Middleware Baseado em Componentes e Orientado a Recursos para Redes de Sensores sem Fio[1]

Rodrigo P. M. de Araújo[1], Jesús M. T. Portocarrero[2], Flávia C. Delicato[2], Paulo F. Pires[2], Luci Pirmez[2], Thais Batista[1], Silvana Rossetto[2], Ana Liz S. Oliveira[1]

[1]DIMAp– UFRN [2]PPGI-DCC/IM – Universidade Federal do Rio de Janeiro

{fenrrir,jesus140,fdelicato,paulo.f.pires,luci.pirmez,thaisbatista,silvana.rossetto,analiz}@gmail.com

Abstract. *Midgard is a component-based WSN middleware designed using microkernel and REST architectural patterns whose main goals are: (i) to provide easy access to data generated by WSN, (ii) to expose such data as Web resources, and (iii) to provide application developers with capabilities to instantiate only required specific services, thus generating a customized middleware and saving node resources.*

Resumo. *Midgard é um middleware para RSSFs baseado em componentes que adota os padrões arquiteturais microkernel e REST. Seus principais objetivos são: (i) prover fácil acesso via Web aos dados gerados pela RSSF, (ii) tratar tais dados como recursos Web; e (iii) prover aos desenvolvedores de aplicações capacidades para a instanciação apenas dos serviços específicos exigidos pela aplicação, dessa forma gerando um middleware customizado e economizando recursos dos nós.*

1. Introdução

Atualmente há uma ampla gama de aplicações para Redes de Sensores Sem Fio (RSSF), variando de monitoramento ambiental a detecção de danos em estruturas. As primeiras aplicações para RSSF possuíam requisitos simples, não demandando a necessidade do uso de infraestruturas de software complexas. Além disso, as RSSF eram normalmente projetadas para atender aos requisitos de uma única aplicação alvo. Todavia, a rápida evolução da área e o aumento da complexidade dos sensores e das aplicações implicaram a necessidade do uso de sistemas de middleware específicos para essas redes. Um middleware para RSSF deve ser capaz de prover os serviços necessários para aplicações baseadas em sensoriamento, e considerar, além dos requisitos específicos das aplicações, as características inerentes dos nós dessas redes (como recursos limitados de energia, memória e CPU); do ambiente no qual os nós estão inseridos; e do contexto dinâmico de execução [Wang et al, 2008]. Nesse contexto, para atender aos cenários de RSSF emergentes, apresentamos o *Midgard*, middleware especificamente concebido

[1] Trabalho parcialmente financiado pela Rede Nacional de Ensino e Pesquisa e pelo CNPQ (processos 4781174/2010-1,309270/2009-0,311363/2011-3,470586/2011-7, 311515/2009-6 e 307269/2010-8).

para RSSF, baseado em um modelo de componentes [Maia, 2007] e nos padrões arquiteturais de microkernel [Schmidt et al, 2000] e REST [Fielding, 2000]. A adoção de um modelo de componentes simplifica a tarefa de programação, promovendo o desacoplamento e a modularização das funcionalidades dos sistemas em componentes bem definidos; e facilita o gerenciamento das funcionalidades internas do middleware, proporcionando a separação das responsabilidades e características do sistema em unidades distintas, de forma a permitir identificar as partes do sistema relacionadas a cada funcionalidade. Como vantagem, obtém-se potencialmente um alto índice: (i) de reuso, onde os componentes podem ser reaproveitados; (ii) de capacidade de evolução, sendo possível a substituição de elementos internos do componente sem interferir nos demais elementos do sistema; e (iii) de especialização, no tocante a customização de componentes. Além de adotar um modelo de componentes, o Midgard utiliza o padrão microkernel. O microkernel pode ser definido como um componente que encapsula o núcleo do sistema e é responsável por inicializá-lo. Os demais serviços são inicializados e removidos de acordo com as necessidades da aplicação. Essas características, aliadas à adoção do modelo de componentes, são desejáveis no contexto de RSSF, pois permitem poupar recursos dos nós sensores. Finalmente, no que diz respeito ao acesso aos dados coletados na RSSF, o Midgard utiliza o padrão REST [Fielding, 2000], o qual define uma forma leve de comunicação entre aplicações baseada em padrões adotados na Web. Usando REST é possível acessar os dados da RSSF como recursos Web, provendo uma camada de abstração que oculta do usuário particularidades dos dispositivos de hardware e plataformas de software das RSSF e permitindo a integração dos dados coletados com outras aplicações Web. Além disso, ao prover uma interface de acesso a dados uniforme, promove-se a interoperabilidade de redes de tecnologias distintas, atendendo a tendência atual de construir sistemas heterogêneos envolvendo múltiplas redes e aplicações [Wang et al, 2008]. Em suma, os dois principais objetivos da abordagem de projeto adotada são: prover fácil acesso via Web aos dados gerados pela RSSF, tratando-os como recursos Web, e prover aos desenvolvedores de aplicações para RSSF a capacidade de instanciar apenas serviços específicos exigidos pela aplicação.

2. Midgard

O Midgard oferece um serviço de comunicação baseado em REST e, para lidar com o ambiente dinâmico e a necessidade de otimização de recursos em RSSFs, provê os serviços de inspeção/adaptação e de configuração. Novos serviços podem ser especificados por terceiros e incorporados usando-se o modelo de componentes e as interfaces de programação do Midgard. O serviço de inspeção e adaptação fornece capacidades reflexivas [Kon et al, 2002] ao middleware, monitorando seu comportamento dinamicamente, buscando informações sobre seus componentes por meio da ocorrência de eventos, e as disponibilizando para a aplicação quando necessário. O serviço de configuração configura os nós sensores de acordo com requisitos da aplicação e busca satisfazer os requisitos da aplicação ao selecionar os protocolos de comunicação, o ciclo de trabalho dos nós e a topologia lógica a serem adotados, a partir de políticas de adaptação e perfis de componentes. Uma política pode definir que determinadas tarefas ou eventos sejam inicializados, pausados ou destruídos, permitindo definir o próximo estado interno do Midgard, conferindo o recurso de adaptação. Um perfil de componentes consiste na coleção de componentes que podem ser usados simultaneamente como configuração do Midgard. Assim, um perfil de

componentes consiste em um conjunto nomeado que define qual implementação de determinada interface deve ser utilizada pelo middleware.

2.1. Arquitetura do Midgard

O objetivo principal de adotar o padrão microkernel é tornar o núcleo do Midgard o menor possível a fim de garantir que o mínimo consumo de memória. No processo de inicialização, o Midgard deve executar apenas um núcleo mínimo. Após a inicialização desse núcleo, os demais componentes necessários para a execução da aplicação são inicializados seletivamente, permitindo a carga de uma versão customizada do middleware, talhada para os requisitos específicos da aplicação a ser executada nos nós. A Figura 1 apresenta a arquitetura do Midgard, ilustrando a pilha de serviços que trabalham acima do microkernel, bem como a relação de dependência entre os serviços.

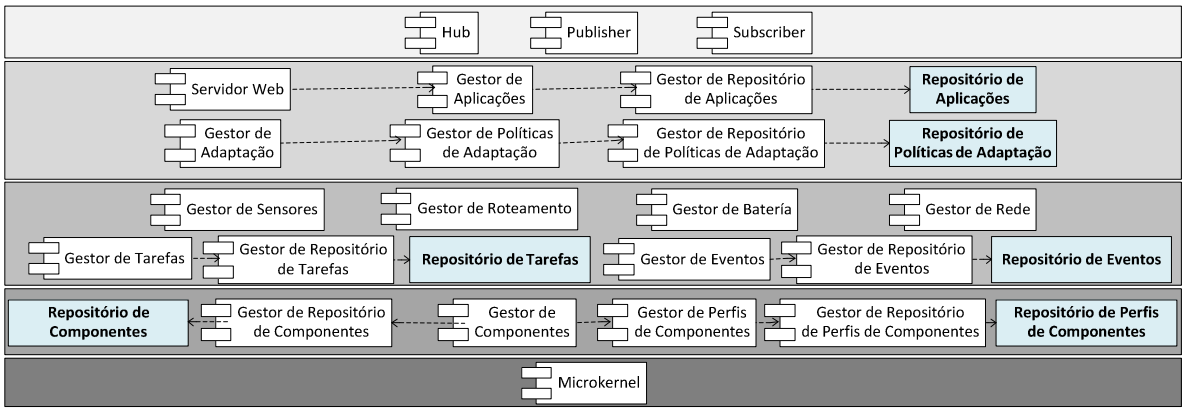

Figura 1. Arquitetura do Midgard

O microkernel inicializa o Midgard e os serviços básicos, que permitem a entrega das informações coletadas pelos nós sensores endereçadas às aplicações clientes, e que precisam permanecer ativos (i.e. na memória principal e em execução) durante todo o seu ciclo de vida. Tais serviços básicos são implementados como componentes gestores no Midgard. Portanto, durante o processo de boot, o Microkernel inicializa os Gestores de: Repositório de Componentes, Componentes e Perfis de Componentes. Por fim, o Gestor de Aplicações é iniciado com o objetivo de configurar o Midgard de acordo com os requisitos das aplicações que serão instaladas no nó. Durante o processo de configuração realizado pelo Gestor de Aplicação, os Gestores de Eventos, de Tarefas, de Roteamento, e de Sensoriamento são inicializados apenas no caso deles serem necessários para atender aos requisitos da aplicação. Os demais componentes permanecerão inativos até que eventualmente sejam requeridos pela aplicação. O Midgard possui um conjunto de repositórios de dados, os quais possuem gestores responsáveis por gerenciar o uso do repositório associado, servindo de interface entre os demais componentes do Midgard e os dados contidos no mesmo.

2.2. O Modelo de Componentes do Midgard

Esta Seção apresenta uma visão geral das partes estrutural (estática) e comportamental (dinâmica) do modelo de componentes do Midgard. Tal modelo consiste em uma implementação do FLEXCM [Filho et al, 2007], cuja escolha foi motivada pelo fato de se tratar de um modelo de componentes leve, voltado, desde o princípio, para sistemas embarcados e adaptativos. O modelo visa atender aos requisitos de permitir carga,

instanciação e conexão dinâmica dos componentes. É importante esclarecer que por carga dinâmica considera-se a carga em memória RAM de componentes já instalados no nó. As interfaces necessárias ao funcionamento do modelo de componentes serão descritas a seguir. A interface ILifeCycle tem como objetivo gerenciar o ciclo de vida de um componente por meio das seguintes operações: (i) load, executada após o carregamento de um componente no middleware; (ii) initialize, executada para inicializar um componente; (iii) pause, invocada quando as atividades de um determinado componente devem ser suspensas; (iv) resume, utilizada para retomar as operações do componente, e (v) destroy, utilizada para indicar que o componente será destruído e precisa liberar seus recursos. A interface IBaseComponent é implementada por todos os componentes do Midgard e é responsável por prover metadados que permitem a conexão dinâmica entre os componentes, como por exemplo, quais são as interfaces requeridas e as providas por um componente.

Midgard também possui um mecanismo de comunicação assíncrona baseada em eventos, permitindo que seus componentes percebam a ocorrência de eventos externos e internos, como novos dados chegando na rede, alertas de baixa energia, etc. Tal mecanismo possibilita que o Midgard proveja serviços de inspeção, configuração e adaptação. As interfaces definidas para tratar eventos são: (i) IEvent, define operações que um evento deve realizar; (iii) IEventHandler gerencia o lançamento de novos eventos, devendo ser implementada por componentes que desejem ser notificados da ocorrência de eventos; (ii) IListener, em conjunto com IEventHandler permite que os componentes se comuniquem através de eventos; (iv) IComponent, implementa todas as interfaces descritas e é implementada por todos os componentes providos pelo middleware; (v) IProxyComponent provê um mecanismo para troca de componentes dentro do middleware, permitindo implementar o padrão Proxy [Gama et al, 1995].

2.3. Gerenciamento de Componentes do Midgard

O Gestor de Perfis de Componentes tem como objetivo principal gerenciar o uso dos perfis presentes no Gestor de Repositório de Perfis de Componentes. Entre suas responsabilidades estão gerenciar o perfil de componentes ativo e o ciclo de vida de todos os perfis de componentes. Suas interfaces providas permitem obter referência ao perfil ativo; trocar o perfil ativo e realizar as operações básicas do ciclo de vida dos perfis de componentes, tais como carregar e inicializar, pausar, retomar e destruir. Como exemplo, podem existir implementações alternativas de componentes caracterizadas por consumir mais energia que outras em troca de resultados mais rápidos: um perfil de componentes denominado QoS-driven pode determinar a seleção de tais componentes. Por outro lado, um perfil de componentes denominado Energy-driven solicita a carga de componentes caracterizados por poupar mais energia no funcionamento da rede.

O Gestor de Componentes gerencia todo o ciclo de vida dos componentes do Midgard, sendo responsável por carregar, configurar e inicializar um componente e suas dependências. Além disso, também é o encarregado de realizar a troca de componentes e remover componentes ativos que não são mais requeridos, liberando os recursos de memória. Tal remoção pode ser solicitada por meio de uma política de adaptação, de um perfil de componentes ou simplesmente por meio de outro componente do Midgard. Já a troca de componentes pode ser realizada de três formas distintas, podendo ser solicitada: (i) por meio do nome dos dois componentes envolvidos na troca; (ii) por meio do Proxy

da interface e o novo componente desejado ou (iii) por meio do Proxy da interface, do componente antigo e do novo componente desejado. Por fim, o Gestor de Componentes provê o método resolveComponent, responsável por permitir a resolução de nomes do Midgard, retornando a referência a um componente por meio do seu nome.

O Gestor de Rede, juntamente com o Gestor de Roteamento, fornece informações sobre o estado da rede. Em conjunto, fornecem meios para: (i) permitir a troca do algoritmo de roteamento em uso; (ii) obter dados sobre o algoritmo de roteamento em uso; (iii) descobrir a quantidade de vizinhos de um nó na rede e seus endereços. O Gestor de Roteamento provê acesso ao algoritmo de roteamento em uso pela RSSF. Atualmente, a implementação do Midgard suporta os algoritmos de roteamento AODV (Ad-hoc On Demand Vector [RFC3561]) e LQRP (Link Quality Routing Protocol) [Paschoalino, 2007]. O componente responsável por monitorar o estado da bateria e informar dados sobre a energia residual dos nós é o Gestor de Bateria. O Gestor de Sensores é responsável por gerenciar todos os dispositivos sensores presentes no nó por meio de componentes específicos para cada tipo de sensor, os quais são inicializados de acordo com os requisitos da aplicação.

2.4. Interoperabilidade e Acesso aos dados da RSSF

Como visto, o Midgard adota o padrão REST para facilitar o acesso aos dados da RSSF e prover interoperabilidade com outras redes. REST abrange um grupo de conceitos e princípios, não havendo uma conexão direta entre este paradigma e protocolos específicos. Adotamos neste trabalho o conceito de arquitetura RESTful apresentado em [Richardson e Ruby, 2007]. Tais autores definem o termo Serviços RESTful para designar serviços Web que seguem os critérios defendidos pelo paradigma REST, os quais são usados na implementação do serviço de comunicação do Midgard. Em serviços Web baseados em REST, a interface uniforme para acesso aos recursos é dada pelos métodos definidos no HTTP [RCF 2616], sendo os principais GET, POST, DELETE e PUT. Com esses quatro métodos é possível realizar qualquer operação sobre os recursos providos por um serviço. Assim, serviços web baseados em REST são orientados a recursos [Berenguel et al, 2008]. Há várias estratégias para se disponibilizar uma RSSF como um serviço REST. É de responsabilidade do projetista do middleware determinar a granularidade do que deve ser tratado como recurso REST na rede: (i) a RSSF pode ser vista como um único recurso; (ii) cada nó individual pode ser exposto como recurso; (iii) podem existir diversos recursos em cada nó, como por exemplo, cada uma das unidades de sensoriamento (temperatura, luminosidade, etc) instaladas no nó. Adicionalmente, podem-se executar operações sobre dados individuais gerados, como agregação ou fusão, e disponibilizar os resultados dessas operações como recursos. Agregação e fusão de dados indicam a capacidade de uma RSSF de agregar ou sumarizar dados coletados pelos sensores. Essa é uma funcionalidade importante na rede, pois permite reduzir o número de mensagens transmitidas, minimizando o gasto de energia do sensor [Loureiro et al, 2002].

O serviço de comunicação do Midgard inclui a comunicação entre os nós da RSSF e com redes externas (como a Internet). A comunicação é provida por meio de um Servidor Web e componentes auxiliares, além das interfaces: IWebServer, IHTTPServer, IURLHandler e IWebApplication. Há um nano servidor Web presente em todos os nós sensores, responsável por atender as requisições recebidas. O Midgard

adota a API do protocolo HTTP dentro da própria RSSF (no entanto, não é o protocolo em si, apenas suas operações, já que não são executados TCP ou IP nas camadas subjacentes) para permitir interoperabilidade entre aplicações distintas executando na mesma rede. Para tal, o Midgard provê aplicações Web para acesso aos dados dos sensores presentes no nó. Com essa abordagem, há uma aplicação Web encarregada de tratar as requisições interessadas em dados do sensor de luminosidade, outra para o sensor de temperatura, etc.. Além disso, o desenvolvedor de aplicações também pode instalar novas aplicações Web no nó sensor. Essas aplicações permitem, por exemplo, que dada uma requisição com URL "/battery/level", o nível de bateria seja obtido do componente BatteryManager e enviado à aplicação cliente. É importante ressaltar que as respostas dessas aplicações serão sempre no formato JSON (JavaScript Object Notation) [RFC 4627], permitindo ao cliente consumi-las facilmente. A interface IWebServer, provida pelo Servidor Web, permite a instalação de novas aplicações no Servidor. Como mencionado, o Midgard conta com um conjunto de aplicações Web específico para cada dispositivo de sensoriamento. A interface IWebServer contém os métodos: (i) addWebApplication (instala uma aplicação Web no nó sensor); (ii) removeWebApplication (remove uma aplicação Web do nó sensor); e (iii) listWebApplications (lista as aplicações Web instaladas no nó sensor). O Midgard também especifica a interface IHTTPServer, que tem por finalidade definir as operações do componente responsáveis por atender as requisições HTTP. O método handleRequest dessa interface recebe dois objetos do tipo datagrama como parâmetros. O primeiro datagrama é o conteúdo da requisição enviada ao nó sensor, o qual deve ser processado; e o segundo trata do datagrama a ser preenchido como resposta pelo Servidor HTTP. Uma vez que o Servidor HTTP tenha processado a requisição, o Midgard enviará o datagrama de volta ao nó cliente. No tocante a realização do processamento de uma requisição HTTP, como a RSSF não é acessível diretamente através da Internet, por fazer uso de protocolos de camada de enlace diferentes dos empregados na mesma, a disponibilização de seus serviços exige um nó gateway (funcionalidade do sorvedouro) que possua duas interfaces de rede. A primeira interface deve suportar o protocolo TCP/IP, tornando possível seu acesso por computadores ligados a rede externa. A segunda consiste de uma interface rádio para receber/enviar as informações de/para a RSSF. O nó sorvedouro será encarregado de receber as requisições da Web e repassar para a RSSF, assim como realizar a operação inversa e devolver a Web o resultado produzido na RSSF. Dentro da rede, apenas a parte de implementação da API (operações) do HTTP é implementada pelos nós.

É importante ressaltar que o modo tipicamente pull do protocolo HTTP, onde o cliente deve realizar requisições sempre que desejar obter novos dados, pode ser demasiado custoso em cenários de RSSF. Nessas redes, há diversos tipos de aplicações, com diferentes modelos de entrega de dados, podendo ser de monitoramento periódico, baseadas em consultas síncronas ou em eventos. Em aplicações periódicas ou consultas, o uso do HTTP atende aos requisitos sem onerar demais a rede. Entretanto, para aplicações baseadas em eventos é interessante que se faça uso de um modelo de Publicação-Subscrição, onde uma aplicação cliente precisa se inscrever uma única vez e irá receber novas medições sempre que ocorrerem mudanças (modo push). O HTTP não fornece esse mecanismo de notificação de eventos, tornando-o custoso em comunicação. Diante disso, o Midgard provê suporte ao protocolo Pubsubhubbub [Pubsubhubbub]. Esse protocolo permite que a comunicação entre cliente e servidor seja feita baseada no

modelo Publicação-Subscrição, e para tal define uma terceira entidade na comunicação cliente-servidor. Essa entidade, denominada Hub, tem por objetivo tratar o modelo Publicação-Subscrição nesse cenário. São responsabilidades do Hub, cadastrar os clientes (Subscribers) interessados em receber as notícias, obter os novos dados providos pelo servidor (Publisher) e divulgar os novos dados aos clientes. Os clientes comunicam-se apenas com Hub, primeiro realizam a assinatura do conteúdo com ele, posteriormente, quando o servidor de notícias (Publisher) tiver novo conteúdo, ele deve publicá-lo no Hub. Caso o servidor de notícias não notifique o Hub por um determinado intervalo, o Hub irá realizar essa verificação. Uma vez que o Hub esteja em posse do novo conteúdo do servidor de notícias, o Hub irá divulgá-las a todos os assinantes. O Pubsubhubbub permite, assim, que os clientes não precisem verificar constantemente por novas notícias. O protocolo também elimina a comunicação direta cliente-servidor, fazendo-a sempre entre cliente-hub-servidor.

2.5. Implementação do Midgard na plataforma SunSpot

Na implementação atual do Midgard, toda a especificação descrita anteriormente foi instanciada para a plataforma SunSpot [SunSpot], gerando, portanto sua implementação de referência. A instanciação dos componentes é realizada por meio das APIs. É preciso considerar que há uma perda de desempenho no sensor no momento em que um componente que não está carregado na memória principal passa a ser requerido pela aplicação, e conseqüentemente, nesse momento, ele será carregado na memória principal. O Midgard pode ser configurado para instalar nos sensores só os componentes necessários. Os componentes utilizados pela aplicação estarão em execução na memória principal. Componentes são carregados na memória à medida que são requeridos e descarregados quando não são mais utilizados pela aplicação. Uma restrição imposta para instanciação do Midgard em outras plataformas é que a plataforma alvo suporte a carga dinâmica de componentes nativamente. Algumas plataformas, como por exemplo, o TinyOS [Hill et al. 2000], não suportam alocação dinâmica de componentes na sua forma nativa. Para suportar esse recurso, o TinyOS dispõe de algumas extensões que proporcionam a carga dinâmica de componentes, como por exemplo, o Dynamic TinyOS [Munawar et al, 2000]. O processo de implantação do Midgard funciona da seguinte maneira. Antes de ser construído o arquivo .JAR da imagem de código a ser instalada no nó, o diretório de build (que contém as classes compiladas) sofre uma operação de strip na qual os componentes não necessários à aplicação são removidos. Esse comportamento é importante pois permite economizar espaço na memória flash dos dispositivos, tendo em vista que várias implementações de um dado componente podem nunca ser carregadas. Para que seja possível realizar tal tarefa, a ferramenta faz uso conjunto do Repositório de Componentes com os requisitos da aplicação. O Midgard pode carregar qualquer componente que esteja inserido no repositório de componentes. Uma característica importante do repositório é permitir especificar mais de uma implementação para uma determinada interface de componente e escolher entre elas qual deve ser carregada para memória principal de acordo com as configurações da aplicação. O Repositório de Componentes deve ser implantado no nó sensor junto com a aplicação. Uma importante característica da implantação de componentes do Midgard é seu caráter modular, havendo três tipos de configurações: (i) Configuração Mínima (microkernel + componentes de comunicação), (ii) Configuração de Adaptação (microkernel + componentes de comunicação + Repositório de Aplicações) e (iii)

Configuração de interoperabilidade (microkernel + componentes de comunicação + REST). Tais modelos de configuração serão aplicados e testados no Estudo de Caso apresentado na Seção 3 e permitem adequar a implantação do middleware aos recursos da plataforma onde será usado. Uma contribuição do Midgard consiste em dispor de várias formas para construção de aplicações. As aplicações podem ser desenvolvidas de forma descritiva, apenas selecionando os componentes através da especificação de requisitos. Uma aplicação também pode ser desenvolvida de forma imperativa implementando-se os métodos da interface IApp. Além disso, também é possível agregar funcionalidade apenas adicionando novas aplicações Web no nó sensor. Dessa forma, o Midgard apresenta várias abstrações de programação. Em adição, o Midgard também fornece aplicações já definidas para reuso, assim como esqueletos de aplicações já desenvolvidos. A descrição detalhada do Midgard bem como o código de sua implementação pode ser encontrada em [Araújo, 2011].

3. Estudo de Caso

O estudo de caso consiste no desenvolvimento e execução de aplicações para RSSF em conjunto com o Midgard, com o intuito de demonstrar que os objetivos estabelecidos foram alcançados: (i) capacidade de reuso, (ii) capacidade de evolução, (iii) capacidade de adaptação e (iv) interoperabilidade. Tal estudo foi projetado para quatro cenários, cada qual visando um objetivo. Simulações foram realizadas no ambiente de simulação provido pelo kit de desenvolvimento da plataforma SunSpot, o Solarium [Solarium, 2011] e tiveram duração de dois minutos para cada cenário. As simulações e coleta de dados foram realizadas dez vezes, retirando-se o maior e o menor valor obtido, e calculando-se a média entre os oito valores restantes. Essa operação foi realizada para todas as medidas coletadas durante a simulação.

Cenário 1. O primeiro cenário trata de uma aplicação para medição de temperatura na área monitorada pela RSSF e visa demonstrar como o Midgard atende o objetivo de interoperabilidade proposto. A aplicação desenvolvida tem dois requisitos: realizar medições de temperatura a cada dois segundos; expor os dados coletados via Web para que possam ser consumidos por redes ou aplicações externas. A Figura 2 mostra as linhas de código da aplicação e contém a descrição do Repositório de Aplicações necessário para atender tais requisitos. Para a construção da aplicação por meio do Repositório de *Aplicações***,** é necessário especificar o nome da aplicação e o componente que a representa e que deve ser carregado **(**linhas 3 e 4**)**. O componente *TestApp* ilustrado possui um *template* de aplicação, fornecido pelo Midgard, que consiste numa aplicação padrão. Tal aplicação possui apenas o esqueleto dos métodos, servindo unicamente para representar uma aplicação a ser usada com o Midgard. Para atender os requisitos da aplicação específica a executar na RSSF é necessária a implantação do *Monitor de Rede* (linha 6), para permitir a comunicação com o cliente Web. A linha 7 contempla a implantação do componente responsável pelo *Sensor de Temperatura*. Na linha 12, o *Hub* é especificado como requisito, permitindo ao nó gerenciar subscrições por conteúdo enviadas pela Web. Na linha 17 solicita-se a realização da coleta de dados a cada dois seg., como requisitado. Para concluir as funcionalidades requeridas, instalamos a Aplicação Web responsável por retornar as medições de temperatura através do Servidor Web.

Cenário 2. Nesse cenário foi utilizada a aplicação de medição de temperatura implementada no Cenário 1, porém consideramos que ocorrem mudanças em seus

requisitos. Portanto, trata-se de uma evolução no software a ser instalado nos sensores, possibilitando estimar o esforço necessário pelo desenvolvedor ao estender a aplicação original e demonstrando a capacidade de extensibilidade provida pelo Midgard. O cenário trata da situação do mundo real onde o desenvolvedor deseja implementar uma nova aplicação reaproveitando o que foi implementado na aplicação anterior.

Figura 2. Aplicação do Cenário 1

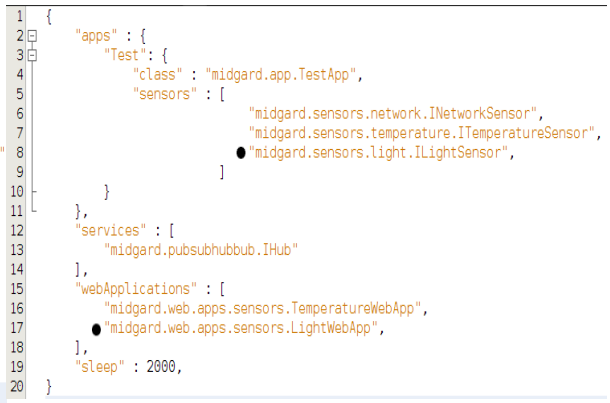

Figura 3. Aplicação do Cenário 2

Os requisitos da nova aplicação são: (i) realizar medições de temperatura e (ii) de luminosidade a cada dois segundos; (iii) prover os dados coletados via Web. Então, além dos requisitos do Cenário 1, a nova aplicação requer “medições de luminosidade”. A Figura 3 mostra a especificação dessa aplicação. As linhas 8 e 17 indicam as mudanças necessárias para atender os novos requisitos onde tem-se a indicação do uso do *Sensor de Luz* e a instalação da aplicação Web responsável por prover os dados de tal sensor, respectivamente. Conforme ilustrado, todos os componentes utilizados no desenvolvimento do Cenário 1 são reutilizados. Tal fato ilustra a facilidade de reuso dos componentes provida pelo Midgard. Com o objetivo de atingir as mudanças de requisitos ocorridas do Cenário 1 para o Cenário 2, o Midgard exigiu do desenvolvedor apenas a inclusão de 2 linhas na especificação dos requisitos da aplicação. O Cenário 2 demonstra também que o Midgard é genérico o suficiente para atender diversos requisitos de aplicações, com o benefício de permitir a customização do middleware de acordo com tais requisitos.

Figura 4. Aplicação do Cenário 3

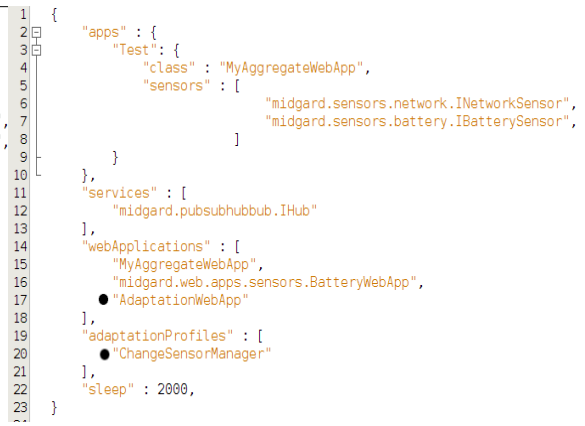

Figura 5. Repositório de Aplic. do Cenário 4

Cenário 3. Ilustra a situação onde o desenvolvedor deseja implantar novos sensores com uma nova aplicação em uma RSSF já em operação. A aplicação desenvolvida possui os

requisitos: (i) comunicação com a aplicação do Cenário 2; (ii) realização de medições do estado da bateria dos nós; (iii) provisão do estado da bateria via Web; (iv) provisão do estado da bateria mais medições de temperatura e luminosidade coletadas na aplicação do Cenário 2 e (v) provisão dos mesmos serviços das aplicações do Cenário 2. Figura 4 ilustra a definição do Repositório de Aplicações para a nova aplicação. A linha 7 define o uso de um Sensor de Bateria, responsável por realizar as medições do estado da bateria. A linha 16 define a aplicação Web que provê os dados da bateria a clientes Web. Além disso, também é informada a necessidade de uso do Hub. Nesse cenário, pela primeira vez é demonstrado como requisito da aplicação a instanciação da aplicação MyAggregateWebApp. Essa aplicação não é provida pelo Midgard e necessitou ser desenvolvida para atender os requisitos. Destaca-se o fato da aplicação MyAggregateWebApp estar especificada nas linhas 4 e 15 do Repositório de Aplicações. Tal especificação se faz necessária para que a aplicação seja automaticamente instalada no Servidor Web, podendo tratar as requisições Web endereçadas a ela. A definição de MyAggregateWebApp apenas na linha 4 não garantiria sua instalação no Servidor Web. Nessa nova aplicação, os dados de temperatura e luminosidade são coletados a partir da medição realizada por sensores já instalados na RSSF. Assim, MyAggregateWebApp é encarregada de responder a essas requisições com os dados que coletou. Este cenário tem como principal objetivo demonstrar que o Midgard provê interoperabilidade entre aplicações distintas dentro da rede. Além disso, também permite ilustrar as facilidades de reuso, extensão e evolução do Midgard. O mecanismo de interoperabilidade entre diferentes aplicações permite que novas aplicações instaladas possam se comunicar com as antigas. Dessa forma, o reuso da aplicação antiga é provido. O reuso de aplicações já implantadas auxilia a evolução da RSSF. Assim, conclui-se que o Midgard atingiu o objetivo de prover interoperabilidade, reuso e evolução.

Cenário 4. Visa demonstrar os recursos de adaptação providos pelo Midgard através da evolução do Cenário 3, adicionando recursos de adaptação à aplicação anterior. A aplicação do Cenário 3 será estendida de forma a permitir que um cliente Web troque a implementação em uso do Gestor de Sensores. Essa troca de componentes ocorre através do uso de políticas de adaptação do Midgard. Para tal, a aplicação do Cenário 4 irá atender requisições na URL “/sensor/changeSensorManager”. Quando tal URL for acessada por um cliente, o Gestor de Sensores Padrão deve ser substituído por um Gestor de Sensores Alternativo. O Gestor de Sensores Alternativo selecionado na adaptação possui a capacidade de economizar energia nos nós aumentando o tempo de sleep durante cada medição. A Figura 5 ilustra o Repositório de Aplicações usado para preencher os requisitos do Cenário 4. Como visto, em tal Repositório encontram-se todos os requisitos apresentados no Cenário 3. Para contemplar os novos requisitos do Cenário 4, o Repositório sofreu alterações mostradas nas linhas 17 e 20. Tais alterações têm por finalidade adicionar uma nova aplicação Web no nó sensor, e carregar a Política de Adaptação necessária. A linha 17 ilustra que a aplicação Web *AdaptationWebApp* foi adicionada. Além disso, entre as linhas 19 e 21, são definidas as políticas de adaptação necessárias. A política utilizada no Cenário 4 é denominada *ChangeSensorManager*.

3.1. Análise e Discussão dos Resultados

Para analisar os resultados obtidos foi usada a abordagem Goal/Question/Metric (GQM) [Basili et al. 1994], a qual é orientada a metas e utilizada em engenharia de software para avaliação de produtos e processos de software. GQM parte do princípio de que toda coleta dos dados deve ser baseada em um fundamento lógico, em um objetivo ou meta,

que é documentado explicitamente. O primeiro passo nessa abordagem é definir metas de alto nível a serem alcançadas na avaliação. Após a identificação das metas, um plano GQM é elaborado para cada meta selecionada. O plano consiste, para cada meta, em um conjunto de questões quantificáveis que especificam as medidas adequadas para sua avaliação [Basili et al. 1994]. As questões identificam a informação necessária para atingir a meta e as medidas definem operacionalmente os dados a serem coletados para responder as questões. Para analisar o estudo de caso foram definidas as duas metas exibidas na Tabela 1. As métricas usadas para analisar tais metas representam de forma quantitativa a eficiência do Midgard. Estas métricas foram aplicadas durante a execução das quatro aplicações geradas para cada um dos cenários apresentados nesta seção. Além disso, as métricas foram aplicadas em uma aplicação monolítica que executa o middleware sem o recurso de customização/alocação seletiva de componentes. Os cenários foram executados usando o simulador solarium em um notebook Core 2 duo de 1.73Ghz e 2Gb de RAM rodando o sistema operacional Ubuntu linux.

Tabela 1. Avaliação dos resultados utilizando GQM

<table>
<tr><td colspan="6">Meta 1: Analisar a viabilidade de instalação, execução e a escalabilidade do Midgard em uma RSSF</td></tr>
<tr><td colspan="2"></td><td>Aplicação</td><td>5 nós</td><td>10 nós</td><td>15 nós</td><td>20 nós</td></tr>
<tr><td rowspan="10">Questão 1.1: O tempo de resposta do Midgard do ponto de vista do usuário é aceitável?

Questão 1.2: Como se comporta o Midgard conforme novos nós são inseridos?

Questão 1.3: Os requisitos de memória exigidos pelo Midgard são aceitáveis considerando as características das RSSF?</td><td rowspan="5">Métrica M1: Tempo de resposta de requisição HTTP (milissegundos) = tempo decorrido entre a chegada de uma requisição HTTP e a resposta da requisição por parte do servidor Web</td><td>Cenário 1</td><td>97</td><td>99</td><td>111</td><td>125</td></tr>
<tr><td>Cenário 2</td><td>95</td><td>95</td><td>107</td><td>129</td></tr>
<tr><td>Cenário 3</td><td>195</td><td>205</td><td>262</td><td>281</td></tr>
<tr><td>Cenário 4</td><td>191</td><td>189</td><td>257</td><td>273</td></tr>
<tr><td>Cenário 4</td><td>57</td><td>87</td><td>109</td><td>115</td></tr>
<tr><td rowspan="5">Métrica M2: Consumo de memória RAM em uso (Kb)</td><td>Cenário 1</td><td>148.752</td><td>182.064</td><td>218.66</td><td>230.512</td></tr>
<tr><td>Cenário 2</td><td>169.458</td><td>221.02</td><td>272.768</td><td>301.272</td></tr>
<tr><td>Cenário 3</td><td>141.656</td><td>315.648</td><td>397.668</td><td>394.792</td></tr>
<tr><td>Cenário 4</td><td>142.228</td><td>340.264</td><td>350.316</td><td>379.812</td></tr>
<tr><td>Monolítica</td><td>207.864</td><td>386.112</td><td>437.14</td><td>450.092</td></tr>
<tr><td colspan="6">Meta 2: Analisar o esforço de programação e a utilidade das capacidades de adaptação e evolução providos pelo Midgard</td></tr>
<tr><td rowspan="10">Questão 2.1: O esforço de programação com o Midgard diminui quando existe código potencialmente reusável?

Questão 2.2: A customização e/ou alocação seletiva de componentes é vantajosa considerando-se os requisitos das RSSF?</td><td rowspan="5">Métrica: Número de Linhas de código fonte M3</td><td>Aplicação</td><td>Descritiva</td><td>Imperativa</td><td>Total</td></tr>
<tr><td>Cenário 1</td><td>19</td><td>0</td><td>19</td></tr>
<tr><td>Cenário 2</td><td>21</td><td>0</td><td>21</td></tr>
<tr><td>Cenário 3</td><td>19</td><td>102</td><td>121</td></tr>
<tr><td>Cenário 4</td><td>45</td><td>153</td><td>198</td></tr>
<tr><td rowspan="5">Métrica M4: Tamanho da Imagem da Aplicação (footprint de memória em Kb)

Obs: Operação de strip do Midgard remove da imagem da aplicação os componentes que não são necessários</td><td colspan="2">Aplicação</td><td>Strip (Kb)</td><td>S/ Strip (Kb)</td></tr>
<tr><td colspan="2">Cenário 1</td><td>180</td><td>228</td></tr>
<tr><td colspan="2">Cenário 2</td><td>184</td><td>228</td></tr>
<tr><td colspan="2">Cenário 3</td><td>180</td><td>228</td></tr>
<tr><td colspan="2">Cenário 4</td><td>184</td><td>228</td></tr>
<tr><td colspan="2">Monolítica</td><td>-</td><td>228</td></tr>
</table>

Quanto a meta 1, os resultados da métrica M1 indicam, como esperado, que em todos os cenários os tempos de resposta das requisições crescem na medida em que o número de clientes aumenta. O pior resultado foi obtido pela aplicação do Cenário 3, onde o tempo de resposta sofreu aumento de 30.6% para o número máximo de clientes,

que corresponde a um aumento de 400% do número inicial de clientes (demonstrando a escalabilidade do Midgard). Além disso, o tempo de resposta da ordem de 300 milissegundos ainda é considerado na literatura como sendo imperceptível do ponto de vista do usuário de aplicações Web típicas. Além disso, observou-se que o tempo de resposta é afetado pela complexidade da aplicação. No tocante à métrica M2, uma vantagem observada do Midgard foi que ele apresentou menor consumo de memória para todos os cenários em comparação como a aplicação Monolítica, uma vez que nesta, todos os componentes do middleware estão em execução. Ainda, o consumo de memória apresentado é aceitável, tendo em vista está abaixo de 50% da capacidade de memória disponível na plataforma SUN Spot. Dessa forma, entende-se que o consumo apresentado de memória é tolerável e não representa uma inviabilização do Midgard.

Com relação a métrica M3 relativa a meta 2, a tabela apresenta o número de linhas de código descritivas, as quais se referem aos requisitos da aplicação descritos em um arquivo em formato JSON, e o número de linhas de código imperativas, as quais se referem à programação em código Java necessária para construir a aplicação. Podemos perceber que, para os Cenários 1 e 2, foi possível construir a aplicação apenas descrevendo os seus requisitos. Isso é possível devido ao fato de tais aplicações terem características comuns a aplicações para RSSF. Dessa forma, os Cenários 1 e 2 são os que exigem menor esforço por parte do programador (respectivamente, 19 e 21 linhas de código). No Cenário 2, para prover as funcionalidades de medição de luminosidade e disponibilização desses dados através da Web, é necessário adicionar apenas duas linhas de código ao código do Cenário 1. Por outro lado, as aplicações do Cenário 3 e 4 exigem maior esforço do programador, devido as características dessas aplicações. O valor de 45 linhas de código obtido pela aplicação do Cenário 4 deve-se ao uso do mecanismo de adaptação do Midgard. Dessa forma, é necessário a aplicação especificar a política de adaptação a qual irá utilizar. Assim, a descrição dessa política, a qual também é escrita em formato JSON, deve ser contabilizada. A métrica M4 demonstra que o tamanho da imagem da aplicação com o Midgard é sempre menor que uma aplicação Monolítica. Entretanto, conforme a complexidade da aplicação aumenta entre os cenários de evolução, o consumo de memória da aplicação também cresce. Esse aumento é justificado pelo uso de mais componentes do middleware para atender os novos requisitos da aplicação.

4. Trabalhos Relacionados

Mires [Souto et al, 2004] é em um middleware para RSSF baseado em serviços e fundamentado no padrão publish/subscribe. Ele assume a existência de apenas uma rede externa interessada em interagir com a RSSF e essa interação está centralizada em um único nó da RSSF. Diferentemente do Midgard, o Mires não trata da interoperabilidade da RSSF com diferentes redes externas. Tais características restringem o acesso à RSSF e sobrecarregam o nó encarregado de realizar essa tarefa. Adicionalmente, o Mires especifica um padrão próprio para comunicação com a rede externa e o Midgard, por sua vez, adota o padrão de comunicação baseado em REST, que permite que a RSSF se comunique com qualquer rede externa que adote esse padrão.

Em [Boonma e Suzuki, 2009] é descrito o TinyDDS, middleware com as características de ser (re)configurável, de código aberto, baseado no padrão de projeto em camadas, e adota interoperabilidade de comunicação para aplicações baseada em

publish/subscribe. O TinyDDS não aborda questões de adaptação nem explora capacidades reflexivas de modo a permitir que a aplicação participe mais ativamente na (re)configuração da rede. A abordagem utilizada pelo TinyDDS é baseada em CORBA. Já o middleware Midgard é baseado no modelo de componentes e na arquitetura baseada em microkernel, permitindo que componentes sejam instanciados, à medida que forem sendo requisitados e descarregados quando não forem mais úteis a aplicação naquele momento. Essas características permitem adaptação e reconfiguração no Midgard.

Em [Delicato, 2005] propõe-se um middleware reflexivo orientado a serviços. Ele provê uma camada de abstração, baseada em Serviços Web, entre as aplicações e a infraestrutura de rede subjacente, além de manter um balanço entre requisitos de QoS das aplicações e tempo de vida da rede. O Midgard, em vez de adotar como em [Delicato,2005] protocolos e padrões de Serviços Web (WS*), como SOAP e XML, usa o padrão REST, que provê uma solução mais leve. Em [Luckenbach et al, 2005], os autores apresentam o TinyRest, um protocolo para integração das RSSF com a Internet. As informações disponibilizadas pela RSSF são acessadas pela Web, mas não diretamente através dos nós sensores, e sim através de um gateway, o qual pode ser considerado como uma camada de middleware. TinyRest usa um mecanismo baseado em HTTP para a troca de informações entre a RSSF e a Internet. Os autores usam requisições HTTP para obter/alterar o estado dos sensores/atuadores via Internet e realizam um mapeamento entre requisições HTTP e mensagens TinyOS. A principal diferença entre o TinyRest e o Midgard consiste na forma de utilização do protocolo HTTP dentro da RSSF. Enquanto o TinyRest faz uso de um protocolo proprietário para a comunicação entre os nós sensores, o Midgard adota a própria API do HTTP. Além disso, Midgard apresenta uma arquitetura em microkernel baseada em componentes, a qual lhe permite adaptar-se aos requisitos específicos das aplicações.

5. Considerações Finais

O Midgard é um middleware concebido especificamente para atuar em RSSF que adota os padrões arquiteturais baseado no modelo de componentes, microkernel e REST. Seu objetivo principal consiste em oferecer aos desenvolvedores recursos de alto nível para a instanciação apenas dos componentes específicos exigidos pela aplicação, customizando o middleware e economizando recursos computacionais nos nós. Além disso, Midgard permite que os dados da RSSF sejam tratados como recursos Web REST, o que possibilita alinhar a proposta ao cenário emergente de Web das Coisas. Mostramos que é possível utilizar RSSF como recursos Web e ainda prover uma arquitetura de software coesa e fracamente acoplada, endereçando interoperabilidade e customização.

Referências

Araújo, R. P. Marques. “Midgard: um middleware baseado em componentes e orientado a recursos para redes de sensores sem fio”. Natal, RN, 2011, 171f. ; il. Dissertação de Mestrado, Disponível em: http://labnet.nce.ufrj.br/?page_id=39.

Basili, V., Caldiera, G., and Rombach D. “Goal, Question Metric Paradigm”, Encyclopedia of Software Engineering, vol. 1, John Wiley and Sons, 1994.

Berenguel, A. L. A.; etal. “Arquitetura AAA em sistemas web baseados em REST”. Global Science and Technology, v. 1, n. 1, dez./mar. 2008, p. 01-07. May 2010.

Boonma, P. and Suzuki, J. “Self-configurable publish/subscribe middleware for wireless sensor networks”. In: Proceedings of IEEE CCNC. IEEE Press, p. 1376-1383, 2009.

Delicato, F.C., “Middleware Baseado em Serviços para Redes de Sensores sem Fio”, Tese de Doutorado, Universidade Federal do Rio de Janeiro, Brasil, Junho, 2005.

Fielding, R. T. “Architectural Styles and the Design of Network-based Software Architectures”. PhD Thesis University of California, Irvine, 2000.

Filho, S. M., Leite, L. E. C., Lemos G., Meira, S. “FLEXCM - A Component Model for Adaptive Embedded Systems”, In: Proc. Of COMPSAC 2007. Beijing, 2007.

Guinard, D. and Trifa, V. Towards the Web of Things: Web Mashups for Embedded Devices, Proc. Of MEM 2009, Madrid, Spain, Apr 2009.

Hill, J., et al. “System architecture directions for networked sensors”. In: Proc. Of ASPLOS 2000, p. 93–104, Massachusetts, 2000.

Kon, Fabio; Costa, Fabio; Blair, Gordon e Campbell, Roy H.. ”The case for reflective middleware”. Commun. ACM 45, 6, 2002, p. 33-38

Loureiro, A. A., et al. Rede de sensores sem fio. In Porto, I. J. Anais do Simpósio Brasileiro de Computação, JAI, 2002, p. 193–234

Luckenbach, T., et al, "Tiny REST - a Protocol for Integrating Sensor Networks into the

Internet," in Proc. of REALWSN'05, 2005.

Maia, R. F. “Um framework para adaptação dinâmica de sistemas baseados em componentes distribuídos”. 2007. Dissertação - DI PUC-RIO, Rio de Janeiro, 2007.

Munawar, W. et al. “Dynamic TinyOS: Modular and Transparent Incremental Code-Updates for Sensor Networks”. Proc of the IEEE ICC 2010, Cape Town, p. 23-27.

Paschoalino, R.; Madeira, E.R.M., “A Scalable Link Quality Routing Protocol for Multiradio Wireless Mesh Networks”. Proc. of WiMan 2007, Honolulu, 2007.

Pubsubhubbub. Disponível em: http://code.google.com/p/pubsubhubbub/.Acesso:set/11.

RFC 3561. Disponível em http://www.ietf.org/rfc/rfc3561.txt. Acesso: Abril 2010.

RFC 4627. Disponível em: http://tools.ietf.org/html/rfc4627. Acesso: Set 2010

Richardson, L.; Ruby, S. “RESTful Web Services”. O’Reilly Media, USA, 2007

Schmidt, D., et al. Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects. John Wiley & Sons, Inc., 2000.

Solarium. Disponível em: https://solarium.dev.java.net. Acesso em: jun 2011.

Souto, E.; Guimaraes, G.; Vasconcelos, G. et al. “A message-oriented middleware for sensor networks”. In: Proc. of the 2nd Workshop on Middleware for Pervasive and Adhoc Computing, Vol. 77, 2004, p. 127-134.

Sunspot. Sun Spot World Disponível em: http://sunspotworld.com/. Acesso: Jan 2011.

Wang, M.M., Cao, J.N., Li, J., Dasi, S.K.: Middleware for wireless sensor networks: A survey. Journal of Computer Science and Technology 23(3), May, 2008, p. 305-326