APLICAÇÃO DE AGENTES MÓVEIS EM REDES DE SENSORES SEM FIO PARA LOCALIZAÇÃO E SEGUIMENTO DE OBJETOS ALVOS MÓVEIS

RODRIGO S. ALLGAYER, CARLOS EDUARDO PEREIRA

Departamento de Engenharia Elétrica, Universidade Federal do Rio Grande do Sul (UFRGS).
Av. Osvaldo Aranha, 103 – Porto Alegre/RS - Brasil
E-mails: allgayer@ece.ufrgs.br, cpereira@ece.ufrgs.br

LEONARDO STEINFELD

Instituto de Ingeniería Electrica, Universidad de la República
Julio Herrera y Reissig 565 - Montevideo - Uruguai
E-mails: leo@fing.edu.uy

LUIGI CARRO, FLÁVIO R. WAGNER

Instituto de Informática, Universidade Federal do Rio Grande do Sul (UFRGS)
Av. Bento Gonçalves, 9500 - Porto Alegre/RS – Brasil
E-mails: carro@inf.ufrgs.br, flavio@inf.ufrgs.br

Abstract – This paper presents the performance analysis of an embedded system for a location and tracking application wireless sensor networks (WSN). It is assumed that the sensor network nodes are static, while the developed software is implemented as mobile agents with the help of the JADE framework. The presented project follows a Model-Driven Development (MDD) methodology using UML (Unified Modeling Language) models. Metrics related to dynamic and static features of the implemented solution are extracted both from the UML models as well as from the generated Java code, allowing a design space exploration in terms of concepts such as performance, memory, energy consumption, etc.

Keywords— Location and tracking, mobile agents, wireless sensor networks.

Resumo— Este artigo apresenta a análise de desempenho de um sistema embarcado baseada em uma aplicação de localização e seguimento de objeto alvo utilizando redes de sensores sem fio (RSSF). Sendo os nós da rede fixos, realiza-se o seguimento do alvo utilizando agentes móveis com o auxílio do framework JADE. A modelagem da aplicação é realizada a partir de uma especificação baseada em modelos (MDE) utilizando a linguagem UML (Unified Modeling Language). Com base na modelagem da aplicação e no software desenvolvido, métricas estáticas e dinâmicas serão extraídas para análise e comparação, permitindo uma exploração do espaço de projeto em termos de desempenho, memória, consumo de energia, e etc.

Palavras-chave— Localização e seguimento, agentes móveis, rede de sensores sem fio.

1 Introdução

As RSSF apresentam-se como uma nova plataforma de computação que combina processamento, sensoriamento/detecção e comunicação dentro de um ambiente físico. O nó sensor, uma nova classe de sistema embarcado em rede, é caracterizado por severas restrições de recursos, especialmente de energia, porque são usualmente alimentados com baterias ou energia coletada do meio em que se encontram inseridos. Com o avanço da tecnologia, a capacidade dos circuitos integrados aumenta e novas aplicações podem ser concebidas, onde previamente seus custos em termos de preço e energia eram inaceitáveis (Steinfeld, 2009).

No entanto, esta crescente complexidade das aplicações que utilizam redes de sensores sem fio logo se torna uma barreira para a adoção dessas redes. Novos modelos de programação são essenciais para o desenvolvimento de aplicações distribuídas complexas, e ao mesmo tempo para obter um nível aceitável de eficiência energética. Os modelos de programação das RSSF atuais não escalam bem de modelos simples de aquisição de dados a modelos de processamento de informação colaborativos. Em um cenário diferente, aplicações distribuídas complexas têm sido desenvolvidas para plataformas (como PDA, computadores portáteis, etc), mas eles não são apropriados para as plataformas de recursos escassos, como os chamados motes de Berkeley (Hill, 2002). A recente disponibilização de máquinas virtuais pra sistemas embarcados de recursos escassos como os nós das RSSF, Darjeeling (Brouwers, 2008) ou Squawk (Simon, 2005) entre outras, possibilita o uso de linguagens com maiores níveis de abstração como Java, permitindo adotar paradigmas de processamento distribuído existentes, como agentes móveis. Ainda que previamente se tenha proposto soluções baseadas no uso de agentes móveis, estes não têm avaliado os custos tanto em termos de tamanho de memória de programa e de dados, tanto de energia, que são conceitos importantes no âmbito de sistemas embarcados.

O projeto e o desenvolvimento de sistemas embarcados apresentam graus de liberdade que devem ser explorados cautelosamente para permitirem o melhor desempenho para a aplicação. A extração de métricas no projeto da aplicação podem auxiliar na exploração destes graus de liberdade proporcionando uma melhor qualidade do projeto de aplicações que necessitam ser otimizadas devido a limitações de consumo de energia e de processamento. Estas características podem ser encontradas em aplicações utilizando RSSF.

Este artigo tem como objetivo a análise de desempenho de um sistema embarcado baseada em uma aplicação de localização e seguimento de objeto alvo utilizando redes de sensores sem fio, buscando realizar uma análise destas novas plataformas. Através da modelagem e implementação, métricas serão extraídas para possibilitarem a análise do desempenho obtido.

O texto encontra-se divido da seguinte forma: no Capítulo 2 será apresentada a metodologia de projeto proposto para este trabalho. Em seguida, no Capítulo 3, será descrito a aplicação abordada por este trabalho e suas características. No Capítulo 4, é apresentado a modelagem da aplicação e, em seguida, no Capítulo 5 são destacadas as métricas extraídas dos modelos desenvolvidos e da implementação realizada. Por fim, no Capítulo 6, são realizadas as conclusões e apresentados os trabalhos futuros.

2 Metodologia

A metodologia utilizada para o desenvolvimento e análise da aplicação parte da modelagem das características da aplicação conjuntamente com uma parte do framework JADE (Java Agent DEvelopment Framework) (Bellifemine, 2007) para definição dos requisitos da aplicação.

O framework JADE, completamente desenvolvido em Java, auxilia na implementação dos agentes. Este consiste em um middleware baseado nas especificaçoes Foundation for Intelligent Physical Agents (FIPA) (FIPA, 2010) e contém um conjunto de ferramentas de depuração e uma interface gráfica. As principais classes necessárias para o desenvolvedor da aplicação são: *Agents* e *Behaviour*. A classe *Agents* deve ser extendida para especializar seu uso na aplicação, agregando as características de um sistema multi-agentes. A classe *Behaviour* implementa as tarefas dos agentes através da definição de comportamentos.

A modelagem da aplicação foi realizada com o desenvolvimento de diagramas de casos de uso, diagramas de classes e diagramas de seqüência. Com base no modelo desenvolvido e nas características da aplicação, foi realizado o desenvolvimento do software da aplicação, em linguagem de programação Java, para posterior execução em uma plataforma PC. As métricas estáticas e dinâmicas do software da aplicação foram extraídas com o auxílio da ferramenta DESEJOS (Mattos, 2005). A partir da instrumentação do código e execução da aplicação, foi possível extrair métricas dinâmicas, utilizando dados de entradas gerados para uma avaliação controlada do software na plataforma alvo.

As ferramentas utilizadas foram:

- MagicDraw UML: criação dos modelos para auxílio no desenvolvimento da aplicação;
- NetBeans: desenvolvimento da aplicação em Java;
- DESEJOS: instrumentação e extração de métricas estáticas e dinâmicas do software da aplicação;
- Matlab: simulação do algoritmo e geração de dados de entrada sintéticos para utilizar na execução controlada do sistema.

3 Descrição da aplicação

A aplicação utiliza uma rede de sensores sem fio para realizar a localização e o seguimento de objetos alvos utilizando agentes móveis. Inicialmente, a aplicação foi baseada no experimento descrito em (Tseng, 2003), que utiliza o paradigma de agentes móveis para localização e seguimento. Todavia, diferentemente daquele trabalho, no qual os agentes de software eram estáticos, no presente trabalho os agentes podem migrar entre os nós sensores, permitindo a localização e seguimento de objetos alvos com a cooperação de nós locais, reduzindo a quantidade de dados trafegando entre os nós da rede.

A rede é composta por nós distribuídos em um ambiente com capacidade de realizar o sensoriamento, processamento e a comunicação com os nós vizinhos, como demonstra a Figura 1. O nó sensor possui um sensor com a capacidade de mensurar a distância em relação ao alvo, sendo possível a utilização de diversos tipos de sensores (som, luz e radiofreqüência), podendo ser ativos ou passivos dependendo das características do objeto alvo. Neste trabalho foi considerado que o objeto alvo emite sinais de radiofreqüência, denominados *beacons*, que serão detectados pelos receptores localizados em cada nó sensor da rede. Dependendo da potência do sinal de radiofreqüência recebido, calcula-se a distância entre o nó sensor e o alvo com base no modelo de propagação de ondas eletromagnéticas. Este prevê que a potência do sinal decai com o aumento da distância de separação entre transmissor e receptor.

Existem três tipos de nós na rede: nó sensor, nó coordenador e objeto alvo, sendo que cada nó possui uma função distinta para a aplicação. O objeto alvo é caracterizado pela emissão de *beacons* na rede. O nó sensor é responsável por realizar o sensoriamento dos sinais enviados pelo objeto alvo e encontra-se em maior número na rede. O nó coordenador possui

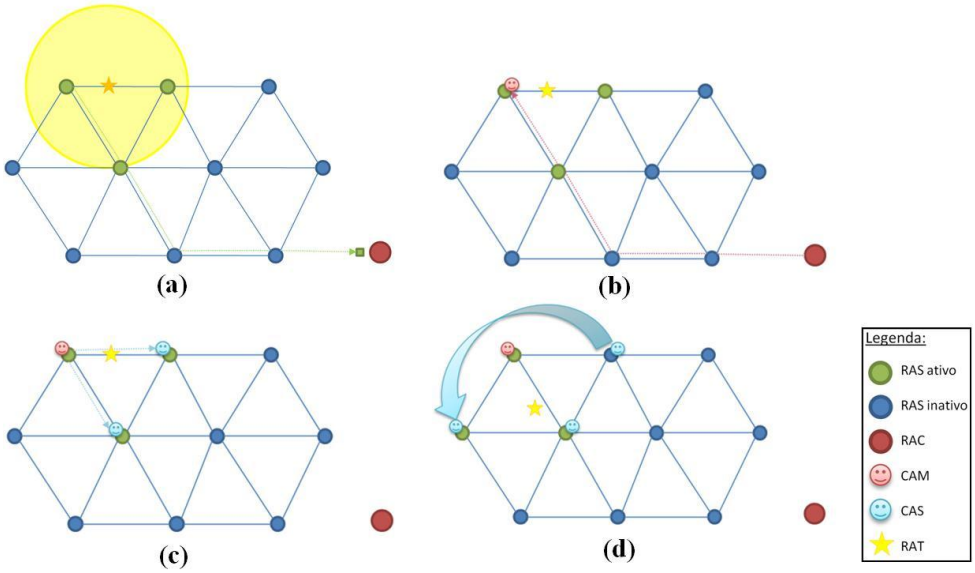

Figura 1. Movimentação dos agentes na rede de sensores para localização e seguimento do objeto alvo.

a função de gerenciar a entrada e saída de agentes na rede e centralizar um banco de dados da trajetória do objeto alvo. Os algoritmos de localização e o seguimento do objeto alvo na rede são realizados pelos agentes, sendo estes baseados no *Framework*.

Nesta aplicação, os agentes podem ser de dois tipos distintos: Agentes Residentes (*Resident Agent*, RA) e Agentes Colaboradores (*Colaborative Agent*, CA). Os agentes RA econtram-se fixos em um nó da rede, podendo ser um *RA_Coordinator* (RAC) quando encontra-se no nó coordenador, ou *RA_SensorNode* (RAS) quando encontra-se em um nó sensor. Eles comunicam-se com os agentes colaboradores quando estes encontram-se no mesmo nó. Os Agentes Colaboradores possuem a capacidade de movimentarem-se entre os nós da rede, sendo responsáveis por realizar o seguimento e o cálculo da posição dos objetos alvos. Eles podem ser do tipo CA_*Master* (CAM), quando possuem a função de coordenar o cluster de localização solicitando dados de outros agentes e verificando quando um agente deve migrar para outro nó, ou *CA_Slave* (CAS) quando possuem a função de colaborar com o agente CAM com dados de distância.

A inserção de agentes na rede é realizada pelo nó coordenador. Quando um nó sensor detecta um *beacon* de um objeto alvo pela primeira vez, este informa o nó coordenador (Figura 1(a)). O nó coordenador aguarda receber a notificação de pelo menos três nós para injetar o CAM na rede. O nó sensor que receberá o CAM será decidido através da menor distância recebida entre o nó sensor e o objeto alvo (Figura 1(b)). O CAM, ao ser inicializado no nó sensor, informa o agente residente que está presente no mesmo nó e disposto a cooperar. O agente residente então passa a enviar as informações de distância ao agente colaborador. Este realiza duas clonagens, gerando os CAS, e os envia aos nós vizinhos (Figura 1(c)). Os CAS são inicializados nos nós sensores e informam ao agente residente, que iniciam o envio das distâncias mensuradas. Em seguida, os CAS enviam as distâncias ao CAM.

Com o objetivo de delimitar o escopo da aplicação, algumas restrições e características foram definidas para a aplicação.

- Apenas um objeto alvo a ser detectado está presente dentro da rede. Essa restrição evita a identificação das trajetórias dos objetos alvos caso elas se cruzem.
- A rede de nós sensores e o objeto alvo encontram-se em uma superfície plana (2-D). Assim, é possível determinar-se a posição do objeto alvo a partir da distância mensurada por três nós sensores. Para o caso de uma superfície 3-D, não abordada neste trabalho, seriam necessários no mínimo quatro valores de distância mensurados pelos nós sensores.
- A relação das distâncias entre nós e o alcance da conexão de radiofreqüência assegura comunicação direta entre pelo menos três nós vizinhos.
- O protocolo de roteamento permite enviar mensagens para o coordenador, sendo sua posição conhecida pelos nós da rede.

3.1 Cálculo da posição do objeto alvo

O cálculo da posição do objeto alvo é realizado através da triangulação entre três nós sensores vizinhos da rede. Por construção, a rede possui uma forma regular, onde os nós são dispostos de uma forma triangular eqüidistantes um do outro. A fórmula para determinar a posição do objeto alvo (x_0,y_0) está representada pela Equação 1, onde é necessário a posição dos três nós sensores (x_i,y_i) e das distâncias destes nós ao objeto alvo (r_i) para $i=1,...,3$.

A execução do algoritmo é responsabilidade do agente CAM. Este possui o conhecimento da posição dos dois nós vizinhos e obtém a distância mensurada pelos agentes CAS em relação ao objeto alvo.

$$p_0 = \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} = A^{-1} \cdot b \qquad (1)$$

Onde,

$$A = 2 \cdot \begin{bmatrix} x_3 - x_1 & y_3 - y_1 \\ x_3 - x_2 & y_3 - y_2 \end{bmatrix} \qquad (2)$$

e

$$b = \begin{bmatrix} (r_1^2 - r_3^2) \\ (r_2^2 - r_3^2) \end{bmatrix} - \begin{bmatrix} (x_1^2 - x_3^2) + (y_1^2 - y_3^2) \\ (x_2^2 - x_3^2) + (y_2^2 - y_3^2) \end{bmatrix} \qquad (3)$$

O cálculo da posição pode impactar no desempenho dos nós sensores que possuem uma capacidade limitada de processamento e de energia, como é o caso de redes de sensores tradicionais. Assim, otimizações podem ser realizadas no algoritmo com relação ao cálculo da posição do objeto alvo, dada pelas Equações de 1 a 3. A matriz A, da Equação 2, permanecerá constante até que um dos agentes colaboradores tenha que realizar uma migração, sendo que o resultado do cálculo da operação inversa da matriz *A* pode ser armazenado até que uma migração ocorra. Com este artifício, será reduzido o número de operações realizadas pelo algoritmo. O mesmo pode ser aplicado para uma parte do vetor *b*, representado pela Equação 3.

3.2 Cálculo da movimentação ("migração") dos agentes

Os agentes colaboradores possuem a característica de mobilidade entre os nós da rede para realizar o seguimento do objeto alvo, minimizando a quantidade de mensagens trocadas entre os nós da rede, sendo um dos objetivos do processamento distribuído ou processamento de informação colaborativa. Esta funcionalidade exige que os agentes conheçam a distribuição da rede e tenham conhecimento para qual dos nós eles devem migrar, continuando próximos do objeto alvo. Na Figura 1(d) está representada a migração dos agentes entre os nós da rede.

Devido à disposição da rede ser em forma triangular com lados eqüiláteros, os agentes podem realizar o cálculo da posição do próximo nó para onde devem migrar utilizando a Equação 4.

$$\begin{aligned} x_4 &= x_1 + x_3 - x_2 \\ y_4 &= y_1 + y_3 - y_2 \end{aligned} \qquad (4)$$

Este cálculo é realizado pelo agente CAM que verifica a distância recebida pelos agentes CAS e a sua própria distância, comparando com um valor limite pré-estabelecido. Na Figura 2 pode-se verificar o objeto alvo atravessando a distância limite determinada para o nó sensor 1 participar do cálculo de localização do alvo. Após o alvo ultrapassar este limite, o agente realizará uma migração para o nó sensor 4.

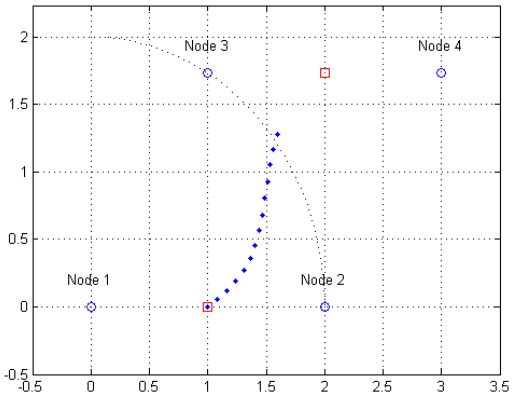

Figura 2. Representação da região limite de sensoriamento do nó sensor 1 sendo ultrapassada pelo objeto alvo.

4 Modelagem e implementação

O desenvolvimento do software da aplicação foi realizado com base nos modelos desenvolvidos da aplicação. Esta prática de desenvolvimento de software baseada na criação e especificação de modelos para o detalhamento da aplicação dá-se o nome de *Model Driven Engineering* (MDE) (France, 2007). Esta abordagem busca a utilização de especificações (denominadas modelos) de alto nível utilizando linguagens de domínio para a descrição e implementação do sistema. Os principais modelos necessários para a descrição de uma aplicação são: diagrama de casos de uso, diagrama de classes e diagrama de interação.

Como a natureza da aplicação apresentada neste trabalho é distribuída, mais um grau de dificuldade foi introduzido para a descrição das diferentes aplicações que serão executadas independentemente nos nós da rede e nos agentes. Outra dificuldade apresentada foi a descrição da interação (comunicação) dos nós da rede, podendo ser de duas maneiras: através de um objeto "rede" ou acessando métodos dos objetos com o qual deseja-se interagir. Para solucionar a natureza distribuída do problema, decidimos modelar a aplicação como um todo, especificando a visão das partes que compõem a rede nos diagramas de caso de uso.

A seguir, serão detalhados os diagramas desenvolvidos para descrever a aplicação apresentada neste trabalho.

4.1 Diagrama de Casos de Uso

O diagrama de caso de uso descreve um cenário que mostra as funcionalidades do sistema do ponto de vista do usuário, apresentando as funcionalidades e os atores envolvidos (OMG, 2010). Neste caso, fo-

ram desenvolvidos diagramas de casos de uso representado os diferentes sistemas envolvidos nesta aplicação distribuída, como podem ser visualizados na Figura 3. A Figura 3(a) ilustra a visão do sistema do ponto de vista da rede, onde os atores são o usuário realizando requisições a rede e o objeto alvo enviando *beacon*s a serem identificados. Esta visão apresenta o sistema como um todo.

Na Figura 3(b) está representada a visão do nó sensor que possui a funcionalidade de detectar o objeto alvo. Em seguida, na Figura 3(c) está representada a visão do coordenador da rede, onde os atores são os nós sensores que identificaram os *beacons* da rede e aguardam o recebimento dos agentes para realizarem a localização e seguimento do objeto alvo. Na Figura 3(d) está representado o diagrama de caso de uso com a visão do agente CAM da rede, onde os atores são incialmente o agente RAS no nó sensor para onde o agente CAM migrou. Posteriormente, com a criação dos agentes CAS, o agente CAM passa a comunicar-se com os agentes CAS e envia os dados de posição do objeto alvo ao coordenador da rede.

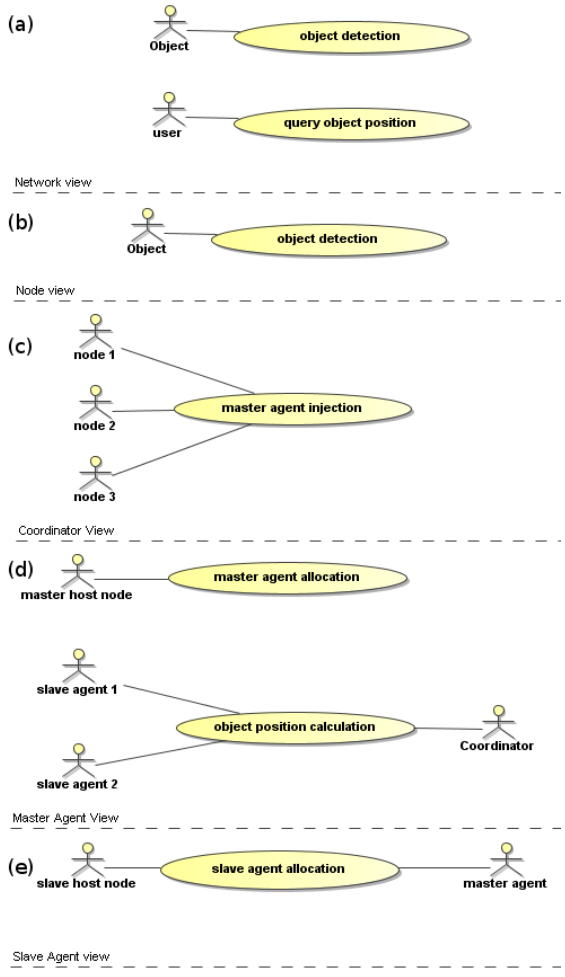

Figura 3. Diagrama de casos de uso da aplicação.

Por fim, a Figura 3(e) apresenta a visão do agente escravo, onde os atores são o agente RAS ao nó que ele encontra-se e, também, o agente CAM ao qual necessita enviar os dados de distância coletados.

4.2 Diagrama de Classes

O diagrama de classes é um dos principais diagramas estruturais da linguagem UML, onde se ilustra classes, interfaces e o relacionamento entre elas (OMG, 2010). Inicialmente, foi criado um modelo conceitual do domínio do problema para melhor visualização, sendo posteriormente desenvolvido o diagrama de classes da aplicação, como pode ser visualizado na Figura 4. O modelo apenas inclui as classes da aplicação e as classes essenciais do framework JADE, não sendo modelado o *framework* inteiro.

No diagrama de classes é possível visualizar a relação entre os diferentes tipos de agentes e seus comportamentos herdados. O código da aplicação desenvolvida estende a classe Agent, sobrescrevendo os métodos *setup()* e *takeDown()*. No método *setup()* está o código que descreve o agente, sendo necessário descrever os seus comportamentos. Os comportamentos descrevem as funções do agente podendo ser de dois tipos: *SimpleBehaviour* ou *CompositeBehaviour*. Neste trabalho foram somente utilizados os comportamentos *SimpleBehaviours*, sendo eles: *CyclicBehaviour* e *TickerBehaviour*. O *CyclicBehaviour* descreve um comportamento cíclico para o agente e, o comportamento *TickerBehaviour*, descreve um comportamento periódico ao agente. O agente pode conter mais de um comportamento, sendo cada um deles representado por uma *thread* diferente.

A comunicação entre os dispositivos foi modelada através de métodos que seriam acessados por outras classes. O objeto RATarget foi modelado juntamente com a aplicação para podermos simular o sistema.

4.3 Diagrama de Interação

O diagrama de interação representa a interação entre os diferentes objetos da aplicação, onde é possível visualizar os métodos e as mensagens de comunicação (OMG, 2010).

A operação do sistema derivada do caso de uso *object detection*, apresentado na Figura 3(b), está representado na Figura 5. Nesta figura encontra-se o primeiro diagrama de seqüência representando a indicação do objeto alvo por parte dos nós sensores a partir da emissão de um *beacon* por parte da instância da classe de objeto alvo (RATarget) que é detectado pelos nós sensores (RASensorNodes). Ao receber um sinal *beacon*, os nós sensores enviam uma mensagem de "*firstBeacon"* ao nó coordenador. Quando o nó coordenador recebe a terceira mensagem de "*firstBeacon"*, ele envia o agente CAM para o nó sensor que detectou o alvo primeiramente. Este inicializa o objeto CAM que, posteriormente, cria os dois agentes CAS e os enviam aos nós sensores vizinhos.

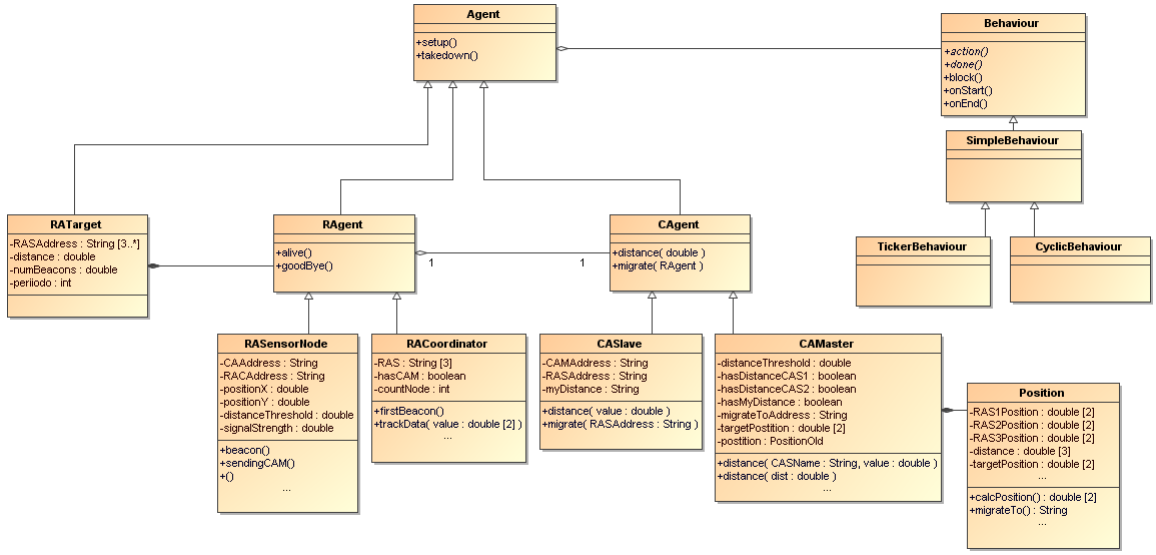

Figura 4. Diagrama de classes da aplicação.

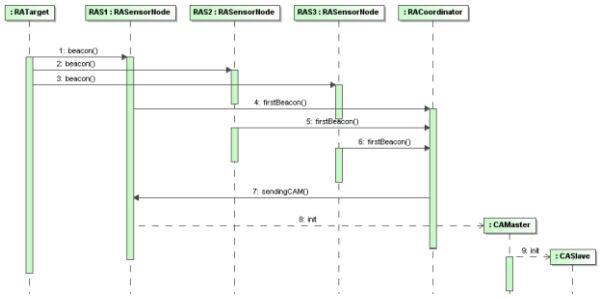

Figura 5. Diagrama de interação demonstrando a percepção do primeiro *beacon*.

O diagrama representado pela Figura 6 apresenta a interação entre os objetos CAM e CAS para a realização do cálculo do posicionamento do objeto alvo com base nas distâncias de cada nó ao alvo. Em seguida, o valor é enviado ao nó coordenador.

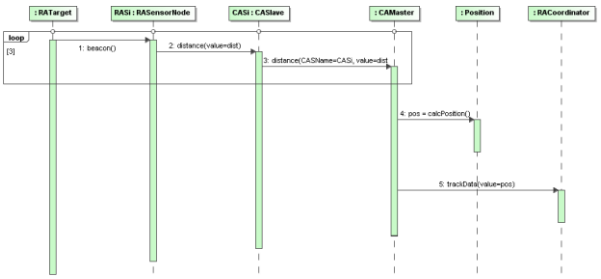

Figura 6. Diagrama de interação demonstrando a interação entre os agentes CAM e CAS.

Finalmente, o diagrama representado pela Figura 7 demonstra a operação de migração de um dos agentes dentro da rede. O CAM da rede compara os valores de distância recebida dos CAS e verifica qual dos nós deve migrar. Posteriormente envia uma mensagem de "*migração*" ao agente correspondente. Este agente informa ao agente residente que irá migrar, com o envio de uma mensagem "*goodBye*", e, após realizar a migração, informa ao novo agente residente que irá residir no mesmo nó com uma mensagem "*alive*".

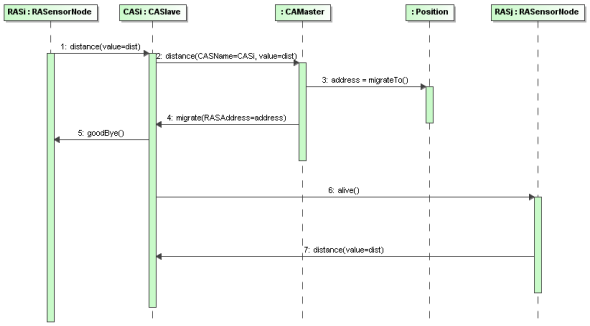

Figura 7. Diagrama de interação demonstrando a *interação entre os agentes CAM e CAS na migração*.

5 Análise das métricas extraídas da implementação

A ferramenta DESEJOS permite extrair tanto métricas estáticas quanto dinâmicas. As métricas dinâmicas são extraídas através da instrumentação dos bytecodes de Java para obter um novo arquivo *.class* com bytecodes anotados em pontos estratégicos (*checkpoints*). Depois, quando a aplicação é executada, diferentes contagens são realizadas para a obtenção de medidas das características físicas (se supõe a execução em um processador FemtoJava Multiciclo e Pipeline (Ito, 2001)): ciclos, energia (mJ) e memória de dados dinâmica. As medidas de memória de dados estática e a memória de programa são extraídas a partir da análise dos bytecodes.

Inicialmente se avaliou o peso relativo em termos estáticos e dinâmicos da aplicação em comparação com o *framework* JADE. Na Tabela 1 pode-se observar que o peso da aplicação desenvolvida não é significativo frente ao total (aplicação e o *framework*), sendo de aproximadamente 1% para as métricas de número de pacotes e de classes, número total de ciclos e instruções executados. Com base no número de vezes que cada instrução é executada e no

consumo para execução de cada instrução é possível calcular a energia consumida pelo algoritmo. Pode se verificar que o desenvolvimento da aplicação fica muito simplificado quando utilizamos um *framework* com diferentes bibliotecas, porém, estas bibliotecas devem estar muito bem desenvolvidas para não impactarem no desempenho da aplicação de forma substancial.

Tabela 1: Peso relativo da aplicação.

	Total	Aplicação	Percentual
Packages	56	1	1,79%
Classes	1460	14	0,96%
Métodos	3980	40	1,01%
Ciclos	2,84E+06	105344	3,71%
Instruções	3,78E+08	1839150	0,49%

Na Tabela 2 estão representados os custos em ciclos e energia dos diferentes métodos da aplicação, sendo que foram escolhidos os primeiros 15 métodos mais relevantes (do total de 40 métodos) em ordem decrescente. Primeiro é observado que existe uma alta correlação entre os ciclos e a energia consumida pelos métodos. Na última coluna mostra-se a potência média para cada método, calculada como a energia dividida pelo número de ciclos. No caso de o método ser chamado somente uma vez, a potência é a potência média consumida pelo método.

Uma potência com o valor maior significa que executa *bytecodes* mais custosos do ponto de vista do consumo.

Na Tabela 3 é possível observar o tamanho de memória de programa utilizado na descrição de cada agente. Uma análise destes números se faz muito importante para o desempenho da aplicação, visto que um sistema multi-agentes utiliza a migração de agentes móveis através da serialização de objetos. Neste caso, deseja-se que os agentes tenham o menor tamanho possível para não impactar na comunicação. No caso de uma rede de sensores sem fio, os módulos de comunicação sem fio consomem muita energia para realizar a transmissão de dados, assim, quanto menor a quantidade de dados a serem transmitidas através dos módulos, mais eficiente se torna o nó sensor e mais vida útil é alcançada.

Tabela 3: Memória de programa dos Agentes.

Classes	Tamanho (bytes)
CAMaster	529
CAMaster$1	846
CAMaster$2	26
Position	1231
CASlave	183
CASlave$1	326
TOTAL	3141

6 Conclusão e trabalhos futuros

O trabalho apresentou a análise de uma aplicação de localização e seguimento de objetos alvos baseada em rede de sensores sem fio utilizando agentes móveis. A análise foi realizada a partir da elaboração de modelos e da implementação da aplicação. As métricas foram extraídas com auxílio da ferramenta DESEJOS para posterior análise e verificação do desempenho obtido.

A aplicação, por apresentar uma natureza distribuída, representou algumas dificuldades iniciais para ser modelada. Inicialmente, pensou-se em uma modelagem dos sistemas sob uma visão particular de cada nó sensor. Porém, não foi possível realizar esta abordagem, pois as métricas para análise deveriam ser extraídas da aplicação inteira. Assim, buscou-se uma modelagem sob a visão da rede, abrangendo todos os nós que compõem a aplicação, o que simplificou a modelagem da comunicação e interação entre os objetos da aplicação.

Tabela 2: Tabela comparativa de ciclos vs. energia por métodos da aplicação

Método	Ciclos	%	Energia	%	Potência
Position.calcPosition	39480	37%	692393	38%	17,5
CAMaster$1.action	25513	24%	428583	23%	16,8
RASensorNode$1.action	17013	16%	306280	17%	18,0
RACoordinator$1.action	6984	7%	84364,3	5%	12,1
CASlave$1.action	5187	5%	141904	8%	27,4
CAMaster$2.action	2451	2%	57846,4	3%	23,6
Position.<init>	1388	1%	13450,3	1%	9,7
Position.migrateTo	1309	1%	23909,1	1%	18,3
RASensorNode.setup	1078	1%	23605,1	1%	21,9
RACoordinator.setup	958	1%	12225,8	1%	12,8
RASensorNode.<init>	497	0%	10588	1%	21,3
CAMaster.<init>	283	0%	4261,04	0%	15,1
RASensorNode$1.<init>	279	0%	4007,46	0%	14,4
CASlave.<init>	58	0%	3932,86	0%	67,8
CAMaster$2.<init>	42	0%	3951,23	0%	94,1

Outra dificuldade na modelagem de sistemas distribuídos foi a comunicação entre os diversos sistemas que compõem a aplicação de rede de sensores sem fio. Por fim, buscou-se modelar a comunicação através do acesso a métodos das classes que representavam os sistemas distintos.

A extração de métricas da implementação da aplicação foi útil para realizar uma aproximação do consumo e do peso da aplicação em comparação ao framework JADE escolhido. Com as métricas, foi possível dimensionar o tamanho de memória do código de cada agente. Esta análise possibilita estimar o consumo de energia que a serialização, utilizando a comunicação sem fio destes agentes, implicaria no consumo da aplicação a cada migração destes agentes. Novas técnicas de migração podem ser estudadas para minimizar este impacto.

Como trabalho futuro, espera-se reduzir o número de restrições estabelecidas nesta abordagem inicial, buscando uma aplicação mais concreta. Outra abordagem será a extração de métricas com base na visão da aplicação que será executada em cada nó individualmente, estimando-se o consumo de energia, de memória e o processamento necessário para cada nó da rede.

Agradecimentos

Agradecimentos ao CNPq e a CAPES pelo apoio e incentivo no desenvolvimento das atividades.

Referências Bibliográficas

Bellifemine, F. L. and Caire, G. and Greenwood, D. (2007). Developing Multi-Agent Systems with JADE (Wiley Series in Agent Technology). Wiley, April 2007.

Brouwers, N. and Corke, P. and Langendoen, K. (2008). Darjeeling, a java compatible virtual machine for microcontrollers. In Proceedings of the ACM/IFIP/USENIX Middleware '08 Conference Companion. New York, NY, USA: ACM, pp. 18-23.

France R. and Rumpe, B. (2007). Model-driven Development of Complex Software: A Research Roadmap. In Proceedings of Future of Software Engineering 2007. IEEE Computer Society: Washington, DC, USA, 37-54.

Hill, J. L. and Culler, D. E. (2002). Mica: A Wireless Platform for Deeply Embedded Networks. IEEE Micro, pp. 12-24, November/December.

IEEE Foundation for Intelligent Physical Agents (2010). http://www.fipa.org/. Acessada em: 11 de março de 2010.

Ito, S. A. and Carro, L. and Jacobi, R. P. (2001). Making java work for microcontroller applications. IEEE Design and Test of Computers, 18(5):100–110.

JADE WebSite (2010). Página da internet: http://jade.tilab.com/. Acessada em: 11 de março de 2010.

Mattos, J. C. and Specht, E. and Neves, B. and Carro, L. (2005). Making object oriented efficient for embedded system applications. In Proceedings of the 18th Annual Symposium on integrated Circuits and System Design, Florianopolis, Brazil. ACM, New York, NY, 104-109.

Object Management Group, Unified Modeling Language (UML) (2010). http://www.uml.org. Acessed 11 March 2010.

Simon, D. and Cifuentes, C. (2005). The squawk virtual machine: Java on the bare metal. In Proceedings of Conference on Object-oriented programming, systems, languages, and applications. New York, NY, USA: ACM Press, pp. 150-151.

Steinfeld, L. and Carro, L. (2009). The Case for Interpreted Languages in Wireless Sensor Networks. In Proceedings of International Embedded Systems Symposium. Langenargen, Germany, 279—289.

Tseng, Y. C. and Kuo, S. P. and Lee, H. W. and Huang, C. F. (2003). Location tracking in a wireless sensor network by mobile agents and its data fusion strategies. Information Processing in Sensor Networks Book, Springer:Berlin, p. 554.