Arquitetura para o desenvolvimento de soluções embarcadas de internet das coisas aplicadas em redes elétricas inteligentes

Glauber da Silva Bernardo, Anderson Dourado Cunha, Elias Teodoro da Silva Júnior
Instituto Federal de Educação, Ciência e Tecnologia do Ceará - IFCE
Departamento Ciência da Computação, 60040-531, Fortaleza - CE, BRASIL
glauber.silva@ppgcc.ifce.edu.br

***Resumo*—As Redes Elétricas Inteligentes (*Smart Grids*) empregam vários dispositivos para monitorar, analisar e controlar a própria rede. Esses dispositivos são implantados em grande quantidade em usinas de energia, centros de distribuição e nas instalações dos consumidores. Uma *Smart Grid* requer conectividade, automação e rastreamento para tais dispositivos. Por outro lado, a Internet das Coisas (*IoT*) tem sido utilizada para prover esses serviços, mas ainda há uma série de desafios a serem superados. Os principais deles se relacionam à alta diversidade de plataformas de hardware e software utilizadas para implementar aqueles computadores embarcados que estão mais próximos da infraestrutura da rede elétrica, chamados dispositivos de borda. Para reduzir o tempo de desenvolvimento dessas aplicações embarcadas, este artigo propõe um *middleware IoT*, denominado *Smart Grids Applications Middleware (SGAM)*, facilitando a criação de soluções embarcadas para controle, monitoramento e medição em Redes Elétricas Inteligentes. Uma aplicação de monitoramento remoto de ativos foi desenvolvido e é apresentado para demostrar o funcionamento da arquitetura proposta.**

***Index Terms*—Redes Elétricas Inteligentes; Internet das Coisas; *Middleware*; Redes *LPWAN*; Aplicações Embarcadas.**

I. Introdução

O conceito de *Smart Grid (SG)* ou Redes Elétricas Inteligentes, cunhado em [1], apresenta uma mudança no paradigma do setor elétrico. Ele leva em conta a necessidade de tornar o sistema de entrega de energia mais interativo, incorporando diferentes fontes de energia na rede, em especial, fontes geradoras descentralizadas, renováveis e intermitentes. Além disso, introduz novos consumidores, como veículos elétricos, e melhora a eficiência e o próprio dimensionamento da rede.

Desde a criação da primeira rede de energia elétrica não tinham ocorrido inovações tão significativas na maneira como essa energia era fornecida ao consumidor. A introdução das Tecnologias de Informação e Comunicação (TIC) nas redes elétricas pôde iniciar um processo retroalimentado de desenvolvimento. Conforme [2] e [3], as primeiras tentativas de se instalar alguma inteligência na rede vieram da medição eletrônica. Os medidores inteligentes foram implantados em todo o mundo substituindo medidores mecânicos para fornecer leituras de energia mais precisas e observáveis remotamente. Posteriormente foram incluídos serviços de flexibilidade tarifária, detecção de adulteração, monitoramento de qualidade de energia e gerenciamento de picos, conexão/desconexão e muitos outros. Atualmente, as Redes Elétricas Inteligentes possuem vários tipos de dispositivos para monitorar, analisar e controlar a rede. Tais dispositivos são implantados em usinas, linhas de transmissão, torres, centros de distribuição e instalações dos consumidores, e seus números chegam a centenas de milhões ou mesmo bilhões [4].

Conforme descrito em [5], a Internet das Coisas pode ajudar a Rede Elétrica Inteligente, suportando várias funções na geração, armazenamento, transmissão, distribuição e consumo de energia. Também auxilia incorporando dispositivos (como sensores, atuadores e medidores inteligentes), bem como fornecendo conectividade, automação e rastreamento para tais dispositivos. No entanto, há ainda uma série de desafios a serem superados para alavancar a ampla disseminação desse paradigma. Principalmente com relação ao desenvolvimento de aplicações e à alta diversidade de tecnologias de *hardware* e *software* desses ambientes [6], [7]. Dentre esses desafios, destacam-se: (*i*) a diversidade de plataformas de desenvolvimento e linguagens de programação, (*ii*) as limitações de desempenho das linhas de comunicação e (*iii*) a heterogeneidade dos diversos *hardwares*, sistemas operacionais e protocolos existentes.

O desenvolvimento das soluções embarcadas de Internet das Coisas para as Redes Elétricas Inteligentes possui requisitos em comuns, tais como: (1) funcionalidades de controle, monitoramento e medição dos dispositivos; (2) interface de comunicação com fio e sem fio; (3) comunicação unidirecional ou bidirecional; (4) protocolos de comunicação; suporte a diversos sensores, atuadores e módulos; e (5) portabilidade da aplicação entre diferentes plataformas de hardware [8]. A partir da identificação desses requisitos em comum, esse trabalho propõe unificá-los em um componente de software para redução do tempo e dos custos de projeto. Propõe-se um *middleware* que seja genérico o suficiente para permitir sua utilização por variados tipos de aplicações feitas nessas redes e que, ao mesmo tempo, seja leve para garantir um bom desempenho e boa integração.

Este artigo descreve um *middleware IoT*, denominado *Smart Grids Applications Middleware (SGAM)*, para a criação de soluções embarcadas de controle, monitoramento e medição em Redes Elétricas Inteligentes. O *middleware SGAM* busca disponibilizar um conjunto de funcionalidades que permitirá

ao desenvolvedor focar principalmente na regra de negócio de sua aplicação. Também facilitará a portabilidade das soluções criadas entre diversas plataformas de prototipação existentes. Utilizando os recursos disponibilizados pelo *middleware*, o desenvolvedor estará livre para implementar sua aplicação independentemente do seu nível de conhecimento da plataforma alvo, pois estará programando focado na *API* do *middleware*. Com isso, objetiva-se que pressões ligadas ao mercado de consumo também sejam atendidas, tais como: a redução do tempo desde a ideia até a disponibilidade do produto (*time-to-market*) e a crescente complexidade das aplicações [9].

O *middleware* desse trabalho executa nos dispositivos de borda, aqueles computadores embarcados que estão no final do processo, ou seja, mais próximos dos equipamentos elétricos da rede.

II. Trabalhos Relacionados

Nesta seção, a literatura foi revisada sob três categorias de pesquisa, nas quais a proposta está inserida: Arquitetura de *middleware IoT*, uso das redes *Low Power Wide Area Network (LPWAN)* em soluções *IoT* para *smart grids* e utilização de sistemas operacionais embarcados visando a diminuição do tempo de projeto e aumento da portabilidade.

Várias plataformas de *middleware* foram propostas e desenvolvidas para resolver diferentes desafios. Em [8], os autores propõem um *middleware* chamado *SmartCityWare* e discutem como a abordagem de *middleware* orientada a serviços pode ajudar a resolver alguns dos desafios de desenvolver e operar serviços de cidade inteligente usando o paradigma da computação na borda da rede (*Fog Computing*). No trabalho realizado em [10], é apresentado o projeto de um *middleware* orientado a eventos para serviços gerais em *smart grid (SG)*. O *middleware* proposto permite interoperabilidade independente de *hardware* entre tecnologias heterogêneas. Um estudo de caso foi desenvolvido para demonstrar a implantação prática. Diversas plataformas comerciais, por exemplo, ThingSpeak[1], NodeRed[2], Geniot[3] e OpenHAB[4] facilitam a implementação de aplicações, e oferecem suporte a várias placas controladoras e serviços. A principal diferença entre o *Middleware SGAM* e outras plataformas de *middleware* propostas é que o *SGAM* auxilia a criação da parte das aplicações que será executada no nodo final com ênfase na comunicação por *LPWAN*. Sua arquitetura adaptável favorece a criação de novas aplicações embarcadas de maneira mais simples pela utilização dos conceitos de reuso de software e permite ser implementada para diferentes plataformas.

Algumas soluções foram propostas integrando aplicações de Internet das Coisas às Redes Elétricas Inteligentes através da utilização das *LPWAN*, objetivando reduzir custo e o consumo de energia dos dispositivos. No trabalho apresentado em [11], os autores exploraram as principais características de *RF Mesh* e *LoRaWAN*, fazendo uma comparação entre elas e discutindo a adequação do *LoRaWAN* à Rede Elétrica Inteligente. Concluem, que para serviços de Internet das Coisas, incluindo Redes Elétricas Inteligentes, a tecnologia emergente *LoRaWAN* representa uma alternativa realmente convergente, que também adota padrões abertos e interoperáveis, e a um custo menor se comparado ao *RF Mesh*. Em [12], os autores mostram que a empresa de energia elétrica nacional da Indonésia (PLN Bali) implementou a tecnologia da *LPWAN* no medidor inteligente. Como resultado, foi observado que a tecnologia *LoRaWAN* se adequou perfeitamente à solução, obtendo precisão no controle dos dados e eficiência operacional. Observa-se a aderência da tecnologia *LoRaWAN* com redes que buscam monitoramento remoto a um consumo baixo de energia, justificando assim, a sua escolha para a infraestrutura de comunicação de dados.

Como plataforma de software, diversos estudos de Internet da Coisas sugerem a utilização de sistemas operacionais embarcados visando a diminuição do tempo de projeto e aumento da portabilidade das soluções. Na produção de [13], um sistema foi proposto para a detecção de queda de idosos. O *ARM Mbed OS* foi utilizado na unidade responsável por transmitir os dados coletados dos sensores para o aplicativo principal. Já no trabalho de [14], um sistema de coleta de energia foi criado e integrado a um *middleware* de Internet das Coisas. A integração foi realizada com o *ARM Mbed OS* por ser um sistema operacional embarcado de código aberto, com *drivers* para diferentes protocolos de comunicação existentes e por oferecer suporte a plataformas de diversos fabricantes. Observa-se que para reduzir o tempo de projeto e, consequentemente, os custos, é crucial que programadores utilizem plataformas que facilitam e aceleram a programação. Para esses fins o *ARM mbed OS* vem se mostrando eficaz e sendo muito utilizado.

III. Arquitetura Proposta

A abordagem do *Middleware SGAM* foi inicialmente contextualizada na taxonomia apresentada em [15] para auxiliar a definição de aspectos arquiteturais. Além disso, o desenho arquitetural do *middleware* seguiu a metodologia de projeto orientado a pontos adaptáveis (*Hot Spot Driven Design*), desenvolvida em [16], que consiste em identificar os pontos fixos e os pontos adaptáveis do domínio de uma aplicação, dando ao *middleware* a capacidade de ser flexível e se moldar às diferentes aplicações e plataformas embarcadas.

Diferentes casos de uso (serviços de controle, monitoramento e medição) fornecidos para projetistas de soluções de Internet das Coisas e Redes Elétricas Inteligentes foram pensados. Com base nesses resultados, a especificação funcional do *middleware* foi estabelecida.

- Arquitetura modular para que possa ser reutilizado em cada nova implementação;
- *API* com suporte a multilinguagem;
- Disponibilidade de diversos protocolos utilizados em sistemas de medição;
- Comunicação básica cabeada ou por redes sem fio, transmissão de dados bidirecional, confiabilidade, segurança e

[1] https://thingspeak.com/
[2] https://nodered.org/
[3] https://www.geniot.io/site/
[4] https://www.openhab.org/

escalabilidade obtidas com as redes *LPWAN*, especificamente *LoRaWAN*;
- Controle dos recursos do sistema, manipulação de dados e escalonamento de tarefas;
- Suporte a diversas plataformas de prototipação.

A. Infraestrutura da Rede

LoRa ® é uma tecnologia de radiofrequência baseada em padrões abertos, de baixo custo, que permite uma comunicação a longas distâncias com consumo mínimo de energia [17]. Foi projetada desde o início para construir plataformas urbanas para Internet das Coisas. Assim, de acordo com [11], o escopo de aplicação é muito abrangente. Dentre as possíveis aplicações estão Redes Elétricas Inteligentes para serviços públicos, incluindo empresas do setor de energia.

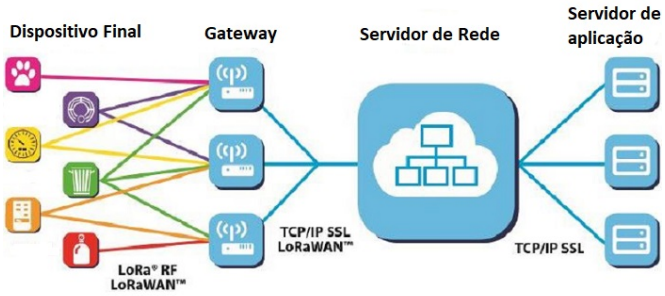

Figura 1. Protocolo de Rede *LoRaWAN* [17]

.

LoRaWAN™ é um padrão aberto de protocolo de rede que utiliza *gateways* se comunicando com os dispositivos *LoRa* e repassando esta informação para os servidores de rede e de aplicação final. Considerando que a arquitetura do *middleware* funcionará sobre o protocolo *LoRaWAN*, a topologia de rede será a definida para esse protocolo. São especificados quatro elementos: nodo (dispositivo) final, *gateway*, servidor de rede e o servidor de aplicação. A Figura 1 mostra a infraestrutura da rede *LoRa*. Vale lembrar que a arquitetura de *middleware* proposta neste artigo, se destina exclusivamente ao sistema embarcado localizado no nodo final.

B. Características do Nodo Final

Uma solução de Internet das Coisas é caracterizada por muitos dispositivos (nodos finais ou de borda) que podem usar algum tipo de *gateway* para se comunicar por meio de uma rede com um servidor corporativo que está executando uma plataforma *IoT* na nuvem. As funções dos dispositivos, *gateways* e plataforma de nuvem são bem definidas e cada uma delas fornece recursos e funcionalidades específicas nas soluções de Internet das Coisas.

O dispositivo ou nodo final é o ponto de partida para uma solução de Internet das Coisas, sendo tipicamente o originador dos dados devido sua interação com o mundo físico. Algumas aplicações, visando a uma redução nos custos ou no uso de energia, por exemplo, utilizam dispositivos (microcontroladores) com muito poucos recursos de memória e desempenho.

A Figura 2 traz um detalhamento da hierarquia de *software* e *hardware* do nodo final, visando a uma melhor compreensão da interação do *middleware SGAM* com as demais camadas.

O nodo é dividido em quatro camadas:
- **Aplicação:** Responsável pela combinação completa de funções e funcionalidades que descrevem bem a regra de negócio de uma aplicação útil ao usuário final;
- **Middleware:** Responsável por disponibilizar uma interface para servir de base para a programação da aplicação do usuário final;
- **Sistema:** O sistema operacional embarcado utilizado para gerenciar os recursos do microcontrolador e seus dispositivos periféricos internos ou externos;
- **Dispositivo:** A camada que descreve os *hardwares* utilizados, podendo englobar recursos e interfaces do próprio microcontrolador ou externas a ele, bem como sensores e atuadores.

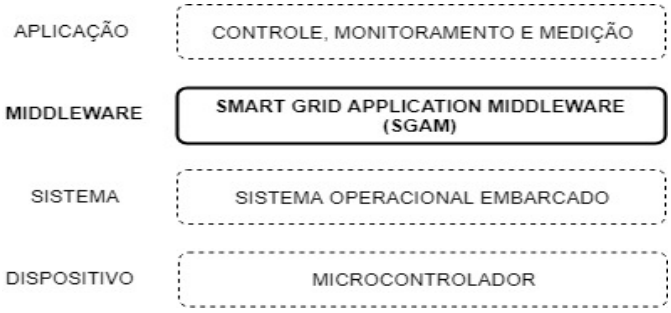

Figura 2. Divisão do nodo em quatro camadas.

A divisão da Figura 2 pretende enfatizar a independência das quatro camadas em relação à criação da aplicação embarcada. Assim, o desenvolvedor cria sua solução com base na *API* disponibilizada pelo *middleware*, o que permitirá que o código da aplicação seja reutilizado em outras plataformas.

C. Smart Grid Application Middleware (SGAM)

O *middleware SGAM* foi concebido em quatro módulos principais, oferecendo desde uma *API* padrão multilinguagens até sua integração com a plataforma utilizada. A arquitetura conceitual do *middleware* é apresentada na Figura 3. Esse modelo conceitual busca ilustrar os aspectos mais significativos do domínio do problema no mundo real. Cada módulo encapsula os detalhes de implementação, ajudando a lidar com a complexidade, diminuindo o acoplamento e, por fim, disponibilizando interfaces claras e coesas para o sistema. Com isso, projetistas compreendem melhor a arquitetura do software produzido. Adicionalmente, os módulos podem ser reutilizados mais de uma vez, dentro do mesmo programa ou por outros programas. A seguir, cada módulo do *middleware SGAM* é apresentado.

- ***API* Padrão Multilinguagens:** É o módulo responsável por oferecer as funcionalidades de controle, monitoramento e medição dos dispositivos conectados à rede

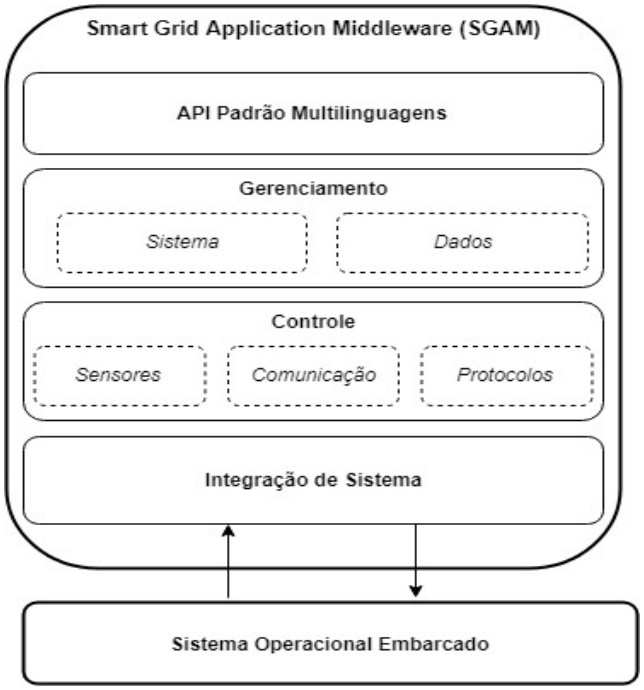

Figura 3. Arquitetura do *SGAM - Smart Grid Application Middleware*.

elétrica. Essencialmente será implementada em C/C++, no entanto, também suportará as linguagens *Java*, *Python* e *JavaScript*.

- **Gerenciamento:** Esse módulo engloba um conjunto de serviços e tarefas que permitem gerenciar os recursos do próprio sistema embarcado (nodo final da rede) e os dados trafegados. O submódulo *Sistema* oferece funcionalidades de coletas e registros do ciclo de vida dos dispositivos. No submódulo *Dados* ocorre a conversão de tipos, além de agregação/desagregação de dados em lotes.
- **Controle:** Esse módulo controla o acesso a sensores, tipos de comunicação e protocolos. O submódulo *Sensores* descreve um modelo de sensor genérico. O submódulo *Comunicação* disponibiliza uma interface de conectividade, que pode ser com fio e sem fio. Já o submódulo *Protocolo* deve prover a descrição de diversos tipos de protocolos para interagir com os dispositivos da rede elétrica, por exemplo, o protocolo *DLMS/COSEM (Device Language Message Specification/Companion Specification for Energy Metering)*, que interage com medidores inteligentes.
- **Integração de Sistema:** É o módulo responsável pela integração entre o *middleware* e o sistema operacional *IoT* da plataforma utilizada. Este módulo responde pela implementação das camadas superiores (controle, gerenciamento e *API*) e pela ”tradução”entre as funcionalidades disponibilizadas pelo *middleware* e o sistema operacional utilizado, conforme mostrado na figura 3.

D. Escolha da plataforma ARM Mbed

A plataforma *ARM Mbed IoT* fornece um sistema operacional embarcado de código aberto (*ARM mbed OS*). Esse *S.O.* possui um grande número de bibliotecas padrão projetadas para desenvolver aplicações *IoT*, incluindo segurança, conectividade e *drivers* para periféricos e sensores. Além disso, esse sistema é suportado por um grande número de plataformas de hardware de vários fabricantes com processadores que variam de *Cortex-M0* a *Cortex-M7*. Por isso, a camada de sistema (Figura 2) será implementada usando o *ARM Mbed OS*.

Como mencionado nos trabalhos de [18] e [19], as arquiteturas *ARM* são predominantemente usadas em dispositivos móveis e sistemas embarcados. Além disso, mais de 30 fornecedores de placas e componentes para dispositivos embarcados dão suporte ao *ARM Mbed OS*, tais como: *STMicroelectronics*, *MultiTech*, *NXP*, *Semtech* e outras [20]. Essa ampla base favorece a criação de aplicações de Internet das Coisas e auxilia a integração com outras soluções em diferentes plataformas.

E. Integração do SGAM com a Plataforma ARM Mbed

O processo de integração com a plataforma alvo se dá pela correta implementação das classes e interfaces que descrevem as funcionalidades presentes nos módulos. Assim, cada módulo é composto por um conjunto de classes e interfaces que devem ser corretamente implementadas para garantir a total compatibilidade entre as *API's* do *middleware* e do sistema alvo. Ao implementá-las adequadamente cria-se um objeto que se torna específico para o sistema operacional utilizado.

IV. Implementação do Middleware

Os módulos **Controle** e **Integração de Sistema** foram selecionados para a implementação funcional da proposta de arquitetura do *Middleware SGAM*. Com a escolha desses módulos é possível: (*i*) verificar a execução do *middleware* no *Mbed OS*, (*ii*) validar o controle e interação com os sensores e (*iii*) avaliar a comunicação utilizando o *LoRaWAN*.

A modelagem dos módulos foi realizada a fim de indicar uma representação formal das principais entidades do *middleware* e seus relacionamentos. O diagrama de classes e a notação da *UML* 2.0 (*Unified Modeling Language*) foram utilizados para essa representação. Para a geração do código-fonte foi utilizada a linguagem C++, utilizando-se dos conceitos *Object-Oriented (OO)*, *templates* e *maps*. A abordagem de reutilização de software baseou-se na estratégia *top-down*, aplicando *design patterns* para sistemas embarcados [21], como *Factory Method* e *Singleton*.

Nessa versão preliminar do *middleware SGAM* o Módulo Gerenciamento não está disponível e o desenvolvedor dispõe somente da linguagem C++.

A. Módulo de Controle

O módulo de controle possui a classe abstrata *control* e uma descrição genérica de um tipo *sensor* e um tipo *communication*. O submódulo protocolo não está implementado, e não será utilizado no caso de uso da seção V. A Figura 4, ilustra

o diagrama de classes do módulo Controle. As classes exibidas são *Control*, *Sensor* e *Communication*.

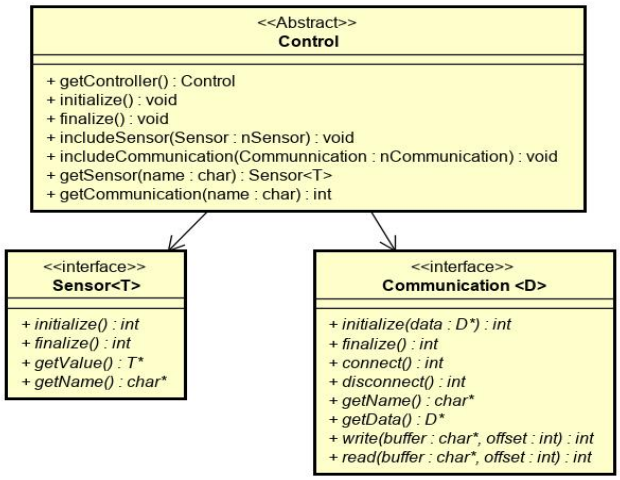

Figura 4. Diagrama de classe - *Control*.

- *Control*: é uma classe abstrata responsável por controlar a inclusão e utilização de novos sensores, tipo da comunicação e protocolos. A classe *Control* possui sete operadores que devem ser implementados:
  - *getController*: retorna uma instância única (*Singleton*) da classe *control*.
  - *initialize*: inicializa os objetos da classe *control*.
  - *finalize*: finaliza os objetos da classe *control*.
  - *includeSensor*: inclui a definição de novos tipos de sensores;
  - *includeCommunication*: inclui a definição de novos tipos de comunicação;
  - *getSensor*: obtém a referência do objeto sensor criado.
  - *getCommunication*: obtém a referência do objeto comunicação criado.
- *Sensor*: é uma *interface* que descreve um sensor genérico. A especialização dessa classe gera a descrição de diferentes tipos de sensores; por exemplo, sensor de giroscópio, sensor de temperatura, sensor de posicionamento global (gps), etc. Quatro operações devem ser implementadas para definir o comportamento do sensor utilizado.
  - *initialize*: operador responsável por inicializar os objetos da classe sensor que foi especializada.
  - *finalize*: operador responsável por finalizar os objetos da classe sensor que foi especializada.
  - *getValue*: retorna os dados da classe especializada. O tipo *T** descreve o modelo de dados utilizado.
  - *getName*: retorna o nome do sensor que foi especializado, por exemplo, um sensor *gyroscope*.
- *Communication*: é uma *interface* que descreve uma comunicação genérica. Aponta especificamente para o servidor de dados da aplicação *IoT*. A especialização dessa classe, gera a definição de um ou mais tipos de comunicação; por exemplo, comunicação *LoRaWAN*. Oito operadores devem ser implementados para definir o comportamento da comunicação utilizada:
  - *initialize*: operador responsável por inicializar os objetos da classe *communication* que foi especializada. O tipo *D** encapsula os parâmetros de inicialização.
  - *finalize*: operador responsável por finalizar os objetos da classe *communication* que foi especializada.
  - *connect*: realiza o *handshake* para a comunicação definida.
  - *disconnect*: responsável por desconectar do tipo de comunicação definida.
  - *getName*: retorna o nome da comunicação que foi especializada, por exemplo, comunicação *LoRaWAN*.
  - *getData*: retorna os dados da classe especializada. O tipo *D** descreve o modelo de dados utilizado.
  - *write*: escreve/envia os dados trafegados na comunicação.
  - *read*: recebe/ler os dados trafegados na comunicação.

B. Módulo de Integração

O módulo de integração é a implementação das classes/interfaces do módulo de controle. Desta forma, a implementação da *Control* gera a classe *ControlImpl*, a *sensor* gera as classes *Gps*, *Gyroscope* e *Temperature* e a *communication* gera a classe *LoRaWanComm*. A Figura 5, ilustra as classes geradas a partir do módulo de controle.

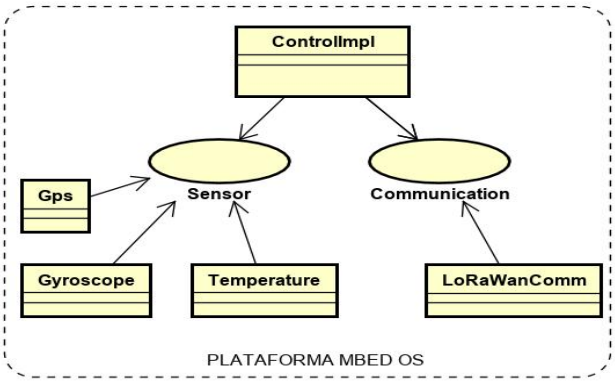

Figura 5. Classes geradas para a plataforma *Mbed OS*.

Após o primeiro passo de integração com a especialização dessas classes, o próximo passo ocorre no nível do sistema operacional embarcado utilizado. Neste caso, a integração é definida com o *Mbed OS*. Assim, o *middleware SGAM* se utiliza da *API* e *Drivers* do sistema operacional embarcado para ter acesso a recursos como interfaces *UART*, *SPI*, *I2C*, *GPIO* e outras. A seguir, são exibidos trechos de códigos utilizados na implementação do *middleware SGAM*.

A implementação da classe *Control* é realizada na classe *ControlImpl*. Conforme é observado na Figura 6, linhas 12 a 14, três tipos de sensores são incluídos para o módulo de controle do *middleware SGAM*, o sensor de temperatura, o

sensor de giroscópio e o sensor *GPS*, com suas respectivas interfaces de comunicação (*A1-Analog IO*; *I2C*; *serial*). A inclusão do tipo de comunicação *LoRaWAN* ocorre na linha 17. Após a inclusão dos sensores e da comunicação, estes objetos estão disponíveis no módulo de controle. Nas linhas 21 a 23, é exibido o método que retorna a referência para o objeto ***control*** que será utilizado pela aplicação para acessar os recursos do *middleware*.

```
9  ControlImpl::ControlImpl() {
10
11   // add Sensor
12   includeSensor( (Sensor<void*>*) new Temperature(A1) );
13   includeSensor( (Sensor<void*>*) new Gyroscope(i2c) );
14   includeSensor( (Sensor<void*>*) new Gps(serial) );
15
16   // add Communication
17   includeCommunication( (Communication<void*>*)new
     LoRaWanComm("") );
18 }
19
20   // returns single instance
21 Control* ControlImpl::getController() {
22   return this;
23 }
```

Figura 6. Trecho de código dos operadores de inclusão e obtenção do *controller*.

Na Figura 7, observa-se a definição de um tipo de sensor, o giroscópio. A definição dos tipos *GPS* e *Temperature* ocorrem de forma similar. A classe *Gyroscope* implementa os operadores da interface *Sensor*. Conforme é observado na linha 19, a interação do sensor com a placa de prototipação ocorre pela interface *I2C*. O tipo *GyroscopeData*, linha 25, descreve o modelo de dados que ele deve retornar, no caso, são inteiros que possuem três eixos: gx, gy e gz. Na linha 29, verifica-se a obtenção da referência *mpu* para o *driver* do sensor de giroscópio. E na linha 30, a declaração do método *getMotion* usado pela classe para obter os valores do sensor.

```
17 class Gyroscope: public Sensor<GyroscopeData> {
18 public:
19     Gyroscope(I2C &i2c);
20     virtual ~Gyroscope();
21
22     virtual int initialize();
23     virtual int finalize();
24
25     virtual GyroscopeData* getValue();
26     virtual const char* getName();
27
28 private:
29     MPU6050 mpu;
30     void getMotion(GyroscopeData* data);
31 };
```

Figura 7. Trecho do código da definição da classe *Gyroscope*.

A implementação da interface *Communication* ocorre de forma similar a interface *Sensor*. Na definição da classe *LoRaWanComm*, conforme é observado na Figura 8, ocorre a declaração dos operadores da interface *Communication*. Destaca-se, na linha 61, o tipo *LoraData*, que possui o modelo de dados necessário para a inicialização do tipo de comunicação *LoRaWAN*. Essa classe de tipo de dados encapsula os parâmetros de inicialização; por exemplo, o tipo de comunicação utilizada: via rádio *LoRa*, o tipo de conexão: *Over-The-Air Activation (OTAA)* ou *Activation By Personalization (ABP)*, seus parâmetros conforme a conexão, porta de leitura e escrita. O *driver* utilizado para interagir com o rádio do sistema, definido na linha 67, é o *LoRaWANInterface* que fornece uma *API C++* para se conectar à Internet através de uma rede *LoRa*.

```
49 class LoRaWanComm: public Communication<LoraData> {
50 public:
51     LoRaWanComm(const char* parameters);
52     virtual ~LoRaWanComm();
53
54     virtual int initialize(LoraData* data);
55     virtual int finalize();
56
57     virtual int connect();
58     virtual int disconnect();
59
60     virtual const char* getName();
61     virtual LoraData* getData();
62
63     virtual int write(const unsigned char* buffer
       , int offset);
64     virtual int read(unsigned char* buffer, int
       offset);
65
66 private:
67     LoRaWANInterface* lorawan;
68     LoraData* data;
69 };
```

Figura 8. Trecho do código da definição da classe LoRaWanComm.

Após a implementação das classes *ControlImpl*, *LoRaWanComm*, *Gyroscope*, *Gps* e *Temperature* é possível gerar o *middleware SGAM* para a plataforma *Mbed OS*.

V. Exemplo de utilização do Middleware

Um caso de uso utilizando o *middleware SGAM* foi projetado para demonstrar o uso da arquitetura proposta. O exemplo desenvolvido realiza o monitoramento remoto dos ativos (equipamentos) da rede elétrica. Com este caso de uso é possível auxiliar os profissionais da companhia elétrica responsáveis pelo monitoramento desses dispositivos. O *middleware* provê acesso a sensores que monitoram grandezas físicas, as quais indicam o estado operacional de ativos, como transformadores e isoladores. Exemplos dessas grandezas são: temperatura, vibração e corrente elétrica. Com esses dados se pode fazer diagnóstico desses ativos ou permitir levantamentos estatísticos. No caso de equipamentos que podem ser removidos, a aplicação pode detectar e alarmar deslocamentos não programados, utilizando Giroscópio e/ou GPS.

A Figura 9 ilustra um exemplo de sistema de monitoramento remoto de ativos de uma rede elétrica utilizando um dispositivo embarcado. No dispositivo é executada uma **aplicação** que obtêm dados dos sensores e os envia para a nuvem, alimentando um sistema de diagnóstico dos ativos.

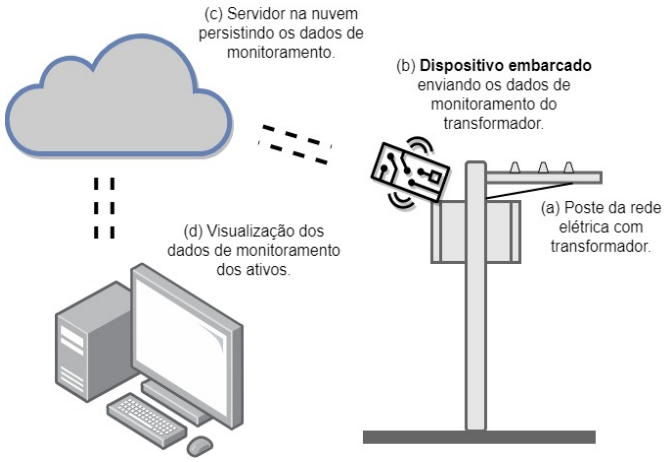

Figura 9. Exemplo de sistema de monitoramento utilizando dispositivo embarcado.

Na Figura 10, um trecho do código da aplicação de monitoramento remoto de ativos utilizando o *middleware* é apresentado. Nas linhas 10 e 11, observa-se a obtenção da referência da classe *control* e sua inicialização. Com o objeto *control* é possível acessar os recursos disponibilizados pelo *middleware*, no caso, sensores e a comunicação. Nas linhas 13 a 17 são obtidas as referências dos sensores (temperatura, giroscópio, GPS) e da comunicação (LoRaWAN), respectivamente. Nas linhas 18 e 19, se inicializa a comunicação *LoRaWAN* com as configurações de conexão definidas no objeto *LoraData*. Em seguida, a aplicação tenta realizar a conexão na rede. Da linha 21 à 23 é obtido o valor do sensor de temperatura (*temp->getValue()*) e é enviado para o servidor de rede (*COMM->write()*). Na linha 24 a aplicação de monitoramento lê a mensagem (*COMM->read()*) enviada pelo servidor de rede. Essa mensagem de resposta pode ser utilizada como comando para ativar a *thread* com o sensor de movimento. Caso haja alguma pertubação nos valores do sensor, este deve iniciar o GPS e enviar suas coordenadas de posicionamento.

```
10  Control* control = ctrl.getController();
11  control->initialize();
12
13  Sensor<void*>* temp = control->getSensor(
    "Temperature");
14  Sensor<void*>* gyro = control->getSensor(
    "Gyroscope");
15  Sensor<void*>* gps = control->getSensor(
    "GPS");
16
17  Communication<void*>* COMM = control->
    getCommunication("LoRAWAN");
18  COMM->initialize(loraData);
19  COMM->connect();
20
21  value = (float*)temp->getValue();
22
23  COMM->write( value, sizeof(value));
24  COMM->read( rx_buffer, sizeof(rx_buffer));
```

Figura 10. Trecho do código da aplicação de monitoramento de ativos.

A. Dispositivos utilizados

As placas utilizadas no experimento foram a *STM32F411RE* e *STM32F429ZI*. Ambas possuem um processador *ARM 32bits Cortex-M4*, no entanto, a *STM32F411RE* atua com frequência de 100MHz, *512 KB de memória Flash* e *128KB SRAM*, enquanto a *STM32F429ZI* possui frequência de 180MHz, *2048 KB de memória Flash* e *256KB SRAM*. Para a comunicação com o servidor é usado um transceptor LoRa da *Semtech SX1272 Low Power Long Range*. Os sensores são: temperatura (*Grove*), giroscópio (*GY-87 MPU 6050*) e de posicionamento global (*GPS ublox NEO6m*). Quanto aos recursos de rede, foi utilizado um *Gateway LoRa RISINGHF 2S008*, configurado no plano de banda Australiana 915MHz, utilizando os canais de 8 a 15 e adicionado ao servidor de rede *LoRaWAN The Things Network (TTN)*.

Tabela I
MEMÓRIA UTILIZADA (*Flash* E *RAM*)

Placa	Memória	Aplicação	Placa	%
STM32F411RE	*Flash*	*113.1 kb*	*512.0 kb*	*22,1%*
	RAM	*13.4 kb*	*96.0 kb*	*13,9%*
STM32F429ZI	*Flash*	123.1 kb	2.0 MB	6,1%
	RAM	30.1 kb	260.0 kb	11,6%

A tabela I mostra os recursos de memória *Flash* (código e constantes) e *RAM* (dados) utilizados na geração da aplicação para duas plataformas de hardware selecionadas. O próprio compilador *ARMCC* do *Mbed* gerou uma aplicação mais otimizada para a plataforma *STM32F411RE* que possui menos recursos disponíveis. Parte dessa otimização ocorreu pela diminuição significativa da seção da memória *RAM* reservada para dados não inicializados. Nas duas plataformas, apenas uma parte dos recursos foi utilizada. Isso indica que a solução gerada poderá ser aplicada em plataformas de prototipação com recursos de hardware mais limitados.

B. Utilização do Middleware SGAM

Para realizar a conexão da aplicação de monitoramento remoto de ativos à rede *LoRaWAN* é necessário adicionar ao servidor de rede (*The Things Network (TTN)*) as configurações do *gateway* e das placas de prototipação. Um fluxo completo de conexão, envio e recebimento de dados foi realizado com cada placa; como forma de validação inicial. A Figura 11 exibe a tela de dados da aplicação adicionada no servidor de rede *TTN*. A Figura 11(a) mostra a mensagem de confirmação de estabelecimento de conexão com a rede *LoRaWAN*. Após a confirmação de conexão, o dispositivo pode enviar dados ao servidor. Na Figura 11(b) se pode verificar a chegada do *payload* no servidor de rede (*UPLINK*) na porta 15, com os dados de temperatura com o valor de 27°C. Finalmente, no campo (c) da Figura 11, está a confirmação de leitura da mensagem enviada pelo servidor de rede (*cmd.value:1111*) para o dispositivo embarcado (*DOWNLINK*).

VI. CONCLUSÃO

Este artigo apresentou o *Smart Grid Application Middleware (SGAM)*, um *middleware* para aplicações embarcadas

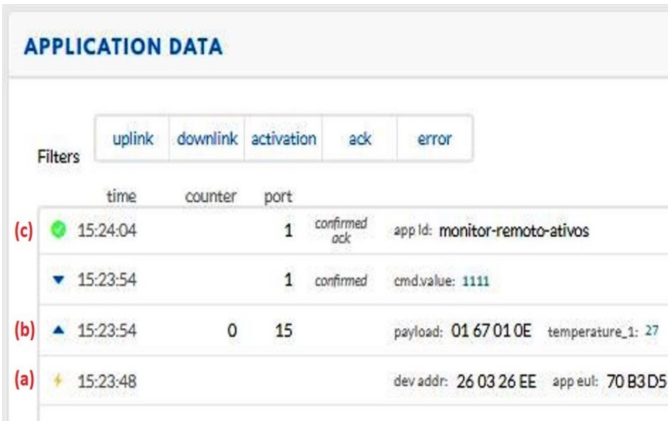

Figura 11. TTN: Exemplo de conexão estabelecida e tratamento de dados bidirecional.

de automatização em redes elétricas inteligentes. A definição da arquitetura conceitual e a modelagem do *middleware* foram baseadas na taxonomia apresentada em [15]. Além disso, o desenho arquitetural do *middleware* seguiu a metodologia de projeto orientado a pontos adaptáveis [16]. Outro fator que foi considerado durante o desenho arquitetural foi possibilitar a integração com as redes *LPWAN* objetivando um consumo baixo de energia na comunicação, permitindo utilizar dispositivos alimentados por baterias. Adicionalmente, se utiliza um sistema operacional de Internet das Coisas como plataforma para disponibilizar recursos de *hardware*, *software* e rede, diminuindo o tempo de desenvolvimento e aumentando a portabilidade das aplicações.

Quanto à implementação, um *middleware* leve e reutilizável foi gerado. Com o *middleware SGAM* foi possível projetar de forma simples uma aplicação de monitoramento para dispositivos da rede elétrica. Parte desses aspectos positivos foram alcançados pela utilização de padrões de projeto direcionados a sistemas embarcados. Com a criação do caso de uso monitoramento remoto de ativos, verificou-se: (*1*) a execução do *middleware* no *Mbed OS*, (*2*) seu controle e interação com diferentes placas de prototipação e diversos sensores e (*3*) a comunicação bidirecional entre o dispositivo final e o servidor de rede utilizando o protocolo *LoRaWAN*. Assim, a proposta de arquitetura se mostra promissora para a criação de aplicações embarcadas para sistemas de monitoramento aplicados em redes elétricas inteligentes.

Como trabalhos futuros a camada de gerenciamento será implementada e validada. Sugere-se ainda projetar outros casos de uso, avaliando os demais módulos da arquitetura. Também se deve verificar os recursos de *hardware* requeridos pelo uso do *middleware*, auxiliando na definição de eventuais limitações quanto ao número de sensores conectados e quais plataformas embarcadas podem ser utilizadas.

Referências

[1] S. M. Amin and B. F. Wollenberg, "Toward a smart grid: power delivery for the 21st century," *IEEE Power and Energy Magazine*, vol. 3, no. 5, pp. 34–41, Sep 2005.

[2] MME, *Relatório, Smart Grid*, Ministério de Minas e Energia, Brasil, abr. 2010.

[3] A. Moscatelli, "Innovative system on chip platform for smart grids and internet of energy applications," in *2016 IEEE Symposium on VLSI Circuits (VLSI-Circuits)*, June 2016, pp. 1–2.

[4] S. E. Collier, "The emerging enernet: Convergence of the smart grid with the internet of things," *IEEE Industry Applications Magazine*, vol. 23, no. 2, pp. 12–16, 2017.

[5] W. Meng, R. Ma, and H.-H. Chen, "Smart grid neighborhood area networks: a survey," *IEEE Network*, vol. 28, no. 1, pp. 24–32, 2014.

[6] M. S. Obaidat, N. R. Reddy, K. P. Venkata, and V. Saritha, "Context-aware middleware architectural framework for intelligent smart grid data management," in *2018 International Conference on Computer, Information and Telecommunication Systems (CITS)*. IEEE, 2018, pp. 1–5.

[7] M. K. Afzal, M. H. Rehmani, A. Pescape, S. W. Kim, and W. Ejaz, "Ieee access special section editorial: The new era of smart cities: Sensors, communication technologies, and applications," *IEEE Access*, vol. 5, pp. 27 836–27 840, 2017.

[8] N. Mohamed, J. Al-Jaroodi, S. Lazarova-Molnar, I. Jawhar, and S. Mahmoud, "A service-oriented middleware for cloud of things and fog computing supporting smart city applications," in *2017 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computed, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*. IEEE, 2017, pp. 1–7.

[9] A. Sangiovanni-Vincentelli, L. Carloni, F. De Bernardinis, and M. Sgroi, "Benefits and challenges for platform-based design," in *Proceedings of the 41st annual Design Automation Conference*. ACM, 2004, pp. 409–414.

[10] E. Patti, A. L. A. Syrri, M. Jahn, P. Mancarella, A. Acquaviva, and E. Macii, "Distributed software infrastructure for general purpose services in smart grid," *IEEE Transactions on Smart Grid*, vol. 7, no. 2, pp. 1156–1163, 2014.

[11] H. G. S. Filho, J. P. Filho, and V. L. Moreli, "The adequacy of lorawan on smart grids: A comparison with rf mesh technology," in *2016 IEEE International Smart Cities Conference (ISC2)*, Sep. 2016, pp. 1–6.

[12] G. Wibisono, S. G. Permata, A. Awaludin, and P. Suhasfan, "Development of advanced metering infrastructure based on lora wan in pln bali toward bali eco smart grid," in *2017 Saudi Arabia Smart Grid (SASG)*, Dec 2017, pp. 1–4.

[13] N. Bhati, "mhealth based ubiquitous fall detection for elderly people," in *2017 8th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, July 2017, pp. 1–7.

[14] D. Balsamo, A. Elboreini, B. M. Al-Hashimi, and G. V. Merrett, "Exploring arm mbed support for transient computing in energy harvesting iot systems," in *2017 7th IEEE International Workshop on Advances in Sensors and Interfaces (IWASI)*, June 2017, pp. 115–120.

[15] L. F. Rahman, T. Ozcelebi, and J. J. Lukkien, "Choosing your iot programming framework: Architectural aspects," in *2016 IEEE 4th International Conference on Future Internet of Things and Cloud (FiCloud)*, Aug 2016, pp. 293–300.

[16] P. Wolfgang, "Design patterns for object-oriented software development," *Reading Mass*, vol. 15, 1994.

[17] LoRa, *LoRaWAN™ What is it?*, LoRa-Alliance, San Ramon, Califórnia, EUA, nov. 2015.

[18] W. Zhu, "Teaching assembly programming for arm-based microcontrollers in a professional development kit," in *2017 IEEE International Conference on Microelectronic Systems Education (MSE)*, May 2017, pp. 23–26.

[19] M. Hossein, A. Hemmat, O. A. Mohamed, and M. Boukadoum, "Towards code generation for arm cortex-m mcus from sysml activity diagrams," in *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2016, pp. 970–973.

[20] ARM Mbed. (2019) Development boards, mai 2019. ARM Mbed. [acessado 03-mai-2019]. [Online]. Available: https://os.mbed.com/platforms/

[21] B. P. Douglass, *Design patterns for embedded systems in C: an embedded software engineering toolkit*. Elsevier, 2010.