Análise de Desempenho da Utilização do Framework AFME em uma Aplicação de Seguimento de Trajetória para Rede de Sensores sem Fio utilizando Agentes Móveis

EDISON PIGNATON DE FREITAS, BERNHARD BÖSCH, TONY LARSSON

School of Information Science, Computer and Electrical Engineering
Halmstad University – Halmstad – Suécia
E-mails: {edison.pignaton, tony.larsson}@hh.se, berbos09@student.hh.se

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— Advances on wireless communication and sensor systems enabled the growing usage of Wireless Sensor Networks. This kind of network is being used to support a number of new emerging applications, thus the importance in studying the efficiency of new approaches to program them. This paper proposes a performance study of an application using high-level mobile agent model for Wireless Sensor Networks. The analysis is based on a mobile object tracking system, a classical WSN application. It is assumed that the sensor nodes are static, while the developed software is implemented as mobile agents by using the AFME framework. The presented project follows a Model-Driven Development (MDD) methodology using UML (Unified Modeling Language) models. Selected metric of the implemented solution, i.e. CPU utilization, memory and energy usage, are extracted from the deployed application, allowing a design space exploration in terms of performance, memory and energy consumption.

Keywords— Wireless Sensor Networks; Multi-agents; Overhead; Energy Consumption.

Resumo— Avanços na área de comunicação sem fio e rede de sensores permitiram um crescimento do uso de Rede de Sensores sem Fio. Este tipo de rede está sendo utilizada em diversas aplicações emergentes, o que justifica a importância do estudo de novos métodos de programação visando explorar o desempenho destas redes. Este artigo apresenta um estudo de desempenho de uma aplicação utilizando uma abordagem alto-nível e agentes móveis. A análise é baseada em um sistema de seguimento de objetos alvos utilizando rede de sensores sem fio. Nesta aplicação, os nós-sensores são estáticos enquanto que a aplicação utiliza o framework AFME para programação dos agentes. O projeto apresentado segue a metodologia de desenvolvimento baseado em modelos (Model-Driven Development - MDD) utilizando modelos UML (Unified Modeling Language). Métricas relacionadas à utilização do processador, bem como de memória e energia são extraídas do sistema implementado, permitindo a exploração do espaço de projeto em termos do uso de memória, desempenho no processamento e consumo de energia dos nós da rede.

Palavras-chave— Rede de Sensores sem Fio; Muti-agentes; Overhead; Consumo de energia.

1 Introdução

Rede de sensores sem fio (RSSF) são sistemas embarcados usados para implementar um grande número de aplicações em diferentes áreas, incluindo aquelas em que as soluções com fios não são adequadas. Seu uso é potencializado quando estas redes são utilizadas de forma que os nós sensores possam coletar informações de forma colaborativa. Alguns exemplos da utilização destas redes são monitoramento de animais selvagens, recursos hídricos, perímetro de segurança ou coleta de dados para prevenção de desastres [1].

Apesar do enorme potencial de uso destas redes, a implementação real de sistemas baseados em RSSF apresentam grandes desafios, particularmente em relação ao uso dos recursos energéticos dos nós sensores. Normalmente, nós sensores têm como fonte de energia as baterias, que são recursos de energia limitada. Mesmo no melhor dos casos em

que os nós sensores são capazes de coletar a energia do meio em que se encontram, como por exemplo sensores piezoelétricos e fotovoltaicos, a energia disponível é ainda uma preocupação. Além disso, a capacidade de processamento dos nós sensores é limitada, apresentando um desafio para a execução de aplicações complexas em um único nó. Com base neste aspecto e na própria natureza distribuída das aplicações de rede de sensores, artifícios de programação distribuída são altamente recomendados e utilizados [2].

Como consequência dessa crescente complexidade das aplicações de RSSF, o desenvolvimento de novos modelos de programação é um passo essencial no sentido de soluções que minimizem essa complexidade, porém mantendo a eficiência energética. Apesar de existirem modelos utilizados para sistemas distribuídos que poderiam resolver os problemas relacionados com a natureza distribuída das aplicações de RSSF, eles não abordam as necessidades relacionadas com a energia e os recursos limitados de processamento apresentados por nós das RSSF. Assim, esta pesquisa visa o desenvolvimento da adaptação desses modelos à realidade das RSSF, sendo a modelagem da aplicação baseada em agentes de software móveis [4]. Pela própria natureza distribuída dos agentes móveis, eles encaixam-se perfeitamente aos requisitos distribuídos de aplicações de RSSF. Um exemplo deste tipo de uso é apresentado em [5], em que os bombeiros utilizam RSSF com agentes móveis para monitorar a progressão do fogo. No entanto, uma desvantagem de abordagens como [5] é a natureza *ad-hoc* da solução e do paradigma de programação de baixo nível usada para implementá-lo, o que torna difícil evoluir ou adaptar esses sistemas. Assim, há uma necessidade de linguagens de alto nível de programação e de metodologias que permitam a adoção de soluções flexíveis de agentes móveis. Esta alternativa está se tornando possível com o surgimento de máquinas virtuais para sistemas embarcados de recursos escassos, como por exemplo Squawk [6], permitindo o uso de linguagens de programação com níveis mais altos de abstração como a linguagem Java. Esses avanços combinados com o uso de métodos para abstração de hardware, como o desenvolvimento baseado em modelos (Model-Driven Development - MDD) [7], permitem que os projetistas de sistemas considerem diferentes alternativas para a implementação em fases iniciais do projeto do software. Além disso, esses métodos permitem a rastreabilidade de mudanças em toda a concepção e execução, o que proporciona uma suave transição entre diferentes versões de software, quando mudanças são necessárias. Assim, uma análise sobre estas questões é necessária para indicar como a adoção destes métodos de abstração de alto nível podem ser melhorados no domínio das RSSF.

Este artigo visa analisar o desempenho de um sistema de RSSF com agentes de software para utilizando uma metodologia MDD. Para realizar esta análise, uma aplicação de seguimento de objetos alvos utilizando RSSF foi escolhida [8]. O objetivo é avaliar o desempenho da utilização de agentes móveis em RSSF com base em métricas extraídas da aplicação implementada que medem o desempenho do processamento, consumo de energia e espaço de memória utilizado. O resultado destas métricas fornece informações valiosas para a exploração do espaço de projeto, auxiliando os desenvolvedores de sistemas em suas decisões e escolhas durante a concepção e implementação do mesmo. A implementação foi realizada utilizando o framework AFME [9], que fornece o suporte para os agentes. Este framework é desenvolvido em linguagem Java e possui suporte para execução nos nós sensores SunSPOT [10].

O artigo está organizado da seguinte forma: a Seção 2 descreve a metodologia utilizada para realizar esse estudo. Na Seção 3, a aplicação utilizando agentes móveis é apresentada. A Seção 4 descreve a modelagem da aplicação. As modalidades de execução das simulações realizadas com os respectivos resultados obtidos são apresentados na Secção 5. Finalmente, a Seção 6 apresenta as conclusões e trabalhos futuros.

2 Metodologia

O roteiro para o estudo proposto é composto por três etapas principais: 1) modelagem da aplicação, 2) implementação do aplicativo e 3) métricas de avaliação.

Na primeira etapa, o aplicativo de seguimento de objetos alvos é modelado utilizando diagramas de caso de uso UML para estudar os requisitos do sistema e identificar as suas funcionalidades. Assim, com base nas informações obtidas a partir desta primeira análise, o sistema foi modelado com base nas características fornecidas pelo framework AFME [9]. O AFME foi desenvolvido para permitir a criação de agentes para execução em dispositivos móveis e de recursos limitados. Ele utiliza a especificação FIPA que permite a interoperabilidade com outros ambientes.

Baseado no modelo desenvolvido, a segunda etapa apresenta o desenvolvimento da utilizando linguagem de programação Java e usando a estrutura AFME para execução nos nós SunSPOTs.

A terceira etapa foi realizada através da aquisição das métricas dinâmicas da aplicação desenvolvida através da instrumentação do código para obter medidas de desempenho durante a execução do sistema na plataforma SunSPOT. Finalmente, os dados adquiridos são apresentados e discutidos.

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 [3]. Esta abordagem

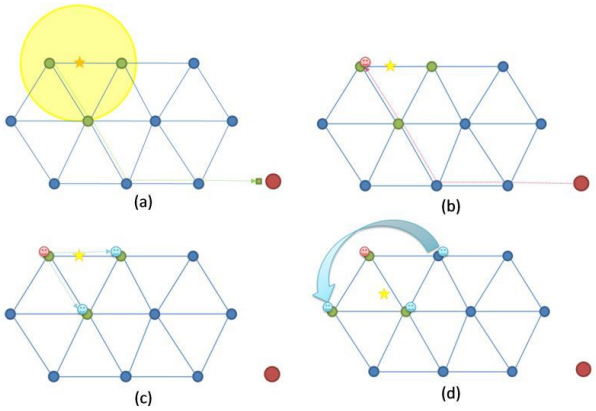


Fig.1. Movimentação dos agentes na rede e o seguimento do objeto alvo.

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 utilizando radiofrequência. 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 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.

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 equidistantes um do outro. 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.

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. 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.

4 Modelo e implementação da aplicação

O software foi desenvolvido baseado em modelos que descrevem a aplicação. Primeiramente, considerando-se a aplicação escolhida como estudo de caso deste trabalho como um sistema colaborativo, no qual partes distintas são executadas de maneira independente em diversos nós da rede, cada uma destas partes deve ser modelada. Um outro ponto importante se refere a questão da interação entre nós da rede através da comunicação de mensagens, a qual pode ser modelada de duas maneiras: a primeira seria na qual a comunicação se

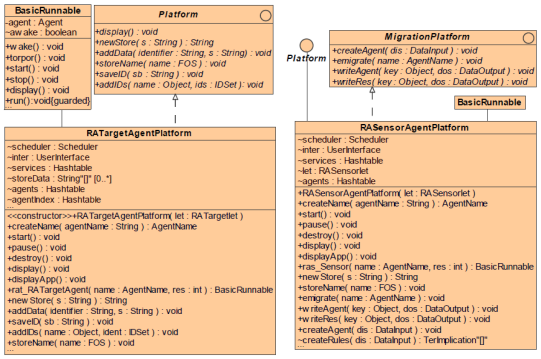
dá através de um objeto representando a rede, através do qual todas as mensagens na rede são enviadas; ou modelando o canal de comunicação entre objetos como associações entre objetos que precisam trocar mensagens. Desta forma os objetos podem chamar os métodos apropriados de cada um. Esta segunda opção traduz melhor a natureza distribuída de aplicações de RSSF, como é o caso do seguimento de objetos alvos estudado neste trabalho. Além disto, esta alternativa é mais simples de ser implementada do que a primeira, devido ao fato de evitar um elemento intermediário no sistema. Por estas razões a segunda opção foi a escolhida para ser implementadas.

O diagrama de classes é um dos mais importantes diagramas estruturais da UML [11]. Neste diagrama, classes, interfaces, e relacionamentos entre estes elementos são representados. O modelo desenvolvido foi divido em três diagramas de classes, um para cada tipo de nó, como por exemplo o nó sensor, nó coordenador e o objeto alvo. Além destes, outros diagramas de classes foram criados para os agentes CA. O modelo inclui classes pertencentes a aplicação além de classes essenciais do framework AFME. Note que o framework tem muitas outras classes que não são apresentadas neste artigo porque ou elas não foram usadas neste projeto ou porque elas não são essenciais a compreensão da estrutura do software. Em seguida são apresentadas partes dos diagramas de classes desenvolvidos, as quais foram escolhidas para mostrar as interações mais importantes entre classes do framework e da aplicação. A Figura 2 mostra um diagrama para as classes que dão suporte a execução dos agentes nos três tipos de nós, i.e. objeto alvo (Figura 2(a)), nó sensor e coordenador (Figura 2(b)).

O diagrama de classes da Figura 2(a) mostra a classe `RATargetAgentPlatform`, a qual implementa a interface `Platform` da API AFME. Esta interface provê as funcionalidades básicas para o agente que está hospedado em um nó. A classe contem uma instância da classe `BasicRunnable`, a qual provê as funcionalidades básicas para execução de um agente, atualizando suas crenças e prosseguindo seu processo de controle. Esta instância é que representa o agente `RATarget` no objeto alvo. A Figura 2(b) apresenta um diagrama de classes similar para os nós sensor e coordenador. No caso destes nós, a classe `RASensorAgentPlatform` além de implementar a interface `Platform`, deve implementar também a interface `MigrationPlatform`. Esta classe irá instanciar um objeto `RASensorNode` para o nó sensor e um objeto `RACoordinator` para o nó coordenador a partir da classe `BasicRunnable`. A interface `MigrationPlatform` provê as funcionalidades necessárias para os dois tipos de nós serem capazes de enviar e receber agentes móveis que compõe a aplicação, por exemplo os CA_Master e CA_Slave.

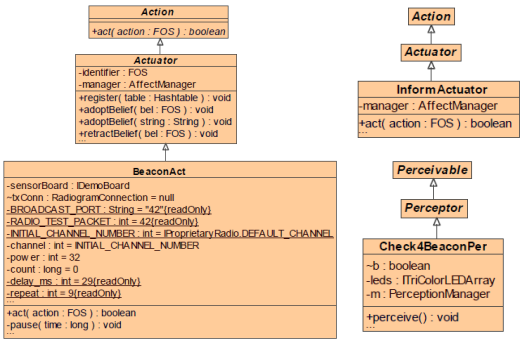
As classes apresentadas até agora apenas proveem suporte para a execução dos agentes nos nós. Utilizando-se AFME, a semântica dos agentes é realmente expressa em classes representando as percepções dos agente, os Perceptors, e as classes que implementam as suas ações, os Actuators. A Figura 3(a) apresenta as classes que modelam o atuador do RA_Target, enquanto a Figura 3(b) apresenta o actuator e o perceptor do RA_Sensor.

Note que o RA_Target não apresenta perceptors e apenas um actuator, o `BeaconAct` na Figura 3(a), que é responsável pela ação de enviar os beacons que serão percebidos pelos RASensorNode, através da funcionalidade implementada na classe `Check4BeaconPer` apresentada na Figura 3(b). Esta percepção será armazenada nas crenças do agente e será informada a outros agentes residentes no nó através da funcionalidade implementada na classe `InformActuator` (Figura3(b)).



(a) (b)

Fig. 2. Diagrama para as classes básicas de suporte em cada tipo de nó: (a) Objeto Alvo; (b) Nó Sensor e Nó Coordenador.



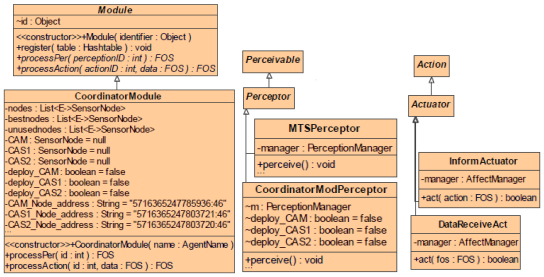
(a) (b)

Fig. 3. Diagrama de classes: (a) RA_Target Actuator; (b) RA_Sensor Perceptor e Actuator.

A Figura 4 apresenta o diagrama de classes com os perceptors e o actuator do RA_Coordinator, além de uma classe auxiliar que extende a classe `Module` da API do AFME.

A classe `CoordinatorModule` apresentada na Figura 4(a) é responsável por introduzir os agentes CA na rede. O resultado desta introcução é percebido pelo `CoordinationModPerceptor` (Figura 4(b)), o qual mantém o histórico destes agentes CA após sua entrada na rede. Isto é feito através da atualização das crenças do RACoordinator a respeito do estado dos agentes CA. Como explicado

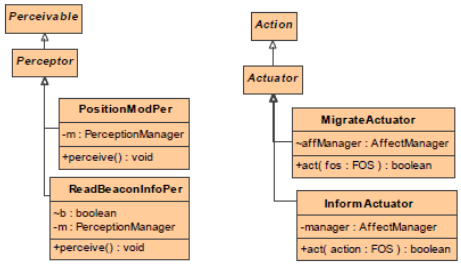
anteriormente, a introdução dos agentes CA apenas se dá após os RASensorNodes terem enviado uma mensagem sobre o recebimento do primeiro beacon do objeto alvo para o RACoordinator, o qual trata esta informação através de seu actuator `DataReceiveAct` (Figura 4(c)).



(a) (b) (c)

Fig. 4. Diagrama de classes para o RA_Coordinator: (a) Module; (b) Perceptors; (c) Actuators.

A Figura 5 apresenta os perceptors e actuators do agente CA_Master. A classe `ReadBeaconInfoPer` é responsável por atualizar as crenças do agente quando acontece o recebimento de um beacon do objeto alvo, enquanto o `PositionModPer` atualiza a informação a respeito da posição do objeto (Figura 5(a)). A classe `MigrateActuator` é responsável pela migração do agente quando o objeto algo deixa a região coberta pelo nó que atualmente está hospedando o agente, enquanto a classe `InformActuator` é responsável por enviar mensagens aos CA_Slaves hospedados pelos outros nós sensores.



(a) (b)

Fig. 5. Diagrama de classes do CA_Master: (a) Perceptors; (b) Actuators.

O diagrama de classes para o CA_Slave é muito similar ao apresentado para o CA_Master, porém com diferenças na semântica implementada em seus actuators e perceptors, como o envio de mensagens a respeito da posição do objeto alvo para o CA_Master e a decisão de migração ao receber a respectiva mensagem do CA_Master.

5 Resultados: Análise das Métricas

O software desenvolvido foi avaliado através da análise de métricas dinâmicas. Estas métricas representam informações sobre a execução da aplicação e foram extraídas através de instrumentação de seu código. As medições realizadas proveem informação sobre o custo associado a execução da aplicação em termos de tempo de utilização da CPU, consumo médio de energia e utilização de memória.

O protótipo para realização dos experimentos foi desenvolvido utilizando-se seis nós SunSPOT: um deles representando o objeto alvo a ser detectado por outros quatro nós, e um último nó foi utilizado como referência. O nó de referência não tinha nenhum agente sendo executado durante todo o tempo do experimento e sua configuração foi denominada Configuração 1. O nó representando o objeto alvo executou o agente RA_Target, sendo chamado Configuração 2. Os demais nós executaram inicialmente o agente RA_SensorNode para realizar a detecção do objeto alvo, o qual estava inicialmente fora da faixa de cobertura destes quatro nós, e tiveram esta configuração denominada Configuração 3. Em seguida, os nós foram classificados de acordo com sua carga de processamento e comunicação, dependendo dos agentes que hospedavam e executavam. Quando o objeto alvo entrou na faixa de cobertura da rede, três nós realizaram a detecção, sendo denominados Configuração 4. Finalmente, estes três nós realizaram o acompanhamento do alvo, dois executando o agente CA_Slave, Configuração 5, e um executando o CA_Master, Configuração 6. O agente RA_Coordinator foi executado em um PC representando a estação base.

A Tabela 1 apresenta os resultados para a utilização de tempo de CPU e corrente consumida, medida diretamente ligada ao consumo de energia. A Configuração 2 não apresentou significante consumo de recursos de processamento, mas sim de energia, o que é explicável pelo fato do objeto alvo periodicamente enviar beacons para a rede, o que requer uso da energia pelo módulo de comunicação. A Configuração 3 tem uma utilização de CPU muito pequena se comparada a Configuração 1, o que é explicável pelo fato de que ela representa um nó que está apenas esperando a recepção de beacons. Ela também não apresenta um alto consumo de energia. No entanto, quando o objeto alvo entra no faixa de cobertura da rede, os nós sensores que efetuam a sua detecção, Configuração 4, apresentam um maior consumo de energia, o que é explicável pelo envio da mensagem para o coordenador a respeito do primeiro beacon. A utilização de CPU também aumenta, mas não significantemente. Com a inserção dos agentes móveis CA e consequentes alterações para Configurações 5 e 6, nota-se um aumento significativo na utilização de CPU e também no consumo de energia. Isto é explicável pela execução dos cálculos realizados pelo agentes CA, bem como pela comunicação entre eles.

A Tabela 2 apresenta os resultados sobre a utilização de memória em cada nó de acordo com sua configuração e espaço de memória livre. Em termos de memória, o peso da abordagem orientada a agentes é pequeno se comparado à utilização de CPU e consumo de energia. Como esperado, o sensor

executando o agente CA_Master utilizou uma maior quantidade de memória, mas a diferença em relação as demais configurações não foi muito significante.

Tabela 1. Resultados sobre a utilização de CPU e Consumo de Corrente.

Configuração	Utilização de CPU (%)	Consumo de Corrente (mA)
1 - Sem agente – Nó de Referência	1,52	59,7
2 – Objeto Alvo	17,26	74,4
3 - Sem CA / Sem Alvo presente	3,44	65,8
4 – Sem CA/ Alvo presente	15,73	71,7
5 - CA_Slave	28,32	73,9
6 - CA_Master	60,85	85,1

Tabela 2. Resultados sobre a utilização de memória

Configuração Memória	Conf 2 Alvo	Confs 3, 4 Sensor	Conf 5 CAS	Conf 6 CAM
Livre (Bytes)	337164	334852	320300	301164
Utilizada (Bytes)	106359	108668	123221	142358

Pela análise dos resultados obtidos, é possível se avaliar o overhead imposto pela adoção de uma abordagem orientada à agentes na implementação de uma aplicação RSSF. Esta informação pode ser utilizada por um desenvolvedor para avaliar alternativas para se reduzir o overhead, como por exemplo, observando o alto custo no nó que executa do agente CA_Master, o desenvolvedor pode considerar a redistribuição da computação nele realizada e aloca-la em outros nós que executam o CA_Slave. Outro possibilidade é com relação à troca de mensagens entre os nós, que terá impacto na arquitetura da aplicação.

6 Conclusão

Este artigo apresentou a utilização de agentes móveis na implementação de aplicações de RSSF. A clássica aplicação de seguimento de alvos foi escolhida como estudo de caso, a qual foi implementada, instalada em nós sensores reais, os SunSPOTs, e avaliada. Os resultados alcançados indicam o significante consumo de recursos de processamento e moderado consumo de energia. A utilização de memória não representou peso significativo para esta aplicação em especial sendo executada sobre a plataforma SunSPOT. Estes resultados podem ser utilizados como base para novos projetos nos quais decisões de alto nível podem ter seu impactos considerados no desempenho do sistema.

Existem várias tarefas a serem executadas para dar prosseguimento a este trabalho. Uma delas é a implementação da mesma aplicação utilizando outros frameworks para agentes, e com isto analisar-se o overhead específico imposto pelos diferentes frameworks, e compará-los. Além disto, outras métricas podem ser extraídas, como métricas estáticas para serem compararadas nas diferentes implementações. Outro trabalho a ser realizado é a divisão do consumo de energia para diferentes cargas de processamento, o que vai permitir uma melhor análise e entendimento do impacto dos diferentes frameworks para agentes no uso do canal de comunicação, apontado como o maior responsável pelo consumo de energia em RSSF.

Referências Bibliográficas

1. Arampatzis, T., Lygeros, J., Manesis, S.: A Survey of Applications of Wireless Sensors and Wireless Sensor Networks. Proceedings of the 13th Mediterranean Conference on Control and Automation, Limassol, Cyprus, pp. 719--724 (2005).
2. Zhao, F., Guibas, L.: Wireless Sensor Networks: An Information Processing Approach. Elsevier (2004).
3. Allgayer, R. S. ; Steinfeld, L. ; Carro, L. ; Wagner, F. R.; Pereira, C. E. . Aplicação de agentes móveis em rede de sensores sem fio para localização e seguimento de objetos alvos móveis. Congresso Brasileiro de Automática - CBA2010, Bonito-MS, v. 01. p. 1513-1520 (2010).
4. Lange, D. B. and Oshima, M.: Seven Good Reasons for Mobile Agents. Communications of the ACM, Vol. 42, No. 3, pp. 88--89 (1999).
5. Fok, C.-L.; Roman G.-C., Lu, C.. Rapid Development and Flexible Deployment of Adaptive Wireless Sensor Network Applications. Proc. of the 24th ICDCS, pp.653--662 (2006).
6. Simon, D.; Cifuentes, C.: 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, (2005).
7. France R. and Rumpe, B.: Model-driven Development of Complex Software: A Research Roadmap. In Proceedings of Future of Software Engineering 2007. IEEE Computer Society: Washington, DC, USA, pp. 37--54 (2007).
8. Tseng, Y. C. and Kuo, S. P. and Lee, H. W. and Huang, C. F. Location tracking in a wireless sensor network by mobile agents and its data fusion strategies. Information Processing in Sensor Networks Book, Springer: Berlin, pp. 554 (2003).
9. Muldoon, C., O'Hare, G.M.P., Collier, R., O'Grady, M. J.: Agent Factory Micro Edition: A Framework for Ambient Apps. Comp. Science, Springer: Berlin, pp. 727--734 (2006).
10. SunSPOTWorld. http://www.sunspotworld.com. Acessado em: abril de 2011.
11. Object Management Group, Unified Modeling Language (UML) (2011). http://www.uml.org. Acessado em: março de 2011.