GERAÇÃO AUTOMÁTICA DE CÓDIGO PARA REDES DE SENSORES SEM FIO BASEADO EM COMPONENTES DE SOFTWARE

Alyson de Jesus dos SANTOS(1); Marcus de Lima BRAGA (1); Vicente Ferreira de LUCENA Júnior (1)
Universidade Federal do Amazonas, Av. Gen. Rodrigo Otávio 3000, Campus UFAM - FT, 69077-000 - Manaus - AM
{alysondejesus, marcuslbraga, vicente.ferreira.jr}@gmail.com

RESUMO

Este artigo propõe o desenvolvimento de uma ferramenta intitulada Geração Automática de Código para Redes de Sensores Sem Fio (GAC-RSSFs), cujo objetivo principal é possibilitar a alta produtividade no desenvolvimento de aplicações para as Redes de Sensores Sem Fio (RSSFs) utilizando modelos especificados na Arquitetura Dirigida a Modelos (*Model Driven Achitecture - MDA*). GAC-RSSFs recebe como entrada um modelo baseado em componentes de uma aplicação criada pelo projetista e só então realiza a geração automática de código em linguagem *nesC* (*network embedded systems C*), seja a configuração ou o módulo das aplicações.

Palavras-chave: Arquitetura Dirigida a Modelos, Geração Automática de Código, *nesC*, Redes de Sensores Sem Fio.

1 INTRODUÇÃO

Avanços recentes em miniaturização de hardware e comunicação sem fio criaram um novo paradigma em monitoração de ambientes [1]. Dispositivos pequenos e de baixo custo, dotados de unidades de processamento, comunicação e sensoriamento denominados nós sensores podem colaborar através do meio sem fio. Esses dispositivos são capazes de medir características do ambiente como pressão, temperatura, luminosidade e velocidade [2]. Esses dados podem ser processados internamente na rede e depois enviados a um ponto de acesso, o qual possui uma visão global da rede. Os nós sensores podem ser utilizados em várias aplicações, como detecção de incêndios, monitoração ambiental, monitoração e controle industrial, agricultura de precisão e rastreamento de alvos. Uma coleção de nós sensores trabalhando em uma aplicação compõem uma RSSF.

Programar os nós de uma RSSF é uma tarefa difícil e trabalhosa, e conta com duas categorias de desenvolvedores, que precisam criar as mais diversas aplicações, a custo de muito tempo e esforço de codificação do projeto. Temos na primeira categoria, os desenvolvedores especialistas, conhecedores em nível de detalhes do modelo de programação das RSSFs e da especificação da linguagem em particular. Na segunda categoria, os desenvolvedores inexperientes que precisam construir aplicações, porém tem pouco conhecimento tanto do modelo de programação como da especificação da linguagem. Todas as categorias apresentadas são importantes e devem ser consideradas, principalmente quando se trata de acelerar e facilitar o processo de desenvolvimento de software. Reduzir o tempo e o esforço dos desenvolvedores na codificação do projeto, sem diminuir a qualidade do produto final (software), vem sendo uma busca incessante dos engenheiros de software. Muitas técnicas, modelos, metodologias e ferramentas foram e estão sendo desenvolvidas para aumentar a produtividade dos desenvolvedores na construção de aplicações para sistemas embarcados.

Uma das iniciativas proposta é a utilização da *MDA* criado pelo *Object Management Group* (*OMG*) [3]. O desenvolvimento orientado por modelos é foco de muitos estudos nos últimos anos; o interesse cresce à medida que se torna uma proposta viável no processo de desenvolvimento de software e na missão de prover soluções para o problema da produtividade do desenvolvedor. O uso de representações gráficas precisas, porém abstratas de algoritmos, de tal forma que permita a construção de sistemas completos a partir de modelos que podem ser entendidos muito mais rapidamente e profundamente do que os códigos de linguagem de programação é a essência da *MDA*.

A *MDA* apoia todo o ciclo de vida do desenvolvimento de aplicações, através da utilização de modelos, como eles devem ser usados e o relacionamento entre eles durante todas as fases do ciclo. Dessa maneira, a maior parte do esforço ficará em cima do modelo, e não mais nos códigos fontes. Dentre os benefícios propostos estão: o ganho de produtividade através da geração de código a partir de modelo, portabilidade a outras plataformas e interoperabilidade.

A ferramenta *Eclipse Model Framework* (*EMF*) é um *plugin* que proporciona apoio automatizado ao método *MDA*. Esse plugin é responsável pela transferência harmoniosa de informações (modelos) criada na ferramenta *EMF*, utilizada em outra ferramenta, no caso, a baseada em *templates*, o *Java Emitters Template* (*JET*).

O texto deste artigo está organizado da seguinte forma: a seção 2 apresenta conceitos que envolvem a ferramenta. A seção 3 é apresentado o ambiente proposto nesse artigo, descrevendo alguns dos seus benefícios, limitações e mencionando algumas questões sobre a implementação. A seção 4 mostra os trabalhos relacionados à ferramenta GAC-RSSFs, e finalmente a seção 5 apresenta as considerações finais e os trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Nesta seção apresenta-se o estado da arte sobre *MDA*, *TinyOS*, linguagem *nesC* e Arquitetura da Plataforma Eclipse, utilizados na ferramenta GAC-RSSFs.

2.1 Arquitetura Dirigida a Modelos

MDA [4, 5, 6] é um padrão baseada em modelos. O modelo significa uma representação de parte de uma funcionalidade, estrutura ou comportamento de um sistema e precisa estar relacionado, sem ambigüidades, a uma definição de sintaxe e semântica.

A *MDA* possui alguns tipos de modelos para descrição de um sistema. Os principais são: O Modelo Independente de Plataforma (*PIM*) que fornece informações sobre a estrutura e a funcionalidade de um sistema, porém, abstraindo os detalhes técnicos. O Modelo Específico de uma Plataforma (*PSM*) que funciona como uma extensão ao *PIM*, incluindo os detalhes específicos de uma tecnologia ou plataforma. Existe também o Modelo Independente de Computação ou Modelo do Domínio (*CIM*), que representa a visão de um sistema sem levar em consideração detalhes relativo à sua computação.

2.1.1 Transformações de Modelos

A transformação de modelos na *MDA* (Figura 1) é o processo pelo qual a partir de especificações com um alto nível de abstração e independência de plataforma seja possível gerar, de forma automática, sistemas completos para múltiplas plataformas. Os tipos de transformações são:

- *CIM* para *PIM*: não há transformações automáticas;
- *PIM* para *PIM*: o objetivo é aprimorar, filtrar ou especializar modelos, sem introduzir detalhes dependentes de plataforma;
- *PIM* para *PSM*: o objetivo é utilizar um *PIM* suficientemente refinado e projetá-lo para um modelo de ambiente de execução;
- *PSM* para *PSM*: o objetivo é refinar o modelo específico de uma plataforma;
- *PSM* para *PIM*: o objetivo é extrair modelos independentes de plataforma a partir de implementações existentes.

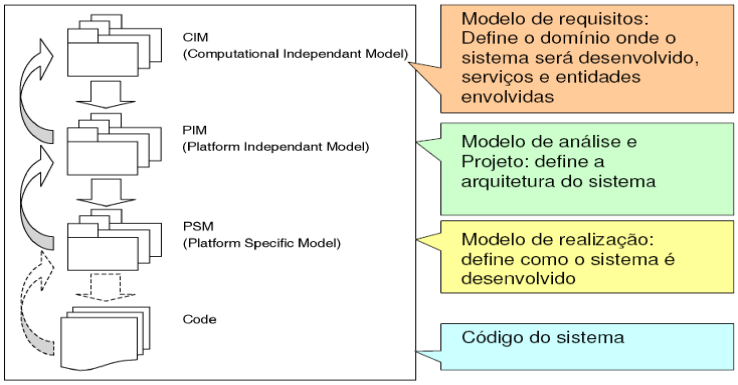

Figura 1 – Transformações de modelos na *MDA*.

2.2 *TinyOS*

O *TinyOS* [7, 8, 9] é um sistema operacional desenvolvido pela Universidade da Califórnia que possui uma arquitetura baseada em componentes e foi especialmente projetado para redes de sensores. As suas aplicações são escritas em um dialeto de C, denominado *nesC*, que foi desenvolvido com o objetivo de ser eficiente em relação às restrições de memória dos sensores e permitir rápida implementação.

Aplicativos escritos em *nesC* são compostos por componentes [10, 11, 12], que possuem três abstrações computacionais:

- *Commands*: requisição ao componente para execução de algum serviço;
- *Events*: sinalização do componente indicando o fim da execução de um serviço;
- *Tasks*: representam concorrência interna no componente.

2.3 Linguagem *nesC*

A linguagem *nesC* [13, 14, 15] é baseada em componente. O uso de componentes diminui o tempo de desenvolvimento de aplicações e permite a sua reusabilidade.

Existem dois tipos de componentes: configuração e módulo. As configurações definem como os componentes estão conectados, enquanto os módulos são as implementações das interfaces dos componentes.

2.3.1 Interfaces

As interfaces são pontos de acesso aos componentes e devem obedecer a um padrão e conter apenas as assinaturas dos *commands* e *events*. No exemplo a seguir é mostrada uma parte da interface ***Clock*** (Figura 2).

```
interface Clock {
command result_t setRate(char interval,
                         char scale);
event result_t fire();
}
```

Figura 2 – Interface Clock.

2.3.2 Módulos

Os módulos contêm o código da aplicação implementando uma ou mais interfaces. Para exemplificar o uso dos módulos, o módulo ***BlinkM*** é apresentado na figura 3. Ele provê a interface StdControl, ou seja, precisa implementar os comandos init, start e stop dessa interface. O componente usa as interfaces Clock e Leds.

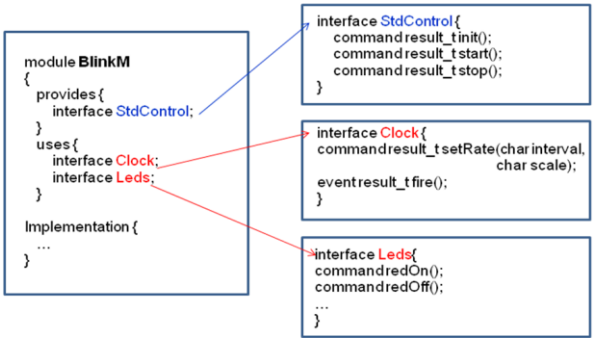

Figura 3 – Módulo da aplicação *BlinkM*.

2.3.3 Configurações

Toda aplicação em nesC precisa da configuração cuja responsabilidade é a ligação entre os componentes. Na ligação, as interfaces providas por um componente são conectadas às interfaces usadas por outros.

```
configuration Blink {
}

implementation {
   components Main, BlinkM, ClockC, LedsC;

   Main.StdControl -> BlinkM.StdControl;
   BlinkM.Clock -> ClockC;
   BlinkM.Leds -> LedsC;
}
```

Figura 4 – Configuração da aplicação *Blink*.

A implementação da configuração especifica os componentes que serão referenciados pelo atual componente e como eles serão conectados. Na figura 4, a configuração da aplicação *Blink* referencia os componentes *Main*, *BlinkM*, *ClockC* e *LedsC*.

3 A FERRAMENTA GAC-RSSFs

Nesta seção são apresentadas as funcionalidades, as etapas do processo de desenvolvimento e os detalhes relacionados a implementação da ferramenta GAC-RSSFs.

GAC-RSSFs é um ambiente de modelagem específico de domínio que atende a modelagem de sistemas baseados em componentes. Os componentes e as interfaces são representados na interface gráfica do usuário, de maneira que estabelecendo uma ligação entre componentes e uma interface requerida por um, é fornecida por outro. O metamodelo é especificado na notação UML [16, 17, 18], usando o diagrama de classe.

3.1 Etapas do Processo de Desenvolvimento da Ferramenta GAC-RSSFs

A seguir, são descritas, de forma resumida, as cinco etapas realizadas no processo de desenvolvimento da ferramenta:

1. Definição do Modelo de Domínio - o metamodelo é especificado usando o *Essencial MetaObject Facility* (EMOF), uma linguagem de definição de metamodelo usada para definição da UML e integrante do *plugin* EMF.
2. Definição do Modelo Gráfico - esta etapa concentra-se na definição das entidades e das propriedades, assim como dos relacionamentos que podem ser expressos na ferramenta. Para isto, foram levadas em consideração as entidades e os relacionamentos definidos no metamodelo da linguagem.
3. Definição do Modelo de Ferramenta - esta etapa é dedicada para a especificação de quais elementos farão parte da paleta da ferramenta. Para isso, essa etapa recebe como entrada o modelo gráfico determinado anteriormente.

4. Definição do Mapeamento - a atividade desenvolvida nesta etapa se concentra na construção de um mapeamento entre os modelos gráfico e o da ferramenta. O mapeamento gerado é usado como entrada em um processo de transformação que tem como objetivo criar um modelo específico de plataforma.

5. Geração da Ferramenta - seguindo uma abordagem generativa, o próximo passo consiste na geração do código da ferramenta levando em consideração o modelo específico de plataforma gerado na etapa anterior. Nesta etapa, é feito uso do *plugin Graphical Model Framework* (*GMF*) que fornece um componente generativo e uma infra-estrutura *runtime* para desenvolver editores gráficos baseado no *EMF* e *Graphical Edition Framework* (*GEF*).

3.2 Implementação da Ferramenta

A Figura 5 mostra uma visão geral da interface gráfica de GAC-RSSFs juntamente com alguns dos seus componentes destacados com letras. Cada componente desempenha uma funcionalidade específica dentro da ferramenta, sendo discutida brevemente a seguir:

- *Package Explorer* (A) - para cada novo projeto de modelagem que é criado, surge a necessidade de criar e importar arquivos que são utilizados durante o processo de modelagem (bibliotecas, documentos de texto, imagens, e etc). O *package explorer* tem como funcionalidade central permitir a organização dos arquivos em uma estrutura de árvore a fim de que seja possível um melhor gerenciamento e manipulação dos arquivos;

- *Modeling View* (B) - os modelos que são criados precisam necessariamente ser visualizados com o objetivo de atender dois requisitos básicos quando se usam modelos: compreensibilidade e a comunicação. Diante dessa necessidade, a *modeling view* permite aos desenvolvedores visualizarem e editarem os modelos de forma interativa;

- *Palette* (C) - os construtores que resultam do diagrama de classe proposto por GAC-RSSFs encontram-se na palette. Observando a figura 5 é possível identificar o elemento *Component* (*Main*, *BlinkM*, *LedsC* e *ClockC*), sendo que seus relacionamentos podem ser estabelecidos entre os construtores presentes na *palette* são representados por Port, que liga os componentes e especifica a interface. Podemos observar ainda a interface *StdControl* relacionada aos componentes *Main* e *BlinkM*.

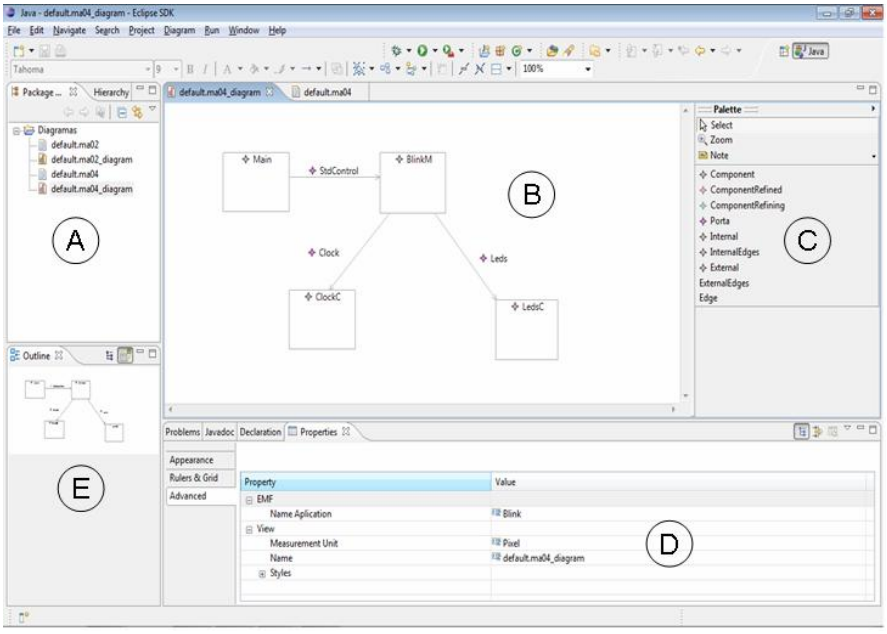

Figura 5 – Visão geral da ferramenta GAC-RSSFs.

- *Properties View* (D) - as preocupações sempre constante durante a elaboração do metamodelo de GAC-RSSFs consistem na identificação e representação das características dos conceitos presentes no paradigma baseado em componentes. Observando a Figura 5, a propriedade *Name Aplication* é atribuído o valor *Blink*.

- *Outline View* (E) - uma vez que um modelo tenha sido criado, um *overview* da distribuição dos elementos presentes no mesmo poderá ser visualizado através da *outline view*.

3.3 Benefícios e Limitações

A falta de um ambiente que dê suporte ao desenvolvimento de aplicações para as RSSFs representa um ponto chave, haja vista que sem o apoio de uma ferramenta o trabalho torna-se difícil de ser realizado e custoso. Desta forma, os benefícios que são obtidos com a ferramenta proposta emergem diante da resolução de alguns desafios encontrados até então. A seguir são destacados alguns desses desafios que serão superados:

- Não é necessário conhecer a linguagem em detalhes - os programadores inexperientes não precisam conhecer a fundo sobre a linguagem de programação nesC, mas ter o entendimento do paradigma orientado a componentes. Nessas condições, o programador ainda conseguirá gerar a configuração e o módulo das aplicações de RSSFs, reduzindo erros de codificação.
- Validação do modelo - todo modelo criado deve ser validado em relação às restrições definidas no metamodelo da linguagem. Caso estas restrições não sejam respeitadas, o modelo criado não estará condizente com as definições do metamodelo, podendo até mesmo não representar uma aplicação para as RSSFs.
- Não está restrito à biblioteca de componentes do *TinyOS* - os programadores podem construir seus próprios componentes ou reutilizar os do TinyOS.
- Utilização de modelos - os modelos são documentação que representam a visão gráfica do código.

A ferramenta GAC-RSSFs foi projetada de forma a capturar o metamodelo representado no diagrama de classe e realizar a geração de código fonte na linguagem nesC, com base no método *MDA*. Tratar o comportamento dinâmico das tarefas, assim como, a simulação e verificação formal são as limitações da ferramenta.

4 Trabalhos Relacionados

Nesta seção são discutidos os trabalhos relacionados que foram selecionados com base em dois critérios: a relevância do trabalho e a sua contribuição para o desenvolvimento do trabalho proposto.

"Aplicando Model-Driven Development à Plataforma GPGPU" [19] é uma ferramenta que aplica princípios do *Model Driven Development* (MDD) ao desenvolvimento de aplicações para *Graphics Processing Units* (GPUs), visando produzir um ambiente mais adequado para a construção desse tipo de software. É visualizada uma aplicação como um modelo de entrada, e gerado automaticamente parte significativa do código da aplicação. O código gerado é expresso na linguagem definida por *Compute Unified Device Architecture* (CUDA), uma plataforma de programação para *General-Purpose computation on GPU* (GPGPU).

"Geração de Aplicação Web usando a MDA**"** [20] é uma ferramenta que apresenta o método *MDA* como forma de desenvolvimento de aplicações web e geração de código de aplicações a partir de modelagem UML. A ferramenta gera um modelo XMI como saída que é exportado para a ferramenta AndroMDA. AndroMDA aplica um conjunto de *templates* denominado cartucho, gerando o código fonte para uma plataforma escolhida, como por exemplo, *Spring*, *Struts*, *EJB*, *Hibernate* e *JSF*.

"Labview Wireless Sensor Network Pionner" [21] é uma ferramenta comercial que trabalha com a abordagem gráfica de programação que simplifica bastante o processo de criação de aplicações embarcadas para nós sensores. A ferramenta inclui estruturas básicas de programação, tais como condicional e de repetição, além dos componentes gráficos.

4.1 Estudo Comparativo

A Tabela 1 foi elaborada com o intuito de facilitar as comparações das ferramentas apresentadas nesse trabalho. Os critérios a seguir foram utilizados nas análises de comparação das ferramentas:

a) Foco - grupos de definições das ferramentas quanto ao enfoque;

b) SO (Sistema Operacional) - especifica em quais sistemas operacionais a ferramenta atua;

c) Gratuidade - identifica se a ferramenta é gratuita ou comercial;

d) Código Aberto - demonstra se a ferramenta possui ou não código aberto;

e) Extensível - identifica se é possível customizar a ferramenta;

f) *PIM* e *PSM* - se as ferramentas implementam esses modelos da *MDA*.

Tabela 1 – Comparação de Ferramentas.

Comparativo	**Aplicando MDD à Plataforma GPGPU**	**Geração de Aplicação Web usando a MDA**	**Labview Wireless Sensor Network Pioneer**	**GAC-RSSFs (Desejável)**
Foco	Processadores	Aplicações na web	Redes de Sensores	Redes de Sensores
Sistema Operacional	Windows e Linux	Windows e Linux	Windows e Linux	Windows e Linux
Gratuidade	Sim	Sim	Não	Sim
Código Aberto	Sim	Sim	Não	Sim
Extensível	Sim	Sim	Não	Sim
PIM	Sim	Sim	Não	Sim
PSM	Sim	Sim	Não	Sim

Observa-se que na tabela 1 a ferramenta "Aplicando Model-Driven Development à Plataforma GPGPU" tem foco voltado para o universo dos processadores, utiliza a metodologia *MDD* e realiza geração de código para a linguagem de baixo nível, denominada *CUDA*. A ferramenta "Geração de Aplicação Web usando a MDA" está voltada para um contexto diferente da ferramenta citada anteriormente. Ela é empregada no desenvolvimento e geração de código de aplicações web, na qual apresenta-se o método *MDA*. A ferramenta "Labview Wireless Sensor Network Pionner" tem foco diferente das duas (2) ferramentas citadas, pois é utilizada no contexto das RSSFs, é proprietária e não segue nenhuma metodologia que contemple a abordagem baseada em modelo. Essa ferramenta não é extensível, e gera código em linguagen nesC, apenas a configuração da aplicação. Com o intuito de gerar código para as RSSFs, seja a configuração e o módulo das aplicações em linguagem nesC, e ainda usar uma abordagem que tem como premissa o desenvolvimento baseado em modelos surge a ferramenta GAC-RSSFs que visa preencher essa lacuna.

5. Conclusão e Trabalhos Futuros

A ferramenta apresentada neste trabalho possibilitará automatizar o processo de construção de aplicações para RSSFs, tendo como foco a geração de código em nesC a partir de modelos baseados em componentes representados através de uma interface gráfica para o projetista de aplicações. A interface gráfica facilitará a montagem de aplicações, fazendo uso dos elementos do domínio: componente e interface disponibilizados na paleta de componentes. A abordagem dirigida a modelos adotada neste trabalho proporciona o aumento do nível de abstração da aplicação, contribuindo para a diminuição de erros por parte do desenvolvedor, algo comum quando se está lidando com programação de baixo nível, que normalmente trabalha com códigos não triviais.

Com o desenvolvimento da ferramenta pretende-se atingir os seguintes objetivos: i) consistência ao realizar transformações sobre os modelos existentes; ii) facilidade em criar aplicações para RSSFs em um ambiente gráfico (mais precisamente na linguagem nesC); iii) facilidade de extensão por meio de plugins; iv) padronização.

Como direções futuras deste trabalho propõem-se: (i) estender a ferramenta GAC-RSSFs para que seja capaz de verificar propriedades de uma aplicação, utilizando o formalismo da Rede de Petri para testar a corretude

da aplicação; (ii) gerar código para as RSSFs obtidos a partir de especificações em Rede de Petri; (iii) navegar entre as especificações de Rede de Petri e as Baseada em Componentes, ou seja, tendo como modelo de entrada uma aplicação especificada em Rede de Petri e obter como saída a representação do Modelo Baseado em Componentes.

REFERÊNCIAS

[1] ESTRIN, D.; GOVINDAN, R.; HEIDEMANN, J. S.; KUMAR, S. "**Next century challenges: Scalable coordination in sensor networks**" in *Mobile Computing and Networking*, 1999, pp. 263–270. Disponível em: <citeseer.nj.nec.com/estrin99next.html> Acesso em: 17/04/2010.

[2] AKYILDIZ, I.F.; SU, W.; SANKARASUBRAMANIAM, Y.; CAYIRCI, E. "**Wireless sensor networks: A survey**" *Computer Networks*, Mar. 2002.

[3] Object Management Group. 2003. Disponível em: <http://www.omg.org> Acesso em: 15/03/2008.

[4] MELLOR, J. S. ***MDA Destilada. Princípios de Arquitetura Orientada por Modelos.*** Rio de Janeiro: Editora: Ciência Moderna Ltda., 2005.

[5] MAIA, N. E. N. ***Odyssey-MDA: Uma Ferramenta para Transformações de Modelos UML.***

[6] MILLER, J.; MUKERJI, J. *MDA Guide Version 1.0.1. Object Management Group.* Disponível em: <http://www.omg.org/mda> Acesso em: 15/03/2008.

[7] **TinyOS.** Disponível em: <http://www.tinyos.net/> Acesso em: 21/01/2008.

[8] HILL, J. ***System Architecture for Wireless Sensor Networks.*** PhD thesis, Univerisy of California, Berkeley, 2003.

[9] BEATRYS, R. L. **Arquitetura para Redes de Sensores Sem Fio.** Disponível em: <http://www.sensornet.dcc.ufmg.br/pdf/mc-sbrc2004.pdf > acesso em 26/02/2009.

[10] CHEESMAN, J.; DANIELS, J. ***UML Components, a simple process for specifying component-based software.*** 1 ed. Reading: Addison-Wesley, 2001.

[11] D'SOUZA, D.; WILLS, A. ***Objects, components and frameworks: The Catalisys Approach.*** *1 ed.* Reading: Addison-Wesley, 2001.

[12] GIMENES, I. M. S.; HUZITA, E.H.M. ***Desenvolvimento Baseado em Componentes: Conceitos e Técnicas.*** Rio de Janeiro: Editora Ciência Moderna Ltda., 2005.

[13] GAY, D.; LEVIS, P.; BEHREN, V. R.; WELSH, M.; BREWER, E.; CULLER, D. **The nesC language: A holistic approach to networked embedded systems.** Disponível em: <http://www.cs.berkeley.edu/~pal/pubs/nesc.pdf> acesso em 04/02/2008.

[14] Hill, J; SZEWCZYK, R.; WOO, A.; HOLLAR, S.; CULLER, D. E.; PISTER, K. S. J. **System Architecture Directions for Networked Sensors**. Disponível em: <http://www.tinyos.net/papers/tos.pdf> acesso em 01/04/2008.

[15] GAY, D. LEVIS, P.; CULLER, D. BREWER E. **nesC 1.1 Language Reference Manual**. Disponível em: <http://nescc.sourceforge.net/papers/nesc-ref.pdf > acesso em 01/04/2008.

[16] BOOCH, G; et al. **The Unified Modeling Language User Guide**. 1998.

[17] FOWLER, M.; SCOTT, K. **UML Essencial: um breve guia para a linguagem padrão de modelagem de objetos.** Porto Alegre: Bookman, 2000.

[18] MELO, A. C. **Desenvolvendo Aplicações com UML.** Rio de Janeiro: Brassport, 2002.

[19] JÚNIOR, A. J. C. **Aplicando *Model-Driven Development* à Plataforma GPGPU.** Disponível em: <http://www.cin.ufpe.br/~tg/2008-2/ajcj.pdf> acesso em 24/05/2010.

[20] GUIMARÃES, G. B.; NETO, R. M. **Geração de Aplicação Web usando a MDA.** Disponível em: <http://www.cci.unama.br/margalho/portaltcc/tcc20082s/pdf/claudio02.pdf > acesso em 24/05/2010.

[21] **Labview Wireless Sensor Network Pioneer.** Disponível em: <http://zone.ni.com/devzone/cda/tut/p/id/8981 > acesso em 02/07/2010.