# WISDOM: Desenvolvimento Multiplataforma e Visual para Redes de Sensores sem Fio

Breno A. D. Vitorino, Vinícius C. de Almeida, Luiz F. M. Vieira,
Marcos A. M. Vieira, Antônio O. Fernandes, Diógenes C. da Silva Jr.

Departamento de Ciência da Computação e Departamento de Engenharia Elétrica
Universidade Federal de Minas Gerais
Avenida Antônio Carlos, 6627 – Pampulha
Belo Horizonte – MG – Brazil
{vitorino, makish, lfvieira, mmvieira, otavio}@dcc.ufmg.br, diogenes@eee.ufmg.br

***Resumo*— Por meio de um novo paradigma de computação, as redes de sensores sem fio revolucionam a monitoração de ambientes, possibilitando aplicações nas mais diversas áreas do conhecimento. Entretanto, o processo de criação de aplicações para essas redes ainda não é simples, sendo bastante dependente do sistema operacional e do *hardware* alvo. Para suprir essa lacuna, desenvolveu-se neste trabalho um novo modelo de programação e uma ferramenta capaz de, a partir de uma especificação baseada em módulos, gerar o código-fonte de uma aplicação para uma plataforma computacional escolhida pelo usuário. Essa nova abordagem facilita a programação e permite a geração de código eficiente, atendendo às restrições severas dessas redes. Dois casos de uso demonstram o potencial do WISDOM para ajudar os desenvolvedores de aplicações para essas redes.**

## I. 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 e luminosidade [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, agricultura de precisão e rastreamento de alvos. Uma coleção de nós sensores trabalhando em uma aplicação compõem uma rede de sensores sem fio (RSSF).

Ao programar os nós de uma RSSF, o desenvolvedor de *software* deve lidar com computação concorrente. Por exemplo, uma aplicação tem que ser capaz de rotear mensagens enquanto aplica um filtro aos dados coletados antes de enviá-los pelo rádio. Aspectos de concorrência, como escalonamento e condições de corrida, são lidados diretamente pelo desenvolvedor, utilizando rotinas de um sistema operacional. Essas limitações levam a programas difíceis de entender e susceptíveis a erros à medida que se tornam mais complexos.

O desenvolvedor também deve lidar com características específicas dos componentes de *hardware*. Para programar uma aplicação para enviar uma mensagem, ele deve conhecer a interface de programação de aplicativos (API) do componente de comunicação, fornecida pelo sistema operacional, ou mesmo programar esse componente em linguagem de máquina. Como ainda não existe nenhuma API padrão para RSSF, isso impossibilita o reuso desse código ao associá-lo a outro sistema operacional.

Para superar essas dificuldades, apresenta-se neste artigo WISDOM, uma ferramenta visual capaz de gerar código nativo para uma dada plataforma computacional, a partir de uma especificação de acordo com um novo modelo de programação proposto. Plataforma computacional é definida neste trabalho como a combinação do sistema operacional e do *hardware* utilizados por uma dada aplicação. Em uma aplicação modelada com o WISDOM, cada módulo abstrai uma conjunto de funcionalidades correlatas, e as ligações entre os módulos definem os fluxos de execução e dados. Módulos pré-definidos possuem métodos para acessar os dispositivos de *hardware*, como os sensores e rádio. Usuários do WISDOM podem especificar seus próprios módulos para determinar o comportamento da aplicação.

Os objetivos do WISDOM são simplificar o desenvolvimento de aplicações para RSSFs e prover um modelo de programação multiplataforma, ou seja, que permita que uma aplicação possa ser especifica e codificada uma única vez para diversas plataformas computacionais. O modelo de programação adotado pelo WISDOM assume somente o entendimento da linguagem ANSI-C. Nenhuma biblioteca em C deve ser usada ou aprendida, pois a própria ferramenta se encarrega de abstrair o acesso aos dispositivos de *hardware*. A programação em módulos previsto por esse modelo permite o reuso de código, sendo possível criar uma biblioteca de módulos comumente utilizados nas aplicações. Essa ferramenta também permite a visualização de todos os módulos e suas dependências através de um ambiente gráfico.

O modelo de programação do WISDOM contém uma abstração intuitiva para concorrência. Nenhum dado é explicitamente compartilhado entre módulos. Quando um módulo determina que um dado está pronto, ele coloca esse dado à dis-

posição dos demais e imediatamente retorna ao processamento de novos dados de entrada. Desse modo, é possível obter alta concorrência através de uma execução baseada em eventos.

O texto deste artigo está organizado da seguinte forma. A seção II apresenta conceitos importantes sobre RSSFs e a descrição das plataformas computacionais atuais. A seção III descreve o modelo de programação WISDOM, e a seção IV exibe a ferramenta WISDOM, a qual utiliza esse modelo. A seção V apresenta dois estudos de caso do WISDOM, os quais mostram o potencial dessa ferramenta para o desenvolvimento de aplicações para RSSFs. A seção VI mostra os trabalhos relacionados ao WISDOM, e finalmente a seção VII apresenta as considerações finais e direções futuras.

## II. Redes de Sensores sem Fio

RSSFs são compostas por nós sensores, os quais podem pertencer a diferentes plataformas, formando RSSFs heterogêneas. A tendência atual é a integração de soluções. Programar cada plataforma computacional requer o desenvolvimento de código diferente para uma mesma aplicação dessa RSSF. Para evitar esse retrabalho, WISDOM define um novo modelo de programação multiplataforma.

Atualmente, as RSSFs são um rico campo de pesquisa [3], [4] e, enquanto a tecnologia de micro sistemas eletromecânicos evolui, mais plataformas computacionais são desenvolvidas [5]. Apesar dessas plataformas terem suas particularidades, elas compartilham alguns conceitos básicos que são utilizados para construir o modelo de programação proposto. A seguir, serão descritas quatro plataformas conhecidas, suas particularidades e características comuns:

### A. TinyOS e Mica motes

Essa plataforma computacional, desenvolvida na Universidade da California, Berkeley, atualmente é a mais difundida. O nó sensor Mica2, pertencente à família Mica motes, possui um processador ATMEL ATMEGA, transceptor de rádio CC1000, uma placa de sensores, memória EEPROM externa e bateria [6]. A placa de sensores padrão inclui sensores de temperatura e luz, acelerômetros de dois eixos e um magnetômetro. TinyOS foi o primeiro sistema operacional desenhado para executar nesse *hardware*.

O TinyOS [7] contém um escalonador dirigido por eventos. Esse escalonador introduz os conceitos de eventos e tarefas. Eventos são abstrações para eventos de *hardware*, como a amostragem de um sensor ou recebimento de mensagens, ou de *software*. Os eventos de *hardware* são as respostas de requisições não blocantes a serviços do *hardware*. Eventos possuem prioridade alta, não são interruptíveis e interrompem e escalonam tarefas. Tarefas contêm a computação principal da aplicação e são disparadas pelo escalonador. Tarefas não interrompem outras tarefas e, se necessário, podem escalonar outras tarefas.

Aplicações do TinyOS são escritas em nesC [8], uma extensão da linguagem C. nesC inclui uma sintaxe para expressar tarefas, eventos e componentes. Componentes agregam funcionalidades similares de uma aplicação, permitindo modularizar essa aplicação. WISDOM gera código em nesC quando essa plataforma é selecionada.

### B. Mantis OS e Nymph

Essa plataforma computacional foi desenvolvida pelo grupo Mantis da Universidade do Colorado, Boulder. O sistema operacional do Mantis foi desenhado primeiramente para executar no nó sensor Nymph, desse mesmo grupo. Nymph [9] é inspirado na arquitetura do Mica e possui o mesmo processador e transceptor de rádio. A diferença principal entre as duas arquiteturas é que as interfaces de programação e sensoriamento do Nymph estão na mesma placa do processador e rádio.

O sistema operacional Mantis [9] provê um subconjunto da API para *threads POSIX*. *Threads* são escalonadas para execução quando um temporizador do sistema operacional é ativado ou em chamadas de sistema ou operações de semáforo. Acesso ao *hardware* é realizado por chamadas blocantes. Esse escalonador claramente não é dirigido por eventos, mas é possível simular um instanciando uma *thread* para cada evento de *hardware*.

O sistema operacional Mantis exporta uma API em linguagem C que pode ser facilmente incorporada ao WISDOM a fim de implementar programas usando essa plataforma computacional.

### C. PeerOS e EYES

O projeto de pesquisa do grupo europeu EYES desenvolveu sua própria plataforma computacional: o sistema operacional PeerOS e o nó sensor EYES. O nó sensor EYES [10] usa o microcontrolador MSP430 como seu processador principal e o transceptor de rádio TR1000. Ele também possui memória externa, bateria e placa de sensores, similares aos do Mica mote.

PeerOS possui um escalonador de tarefas preemptivo baseado no algoritmo EDFI [11]. Esse algoritmo garante algumas características de tempo real com trocas de contexto rápidas.

Essa plataforma computacional pode ser integrada ao WISDOM se for utilizada a mesma metodologia sugerida ao Mantis. A API do PeerOS é escrita em C e pode ser facilmente suportada pela ferramenta WISDOM.

### D. Yatos e BEAN

O sistema operacional Yatos e o nó sensor BEAN foram desenvolvidos pela Universidade Federal de Minas Gerais através do projeto SensorNet. BEAN [12] possui o microcontrolador MSP430F149, o transceptor de rádio CC1000 e o conector para uma placa de sensores. Esse nó sensor é o primeiro a permitir medir a corrente em cada componente.

Yatos [13] é um sistema operacional dirigido por eventos desenvolvido para o BEAN. Ele possui os conceitos de eventos e tarefas, similares ao do TinyOS. Desenvolvedores que utilizam o Yatos podem declarar quais eventos de *hardware* são necessários à aplicação e associar tarefas a cada um desses eventos. Quando um desses eventos ocorrer, as tarefas associadas são escalonadas em uma fila de prioridades. O YATOS

utiliza os vários LPM (“low power modes”) suportados pelo BEAN para economizar energia, que é o recurso mais precioso em RSSF. Quando não há tarefas no escalonador, o Yatos põe o BEAN no modo de maior economia de energia.

A API do Yatos é escrita em C e a linguagem ANSI-C é utilizada para escrever aplicações para esse sistema operacional.

## III. Modelo de Programação WISDOM

O modelo de programação WISDOM permite definir o comportamento da comunicação, sensoriamento e processamento de um nó sensor como parte de uma RSSF. A partir dessa definição, a ferramenta WISDOM é capaz de gerar código nativo para qualquer plataforma computacional que ela suportar. Antes de descrever os conceitos, será apresentado as hipóteses adotadas pelo modelo de programação WISDOM. Essas hipóteses garantem sua independência de plataforma computacional. Elas não são restritivas: todas as plataformas descritas na seção II estão de acordo com as seguintes hipóteses:

- A lógica de negócios da aplicação pode ser expressa através da linguagem ANSI-C. Através da ferramenta WISDOM, essa especificação será integrada às funcionalidades do sistema operacional para construir a aplicação final.
- O sistema operacional suporta escalonamento dirigido por eventos. Toda comunicação com os componentes de *hardware* deve ser feito de forma não blocante. Dessa forma, uma interrupção de *hardware* é disparada quando algum dispositivo conter dados válidos para a aplicação.

Uma aplicação desenvolvida no WISDOM é composta de módulos. Um módulo é uma abstração de um conjunto de funções correlatas. Por exemplo, existem módulos para acessar um sensor específico, para enviar e receber mensagens pelo transceptor, ou para filtrar algum conjunto de dados.

Existem dois tipos de módulos: módulos de sistema e módulos de usuário. Módulos de sistema são módulos pré-construídos que abstraem o acesso ao *hardware* do nó sensor. Eles iniciam qualquer computação da aplicação, estimulados pelos dados recebidos do ambiente. Seus campos e métodos não podem ser modificados pelo desenvolvedor; os cabeçalhos de seus métodos são definidos a priori e sua implementação é dependente da plataforma. Módulos de usuário, por outro lado, pode ser livremente especificados pelo desenvolvedor. Eles abstraem as funcionalidades da aplicação não cobertas pelos módulos de sistema, como filtragem de dados e tratamento de mensagens.

Todos os módulos são compostos de campos e métodos. Métodos são os procedimentos cujo acesso é disponível aos outros módulos, e campos são variáveis internas aos módulos, acessadas apenas pelos métodos do módulo que as definiu. Todos os campos e métodos são descritos em linguagem ANSI-C. Existem dois tipos de métodos: métodos sinalizadores e métodos receptores.

Um método sinalizador é definido para cada evento de interesse da aplicação. Um método sinalizador é disparado toda vez que ocorre o evento de interesse associado. Em módulos de sistema, esses métodos correspodem a eventos de *hardware*, como a recepção de dados através do transceptor ou uma leitura de um sensor, e são disparados em resposta a interrupções de *hardware*. Em módulos de usuário, eles correspodem a eventos de *software*, como a conclusão de alguma computação ou a ocorrência de uma exceção, e são chamados pelos métodos receptores de seu módulo na ocorrência do evento.

Os métodos receptores são os que contêm a computação responsável por tratar um evento de interesse da aplicação. Em módulos de sistema, incluem funcionalidades como enviar mensagens pelo transceptor ou alterar o estado dos leds do nó sensor. Em módulos de usuário, por sua vez, podem filtrar dados de entrada ou tratar uma exceção. Esses métodos são executados por um ou mais métodos sinalizadores associados a ele.

A associação entre métodos sinalizadores e métodos receptores denomina-se conexão. Uma conexão entre um método sinalizador X e um método receptor Y indica que Y é executado toda vez que X é executado. As conexões são definidas estaticamente, durante o desenvolvimento da aplicação. Uma conexão entre métodos sinalizadores e receptores é possível apenas quando o tipo e a ordem da declaração de seus parâmetros é igual. A cardinalidade possível para uma conexão é de muitos para muitos, tal que um método sinalizador pode invocar muitos métodos receptores em uma ordem arbitrária, e um método receptor pode ser disparado em resposta a múltiplos métodos sinalizadores.

O conjunto de módulos escolhidos e suas conexões definem o fluxo de execução e de dados da aplicação. O modelo trabalha com uma execução dirigida por eventos. Toda computação é iniciada por interrupções de *hardware*, as quais correspondem aos métodos sinalizadores dos módulos de sistema. Esses métodos sinalizadores repassam dados para os métodos receptores associados. Dessa forma, o fluxo de execução define onde o dado vai ser processado e transferido. Um fluxo de execução parte de um módulo de sistema, continua por zero ou mais módulos de usuário, até atingir outro módulo de sistema. Esse último módulo é responsável por transmitir o dado para fora do nó sensor, pelo meio sem fio (para outros nós) ou por conexões cabeadas (para nós conectados ao ponto de acesso).

## IV. Ferramenta WISDOM

A ferramenta WISDOM auxilia o desenvolvimento de programas para nós sensores de uma RSSF, de acordo com o modelo de programação descrito na seção anterior. Ele possui uma interface gráfica, através da qual é possível inserir módulos em uma aplicação, definr métodos sinalizadores e receptores, estabelecer conexões entre eles, e finalmente gerar código para uma plataforma computacional. Atualmente WISDOM suporta as seguintes plataformas: TinyOS/Mica Motes e Yatos/BEAN. Ela foi implementada na linguagem Java, e possui código aberto.

As principais características do WISDOM são:

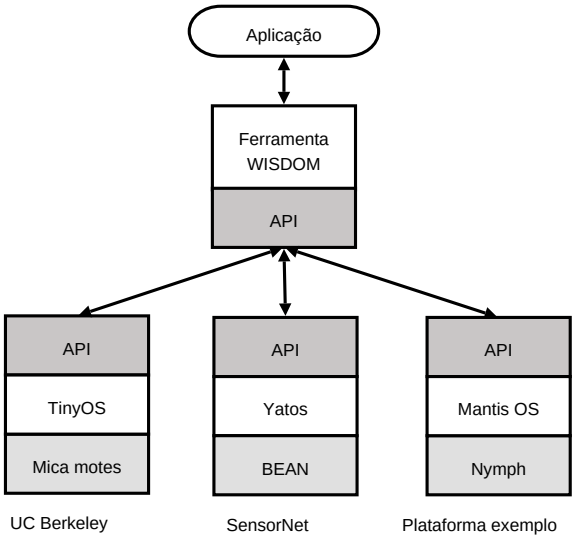


Fig. 1. Geração de código para múltiplas plataformas.

- Suporte à programação visual. A visualização dos módulos e suas conexões permite ao desenvolvedor ter uma melhor visão do fluxo de execução e dados da aplicação, o que implica em redução do tempo de desenvolvimento e testes. Espera-se que essa característica possa facilitar a programação para não-especialistas, abstraindo aspectos como criação e gerenciamento de tarefas concorrentes e o aprendizado de uma nova linguagem e/ou sistema operacional.
- Geração de código para múltiplas plataformas. O modelo de programação WISDOM não assume nenhuma plataforma computacional específica. Então, a partir de uma especificação de uma aplicação utilizando esse modelo, a ferramenta WISDOM pode analisá-la e gerar código para uma dada plataforma suportada (figura 1). Essa característica também permite portabilidade de código, o que é importante para RSSFs heterogêneas.
- Extensibilidade. A ferramenta WISDOM foi desenhada para ser estendida para gerar código para outras plataformas, além daquelas atualmente suportadas. Já existe um conjunto de classes que provê a funcionalidade básica para adicionar novas plataformas.

Nas próximas subseções será detalhada a arquitetura e as características da ferramenta WISDOM.

### *A. Arquitetura de Software*

WISDOM foi implementado em quatro camadas: apresentação, controle, núcleo e plataforma. Essas camadas e suas relações são exibidas na figura 2.

A camada de apresentação é responsável pela interface da ferramenta WISDOM. Ela contém todas as telas, as representações gráficas dos módulos e conexões, e o gerenciamento desses elementos gráficos na janela.

A camada de controle faz a ligação entre as camadas de visualização e núcleo. Isso significa que essa camada instancia e utiliza as classes do núcleo para cada ação do usuário da

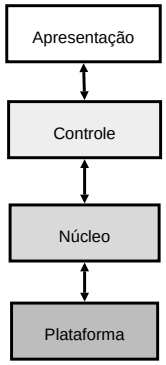


Fig. 2. Arquitetura de software da ferramenta WISDOM.

interface. Essa camada simplesmente desacopla a interface da lógica de negócio da ferramenta.

A camada núcleo efetivamente implementa o modelo de programação. Ela contém as classes que implementam os módulos de sistema e usuário, métodos receptores e sinalizadores, e conexões. A classe Application contém a coleção de módulos e conexões que formam uma aplicação. Essa classe pode ser salva ou carregada de um arquivo se for conveniente ao usuário. Essa camada também contém a lógica básica para geração de código, selecionando as classes da camada plataforma adequadas à plataforma computacional escolhida.

A camada plataforma contém código específico para cada plataforma computacional suportada pelo WISDOM. Existem classes que contém a implementação de cada módulo de sistema disponível na ferramenta. Ela também contém a lógica para gerar as chamadas de sistema adequadas e código para implementar cada conexão estabelecida na aplicação.

### *B. Descrição e uso da ferramenta*

Um módulo nessa ferramenta é representado por um retângulo com três campos: um campo identificador, que contém seu nome e um ícone que representa sua funcionalidade; um campo de métodos sinalizadores, que mostra o conjunto corrente de métodos sinalizadores desse módulo; um campo de métodos receptores, que contém todos os métodos receptores desse módulo.

A ferramenta WISDOM permite inserir módulos de sistema e usuário. A coleção de módulos de sistema disponíveis é mostrada em uma caixa de seleção para o usuário. Para criar novos módulos de sistema, é necessário estender a ferramenta com novas classes que definam esse novo módulo e determinar as regras de geração de código para cada plataforma suportada. Há também um repositório de classes de módulos de sistema que são utilizados comumente em aplicações. Um novo módulo de usuário pode ser facilmente criado escrevendo seu nome, descrição e métodos sinalizadores e receptores numa janela de criação de módulos. Após a criação desse módulo, essas informações podem ser modificadas caso necessário.

Uma conexão entre módulo é graficamente apresentada como uma seta entre o método sinalizador de um módulo para o método receptor de outro módulo. As conexões são criadas

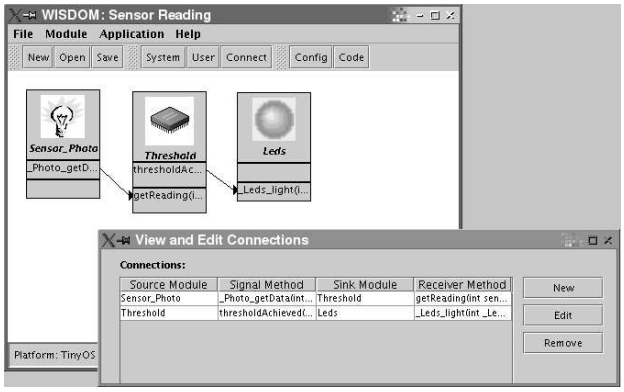


Fig. 3. Gerenciamento de conexões.

selecionando-se um módulo fonte, seu método sinalizador, o módulo de destino e seu método receptor. A conexão é então inserida em uma lista de conexões de uma aplicação, a qual pode ser visualizada ou modificada em uma janela (figura 3). Após confirmar as modificações nessa janela, a aplicação é atualizada com a nova seta (ou com a remoção de setas, se alguma das conexões foi removida da lista).

O código dos métodos receptores pode sempre ser modificado clicando-se em qualquer método receptor no campo de métodos receptores de um módulo. O código deve ser escrito em ANSI-C. Como o modelo não supõe que o sistema operacional possua gerenciamento de memória, alocação dinâmica de memória não é possível. Se o usuário necessitar manter dados entre execuções de um método receptor, ele pode declarar variáveis fora do escopo do método. Essas variáveis são globais, mas não são acessíveis por outros métodos, portanto isso proíbe o acesso compartilhado dessas variáveis.

Após selecionar os módulos de usuário e de sistema, construir as conexões entre eles e definir o corpo dos métodos receptores para novos módulos de sistema, a ferramenta está pronta para gerar código. O usuário escolhe a plataforma computacional alvo, e a ferramenta WISDOM traduz a especificação em código nativo para essa plataforma computacional. Múltiplos arquivos, prontos para serem compilados, são gerados após esse passo. O código binário pode ser então gravado na memória do nó sensor por um dispositivo JTAG, por exemplo.

### C. Geração de Código

A tradução da especificação da aplicação para código fonte para a plataforma computacional alvo envolve quatro passos: geração de código de inicialização, geração de código dos módulos de sistema, geração de código das conexões, e análise de precedência de código.

O código de inicialização contém algum conjunto de chamadas que devem ser realizadas antes da execução da aplicação. Essas rotinas compreendem a inicialização do hardware, como a freqüência do relógio do processador ou a habilitação de alguns componentes de *hardware*.

Outro conjunto de rotinas são necessárias para acessar os módulos de *hardware*. Essas rotinas são geradas de acordo com os módulos de sistema usados na aplicação. Por exemplo, o módulo de sistema Radio possui os métodos send e receive, os quais são preenchidos pelas chamadas de sistema da camada de enlace que enviam e recebem mensagens do rádio. Tarefas, tais como recebimento de mensagens e conversões analógicas-digitais, são realizadas de forma não blocante. Isso significa que essas tarefas serão escalonadas e os componentes de hardware são responsável por assinalar uma interrupção de *hardware* quando a tarefa estiver pronta.

Em seguida há a geração de código das conexões. Para cada conexão, a ferramenta WISDOM dispara uma tarefa para cada método receptor associado a um método sinalizador. O disparo é realizado apenas se esse método receptor já não estiver presente na fila do escalonador. Se for este o caso, o evento será simplesmente ignorado pelo método receptor. Na realidade, existe um esquema de passagem de mensagens com uma fila de tamanho um. Essa estratégia não aloca mais memória do que a exigida pelo usuário, mas esse usuário deve estar atento à taxa de amostragem dos componentes de *hardware* para evitar a perda de muitos eventos.

Finalmente, o código é escrito para os arquivos respeitando-se a ordem de precedência das chamadas. Na linguagem C, não é possível chamar uma subrotina se ela não foi previamente declarada no código. Essa precedência é alcançada escrevendo-se primeiro os métodos sinalizadores dos módulos de sistema, e depois percorrendo as conexões entre métodos sinalizadores e receptores até que não haja método sinalizador conectado em um dado módulo.

## V. Resultados

Nessa seção serão mostradas duas aplicações geradas com o auxílio do WISDOM. Uma aplicação amostra o sensor de luz e exibe os valores amostrados nos leds. A outra coleta os dados de luminosidade e os envia pela rede. Seu algoritmo de roteamento implementa fusão de dados nos nós intermediários. É importante ressaltar que não é o objetivo apresentar as melhores soluções para as aplicações propostas, mas mostrar o potencial da ferramenta em facilitar a programação dos nós sensores e demonstrar o modelo de programação na prática.

Ambas aplicações foram testadas nos nós sensores Mica2 executando TinyOS. Também foi gerado código para a plataforma computacional Yatos/BEAN, o qual foi testado por ferramentas de simulação.

### A. Estudo de caso: Leitura do Sensor

O objetivo dessa aplicação é mostrar nos leds do nó sensor o valor (em binário) das leituras do sensor de luz que ultrapassarem um limiar definido pelo desenvolvedor. Somente os bits mais significantes são mostrados nos leds, quantos forem disponíveis no nó sensor. No nó sensor Mica existe três leds disponíveis, e o bit mais significativo é mostrado no led vermelho. Com uma aplicação simples será mostrado cada passo necessário para compor um programa na ferramenta WISDOM.

Primeiramente, escolheram-se os módulos Sensor_Light e Leds. Eles encapsulam os componentes de *hardware* necessários. Criou-se um módulo Threshold que irá acender os leds

somente quando as leituras de luminosidade ultrapassarem o limiar. Esse módulo de usuário possui um método receptor, getReading, que obtém as leituras do sensor e confronta esses valores com o limiar. Threshold possui um método sinalizador, thresholdAchieved, que é invocado por getReading cada vez que o limiar é ultrapassado. Esses três módulos são o suficiente para compor esta aplicação.

Após escolher todos os módulos necessários, deve-se conectá-los. Conectar o método sinalizador de Sensor_Light com getReading em Threshold significa que todas as leituras do sensor serão tratadas por esse método receptor. Conectar thresholdAchieved de Threshold com o método receptor de Leds transferirá os valores acima do limiar para os leds convenientemente.

Nesse momento, aplicação está pronta. A ferramenta WISDOM pode gerar código para a plataforma alvo, o qual deve ser depois compilado e carregado no nó sensor. O resultado final é exibido na figura 4. O código gerado para as plataformas TinyOS/Mica motes e Yatos/BEAN são mostrados nas listagens 1 e 2.

Programa 1. Código da aplicação Leitura do Sensor para a plataforma computacional TinyOS/Mica motes.

```
module SensorLedsM {
        ...
}
implementation {
  command result_t StdControl.init()
        { ... }
  command result_t StdControl.start()
        { ... }
  command result_t StdControl.stop()
        { ... }
  ...

  task _Leds_light() { ... }
  void thresholdAchieved(int nivel) { ... }
  task void getReading() { ... }
  void _Temp_getData(int _Temp_data) { ... }
  event result_t TempTimer.fired() { ... }
  async event result_t TempADC.dataReady(uint16_t data) {
      ... }
}
```

### B. Estudo de caso: Fusão de Dados

Nesse estudo de caso, apresenta-se como a ferramenta WISDOM lida com comunicação entre nós sensores. Nessa aplicação, os nós sensores possuem diferentes papéis na rede, os quais colaboram entre si, o que não é um cenário incomum nessas redes.

Existem três papéis: coletores, agregadores e ponto de acesso. Os nós sensores coletores apenas coletam as leituras do sensor de luz e transmitem esses dados aos seus agregadores através do radio. Cada coletor associa-se a um agregador, e esse assinalamento é estático. Um nó sensor é agregador se seu endereço módulo GROUP_SIZE é igual a zero. GROUP_SIZE é uma constante que define quantos coletores existirão para cada agregador. O papel do agregador é receber as amostragens dos seus coletores e fundi-los, transmitindo ao ponto de acesso apenas a média dos últimas dez leituras. O ponto de acesso

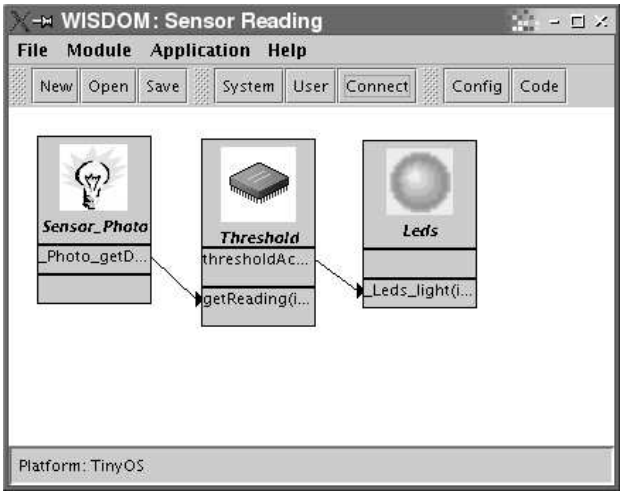


Fig. 4. Aplicação de leitura do sensor na ferramenta WISDOM.

irá receber as mensagens dos diversos coletores e reenviá-las através de uma porta serial de um PC comum. O computador lê as mensagens da serial e as exibe num gráfico, para melhor visualização. A topologia descrita é ilustrada na figura 5.

Programa 2. Código da aplicação Leitura do Sensor para a plataforma computacional BEAN/Yatos.

```
#include "SensorLeds.h"
#include "SO.h"
...
void _Leds_light(evento_t evt) { ... }
void thresholdAchieved(int nivel) { ... }
void getReading(evento_t evt) { ... }
void _Temp_getData(int _Temp_data) { ... }
void tempoTask_Sensor_Temp(evento_t evt){ ... }
void sensorTask_Sensor_Temp(evento_t evt){ ... }
void main(){
  Microcontrolador_IniciaHardware();

  Tarefa_Declara(&getReading,128,getReading);

  Tarefa_AtribuiEvento(
    Tarefa_Declara(&tempoTask_Sensor_Temp,128,
      tempoTask_Sensor_Temp)
  ,evTEMPORIZADOR0,tpdPERIODICO,1000);

  Tarefa_AtribuiEvento(
    Tarefa_Declara(&sensorTask_Sensor_Temp,128,
      sensorTask_Sensor_Temp)
  ,evSENSORADC0,tpdNULO,pdNulo);

  Microcontrolador_IniciaSO();
}
```

Todo esse cenário será descrito numa única aplicação WISDOM, na qual os nós sensores estão sempre testando seu endereço para verificar seus papéis na rede. Esse cenário também poderia ser descrito em três aplicações, um para cada papel, mas a primeira abordagem foi adotada pois ela permite testar a aplicação primeiramente em um simulador como o TOSSIM [14]. O TOSSIM é capaz de simular uma RSSF na qual cada um dos nós está executando o mesmo programa.

Primeiramente, selecionaram-se os módulos necessários para essa aplicação. Foi utilizado os módulos de sistema Radio, Sensor_Light e IOPorts. O módulo Radio é responsável

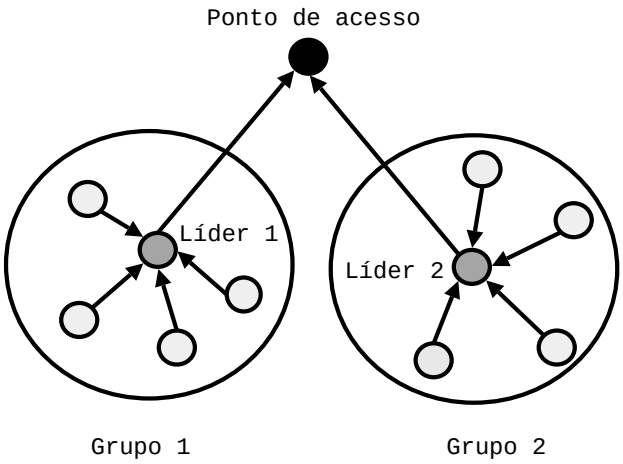


Fig. 5. Topologia da rede para a aplicação Fusão de Dados. Note que, para este exemplo, GROUP_SIZE é igual a 5.

por enviar e receber mensagens, Sensor_Light por coletar leituras de luminosidade e IOPorts por transportar as mensagens pela porta serial até o PC. Também foram criados dois módulos de usuário: MeanFusion e Base. O último é responsável por repassar as mensagens se o nó sensor é um ponto de acesso, e o primeiro contém o algoritmo de roteamento e as regras de fusão de dados.

Programa 3. Código da aplicação Fusão de Dados para a plataforma computacional TinyOS/Mica motes.

```
module SensorRadioM {
        ...
}
implementation {
  command result_t StdControl.init() { ... }
  command result_t StdControl.start() { ... }
  command result_t StdControl.stop() { ... }
  task void send() { ... }
  event result_t SendMsg.sendDone(TOS_MsgPtr msg, bool
       success) { ... }
  task void uartSend() { ... }
  event result_t UARTSend.sendDone(TOS_MsgPtr msg, bool
       success) { ... }
  void fuseReady(Packet* _Fusion_packet) { ... }
  void processReading(Packet* _Base_packet) { ... }
  task void receiveReading() { ... }
  task void collectRadio() { ... }
  task void collectSensor() { ... }
  void recv(Packet* _Radio_recvPacket) { ... }
  event TOS_MsgPtr ReceiveMsg.receive(TOS_MsgPtr
       _Radio_recv_packet){ ... }
  void _Photo_getData(int _Photo_data) { ... }
  event result_t PhotoTimer.fired() { ... }
  async event result_t PhotoADC.dataReady(uint16_t data) {
       ... }
}
```

As conexões entre os módulos são estabelecidas da seguinte maneira. Sensor_Light foi conectado ao método receptor em MeanFusion que define o comportamento dos coletores. Da mesa forma, o método sinalizador receive do módulo Radio foi conectado a outro método receptor em MeanFusion responsável pelo algoritmo dos agregadores. Ambos os métodos receptores de MeanFusion chamam seu método sinalizador

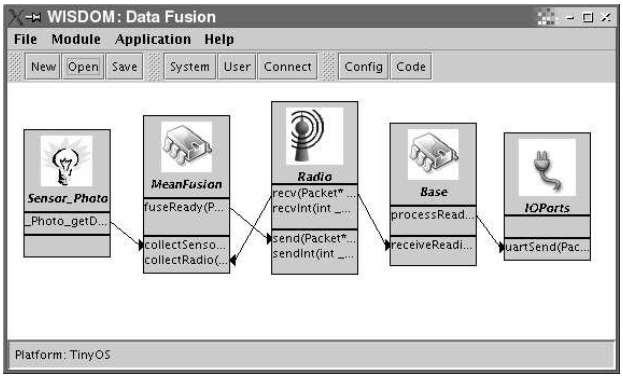


Fig. 6. Aplicação Fusão de Dados modelada na ferramenta WISDOM

para indicar que uma mensagem está pronta para ser enviada pela rede. Logo, esse método sinalizador foi conectado ao método receptor send de Radio. O módulo Base determina a lógica de negócio do ponto de acesso, e está conectado aos módulos Radio e IOPorts.

Esta aplicação na ferramenta WISDOM é mostrada na figura 6. Seu código fonte para as plataformas computacionais TinyOS/Mica motes e Yatos/BEAN, gerados pela ferramenta WISDOM, é mostrado nas listagens 3 e 4.

Programa 4. Código da aplicação Fusão de Dados para a plataforma computacional Yatos/BEAN.

```
#include "SensorRadio.h"
#include "SO.h"
...
void send(evento_t evt) { ... }
void uartSend(evento_t evt) { ... }
void fuseReady(Packet* _Fusion_packet) { ... }
void processReading(Packet* _Base_packet) { ... }
void receiveReading(evento_t evt) { ... }
void collectRadio(evento_t evt) { ... }
void collectSensor(evento_t evt) { ... }
void recv(Packet* _Radio_packet) { ... }
void radioTask_receive(evento_t evt){ ... }
void _Photo_getData(int _Photo_data) { ... }
void tempoTask_Sensor_Photo(evento_t evt){ ... }
void sensorTask_Sensor_Photo(evento_t evt){ ... }
void main(){ ... }
```

## VI. TRABALHOS RELACIONADOS

O modelo de programação WISDOM foi desenhado usando abordagens usuais para RSSFs, como aplicações dirigidas por eventos [15] e modularidade [8]. O conceito de métodos receptores e sinalizadores é análogo à especificação das entradas e saídas de um módulo, mas sua visualização em uma ferramenta foi inspirado em QT [16].

Foram investigadas as plataformas computacionais que se aplicam ao WISDOM. Elas incluem as plataformas TinyOS/-Mica Motes [7] Yatos/BEAN [13] atualmente suportadas. Outras plataformas como Mantis [9] e PeerOS [11] também possuem os pré-requisitos estabelecidos pelo modelo de programação WISDOM: suporte à linguagem ANSI-C e escalonamento dirigido por eventos. Espera-se que essas plataformas

possam tirar vantagem da ferramenta WISDOM para rápido desenvolvimento de aplicações.

Esta é a primeira ferramenta para programação visual para RSSF que gera código para múltiplas plataformas. Mas existem modelos de programação para RSSFs, como o TinyGALS [17]. Seu objetivo é tratar problemas de concorrência comuns em aplicações escritas em TinyOS. Ele oferece uma linguagem que permite a separação da parte síncrona da assíncrona. A parte síncrona é escrita em uma linguagem imperativa semelhante a C, enquanto a comunicação assíncrona é especificada como conexões entre componentes. Um compilador traduz as conexões em estruturas de baixo nível, como passagem de mensagens através de filas e variáveis protegidas.

Apesar da linguagem TinyGALS possuir muitas similaridades com o modelo de programação WISDOM, este trabalho acabou por estender os conceitos do TinyGALS. O TinyGALS foi construído sobre os conceitos de nesC e TinyOS. O modelo de programação WISDOM, por sua vez, é independente de plataforma, tal que os seus benefícios podem ser estendidos para as plataformas computacionais futuras. A ferramenta visual e a adoção da linguagem ANSI-C fazem dessa abordagem mais simples de aprender para programadores novatos em RSSFs.

## VII. Conclusão

Neste trabalho foi apresentado o WISDOM, uma ferramenta e um modelo de programação para nós de RSSFs. O desenvolvedor pode rapidamente construir programas através da ferramenta WISDOM. Ele pode selecionar quais módulos sua aplicação necessita e conectá-los para definir o fluxo de execução dessa aplicação. Uma interface gráfica permite a visualização e modificação desses módulos e de suas conexões de maneira intuitiva. Associou-se um novo modelo de programação, independente de plataforma, à ferramenta WISDOM.

Mostraram-se algumas aplicações-exemplo modeladas no WISDOM. O ambiente visual permitiu a melhor compreensão de como a aplicação funcionará no nó sensor e como seus módulos interagem entre si. Os exemplos foram testados nos nós sensores, validando o código gerado pela ferramenta.

Como direções futuras deste trabalho, visualiza-se a criação de uma biblioteca de módulos de usuário com algoritmos úteis para diferentes aplicações. Propõe-se também que a ferramenta WISDOM seja estendida para que seja capaz de verificar propriedades de uma aplicação, modelando os módulos e suas conexões como uma rede de Petri.

## VIII. Agradecimentos

Este trabalho foi parcialmente apoiado pelo CNPq, processos 55.2111/2002-3, 18.0381/2003-2, 18.0380/2003-6 e 830107/2002-9.

## Referências

[1] D. Estrin, R. Govindan, J. S. Heidemann, and S. Kumar, "Next century challenges: Scalable coordination in sensor networks," in *Mobile Computing and Networking*, 1999, pp. 263–270. [Online]. Available: citeseer.nj.nec.com/estrin99next.html

[2] I.F.Akyildiz, W. Su, Y. Sankarasubramaniam, and E.Cayirci, "Wireless sensor networks: A survey," *Computer Networks*, Mar. 2002.

[3] A. A. F. Loureiro, J. M. S. Nogueira, L. B. Ruiz, and R. A. F. Mini, "Redes de sensores sem fio," in *Curso da XXI Jornada de Atualização em Informática - XXII Congresso da SBC, Florianópolis/SC.*, July 2002.

[4] L. B. Ruiz, L. H. Correia, L. F. M. Vieira, D. F. Macedo, E. Nakamura, C. M. Figueiredo, M. A. M. Vieira, E. H. Mechelane, D. Camara, A. A. F. Loureiro, J. M. Nogueira, and D. C. da Silva Jr., "Arquitetura para redes de sensores sem fio," *Minicurso, XXII Congresso da SBC*, May 2004.

[5] M. A. M. Vieira, D. C. da Silva Jr., and C. N. Coelho, "Survey on wireless sensor network devices," *IEEE International Conference on Emerging Technologies and Factory Automation (ETFA) 2003*, Sept. 2003.

[6] M. Horton, D. E. Culler, K. Pister, J. Hill, R. Szewczyk, and A. Woo, "Mica: the commercialization of microsensor motes." *Sensors Online*, Apr. 2002. [Online]. Available: http://www.sensorsmag.com/articles/0402/40

[7] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. E. Culler, and K. S. J. Pister, "System architecture directions for networked sensors," in *Architectural Support for Programming Languages and Operating Systems*, 2000, pp. 93–104. [Online]. Available: citeseer.nj.nec.com/382595.html

[8] D. Culler, D. Gay, P. Levis, R. von Behren, M. Welsh, and E. Brewer, "The nesc language: A holistic approach to networked embedded systems," in *Conference on Programming Language Design and Implementation of ACM SIGPLAN 2003*, 2003. [Online]. Available: http://www.cs.berkeley.edu/~pal/pubs/nesc.pdf

[9] H. Abrach, S. Bhatti, J. Carlson, H. Dai, J. Rose, A. Sheth, B. Shucker, and R. Han, "Mantis: System support for multimodal networks of in-situ sensors," *2nd ACM International Workshop on Wireless Sensor Networks and Applications (WSNA)*, 2003. [Online]. Available: http://mantis.cs.colorado.edu/

[10] L. van Hoesel, S. Dulman, P. Havinga, and H. Kip, "Design of a low-power testbed for wireless sensor networks and verification," University of Twente, Tech. Rep., Sept. 2003.

[11] J. Mulder, "Peeros preemptive eyes real time operating system," Master's thesis, University of Twente, Apr. 2003.

[12] M. A. M. Vieira, "Bean: A computer platform for wireless sensor network," Master's thesis, Universidade Federal de Minas Gerais, Belo Horizonte, Brasil, May 2004. [Online]. Available: http://www.dcc.ufmg.br/~mmvieira/publications/bean.pdf

[13] L. F. M. Vieira, "Yatos e wisdom: Plataforma de software para redes de sensores," Master's thesis, Universidade Federal de Minas Gerais, Belo Horizonte, Brazil, May 2004, (in Portuguese).

[14] P. Levis, N. Lee, M. Welsh, and D. Culler, "Tossim: accurate and scalable simulation of entire tinyos applications," in *Proceedings of the first international conference on Embedded networked sensor systems*. ACM Press, 2003, pp. 126–137.

[15] S. Tilak, N. Abu-Ghazaleh, and W. Heinzelman, "A taxonomy of wireless microsensor network models," *ACM Mobile Computing and Communications Review*, 2002. [Online]. Available: citeseer.nj.nec.com/tilak02taxonomy.html

[16] Trolltech, "Signals and slots," *Qt Reference Documentation*, 2003. [Online]. Available: http://doc.trolltech.com/3.2/signalsandslots.html

[17] E. Cheong, J. Liebman, J. Liu, and F. Zhao, "Tinygals: a programming model for event-driven embedded systems," in *Proceedings of the 2003 ACM symposium on Applied computing*. ACM Press, 2003, pp. 698–704.