madden, S., Polastre, J., Szewczyk, R., Whitehouse, K., Woo, A., Gay, D., Hill, J., Welsh, M., Brewer, E., and Culler, D. (2004). Tinyos: An operating system for sensor networks. In *in Ambient Intelligence*. Springer Verlag.

[Lin and Levis 2008] Lin, K. and Levis, P. (2008). Data discovery and dissemination with dip. In *IPSN '08: Proceedings of the 7th international conference on Information processing in sensor networks*, pages 433–444, Washington, DC, USA. IEEE Computer Society.

[Madden et al. 2005] Madden, S. R., Franklin, M. J., Hellerstein, J. M., and Hong, W. (2005). Tinydb: an acquisitional query processing system for sensor networks. *ACM Trans. Database Syst.*, 30(1):122–173.

[Munawar et al. 2010] Munawar, W., Alizai, M. H., Landsiedel, O., and Wehrle, K. (2010). Dynamic tinyos: Modular and transparent incremental code-updates for sensor networks. In *Proceedings of the IEEE International Conference on Communications (ICC)*, Cape Town, South Africa, May 23-27.

[Newton et al. 2007] Newton, R., Morrisett, G., and Welsh, M. (2007). The regiment macroprogramming system. In *IPSN '07: Proceedings of the 6th international conference on Information processing in sensor networks*, pages 489–498, New York, NY, USA. ACM.

[Shnayder et al. 2004] Shnayder, V., Hempstead, M., Chen, B.-r., Allen, G. W., and Welsh, M. (2004). Simulating the power consumption of large-scale sensor network applications. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, SenSys '04, pages 188–200, New York, NY, USA. ACM.