Desenvolvimento de Modelos de Armazenamento em Sensores com Reutilização de Código

Alexandre R. Ordakowski[1], Marcos A. Carrero[3], Martin A. Musicante[2], Aldri L. dos Santos[1], Carmem S. 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

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

Resumo. *Sensores são componentes essenciais da Internet das Coisas (IoT). Em consonância com a tendência de armazenar dados próximos às fontes de dados, preconizada pela computação de borda e de névoa, alguns sensores podem desempenhar o papel de repositório dos dados capturados. Dado o crescimento da quantidade de dispositivos e de aplicações para IoT, há a necessidade de investigar e desenvolver novos modelos de armazenamento para sensores. No entanto, poucos trabalhos propõem uma abordagem que trate da modelagem e implementação de sistemas de armazenamento e consulta dos dados da rede de uma maneira sistemática. Este artigo investiga a aplicação de um modelo de componentes para a geração de código para os dispositivos sensores, com a elaboração do framework RCBM-S (RCBM for Sensor devices). No RCBM-S a orquestração dos componentes é baseada em máquinas de estados. Um estudo de caso mostra a reutilização de código promovida pelo RCBM-S para códigos em nesC para o sistema TinyOS.*

1. Introdução

Os sensores são componentes computacionais de coleta de dados essenciais ao provimento da Internet das Coisas (IoT). Previsões apontam que a IoT, em suas diversas formas de redes, atingirá a marca de 20 bilhões de objetos conectados em 2020[1]. Em razão do grande volume de dados trocados, o armazenamento eficiente da enorme quantidade de dados gerados por estes dispositivos tem sido um tópico de investigação de vários trabalhos recentes. Uma técnica comum, preconizada pela computação de borda e de névoa, é processar e armazenar os dados em uma extremidade próxima da fonte de dados. Em consonância com esta tendência, é possível designar alguns dispositivos sensores para desempenhar o papel de repositório dos dados capturados. Em razão do crescimento da quantidade de dispositivos, bem como novas aplicações para IoT, surge a necessidade de investigar e desenvolver novos modelos de armazenamento voltado para as redes de sensores sem fio (RSSF), que são componentes fundamentais da IoT.

Os sensores são dispositivos com recursos normalmente limitados, tanto de bateria, como de processamento e armazenamento. Assim, os sistemas desenvolvidos para as RSSFs buscam considerar as características específicas da aplicação a fim de explorar

[1]https://www.informationweek.com/mobile/mobile-devices/gartner-21-billion-iot-devices-to-invade-by-2020/d/d-id/1323081

seus escassos recursos. Ou seja, cada aplicação requer uma análise e desenvolvimento de modelos de armazenamento específicos. Contudo, o desenvolvimento de tais sistemas é muitas vezes complexo. Observa-se que poucos trabalhos na literatura propõem uma abordagem que trate da modelagem e implementação de sistemas de armazenamento e consulta dos dados da rede de uma *maneira sistemática*, que são requisitos necessários para atender à grande demanda por serviços ubíquos de grande escala. Estas questões foram tratadas por [Carrero et al. 2017] através do desenvolvimento do framework RCBM, para apoiar o desenvolvimento de *códigos de simulação* para RSSFs.

Este artigo investiga a aplicação de técnicas similares para a geração de código para implantação nos *dispositivos sensores*, com a elaboração do framework RCBM-S (*RCBM for Sensor devices*). Ao contrário do RCBM, que foca no simulador de redes NS-2, a linguagem utilizada pelo RCBM-S é o nesC. O nesC é atualmente uma das linguagens de programação mais empregadas para RSSFs [Gay et al. 2014] e foi desenvolvido para o sistema operacional (SO) TinyOS. O RCBM-S promove a reutilização de código com a especificação de componentes, cuja orquestração é expressa por uma máquina de estados. Esta máquina permite que o fluxo de execução da aplicação seja modelada com um alto nível de abstração.

O restante do artigo está organizado da seguinte forma: a Seção 2 apresenta trabalhos relacionados; o framework RCBM-S e o estudo de caso que determina o porcentual de reuso obtido são apresentados nas Seções 3 e 4, respectivamente; a Seção 5 finaliza o artigo apresentando trabalhos futuros.

2. Trabalhos Relacionados

Grande parte dos modelos de desenvolvimento propostos que são para sensores levam em conta máquinas de estados, dada a natureza dos dispositivos, baseados em eventos. Dentre eles podem ser citados o Tokenit [Taherkordi et al. 2015] e o *Communicating X-Machine* (CXM) [de Lima Braga et al. 2010]. As estratégias usadas pelo Tokenit integram um ambiente de modelagem e implementação, no qual a máquina de estados é descrita no formato XML e o compilador do *framework* gera automaticamente código para a plataforma de execução do SO Contiki. Já o *Communicating X-Machine* propõe a definição de *X-Machines Stand-Alone* (isoladas) para a modularização da aplicação, e o uso do método formal CXM para promover a troca de mensagens entre os módulos.

O Wiselib [Baumgartner et al. 2010] é um *framework* para o desenvolvimento de aplicações para RSSF a partir de *templates* genéricos que geram código para diferentes plataformas de dispositivos, como o TinyOS e Contiki. Apesar de fornecer um ambiente de desenvolvimento favorável à reutilização de aplicações para plataformas heterogêneas, o Wiselib não fornece um arcabouço para o reaproveitamento de códigos de aplicações distintas. O RCBM (*Reusable Component-Based Model*) [Carrero et al. 2017] é um modelo baseado em componentes, desenvolvido para sistemas de armazenamento em RSSF em ambientes de *simulação*. Ele promove a reutilização de componentes de software e adota uma máquina de estados com transições baseadas em eventos e lógica para a especificação do fluxo da aplicação. O objetivo do RCBM-S é adotar o mesmo formalismo para o desenvolvimento de códigos para *dispositivos sensores*. Assim, a mesma especificação pode ser usada para desenvolver o código para a validação por simulação, e para a implantação do código em dispositivos para ambientes de sensoriamento reais.

Dentre os trabalhos relacionados, o Wiselib e o X-machine focam no desenvolvimento de código em múltiplas plataformas. No entanto, eles consideram diferentes plataformas de dispositivos sensores e não em um *framework* que atenda tanto a simulação para a validação quanto o desenvolvimento para dispositivos sensores. É possível notar que o RCBM-S contém diversas semelhanças com a ferramenta *Communicating X-Machine*. Entretanto, além do RCBM-S abranger mais plataformas e gerar códigos para nesC e OTcl, ele também utiliza um método formal diferente do utilizado pelo *Communicating X-Machine*. Ambas as ferramentas focam na utilização de máquinas de estado que se comunicam e realizam a transições por eventos, porém o RCBM-S propõe a transição de estados lógica, que é resultante do processamento de informações no estado atual.

3. O Framework RCBM-S

O RCBM-S é um framework para dar suporte ao desenvolvimento de modelos de armazenamento de dados em sensores baseado em componentes, cujo fluxo de execução é especificado por uma máquina de estados. A Figura 1 (a) apresenta a arquitetura do RCBM-S. Embora a arquitetura seja similar à proposta para o RCBM [Carrero et al. 2017], ela foi completamente re-implementada para o sistema TinyOS, na linguagem nesC. A arquitetura tem 3 camadas: especificação (bloco superior), implementação (bloco intermediário) e comunicação (bloco inferior). O objetivo de separar a especificação da implementação é promover a reutilização de código, com a criação de componentes conectáveis, desde que sua interface seja mantida. A camada de comunicação fornece a infraestrutura de comunicação entre os dispositivos. No caso das RSSFs a comunicação é realizada via rádio. Podem ser identificados três tipos de componentes: (i) componentes de biblioteca, que implementam funcionalidades úteis para os modelos, como funções de agregação; (ii) componentes de aplicação, que compõem o modelo de armazenamento e possuem um conjunto de funções que podem ser chamadas por outros componentes ou pelo coordenador; (iii) coordenador: controla o fluxo de execução do sistema, implementando uma máquina de estados.

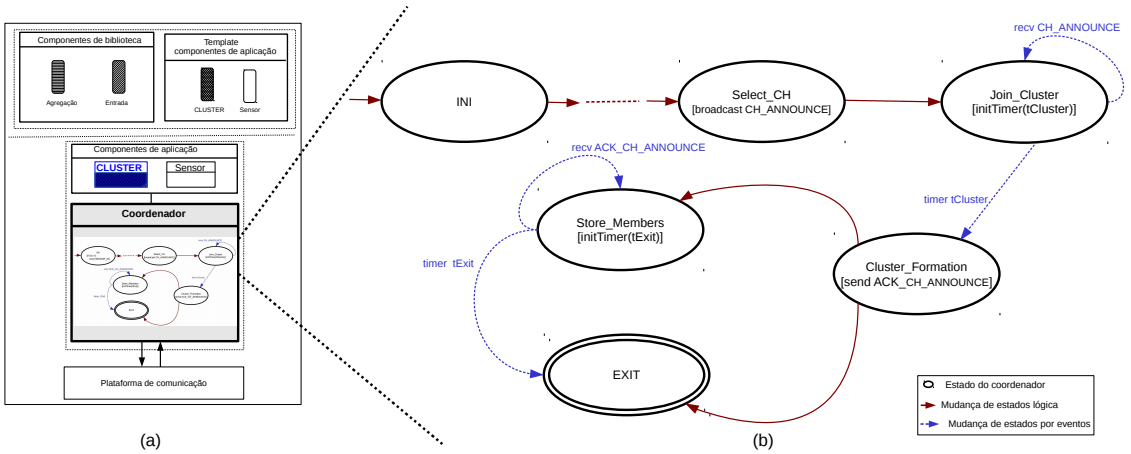

Figura 1. Arquitetura do RCBM-S

Componentes de Biblioteca. Os componentes de biblioteca fornecem funções genéricas e úteis para o desenvolvimento de aplicações. A linguagem nesC já provê alguns, como o componente `Timer`, que fornece um cronômetro que pode ser usado para gerar eventos

em intervalos de tempo regulares. Mas é possível definir novos componentes, como o de funções de agregação, cuja interface é apresentada na Listagem 1. Este componente foi desenvolvido para trabalhar com conjuntos de dados retornando um único valor, como o máximo (max), o mínimo (min), a média (avg) ou a soma (sum) dos seus elementos.

```
1 interface Aggregation {
2     command int max( double val[ ] );
3     command int min( double val[ ]  ] );
4     command double avg( double val[ ] );
5     command int sum( double val[ ] ); }
```

Listing 1. Interface do Componente de Biblioteca de Agregação.

Componentes da Aplicação. Os componentes de aplicação estão diretamente ligados às entidades do seu domínio. Tendo em vista que o RCBM-S foca em modelos de armazenamento de repositório, os componentes de aplicação incluem: (i) dispositivo sensor; e (ii) *cluster*, que é formado por um conjunto de sensores (*cluster-members* - CM) e um sensor que desempenha o papel de *cluster-head* (CH). O CH é responsável por armazenar informações de todo o *cluster*, bem como responder as requisições de consultas. A Listagem 2 apresenta a interface para a entidade *cluster*. São definidas duas funções: a eleição do CH (`selectCH`) e a associação de um membro a um *cluster* (`join`).

```
1 interface Cluster {
2   command int selectCH( int neighbors[] );
3   command int join( int candidates[] );}
```

Listing 2. Template do componente *Cluster*.

Coordenador. O coordenador define o fluxo de execução da aplicação, que é baseado em máquinas de estados, como apresentado na Figura 1 (b). Esta máquina possui transições lógicas (oriundas de um processamento lógico dentro de algum estado) e transições por eventos (que acontecem de maneira reativa na ocorrência de um evento) [Carrero et al. 2018]. O exemplo ilustra parte do processo de formação de *clusters* para o armazenamento de dados no modelo repositório. Na implementação atual do RCBM-S, o coordenador é implementado por funções que tratam as mudanças de estado por um `Timer` (`timer.fired`), o recebimento de mensagens (`receive`) e os estados.

A Listagem 3 apresenta as funções `timer.fired` e `receive`, correspondentes à máquina de estados da Figura 1. O `timer.fired` é um evento disparado ao final de um `Timer`. Por exemplo, no estado *Join_Cluster* é iniciado um `Timer` *tCluster* que, quando finalizado, promove uma transição por evento para o estado *Cluster_Formation*. Isto correponde às Linhas 2-3 da Listagem 3. O evento **receive** trata do recebimento de mensagens. Na Listagem 3, ao receber uma mensagem, é realizado seu desempacotamento e a verificação do tipo da mensagem (Linhas 9 e 11). Embora neste exemplo, a ação seja o armazenamento do conteúdo da mensagem, uma possível ação é acionar uma transição de estados.

```
1 event void Timer0.fired(){
2     if (currentState == JOIN_CLUSTER){
3      state_Cluster_Formation();
4     } else if (currentState == STORE_MEMBERS){
5      state_Exit();}}
6
```

```
7  event message_t* receive(message_t* msg, void* content){
8      GENERAL_MSG* btrpkt = (GENERAL_MSG*)content;
9      if (btrpkt->msgId == CH_ANNOUNCE) {
10         storeMSG(CANDIDATE, btrpkt->sensorId);
11     } else if (btrpkt->msgId == ACK_CH_ANNOUNCE) {
12         storeMSG(MEMBER, btrpkt->sensorId); }}
```

Listing 3. Implementação do coordenador.

Cada estado da máquina corresponde a uma função do coordenador. A Listagem 4 apresenta como exemplo o estado `state_Cluster_Formation`. Ela chama a função `join` do componente `Cluster` que determina qual o CH para o sensor (Linha 3). Caso o CH seja o próprio sensor, ele faz uma transição lógica para o estado `state_Store_Members` (Linhas 4-5). Caso contrário, o sensor transmite a mensagem `ACK_CH_ANNOUNCE` informando que será membro do cluster com líder CH) e muda para o estado `state_Exit`(Linhas 7-8).

```
1  void state_Cluster_Formation(){
2      currentState = CLUSTER_FORMATION;
3      CH = call Cluster.join(candidates_CH);
4      if (CH == TOS_NODE_ID ){
5          state_Store_Members();
6      } else {
7          sendMsg(ACK_CH_ANNOUNCE, CH);
8          state_Exit(); }}
```

Listing 4. Implementação de um estado.

4. Estudo de Caso

Para determinar a efetividade do RCBM-S na promoção da reutilização de código, foram implementados dois modelos de armazenamento em sensores baseados em *cluster*: LCA e LEACH. A Tabela 1 apresenta os resultados obtidos. Como os modelos usam o mesmo fluxo de execução para a formação de *clusters* (apresentado na Figura 1), seus coordenadores são praticamente idênticos, com uma semelhança de mais de 95% das linhas de código. Já na implementação dos componentes, a semelhança é em torno de 75%, visto que os modelos aplicam diferentes critérios para a seleção de CHs e criação de *clusters*.

O LCA elege como CH o nó com o menor identificador único (ID) dentre seus vizinhos e na fase de formação dos *clusters*, os nós não eleitos se unem ao CH no seu alcance com o menor ID. Já o LEACH elege o CH de maneira probabilística, dividida em rodadas, tal que haja uma alternância de CHs e cada sensor do *cluster* seja eleito um número de vezes semelhante. Para a formação dos *clusters*, os sensores se juntam ao CH que tem a maior intensidade de sinal das mensagens de anúncio de CH recebidas. Dadas estas particularidades, grande parte das linhas de código que diferem na implementação dos modelos deve-se ao código das funções `selectCH` e `join`.

O estudo de caso mostra que o desenvolvimento baseado em componentes facilita o desenvolvimento dos modelos, delimitando de forma clara os trechos de código que precisam ser alterados para o desenvolvimento de novos modelos de armazenamento. Estes resultados comprovam que o framework RCBM-S pode promover um grande ganho na reutilização e estruturação do código-fonte de aplicações voltadas ao armazenamento de dados em dispositivos sensores.

Tabela 1. Reuso de código na implementação dos modelos.

	Componentes			Coordenador		
Modelo de Sistema	**Total de Linhas**	**Qtd. Linhas Idênticas**	**% Linhas Idênticas**	**Total de linhas**	**Qtd. Linhas Idênticas**	**% Linhas Idênticas**
LCA	113	89	78.76 %	193	192	99.48 %
LEACH	119	89	74.78 %	199	191	95.97 %

5. Conclusão

Este artigo apresentou o framework RCBM-S, que tem como objetivo apoiar o desenvolvimento de modelos de armazenamento em RSSFs para a implementação do código em dispositivos sensores. Ele facilita a implementação de modelos que se adequem às particularidades das aplicações e que explorem de forma eficiente os escassos recursos dos dispositivos sensores. O estudo de caso mostrou que a proposta promove a reutilização de código de 75% dos componentes e 99% do coordenador no desenvolvimento dos modelos LCA e LEACH. No trabalho de [Carrero et al. 2019] foi proposta a linguagem chamada SLEDS, para o desenvolvimento do coordenador do *framework* RCBM no simulador de redes NS-2. Ela é baseada no modelo de máquinas de estado. Como trabalho futuro é planejada a tradução automática de um programa SLEDS para nesC. Assim, um mesmo código na linguagem SLEDS poderá ser utilizado tanto para a geração de código de simulação, bem como para a geração de código para dispositivos sensores.

Referências

Baumgartner, T., Chatzigiannakis, I., Fekete, S., Koninis, C., Kroller, A., and Pyrgelis, A. (2010). Wiselib: A generic algorithm library for heterogeneous sensor networks. In *European Conference on Wireless Sensor Networks*, pages 162–177. Springer.

Carrero, M., Zamproni, K., Musicante, M. A., Santos, A., and Hara, C. (2018). Uma máquina de estados para especificação de códigos de simulação para redes de sensores sem fio urbanas. In *Simpósio Brasileiro de Computação Ubíqua e Pervasiva*.

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.

Carrero, M. A., Musicante, M. A., dos Santos, A. L., and Hara, C. S. (2019). Sleds: A dsl for data-centric storage on wireless sensor networks. *Communications in Computer and Information Science*, 926:74–89.

de Lima Braga, M., de Jesus dos Santos, A., and de Lucena Junior, V. F. (2010). Modelagem e geração de código para redes de sensores sem fio usando communicating x-machine. In *Proc. of the 9th Int. Information and Telecommunication Technologies Symposium*.

Gay, D., Levis, P., Von Behren, R., Welsh, M., Brewer, E., and Culler, D. (2014). The nesc language: A holistic approach to networked embedded systems. *Acm Sigplan Notices*, 49(4):41–51.

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*, pages 52–61. IEEE.