Uma Máquina de Estados para Especificação de Códigos de Simulação para Redes de Sensores sem Fio Urbanas

Marcos Carrero[1,3], Katriny Zamproni[1], Martin A. Musicante[2], Aldri Santos[1], Carmem Hara[1]

[1]DINF – Universidade Federal do Paraná – UFPR – Paraná, Brasil

[2]DIMAp – Universidade Federal do Rio Grande do Norte – UFRN – Natal, Brasil

[3]FAE Centro Universitário – Paraná, Brasil

{macarrero,kz14,aldri,carmem}@inf.ufpr.br, mam@dimap.ufrn.br

Abstract. *Ubiquitous urban sensing systems face challenges related to the large number of sensors in the network and to the dynamicity of data generation. Individually, sensor activities are triggered in response to events. However, modeling the collaborative process requires the execution of functions that start not only in response to an event, but also based on a logical condition. This paper proposes a state machine with two types of transitions: event-based and logic-based. A case study which models a sensor storage system is presented. Its development using a framework of reusable components shows a correspondence between our machine and the code, which shows the machine usefulness for code development in simulation environments.*

Resumo. *Os sistemas ubíquos de sensoriamento urbano enfrentam desafios relacionados à grande quantidade de sensores na rede e à dinamicidade de geração de dados. Individualmente, os sensores reagem em resposta a eventos. No entanto, a modelagem do processo colaborativo exige mudanças de estado dos sensores não apenas como resposta a um evento, mas também por uma condição lógica. Este artigo propõe uma máquina de estados com dois tipos de transição: por evento e por lógica. Um estudo de caso que considera um modelo de armazenamento em sensores, desenvolvido com o apoio de um framework de componentes reusáveis, mostra uma correspondência entre a máquina proposta e o código, facilitando a sua implementação em ambientes de simulação.*

1. Introdução

O conceito de cidades inteligentes tem por objetivo prover serviços ubíquos para oferecer qualidade de vida e conforto para sua população. Ela é deslumbrada em diversas áreas de aplicação como alertas de congestionamentos de tráfego [Araújo et al. 2014], aplicações médicas [Cremonezi et al. 2017] e casas inteligentes [Filho et al. 2015]. Uma questão primordial aos serviços inteligentes consiste no desafio da coleta e disseminação de grande quantidade de dados nestes ambientes físicos e dinâmicos. As Redes de Sensores Sem Fio (RSSFs) têm oferecido essa infraestrutura para o desenvolvimento de diversos serviços para as redes urbanas em diferentes contextos de aplicação [Carrero et al. 2015].

Os sistemas ubíquos de sensoriamento urbano são tipicamente sistemas orientados a eventos [Muñoz and Leone 2017]. A fim de oferecer serviços ubíquos ao cidadão e ao

gestor público, as RSSFs podem ser usadas como infraestrutura para atender à grande demanda pelas aplicações de sensoriamento em diversos domínios. Dada a natureza reativa das RSSFs, a maioria dos simuladores existentes, dentre eles o NS2, OMNeT++ e TOSSIM, são baseados em eventos. É um modelo intuitivo de programação que associa eventos a ações, mas oferece um **baixo nível de abstração**. Assim, faz-se necessário o uso de ferramentas que reduzam a complexidade do desenvolvimento de sistemas ubíquos orientados a eventos, aumentado a produtividade e minimizando possíveis erros de projetos.

As máquinas de estados (MEs) são uma abordagem usada para tratar a complexidade do desenvolvimento de programas orientados a eventos, possibilitando abstrair detalhes de implementação [Desai et al. 2013]. Os estados são representações lógicas que descrevem o contexto atual de execução. As MEs são frequentemente usadas para especificar códigos para RSSFs, uma vez que os sensores, considerados individualmente, reagem a eventos como a obtenção de uma nova leitura ou o recebimento de uma mensagem. No entanto, sensores e outros dispositivos autônomos frequentemente realizam funções colaborativas, que exigem mudanças de estado dos dispositivos não apenas como resposta a um evento, mas também por uma condição lógica ou por um temporizador. Logo, a modelagem de sistemas com MEs torna-se mais complexa, dificultando sua implementação em simuladores orientados a eventos. Portanto, faz-se necessária uma abordagem de alto nível que associe de maneira clara modelos de máquinas de estados complexas, que tratam diferentes casos de mudanças de estados e que facilitem sua codificação.

Para reduzir o esforço e a complexidade do desenvolvimento de aplicações, [Cañete et al. 2011] propõem um modelo de componentes de software orientado a serviços para desenvolver sistemas complexos a partir de serviços mais simples. No entanto, o modelo proposto não é flexível para a criação de sistemas autônomos, como por exemplo, em aplicações que realizam a formação de agrupamentos na própria rede de sensores. Estratégias propostas por [Krämer et al. 2013] usam máquinas de estados para modelar sistemas orientados a eventos. No entanto, os estudos de caso mostram modelos de máquinas de estados simples, o que não é a realidade encontrada em cenários urbanos.

Este artigo propõe uma máquina de estados para fazer o detalhamento do fluxo de operações em um simulador baseado em eventos, criando dois tipos de transição: baseada em evento e baseada em lógica. Este detalhamento pode então ser utilizado para estruturar o código de simulação. Um estudo de caso descreve uma ME para o DCSSC (*Distributed Clustering Scheme based on Spatial Correlation in WSNs*) [Le et al. 2008], um modelo apropriado para o sensoriamento urbano, que realiza agrupamentos explorando a similaridade de dados. A ME foi implementada usando o *framework* RCBM, que oferece um conjunto de bibliotecas e componentes reusáveis para desenvolver sistemas de simulação para o NS2. O código do DCSSC desenvolvido com o apoio do *framework* RCBM mostrou uma correspondência direta entre os tipos de estados propostos na ME e o seu detalhamento no programa de simulação. Além disso, os resultados mostram uma reutilização de código de até 71.6%, o que demonstra o potencial da proposta para facilitar o desenvolvimento de novas aplicações para RSSFs e outras de natureza reativa e colaborativa, como os sistemas ubíquos e pervasivos.

O restante deste artigo está organizado da seguinte forma. A Seção 2 apresenta os trabalhos relacionados. A Seção 3 descreve o modelo de máquina de estados. A Seção 4 mostra um estudo de caso da aplicação da máquina em um modelo de armazenamento de

RSSF urbana. A Seção 5 apresenta os detalhes da implementação e os resultados obtidos na avaliação de desempenho. A conclusão e trabalhos futuros são discutidos na Seção 6.

2. Trabalhos Relacionados

Na literatura sobre técnicas de desenvolvimento de sistemas para as redes de sensores [Malavolta and Muccini 2014], verificou-se a inexistência de uma abordagem que trate da modelagem e implementação de uma maneira sistemática, requisitos necessários para atender à grande demanda por serviços ubíquos de grande escala. O trabalho proposto por [Krämer et al. 2013] descreve uma biblioteca de desenvolvimento de aplicativos para RSSFs orientada à máquina de estados. Um programa pode ser decomposto em módulos, cada um executando sua própria máquina de estados. A coordenação de vários módulos é feita pela máquina de estados do módulo principal. Embora a biblioteca promova modularidade e reusabilidade de código, a análise de desempenho da biblioteca não demonstra que sua arquitetura seja capaz de implementar sistemas escaláveis e autônomos, requisitos desejados para aplicações urbanas.

O modelo de ME descrita por [Cecílio and Furtado 2012] é composta por duas classes de estados: estados que tratam de requisitos funcionais e estados que tratam de requisitos não-funcionais. Por exemplo, coletar e encaminhar dados é um requisito funcional e reencaminhar dados quando a perda de pacotes for maior que 1% é um requisito não-funcional. Contudo, o foco deste artigo considera os requisitos funcionais encontrados em RSSFs urbanas enquanto a proposta deles lida com os requisitos não-funcionais de ambientes industriais. As estratégias usadas por Tokenit [Taherkordi et al. 2015] integram um ambiente de modelagem e implementação orientado à máquina de estados. O modelo define uma máquina de estados como um conjunto de atividades, que são operações executadas por um estado. As mudanças de estado ocorrem por eventos de *timers* ou eventos assíncronos. A detecção de um movimento, por exemplo, é um evento assíncrono. A máquina de estados é descrita em XML e o compilador do *framework* gera automaticamente código para a plataforma de execução do sistema operacional Contiki. Contudo, a avaliação de desempenho não aborda aspectos sobre a escalabilidade do sistema.

3. Uma Máquina de Estados Orientada a Eventos

Em RSSFs, os sensores podem desempenhar diversas funções, desde a participação no roteamento de mensagens em uma comunicação multi-salto até a atuação como um repositório de dados coletados do ambiente. Estas funcionalidades são em geral acionadas através do recebimento de mensagens, que podem ser modeladas como transições de uma máquina de estados (ME) orientada a eventos. No entanto, existe uma dicotomia entre o entendimento intuitivo de uma transição em uma máquina de estados e o seu funcionamento em uma RSSF, uma vez que a máquina de estados deve representar a funcionalidade de cada sensor *individualmente*. Logo, não há necessariamente uma correspondência entre o estado do sensor que envia uma mensagem e o estado do sensor que a recebe. Ou seja, um sensor pode enviar uma mensagem a partir de um estado e_1, mas o sensor que a recebe pode estar em um estado e_2, que não é necessariamente o próximo estado do sensor origem da mensagem. Assim, a mudança de estado por evento pode não estar necessariamente relacionada ao estado do sensor emissor e o envio de mensagens não necessariamente determina uma mudança de estado. Em suma, não há correspondência direta entre

transições na máquina e a comunicação entre sensores. No entanto, as mudanças de estado do sensor origem e destino são representadas na mesma máquina, que é executada em todos os sensores, como ilustrado pela Figura 1(b). Experiências anteriores de desenvolvimento de código de simulação para RSSF [Furlaneto et al. 2012, Gonçalves et al. 2012], mostraram que esta dicotomia dificulta o desenvolvimento de programas.

O objetivo da ME proposta neste artigo é oferecer suporte para o desenvolvedor especificar o comportamento do sistema antes de iniciar a implementação, através da criação de dois tipos de transição (por evento e lógica), além de uma notação para o envio de mensagens. A ME combina elementos de máquinas de estados finitos, como estados e transições, com princípios de programação reativa dos simuladores de RSSFs, que associam eventos à um conjunto de ações. As ferramentas visuais de modelagem de máquina de estados que seguem o padrão da *UML StateChart*, podem ser usadas para criar as MEs desenvolvidas neste artigo. Um exemplo desta máquina está ilustrada na Figura 1(a). Cada estado possui um nome, como `Ini` e `Wait_First_Sensor_ID`, e pode possuir uma anotação (entre colchetes) sobre o envio de mensagens ou inicialização de temporizadores. As transições podem ser por evento (representadas em azul) ou lógica (representadas em vermelho):

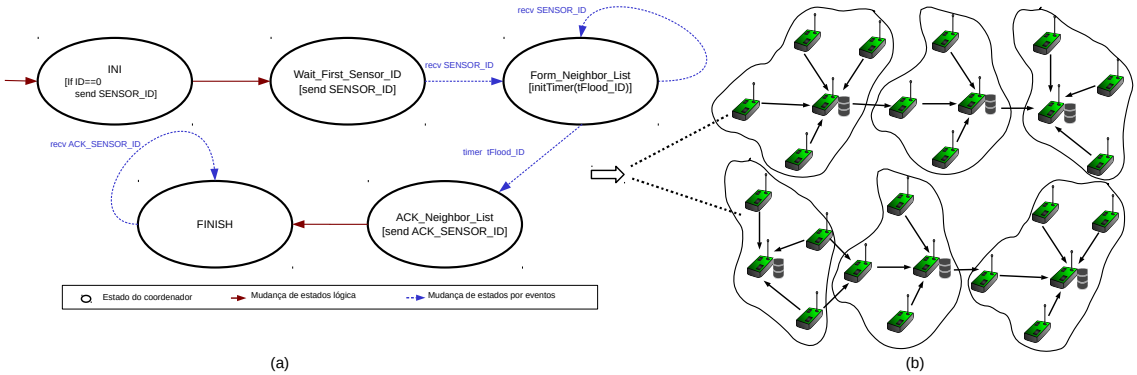

Figura 1. Um modelo de máquina de estados para descoberta de vizinhos

- Mudança de estado por evento: são aquelas que ocorrem quando um sensor recebe algum tipo de mensagem ou então após o tempo de expiração de um temporizador.
- Mudança de estado lógica: é realizada com base no resultado de uma computação.

O exemplo da Figura 1(a) descreve o algoritmo de descoberta de vizinhos por inundação na rede. No estado inicial `INI`, o sensor com identificador (`ID`) zero envia a mensagem `SENSOR_ID` para seus vizinhos. O sensor zero e os demais sensores realizam uma mudança lógica do estado `INI` para o estado `Wait_First_SENSOR_ID`. Os sensores que receberem a primeira mensagem `SENSOR_ID` armazenam o `ID` contido na mensagem e enviam seu identificador para seus vizinhos. Em seguida, realizam uma mudança por evento do estado `Wait_First_SENSOR_ID` para o estado `Form_Neighbor_List`. No estado `Form_Neighbor_List`, os sensores continuam armazenando o `ID` dos demais vizinhos durante um certo tempo t_{Flood_ID}. Quando o tempo expirar, os sensores fazem uma mudança para o estado `ACK_Neighbor_List` e enviam uma mensagem de `ACK` para os sensores conhecidos. Após o envio da mensagem, os sensores realizam mudança lógica para o estado `FINISH`, armazenando as mensagens de `ACK` recebidas. Ao final da inundação, todos os sensores conhecem seus vizinhos.

Este é um exemplo bastante simples, no qual algumas características de RSSFs não ficam tão evidentes. Elas advêm da natureza distribuída das RSSFs, na qual cada sensor executa a máquina de estados de forma independente. Dessa forma, a modelagem destes sistemas, que frequentemente realizam funções colaborativas assíncronas, na qual múltiplas máquinas podem estar em estados distintos, não é uma tarefa trivial. A próxima seção apresenta um estudo de caso que salienta estas características e mostra a efetividade da máquina proposta para a representação do modelo.

4. Estudo de Caso

Um estudo de caso foi desenvolvido para demonstrar o uso da máquina de estados em outros contextos. Como o foco é em aplicações urbanas, foi desenvolvido um estudo de caso a partir do modelo de armazenamento em RSSFs chamado *Distributed Clustering Scheme based on Spatial Correlation in WSNs* (DCSSC) [Le et al. 2008]. O algoritmo proposto constrói e mantém os *clusters* de forma distribuída e dinâmica. A abordagem de agrupamento por similaridade de dados em redes de larga escala torna o DCSSC um modelo apropriado para o sensoriamento urbano.

4.1. O Modelo DCSSC

No modelo DCSSC, cada sensor comunica-se apenas com os sensores que estão à distância de um salto e todos têm os dados relativos ao nível de energia de seus vizinhos. Esta informação é enviada por meio de mensagens do tipo HELLO trocadas periodicamente. O modelo da rede consiste em uma estação-base e N sensores. São atribuídos estados a todos os dispositivos, que determinam seu papel na rede. No início, os sensores estão no estado INI e ao final da fase de construção de agrupamentos eles estarão em um dos seguintes estados: CH (*cluster-head*), GW (*gateway*), EXT (*cluster-extend*) ou MEM (*member*). Os sensores com os estados CH, GW e EXT ficam nesse estado até o fim da fase de construção e são chamados de nodos de *backbone*. Existem também estados temporários, como o INI, GWR (*gateway-ready*) e CHC (*cluster-head candidate*).

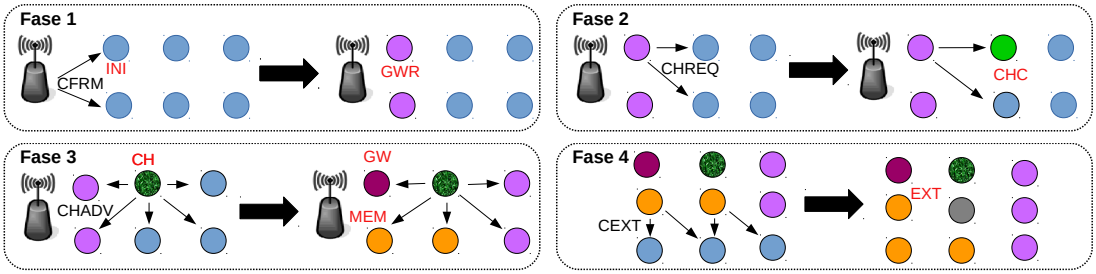

Figura 2. Fases do algoritmo DCSSC

As mensagens de formação de agrupamento incluem sempre a média de valores das leituras aferidas no tempo que precede o envio da mensagem. Baseado no tipo da mensagem, o receptor irá mudar seu estado, criar uma nova mensagem e propagar para seus vizinhos. A nova mensagem também deve conter o identificador (ID) do emissor original da mensagem recebida. A Figura 2 ilustra as fases da simulação que inicia com a construção de agrupamentos, determinada pelos passos que seguem:

Fase 1: A estação-base inicia a formação de agrupamentos fazendo *broadcast* de uma

mensagem do tipo CFRM (*Cluster Formation Message*). Cada sensor que recebe essa mensagem e está no estado INI muda seu estado para GWR, criando dois *timers*, t_{req} e t_{wait}, escolhidos aleatoriamente. Quando o tempo definido por t_{req} expira, sensores no estado GWR fazem o envio de uma mensagem CHREQ (*Cluster Head Request*) para seus vizinhos a fim de encontrar um *cluster-head*.

Fase 2: Um sensor INI ao receber a mensagem CHREQ calcula sua similaridade com o sensor GWR que enviou a mensagem. Os sensores que forem altamente correlacionados mudam seu estado para CHC (candidatos a *cluster-head*). Os nodos CHC criam um novo *timer* t_{adv}, baseado no seu nível relativo de energia. Um sensor CHC se declara como CH fazendo *broadcast* da mensagem CHADV (*Cluster Head Adverstisement*) quando seu timer t_{adv} expirar. Como t_{adv} é inversamente proporcional ao seu nível de energia, sensores com maiores níveis de energia têm maior chance de serem eleitos como líder.

Fase 3: Qualquer sensor em um estado temporário (INI, GWR, CHC), ao receber uma mensagem CHADV, calcula a similaridade com o sensor que enviou a mensagem. Se eles forem altamente correlacionados, o receptor se torna membro (estado MEM) do *cluster* formado por aquele CH. Do contrário, ele vai para o estado GWR, seguindo os mesmos passos definidos anteriormente para um sensor no estado GWR. Todo sensor altamente correlacionado compara o ID da fonte original da mensagem recebida com o seu próprio ID. Se forem iguais, ou seja, o CHADV recebido foi uma resposta ao seu próprio CHREQ, o sensor no estado GWR altera seu estado para GW. Quando o *timer* t_{wait} expira, o sensor no estado GWR muda seu estado para CHC se não houver nenhuma mensagem CHADV proveniente dos seus vizinhos.

Fase 4: Os sensores nos estados GW e MEM propagam a mensagem de formação enviando mensagens de *broadcast* do tipo CEXT (*Cluster Extend*), que incluem a média das leituras do CH que originou a mensagem, e não do sensor atual. Após receber uma mensagem CEXT, todo sensor em estado temporário (INI, GWR, CHC) calcula a sua similaridade com relação ao CH. Se forem altamente correlacionados, ele integra o *cluster* formado e se torna um membro (estado MEM), fazendo com que o transmissor da mensagem CEXT vá para o estado EXT. Caso não seja altamente correlacionado, ele vai para o estado GWR e repete os passos atribuídos a ele. O modelo DCSSC prioriza como líderes de agrupamentos os sensores com maior nível relativo de energia dentre os que apresentam leituras similares, prolongando a vida útil da rede. Os nodos com os papéis de CH, GW e EXT (nodos de *backbone*) têm a função de coletar dados das leituras dos sensores, que enviam seus dados através do *backbone* a cada intervalo de tempo segundo o esquema de escalonamento *Round-Robin*. As leituras recebidas são armazenadas no CH, e podem ser comprimidas a fim de reduzir o espaço necessário de armazenamento. Os dados agregados são transmitidos do CH para o GW do respectivo *cluster*, que encaminha para um sensor vizinho que pertença a outro *cluster*. Dessa forma os dados trafegam na rede através de nodos intermediários, até chegar à estação-base, com menor custo de comunicação.

4.2. A Máquina de Estados do modelo DCSSC

A máquina de estados foi aplicada para descrever a especificação formal da coordenação do fluxo de execução do modelo DCSSC. O fluxo de execução do DCSSC segue uma máquina de estados como representado na Figura 3(b). Todos os nodos iniciam no estado INI, e modificam seu estado ao longo da execução. Neste exemplo, as características distintas da máquina proposta ficam mais evidentes. Por exemplo, todos os sensores iniciam no estado INI e a mensagem CFRM é transmitida por *broadcast* pela estação base. Aque-

les que a recebem passam para o estado GWR, que esperam por um tempo aleatorizado t_{req} para passar para o estado TREQ e enviar uma mensagem CHREQ. Os sensores que recebem esta mensagem estão no estado INI e passam ao estado selectCH, no qual a similaridade de dados com o sensor emissor é calculado para determinar se ele passa ao estado CHC. Caso as condições para mudança de estado para CHC não sejam satisfeitas, o sensor retorna ao estado INI para que possa receber mensagens de outros sensores.

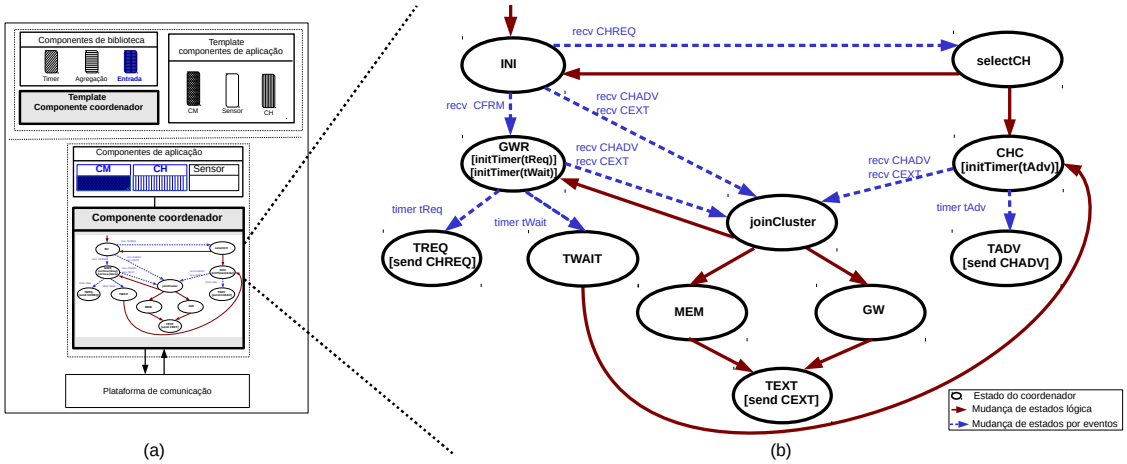

Figura 3. Visão do *framework* RCBM e da máquina de estados do DCSSC

A próxima seção descreve a implementação da máquina de estados do modelo DCSSC. O *framework* RCBM [Carrero et al. 2017], ilustrado pela Figura 3(a), descreve uma metodologia para o desenvolvimento de sistemas de armazenamento para RSSFs orientado a componentes. O RCBM oferece suporte para atender à grande demanda por serviços e aplicações pois possibilita a reusabilidade de código, facilitando o desenvolvimento de novos modelos. O RCBM é um metamodelo que utiliza entidades, propriedades e funções para descrever o que seriam as instâncias desse metamodelo, ou seja, os diversos modelos de armazenamento. No RCBM, cada entidade é associada a um conjunto de componentes que implementam as funcionalidades comuns aos modelos. Por exemplo, para a máquina de estados ilustrada pela Figura3(b), as entidades são os próprios sensores, os agrupamentos formados por *cluster members*(CM) e os líderes do grupo (CH).

5. Implementação

O RCBM é um modelo baseado em componentes reutilizáveis para a simulação de modelos de armazenamento para RSSFs. O RCBM foi implementado no simulador NS2 com o objetivo de promover a reutilização de códigos de simulação a partir de componentes existentes. A arquitetura do RCBM, ilustrada pela Figura 3(a), é composta por três camadas: especificação, implementação e comunicação. O nível superior descreve a camada de especificação, composta pelos componentes de biblioteca, *template* do coordenador e *templates* dos componentes de aplicação. Esses elementos são utilizados da forma como foram especificados, sem a necessidade de alterações realizadas pelo usuário. No nível inferior, estão os componentes de aplicação e o componente coordenador, que de fato serão construídos pelo desenvolvedor, seguindo as definições da camada de especificação. A camada de comunicação é a responsável por interligar as camadas de especificação e de implementação, fornecendo a infraestrutura de comunicação entre os componentes.

O *framework* RCBM já possui a definição de interfaces de funções para os componentes CH e CM, nos quais os estados `selectCH` e `joinCluster` da Figura 3(b), foram adaptados para o modelo DCSSC. No entanto, o desenvolvimento do código de coordenação entre os componentes é responsabilidade do usuário. A partir da máquina de estados ilustrada na Figura 3(b), o componente coordenador, que implementa os diversos estados e transições do modelo RCBM, foi desenvolvido através de duas funções principais no NS2: `recv` e `TimerHandle`. A função `recv` é responsável pelo recebimento de pacotes pelos sensores e trata essas mensagens. É a parte do código que faz as transições do estado do tipo "mudança por evento" e também faz a chamada para os componentes externos. O `recv` é composto por um comando *switch*, com vários *cases*, cada um equivalente a um estado que pode ser atingindo pelas mudanças de estado por evento. Cada mensagem possui um ID, e dentro dela existem alguns parâmetros como o ID do emissor, suas coordenadas e leituras, além do ID do emissor da última mensagem recebida pelo sensor. A função `TimerHandle`, é executada quando existe uma transição ativada quando um *timer* expira e uma nova rodada (ou *round*) é iniciada. O `TimerHandle` cria pacotes e os envia conforme o tipo da rodada atual. Os detalhes da implementação das funções `recv` e `TimerHandle` podem ser verificados no código fonte[1] da implementação do DCSSC.

Analisando o código do NS2, verifica-se que há uma correspondência direta entre os tipos de transição propostos na máquina de estados e o seu detalhamento no componente coordenador do programa de simulação. Esta distinção conceitual de tipos de transição facilita o desenvolvimento de código, mantendo a reutilização promovida pelo *framework* RCBM, como mostram os experimentos detalhados na próxima seção. É importante ressaltar que o componente coordenador deve ser implementado pelo programador e que sua função é fazer a junção dos componentes, coordenando as interações entre eles. O RCBM é mais flexível do que modelos baseados em componentes anteriores para RSSFs, porém possui como limitação a falta de uma especificação formal para a coordenação do fluxo de execução dos componentes. A máquina de estados proposta na Seção 3 supre esta deficiência, com uma proposta para especificar o fluxo de operações do coordenador, facilitando sua implementação.

5.1. Avaliação

Essa seção mostra a avaliação do sistema DCSSC, desenvolvido a partir de técnicas que integram reuso de componentes de software e máquina de estados que descrevem o comportamento dinâmico do sistema. Foram realizados dois experimentos. No primeiro foi usada a métrica de contagem de linhas de código (LOC) para analisar a porcentagem do código que foi possível reutilizar do modelo RCBM. No segundo, analisou-se a métrica CBO para avaliar o grau de acoplamento entre os componentes do sistema.

Reutilização de código: Para analisar a eficácia do modelo RCBM durante o desenvolvimento de código, foram contabilizadas as linhas de código necessárias para implementar o DCSSC. A Tabela 1 apresenta o número total das linhas de código, quantas delas foram criadas, quantas foram reutilizadas e a porcentagem equivalente à reutilização. Para implementar o DCSSC, foi adicionada uma nova biblioteca ao RCBM, que inicializa as leituras dos sensores. O número de linhas equivalentes à implementação do componente

[1]O código completo está disponível em `www.inf.ufpr.br/macarrero/SBCUP2018`.

que inicializa as leituras dos sensores foi inserido na contagem A como linhas novas e na contagem B como linhas reutilizadas. Considerando que o componente que inicializa as leituras dos sensores não existia no modelo RCBM e foi criado para o desenvolvimento deste estudo, seu código pode ser classificado como novo. Entretanto, levando-se em consideração implementações futuras, esse componente pode ser considerado reutilizável, justificando as linhas de sua implementação inseridas como linhas reutilizadas.

Contagem	**Total de linhas**	**Linhas reutilizadas**	**Linhas novas**	**Porcentagem de reuso**
A	1333	905	428	67,9%
B	1333	954	379	71,6%

Tabela 1. Proporção de linhas reutilizadas do RCBM

CBO: Na avaliação do coeficiente de dependência entre os componentes foi aplicada a métrica ***C**oupling **B**etween **O**bject **C**lasses* (CBO), cujo valor está entre 0 e 1. Quanto mais próximo de 1 for o CBO de um certo componente, mais dependente ele é de outras classes. O resultado da avaliação é apresentado na Tabela 2. Percebe-se que o coordenador é altamente dependente de outros componentes, o que é esperado uma vez que ele é o responsável por fazer a junção de todos os componentes. Os componentes de biblioteca por sua vez não são dependentes de nenhum outro componente, e isso garante que seu uso seja independente da implementação do usuário. Os outros componentes utilizados possuem um baixo grau de acoplamento, favorecendo a depuração do código e a correção de erros nos módulos, além de facilitar a inclusão de novos componentes ao RCBM.

Componente	Coordenador	Componentes de biblioteca	Sensor	CM	CH
CBO	1	0	0,33	0,33	0,33

Tabela 2. Análise da métrica CBO para cada componente do modelo DCSSC

6. Conclusão

Este trabalho apresentou uma proposta de máquina de estados para fazer o detalhamento do fluxo de operações em um simulador baseado em eventos. Através de um estudo de caso e a sua codificação no simulador NS2, mostrou-se que há uma correspondência direta entre a máquina proposta e o programa desenvolvido. Os experimentos demonstram que o percentual de reutilização de código utilizando o RCBM para a implementação do DCSSC foi de até 71,6%, considerando o componente de biblioteca como reutilizável, e de até 67,9%, considerando o componente de biblioteca criado como linhas de código novas. Os valores são significativos considerando a complexidade dos códigos de simulação para RSSFs. Como trabalhos futuros podem ser citados: o desenvolvimento de uma linguagem baseada em máquina de estados para a geração automática do coordenador, a realização de estudos experimentais adicionais envolvendo outras métricas e utilização da abordagem de reutilização em outros simuladores, como por exemplo o *Network Simulator 3* (NS3).

Referências

Araújo, G., Tostes, A., de LP Duarte-Figueiredo, F., and Loureiro, A. (2014). Um protocolo de identificação e minimização de congestionamentos de tráfego para redes vei-

culares. *XXXII Simp. Bras. de Redes de Computadores e Sistemas Distribuídos*, pages 207–220.

Cañete, E., Chen, J., Díaz, M., Llopis, L., and Rubio, B. (2011). A service-oriented approach to facilitate wsan application development. *Ad Hoc Networks*, 9(3):430–452.

Carrero, M. A., da Silva, R. I., dos Santos, A. L., and Hara, C. S. (2015). An autonomic in-network query processing for urban sensor networks. In *20th IEEE Symp. on Computers and Communications (ISCC)*, pages 968–973.

Carrero, M. A., Musicante, M. A., dos Santos, A. L., and Hara, C. S. (2017). A reusable component-based model for WSN storage simulation. In *Proceedings of the 13th ACM Symposium on QoS and Security for Wireless and Mobile Networks*, pages 31–38.

Cecílio, J. and Furtado, P. (2012). A state-machine model for reliability eliciting over wireless sensor and actuator networks. *Procedia Computer Science*, 10:422–431.

Cremonezi, B. M., Vieira, A. B., Nogueira, M., and Nacif, J. A. (2017). Um protocolo de alocação dinâmica de canais para ambientes médicos sob múltiplas estações base. In *XXXV Simp. Bras. de Redes de Computadores e Sistemas Distribuídos*, pages 272–285.

Desai, A., Gupta, V., Jackson, E., Qadeer, S., Rajamani, S., and Zufferey, D. (2013). P: safe asynchronous event-driven programming. *ACM SIGPLAN Notices*, 48(6):321–332.

Filho, G. P., Ueyama, J., Faiçal, B. S., Guidoni, D. L., and Villas, L. A. (2015). Residi: Um sistema de decisão inteligente para infraestruturas residenciais via sensores e atuadores sem fio. In *XXXIII Simp. Bras. de Redes de Computadores e Sistemas Distribuídos*, pages 53–66.

Furlaneto, S. S., dos Santos, A. L., and Hara, C. S. (2012). An efficient data acquisition model for urban sensor networks. In *13th IEEE/IFIP Network Operations and Management Symposium (NOMS)*, pages 113–120.

Gonçalves, N. M., dos Santos, A. L., and Hara, C. S. (2012). Dysto-a dynamic storage model for wireless sensor networks. *Journal of Information and Data Management*, 3(3):147–162.

Krämer, M., Bader, S., and Oelmann, B. (2013). Implementing wireless sensor network applications using hierarchical finite state machines. In *10th IEEE Int. Conf. on Networking, Sensing and Control*, pages 124–129.

Le, T. D., Pham, N. D., and Choo, H. (2008). Towards a distributed clustering scheme based on spatial correlation in wsns. In *International Wireless Communications and Mobile Computing Conference*, pages 529–534.

Malavolta, I. and Muccini, H. (2014). A study on mde approaches for engineering wireless sensor networks. In *40th EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO-SEAA)*, pages 149–157.

Muñoz, C. and Leone, P. (2017). A distributed event-based system based on compressed fragmented-iterated bloom filters. *Future Generation Computer Systems*, 75:108–127.

Taherkordi, A., Johansen, C., Eliassen, F., and Römer, K. (2015). Tokenit: Designing state-driven embedded systems through tokenized transitions. In *2015 Int. Conf. on Distributed Computing in Sensor Systems (DCOSS)*, pages 52–61.