Modelagem e Geração de Código para Redes de Sensores Sem Fio Usando Communicating X-Machine

Marcus de Lima Braga[1], Alyson de Jesus dos Santos[1], Vicente Ferreira de Lucena Junior[1]

[1]Centro de Pesquisa e Desenvolvimento de Tecnologia Eletrônica e da Informação – CETELI – Universidade Federal do Amazonas (UFAM), Caixa Postal 69077-000 Manaus – AM – Brasil

Resumo – Redes de Sensores Sem Fio (RSSF) têm atraído a atenção de muitos pesquisadores devido seu caráter pervasivo e flexível utilizado para o monitoramento e controle de fenômenos físicos. Em conseqüência temos o crescimento muito rápido do desenvolvimento de novas aplicações e proporcionalmente aumento de complexidade dos sistemas propostos e do custo de seu desenvolvimento. A dificuldade no desenvolvimento de novas aplicações não se deve somente as suas características restritivas (memória, processamento, energia), mas também por sua natureza distribuída, sua programação de baixo nível e na dificuldade em testar e validar o software. Este panorama é favorável a criação de novas metodologias e ferramentas que dêem suporte ao desenvolvimento de sistemas para tal plataforma. A proposta desta pesquisa é o emprego do método formal *Communicating X-Machine* para modelagem das aplicações em RSSF, no intuito de aumentar o nível de abstração, possibilitando com isso a eliminação de ambigüidades e inconsistências, além de proporcionar o particionamento do sistema para melhor entendimento e conseqüente facilidade de implementação.

I. INTRODUÇÃO

Redes de Sensores Sem Fio (RSSF) são compostas por uma grande quantidade de dispositivos embarcados denominada de nós sensores, com capacidade de processamento, sensoriamento e comunicação sem fio [1,2,3]. Porém essa tecnologia apresenta características restritivas em relação a processamento, memória e energia. As RSSF possuem um grande potencial para aplicações de monitoramento e controle de fenômenos físicos, onde os nós sensores cooperam entre si para percepção de um fenômeno ou evento estudado. Tal potencial tem contribuído para o aumento na diversidade de aplicações e proporcionalmente no aumento da complexidade no desenvolvimento das mesmas. A programação para RSSF tem por característica ser dirigida a eventos e é realizada muito próxima ao sistema operacional exigindo ao desenvolvedor muita concentração em questões de baixo nível, o que tem por conseqüência a distração na aplicação da lógica e a necessidade de uma formação técnica adequada ao domínio da plataforma, fato raro de ser encontrado. Uma solução para problema seria o aumento no nível de abstração, a fim de simplificar a tarefa de programação, sem detrimento da eficiência e possibilitando com que o programador se restrinja apenas na aplicação da lógica. Alguns pesquisadores da área acenam para utilização de métodos formais como solução para este problema. Um método formal proporciona o aumento dos níveis de abstração, fazendo com que o desenvolvedor se preocupe com o que realmente interessa não se detendo em detalhes que não tenham relevância à aplicação, aliado a possibilidade de verificação de corretude da aplicação.

A utilização de um método formal no processo de desenvolvimento de software, permite a descrição (especificação) de sistemas complexos a partir de entidades abstratas (independentes de implementação), construção (por refinamento) e verificação de sistemas com o objetivo de atingir níveis de qualidade mais elevados. Outra característica da utilização de métodos formais é o aumento do nível de abstração, que possibilita a especificação dos dados, comportamento e funções do sistema [4,5]. Alguns métodos formais como Máquinas de Estado Finito ou Redes de Petri, conseguem descrever a dinâmica de um sistema, porém falham para descrever como um dado é afetado em cada operação no diagrama de transição de estados. Outros métodos, como *Statecharts*, conseguem capturar a dinâmica e o comportamento dos dados em cada operação, porém são susceptíveis a diversas interpretações. Em [11] é utilizado Máquina de Estado Finito para o entendimento lógico das aplicações RSSF, isto é conseguido a partir dos códigos das aplicações descritas em *nesC*, o que é o caminho inverso de nossa proposta.

X-Machine é um método formal intuitivo, que consegue descrever os tipos de dados de modo formal e as funções por meio de notação matemática conhecida [6], além de conseguir expressar à dinâmica e o comportamento do sistema (como um dado é afetado após uma operação). Além disso, o conjunto de *X-Machines* pode ser vistos como componentes que podem se comunicar e conseqüentemente formar um sistema maior, o que é uma característica bastante interessante para nossa pesquisa. Outra característica importante das *X-Machines*, é que devido ser um método intuitivo, facilita os estágios cruciais do desenvolvimento de software: modelagem, verificação e testes. Este trabalho propõe a utilização do método formal *Communicating X-Machine* na modelagem e geração de código para aplicações em RSSF.

A seguir, apresentaremos os conceitos que serão utilizados na metodologia proposta (Seções II e III), bem como as descrições das etapas da modelagem das aplicações (Seção IV) e finalmente, será apresentado um exemplo prático existente na plataforma TinyOS (Seção V).

II. MODELO X-MACHINE

Uma X-Machine é uma máquina geral de computação proposta por Eilenberg e extendida por Holcombe [6,7,8] que pode modelar: a) estruturas de memória e b) suas transições são rotuladas com funções. X-Machine é um método formal

que emprega uma abordagem diagramática para modelagem, sendo capaz de modelar tanto os Dados quanto o Controle de um Sistema.

Uma X-Machine (Figura 1) é composta por *stream* de entrada σ e por *stream* de saída γ, onde cada *stream* se configura como fluxo de comunicação por onde passam dados. Cada transição na *X-Machine* faz com um elemento presente no *stream* de entrada σ e o adiciona ao *stream* de saída γ. A X-Machine pode ser definida formalmente como uma óctupla $M = (\Sigma, \Gamma, Q, M, \Phi, F, q_0, m_0)$ onde:

- Σ, Γ são os alfabetos de entrada e saída;
- Q é o conjunto finito de estados;
- M é o conjunto (possivelmente) infinito chamado memória;
- Φ é o tipo da máquina M, um conjunto finito de funções parciais φ que mapeiam uma entrada e um estado de memória numa saída e um novo estado de memória,

$$\varphi: \Sigma \, X \, M \rightarrow \Gamma \, X \, M$$

- F é a função parcial de próximo estado a qual, dado um estado e uma função do tipo Φ, denota o próximo estado. F é normalmente descrita como uma função de transição de estado,

$$F : Q \, X \, \Phi \rightarrow Q$$

- q_0, m_0 são o estado inicial e a memória inicial respectivamente.

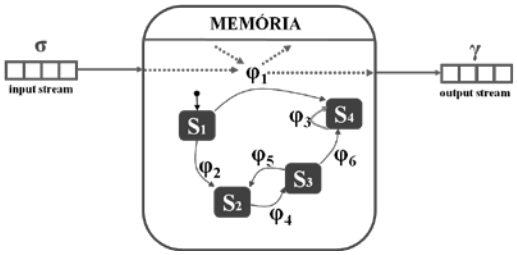

Figura 1. Exemplo abstrato de uma *X-Machine*

As *X-Machines*, embora de natureza abstrata, possuem o mesmo poder computacional das Máquinas de Turing [9] e são expressivas o suficiente para representarem de uma maneira mais precisa a implementação de um sistema [6]. Esta característica é bastante útil para modelagem e facilita na implementação de várias ferramentas, que torna a construção de uma metodologia de desenvolvimento utilizando *X-Machines* mais prática.

Uma outra particularidade das *X-Machines* é o fato de não somente prover uma modelagem formal para um sistema, mas também oferece uma estratégia de teste do modelo [6], permitindo dessa forma uma melhor maturação do software.

As *X-Machines* podem também ser descritas através de uma linguagem de marcação tipo *XML*, denominada *X-Machine Description Language – XMDL*, ao qual será demonstrada na Seção V deste trabalho.

III. COMMUNICATING X-MACHINE

O modelo *Communicating X-Machines* (Figura 2) é uma extensão da *X-Machine*. O mesmo consiste em várias *X-Machines* que podem trocar mensagens entre si (comunicação entre máquinas), de forma que uma função pode ler a entrada a partir de um *stream* de dados de comunicação (mensagem) enviada por outra máquina. Bem como as funções podem escrever a mensagem no *stream* de comunicação de saída de outra máquina. Após a instanciação dos parâmetros de saída a mensagem será enviada ao destino. Podemos perceber que, *Communicating X-Machine* é um método formal que poderá ser usado para modelar o comportamento de um sistema que se comunica [10].

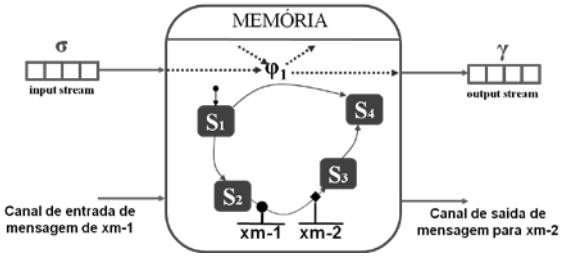

Figura 2. *Communicating X-Machine*

A comunicação de um sistema *Communicating X-Machines* (CXM) com n componentes pode ser descrito por meio de uma matriz de comunicação (CM):

$\mathbf{CXM_n = ((XMC_i)_{i=1,\ldots n}, CM, C_o)}$, onde:

- $\mathbf{XMC_i}$ é um componente *X-Machine* do sistema;
- **CM** é uma matriz *n* X *n*, denominada matriz de comunicação;
- $\mathbf{C_o}$ é o inicio da matriz de comunicação.

Figura 3. Comunicação dos componentes *X-Machine* $\mathbf{XMC_i}$ E $\mathbf{XMC_j}$ através das portas OP e IP

O componentes $\mathbf{XMC_i}$ de $\mathbf{CXM_n}$ é diferente de uma *X-Machine* padrão, pois se utilizam de portas ***IN*** e ***OUT*** para se comunicarem. Essas portas são ligadas a CM que atua como meio de comunicação entre as XMCs. Nas células da CM estão contidas as mensagens trocadas entre as $\mathbf{XMC_i}$ e $\mathbf{XMC_j}$, ou seja, $\mathbf{XMC_i}$ lê a *i*-ésima coluna e escreve na *i*-ésima linha. A CM pode conter um valor indefinido λ, que significa que não existe mensagem, enquanto que nas demais células podem estar vazias. As mensagens podem ser algum tipo definido na memória em todos os componentes XMC, de modo que o envio e recebimento de mensagens requerem um *input* na porta *IP* e um *output* na porta *OP*. Ainda nas portas *IP* e *OP* existe um tipo especial de comunicação que se refere a estados e funções, respectivamente denominados por: *Communicating States* e *Communicating Functions* (Figura 2). As *Communicating Functions* derivam dos *Communicating*

States, aceita um símbolo vazio ε como *input* e produzem um símbolo ε como *output*, sem afetar a memória. As *Communicating Functions* ou lêem um elemento na CM e o carregam para porta *IN*, ou escrevem um elemento que está na porta *OUT* para a CM:

$$cf(\varepsilon,m,in,out,c)=(\varepsilon,m,in',out',c'),\ \text{onde:}$$
$$m \in M,\ in,\ in' \in IN,\ out,out' \in OUT,\ c,c' \in CM$$

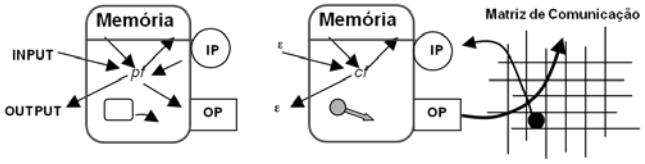

Figura 4. Visão da comunicação entre estados e funções.

As *Communicating Functions* só podem escrever na CM se uma célula contém o símbolo especial λ. Após *Communicating Functions* lerem na CM, a célula recebe o símbolo especial λ. Se a *Communicating Functions* não é aplicável, então ela espera até que o seja. Os processamentos das funções que resultam da mudança de estados afetam os conteúdos das portas *IN* e *OUT*, porém não afetam a CM:

$$pf(\sigma,m,in,out) = (\gamma,m',in',out'),\ \text{onde:}$$
$$\sigma \in \sum,\ m,\ m' \in M,\ in,\ in' \in IN,\ out,\ out' \in OUT,\ \gamma \in \Gamma$$

IV. Construindo aplicações a partir de X-Machine Stand-Alones

Uma alternativa para construção de sistemas é a utilização do modelo de *X-Machines Stand-Alone* (isoladas) como componentes do sistema maior. A abordagem consiste em duas etapas: a) modelar a *X-Machine* independente das outras partes do sistema, ou seja, a *X-Machine* é um componente do Sistema; b) determinar a comunicação entre componentes *stand-alone*.

A abordagem traz diversas vantagens para o desenvolvedor: a) pode re-utilizar modelos pré-existentes; b) podem separar modelagem e comunicação como atividades distintas no desenvolvimento de um sistema de comunicação; c) não necessita modelar um sistema de comunicação a partir do zero; d) pode utilizar-se de ferramentas que trabalhem com *X-Machines Stand-Alone* ou *Communicating X-Machines*.

A. 1° Passo: Modelar X-Machine de Forma Independente

Neste momento, são modelados os comportamentos de um sistema por meio de *X-Machines* em separado, de modo a realizar tal operação independente do todo (sistema). Esta abordagem permite o particionamento do sistema, possibilitando dessa forma um melhor entendimento e o refino na modelagem do mesmo. A idéia principal neste passo é a componentização da *X-Machine*, para que no passo seguinte se comunique com os demais componentes (*X-Machines*).

B. 2° Passo: Construir um Sistema de Comunicação

Nesta abordagem, a CM é substituída por diversos *inputs* de *stream* associados com cada componente *X-Machine*. Esta é outra visão conceitual da mesma entidade, que serve também para as operações assíncronas de uma *X-Machine* individual.

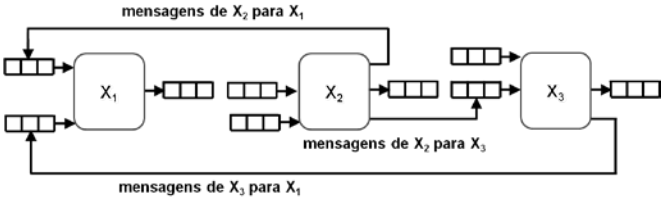

Figura 5. Comunicação com três *X-Machines* - X_1, X_2 E X_3: X_2 se comunica (escreve) com X_1 e X_3, enquanto X_3 se comunica (escreve) com X1.

V. Modelando Blink em Communicating X-Machine

Nesta seção apresentaremos a aplicação da metodologia proposta tomando como foco uma aplicação existente na plataforma *TinyOS*, de modo a verificar a viabilidade técnica da proposta.

O *TinyOS* é composto por um sistema operacional simples, um ambiente de desenvolvimento com código aberto e cuja linguagem de programação é denominada *nesC* (network embedded systems C). As aplicações são descritas através de componentes, que podem ser construídos e combinados, proporcionando a modularidade e reusabilidade de código, características que facilitam a aplicação da metodologia proposta.

Em princípio modelamos a aplicação utilizando *X-Machines* isoladas, isso possibilita particionar o sistema de modo a facilitar o desenvolvimento. Nesta fase é possível se ter uma melhor compreensão dos requisitos, devido ao aumento no nível de abstração, que possibilita o refino da modelagem. A Figura abaixo representa a modelagem da aplicação *Blink* encontrada no *TinyOS*, é possível observar a existência das funções de comunicação entre os componentes (*X-Machine*) que compõe o sistema.

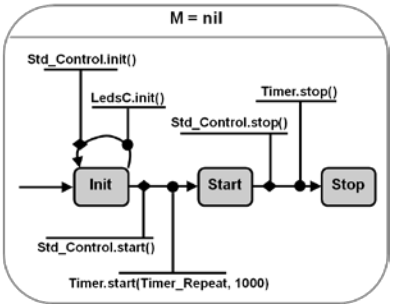

Figura 6. Aplicação *Blink* descrita em *Commnicating X-Machine*

A seguir mostraremos o sistema como um todo apresentando um diagrama de componentes utilizando o método formal. Podemos notar que a referida aplicação é composta por três componentes: *SingleTimer*, *LedsC* e *Main*. Os dois primeiros componentes controlam a freqüência e o *LED* (que irá ser aceso e apagado) respectivamente. O componente *Main* é usado em todas as aplicações *nesC*, para iniciar as aplicações através da interface *StdControl*.

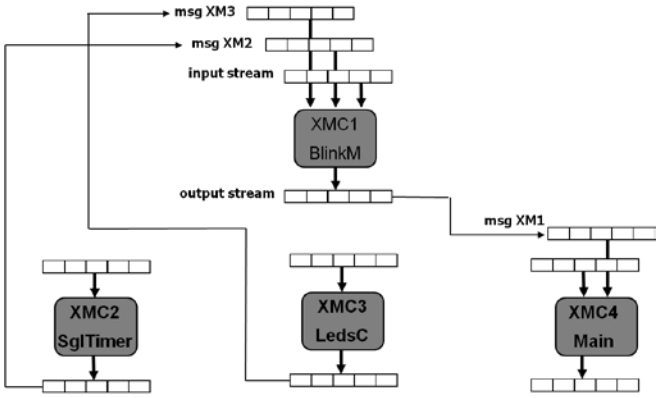

Figura 7. *Blink* descrito em *Communicating X-Machine*.

O modelo pode ser descrito de outra forma, utilizando-se para isto a linguagem *XMDL* e em seguida podemos fazer sua tradução para o *XML*, conforme método apresentado em [6]. A tradução de *XMDL* para *XML*, abre um leque de opções para nossa aplicação, pois por se tratar de uma linguagem que proporciona a organização dos dados de forma hierárquica, possibilita sua tradução para diversas linguagens (Java, C, Lua, dentre outras). A seguir apresentaremos o código que representa o modelo da aplicação *Blink* expresso em *XMDL*:

```
1  # model Blink.
2  # type Interval = natural.
3  # type Type = char.
4  # type messages = {initialized, started, stoped}.
5  # states = {Init, Start, Stop}.
6  # memory nil.
7  # type event = {fired}.
8  # input (Type, Interval).
9  # output (messages).
10 # fun start ((start), (?Type, ?Interval)) =
11    if ?Type == TIMER_REPEAT then
12    (nil,(message)).
13 # fun stop ((stop), (nil)) = ((stop), nil).
14 # transition (Init, init) = Init.
15 # transition (Init, start) = Start.
16 # transition (Start, stop) = Stop.
```

Código 1. Código *Blink* em *XMDL*.

```
1  # communicating of Blink:
2  StdControl.init() writes (init()) to Main.
3     Leds.init() reads from LedsC.
4  StdControl.start writes (start()) to Main.
5     Timer.start() reads from SingleTimer.
6  StdControl.stop writes (stop()) to Main.
7     Timer.stop() reads from TimerC.
```

Código 2. Código de Comunicação *Blink* com componentes.

O código acima é referente a *X-Machine Blink* (Figura 7) e descreve as comunicações dela com as demais *X-Machines* (componentes): *Main*, *SingleTimer* e *TimerC*. A seguir apresentaremos os códigos de comunicação entre os demais componentes (*X-Machines*) da aplicação.

```
1  # communicating of LedsC:
2  LedsC.init() writes (StdControl.init()) to
3     BlinkM.
4  LedsC.redToggle() writes (Timer.fired()) to
5     BlinkM.
```

Código 3. Código de ligação componente *LedsC*.

```
1  # communicating of SingleTimer:
2  StdControl.start() writes
3     (Timer.start(TIMER_REPEAT, 1000)) to
4     BlinkM.
5  StdControl.stop() writes
6     (Timer.stop()) to BlinkM.
```

Código 4. Código de ligação componente *SingleTimer*.

VI. CONCLUSÕES E TRABALHOS FUTUROS

Neste trabalho apresentamos uma proposta de modelagem e geração de código para aplicações de RSSF utilizando o método formal *Communicating X-Machine*. Podemos verificar sua viabilidade na modelagem e implementação de sistemas baseados em eventos, onde seus requisitos podem ser expressos em *XMDL*, possibilitando desta forma a tradução para outra linguagem de marcação conhecida o *XML*. Característica que nos possibilita migrar para outras linguagens de programação como C, Java [6] e *nesC* (a ser apresentado em um trabalho futuro) entre outras.

Outra característica interessante da *Communicating X-Machine*, é o fato de oferecer suporte para o desenvolvimento de sistemas complexos por meio da composição de elementos mais simples, fazendo com que cada parte seja representada por uma *X-Machine*, o que facilitando o processo da análise e da construção do todo. Nossos trabalhos futuros serão a construção de uma ferramenta que automatize este processo e proporcione a geração de código para a plataforma *TinyOS* e a utilização do modelo para Outro ponto a ser explorado é a utilização do modelo formal para verificação do código, fazendo com que as aplicações possam ser refinadas e que os erros de implementação sejam diminuídos.

REFERÊNCIAS

[1] Mottola, L., Picco, G. P. Programming Wireless Sensor Networks: Fundamental Concepts and State of the Art. ACM Trans. Sen. Netw., 4(2): 1-29, 2008.

[2] Sugihara, R., Gupta, R. K. Programming Models for Sensor Networks: A Survey. ACM Transactions on Sensor Networks, vol. 4, issue 2, Mar. 2008.

[3] Akyildiz, I.F., Su, W., Sankarasubramaniam, Y., and Cayirci, E. A Survey on Sensor Networks. IEEE Communications Magazine," Vol. 40, No. 8, pp. 102-116, August 2002.

[4] Paiva, A. Métodos Formais em Engenharia de Software, Universidade do Porto, notas de aulas, 2007.

[5] Reis, C. A. L., Métodos Formais para Especificação de Software, UFPA, 2006.

[6] Kefalas, P., Eleftherakis, G., Kehris, E., Communicating X-Machines: From Theory to Practice. PCI'2001 Proceedings of the 8th Panhellenic Conference on Informatics, pp. 316-335, 2001.

[7] Ogunshile, E. K. A., Automatic Generation of Java Code From Communicating X-Machine specifications, University of Sheffield, 2005.

[8] Kefalas, P., Eleftherakis, G., Kehris, E., Communicating X-Machines: a practical approach for formal and modular specification of large system.

[9] J. Barnard, COMX: a design methodology using communicating Xmachines, Journal of Information and Software Technology 40 (1998) 271–280.

[10] Caldas, R. B., Modelo para Desenvolvimento de Aplicações em Redes de Sensores Sem Fio, Defesa de Proposta de Tese de Doutorado, UFMG, 2007.

[11] Kothari, N., Millstein, T., Govindan, R. Deriving State Machines from TinyOS Using Symbolic Execution Source Information Processing In Sensor Networks. Proc. of the 7th International Conference on Information Processing in Sensor Networks, pp. 271-282.