Paradigma Orientado a Notificações para Aplicações de Internet das Coisas em Cidades Inteligentes

Luis H. P. Figueiredo[1], **Jean M. Simão**[2], **Ana Cristina B. Kochem Vendramin**[1]

[1]Programa de Pós Graduação em Computação Aplicada (PPGCA)
Universidade Tecnológica Federal do Paraná (UTFPR) – Curitiba – PR – Brasil

[2]Programa de Pós Graduação em Engenharia Elétrica
e Informática Industrial (CPGEI)
Universidade Tecnológica Federal do Paraná (UTFPR) – Curitiba – PR – Brasil

luisfigueiredo@alunos.utfpr.edu.br,{jeansimao,criskochem}@utfpr.edu.br

Abstract. *The Internet of Things (IoT) for Smart Cities enables a better use of resources and services. However, the technological challenges for the creation of efficient computational systems increase due to the number and complexity of interactions. This paper proposes a Distributed version of the Notification Oriented Paradigm (NOP) via a publish/subscribe architecture with the Message Queuing Telemetry Transport (MQTT) protocol. Performance of NOP and the object-oriented programming paradigm are compared in a generic IoT application. Results show that by distributing the NOP entities, it is possible to reduce the execution, processing and response times of the application at the cost of higher memory usage.*

Resumo. *A Internet das Coisas para Cidades Inteligentes possibilita um melhor uso dos recursos e serviços. Porém, os desafios tecnológicos para a criação de sistemas computacionais eficientes aumentam devido ao número e complexidade das interações. Este artigo propõe distribuir as entidades do Paradigma Orientado a Notificações (PON) por meio de uma arquitetura publicação/assinatura com o protocolo MQTT. O desempenho do PON e do paradigma de programação orientado a objetos são comparados em uma aplicação genérica de Internet das Coisas. Os resultados mostram que ao distribuir as entidades do PON é possível reduzir o tempo de execução, processamento e resposta da aplicação ao custo de um maior uso de memória.*

1. INTRODUÇÃO

A Internet das Coisas (ou IoT do termo em inglês *Internet of Things*) contribui para a construção das Cidades Inteligentes em diversos aspectos. O uso dos sistemas computacionais e dos dispositivos sensores e atuadores conectados por meio da IoT permite o aprimoramento na forma de interação com o ambiente e um melhor uso dos recursos e serviços, seja em uma pequena residência ou em uma grande metrópole. Como exemplos, pode-se considerar: um sistema de controle de tráfego inteligente, fazendo uso de sensores de contagem de veículos e atuando por meio dos semáforos com o objetivo de melhorar o fluxo de veículos em uma determinada região; e um sistema de iluminação pública inteligente, cujos sensores de luminosidade natural e do fluxo de passagem de

pessoas e veículos podem permitir o controle da intensidade luminosa das lâmpadas, reduzindo o desperdício de energia elétrica.

Porém, se por um lado a sociedade se beneficia dessa revolução promovida pelas aplicações IoT nos diversos âmbitos de uma cidade inteligente, por outro lado essa crescente demanda por desenvolvimento de *software* agrava ainda mais a chamada 'Crise de Software' [Bautsch 2007]. Resumidamente, a 'Crise de *Software*' indica a dificuldade na produção de *software* frente ao rápido crescimento da demanda aliada ao aumento da complexidade dos problemas a serem resolvidos [Ronszcka 2019][Bautsch 2007].

É necessário que as técnicas, ferramentas de projeto, análise e implementação de sistemas, em especial de sistemas distribuídos voltados para IoT, tornem-se cada vez mais fáceis e acessíveis aos desenvolvedores de sistemas, ao mesmo tempo que conduzam a implementações mais eficientes e eficazes. Porém, os paradigmas de programação atualmente dominantes e usuais, derivados dos primordiais Paradigmas Imperativos (PI) e dos Paradigmas Declarativos (PD), apresentam ineficiências para o desenvolvimento e execução de programas, principalmente em ambientes concorrentes e/ou distribuídos [Ronszcka 2019] [Simão and Stadzisz 2008].

O Paradigma Orientado a Notificações (PON) surge justamente como uma alternativa aos paradigmas de programação existentes [Simão and Stadzisz 2008]. Em sua concepção, o PON agrega características desejadas do PI (nomeadamente a orientação a objetos) e PD (nomeadamente a programação lógica) ao mesmo tempo que resolve parte dos seus problemas e limitações facilitando o desenvolvimento de aplicações mesmo em ambientes concorrentes e/ou distribuídos, ao mesmo tempo que as torna mais eficientes no uso dos recursos computacionais [Simão and Stadzisz 2008] [Ronszcka 2019].

Para a programação de aplicações segundo o PON, existem diversos aparatos tecnológicos já desenvolvidos. Enquanto estado da arte, o mais relevante é a linguagem de programação e compilação própria do paradigma, conhecida como tecnologia LingPON [Oshiro et al. 2021]. Enquanto estado da técnica, há um conjunto de *fraweworks* implementados em diversas linguagens de programação orientada a objetos, alterando assim suas formas usuais de utilização. Este trabalho propõe um aprimoramento do *Framework* PON C++ 4.0 [Neves et al. 2021] apresentando, de forma inovadora, o PON no contexto de sistemas distribuídos para IoT por meio de um protocolo de mensagens padronizado e arquitetura publicação/assinatura.

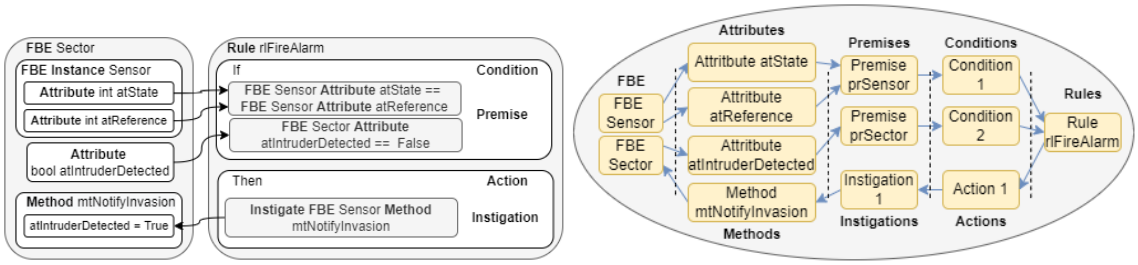
2. PARADIGMA ORIENTADO A NOTIFICAÇÕES

Dentre os principais problemas encontrados em PI e PL, pode-se citar: a redundância temporal que ocorre principalmente nos ciclos implícitos de execução do paradigma imperativo os quais frequentemente avaliam desnecessariamente variáveis que não tiveram seus valores alterados desde o último ciclo de verificação; e o forte acoplamento de código entre expressões causais e estrutura de fatos/dados, a qual não permite ou dificulta a execução paralela e/ou distribuída de instruções. O PON surge nesse cenário como uma alternativa, possuindo mecanismos que eliminam (ou ao menos diminuem) os problemas apresentados sem perder as características desejáveis dos paradigmas como a coesão, a abstração e a clareza de código [Simão and Stadzisz 2008] [Banaszewski 2009].

O principal diferencial do PON, quando comparado ao PD ou PI, é a utilização de um mecanismo de notificação que ocorre por meio de entidades minimalistas, colabo-

rativas e adaptadas para tal. Dessa forma, as notificações são geradas e recebidas pelas entidades apenas nos momentos pertinentes, ou seja, quando de fato há mudanças de estado que demandem uma nova avaliação lógico-causal. Esse mecanismo faz com que os sistemas desenvolvidos em PON sejam mais concisos quanto ao uso de recursos computacionais [Simão and Stadzisz 2008] [Neves et al. 2021].

Em linhas gerais, o PON propõe a divisão da computabilidade em dois grandes grupos relacionados entre si por meio de notificações de seus constituintes, conforme exemplificado na Figura 1 [Simão and Stadzisz 2008] [Banaszewski 2009]: (i) o grupo que trata do processamento facto-execucional por meio de entidades chamadas *Fact-Base-Elements* (*FBEs*). As *FBEs* são as entidades responsáveis por representar entidades do mundo real ou abstrato por meio de estados (*Attributes*) e serviços (*Methods*), de forma análoga (mas, distinta) aos objetos do Paradigma Orientado a Objetos (POO); (ii) o grupo que trata do processamento lógico-causal por meio de entidades chamadas de *Rules*, de forma análoga (mas, distinta) aos Sistemas Baseados em Regras (SBR). Cada *Attribute* é capaz de notificar as *Rules* por meio das *Premises* e *Conditions*, enquanto cada *Method* é capaz de ser instigado pelas *Rules* por meio das *Actions* e *Instigations*.



(a) Interação entre entidades do PON. **(b)** Ciclo de notificações.

Figura 1. Paradigma Orientado a Notificações.
[Oshiro et al. 2021]

A relação entre as entidades constituintes do PON é ilustrada na Figura 1a por meio de um exemplo que utiliza como base um sistema simples de alarme. Como requisito funcional, tem-se um ambiente (*FBE Sector*) no qual se o estado (*atState*) de um sensor (*FBE Sensor*) for igual a um valor de referência (*atReference*) e ainda não estiver alarmado (*atIntruderDetected*), então os usuários serão notificados (*mtNotifyInvasion*).

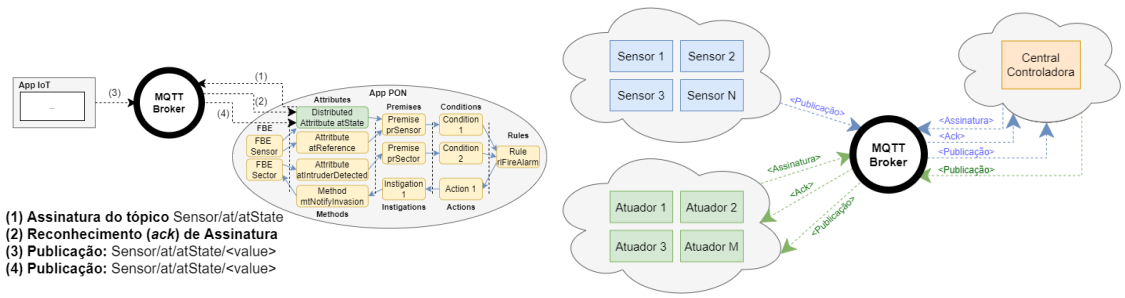
Destaca-se a maneira inovadora de colaboração das entidades do PON por meio de notificações, conforme Figura 1b. Ao mudar de estado, cada *Attribute* de uma instância de um *FBE* notifica apenas as *Premises* pertinentes, que refazem seus cálculos lógicos. Por sua vez, cada *Premise* que mudar de estado lógico notifica apenas as *Conditions* pertinentes, as quais refazem seus cálculos lógicos pelos estados notificados contabilizados. Se a *Condition* é aprovada, ela pode aprovar sua respectiva *Rule*. Esta, quando aprovada, ativa sua *Action*, que notifica suas *Instigations*, as quais instigam os *Methods*. Estes últimos geralmente alteram os estados dos *Attributes*, reativando o fluxo de notificações. Por considerar as alterações dos *Attributes* para desencadear os ciclos de notificações, o PON elimina a redundância temporal, pois as avaliações lógicas passam a ser executadas somente quando existe uma alteração pertinente. Ademais, as conexões entre as entidades ocorrem em tempo de construção dos sistemas [Oshiro et al. 2021] [Ronszcka 2019].

3. PON DISTRIBUÍDO PARA IOT

O presente trabalho propõe a distribuição, experimentação e a comparação do PON em um ambiente de IoT por meio de um sistema de correlação de sensores e atuadores, utilizando arquiteturas e protocolos típicos desse contexto. Esse tipo de aplicação é muito comum em cidades inteligentes, estando presentes, por exemplo, nos sistemas de controle de tráfego inteligente, os quais fazem uso de sensores (contagem de veículos) e atuadores (semáforos) ou em um contexto de iluminação pública inteligente, cujos sensores de luminosidade natural e do fluxo de passagem de pessoas e veículos podem permitir o controle da intensidade luminosa das lâmpadas.

Para permitir a distribuição do PON, alterações foram adicionadas no *Framework* PON C++ 4.0 de modo a permitir que as interações entre todas as entidades do PON aconteçam por meio da publicação ou recebimento de mensagens via rede. Por exemplo, um *Attribute* distribuído pode ter seu estado alterado via recebimento de mensagens de rede e notificar a(s) *Premise*(s) interessada(s) conforme o mecanismo do PON. Como a mensagem que desencadeia as alterações segue o protocolo padronizado, a alteração do *Attribute* distribuído pode ocorrer via alteração de um outro *Attribute* distribuído ou de outro elemento genérico como, por exemplo, um sensor ou aplicação IoT.

Para a comunicação, utiliza-se o protocolo MQTT (*Message Queuing Telemetry Transport*) padronizado pelo OASIS para IoT e um dos protocolos que apresentam maior suporte em plataformas comerciais [Al-Masri et al. 2020]. O MQTT implementa o padrão publicação/assinatura, desacoplando o cliente editor (sensor) do(s) cliente(s) assinante(s) por meio de uma estrutura de tópicos inserida em um elemento intermediário conhecido como *broker*. Quando o cliente editor publica uma mensagem em determinado tópico, esta mensagem é repassada para os clientes assinantes que previamente tenham registrado interesse no respectivo tópico. O *broker* é responsável pela criação e gerenciamento de tópicos os quais permitem a identificação e filtragem de mensagens de acordo com um determinado domínio [mqt]. A correspondência entre cada entidade do PON e os tópicos MQTT segue um padrão de nomenclatura de tópicos, nomeadamente *<domínio>/<entidade PON>/<Identificação Única>/<conteúdo>*. Como exemplo, considera-se o *Attribute* distribuído *atState* da *FBE Sensor* apresentado na Figura 2a, interagindo por meio do tópico "*Sensor/at/atState/*". De forma similar, as demais entidades do PON podem publicar mensagens em tópicos, assinar/registrar em interesse em tópicos e consumir as mensagens em tópicos que sejam de seus interesses.



(a) Interações com o MQTT. **(b)** Exemplo de aplicação IoT.

Figura 2. PON distribuído.

4. EXPERIMENTOS

No cenário proposto para análise, consideram-se N sensores e M atuadores os quais interagem com uma central controladora exclusivamente por meio de mensagens MQTT enviadas e recebidas ao/do *broker*, conforme ilustrado na Figura 2b. Ao detectar uma mudança de estado, os sensores (clientes editores) podem publicar mensagens em tópicos específicos no *broker*, servindo como notificação à central controladora. Por sua vez, os elementos atuadores (clientes assinantes) recebem comandos da central controladora, via *broker*, por meio de mensagens publicadas nos tópicos pertinentes os quais foram previamente subscritos pelos atuadores.

Como os sensores e atuadores possuem uma lógica relativamente simples, necessitando apenas notificar uma mudança de estado e receber comandos da central controladora, optou-se por não considerá-los na análise. Dessa forma, utilizou-se duas aplicações auxiliares sendo uma delas responsável por publicar mensagens de mudanças de estados em seus respectivos tópicos no *broker* e a outra responsável por receber os comandos de controle enviados pela central controladora para os atuadores.

Por sua vez, a central controladora, na qual a lógica de processamento dos eventos acontece, foi implementada utilizando-se dois paradigmas: (i) o Paradigma Orientado a Objetos (POO) distribuído por meio do protocolo MQTT; (ii) o PON via *Framework* PON C++ 4.0 distribuído com MQTT. Para fins de análise, implementou-se também a central controladora nas versões não distribuídas do POO e do PON via *Framework* PON C++ 4.0. Para uma comparação justa entre as implementações distribuídas, utilizou-se o mesmo *broker* de mensagens e bibliotecas de comunicação MQTT. O *broker* utilizado foi o *Mosquitto* e suas respectivas bibliotecas de comunicação em C++ [Light 2017].

Para avaliar o desempenho desses paradigmas em diferentes condições e configurações típicas em ambientes de IoT, foram criados cenários com diferentes percentuais de ativações dos sensores. Para as implementações em PON, cada conjunto de regras lógico-causais foi transcrito para o respectivo conjunto de *Rules*. Para as implementações em POO utilizou-se estruturas decisórias usuais de "se-então" avaliando por meio de chamadas de métodos os estados dos respectivos objetos e, quando pertinente, atuando por meio dos métodos dos respectivos objetos.

Utilizou-se um processador Intel(R) Core(TM) i7-8665U, Quad Core, 2.11 GHz, 16GB de memória RAM DDR4 e sistema operacional *Windows 10 Enterprise*. Para evitar eventuais atrasos na rede de comunicação, avaliou-se todo o sistema na máquina local. As seguintes métricas de desempenho são avaliadas:

- **Tempo médio de execução da aplicação:** tempo total de execução incluindo a inicialização das estruturas (por exemplo, os objetos do POO) e os ciclos de avaliações lógicas dos estados e respectivas alterações;
- **Tempo médio de processamento das mensagens:** tempo de processamento após a alteração de estado de um sensor, sendo a diferença de tempo entre o recebimento de uma mensagem e a finalização da avaliação dessa mensagem. Optou-se por avaliar o tempo médio de processamento das mensagens em duas situações: (i) quando a mensagem recebida provoca alteração no estado de um atuador e; (ii) quando a mensagem recebida não provoca alteração no estado de um atuador;
- **Tempo médio de resposta do sistema:** tempo médio para ativação de um atuador após alteração do respectivo sensor, representando a diferença de tempo entre o

recebimento de uma mensagem MQTT indicando a troca do estado de um sensor e a consequente mensagem MQTT de ativação de um atuador;
- **Uso máximo de recursos de processamento e memória:** alocação de memória máxima necessária para a execução de uma aplicação e o uso máximo de recursos de processamento.

Para avaliar o **tempo médio da execução da aplicação**, utilizou-se o *framework* do *Google benchmark* e o conjunto de testes propostos por [Neves et al. 2021]. Esse conjunto consiste na inicialização de 100.000 sensores e posterior alteração do valor de porcentagem de sensores (0,001, 0,01, 0,1, 1, 10 e 100) e suas respectivas avaliações lógicas decorrentes. Para o PON distribuído e POO distribuído, considerou-se o uso de sensores alteráveis via mensagens MQTT e via código local.

Para a avaliação do **tempo médio de processamento** das mensagens utilizou-se a biblioteca *chrono* para a medição da diferença de tempo entre o recebimento de uma mensagem MQTT até a finalização desse processamento e liberação para o processamento de uma nova mensagem. Para a avaliação do **tempo de resposta do sistema**, utilizou-se o *software Wireshark* para o registro dos pacotes recebidos e enviados e um *script* auxiliar para processamento dos dados gerados pelo *Wireshark*, contabilizando o tempo entre uma requisição e sua consequente resposta. Em ambos experimentos, utilizou-se um conjunto de 1.000 sensores com correspondência para 1.000 atuadores sendo que a ativação de um sensor provoca a ativação de um correspondente atuador. A ativação do sensor ocorre com base na comparação de um valor inteiro recebido na mensagem MQTT com um valor interno, iniciado em zero e incrementado a cada ativação. As mensagens de alteração dos sensores foram produzidas por uma aplicação externa responsável por publicar mensagens de acordo com os campos e valores esperados a cada 0,01 segundo. Cada avaliação foi feita com base na alteração dos 1.000 sensores, duas vezes cada, totalizando 2.000 ciclos.

Para avaliar o **uso de processamento e memória** utilizou-se a ferramenta de diagnóstico disponibilizada no ambiente de desenvolvimento *Microsoft Visual Studio 2019*, considerando-se o cenário com 100.000 sensores e 100% de taxa de alteração.

5. RESULTADOS

O tempo médio de execução de uma aplicação variando o percentual de sensores que têm seu estado modificado pode ser visto na Figura 3. Observa-se que as implementações em PON via *Framework* PON C++ 4.0 e sua versão distribuída apresentaram um melhor desempenho em relação ao POO quando o percentual de ativações foi baixo e um desempenho inferior para taxas de ativações mais altas. Ademais, é possível observar que o tempo de execução da versão distribuída do *Framework* PON C++ 4.0 foi ligeiramente superior ao da versão não distribuída o que pode ser considerado dentro do esperado, visto que foram adicionadas estruturas auxiliares para distribuição das entidades do PON.

Os resultados da avaliação do tempo médio de processamento das mensagens e do tempo médio de resposta do sistema são apresentados na Figura 4. Os resultados mostram que o PON em sua versão distribuída apresentou tempos de processamento e resposta menores que o POO nos dois cenários: o tempo de processamento foi 20% menor no cenário com a ativação de atuadores e 69% menor no cenário sem a ativação de atuadores; o tempo de resposta foi, em média, 16% menor. Esses resultados demonstram que o

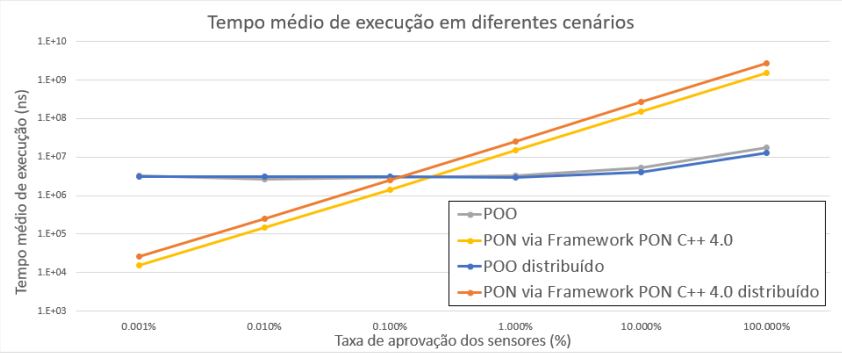
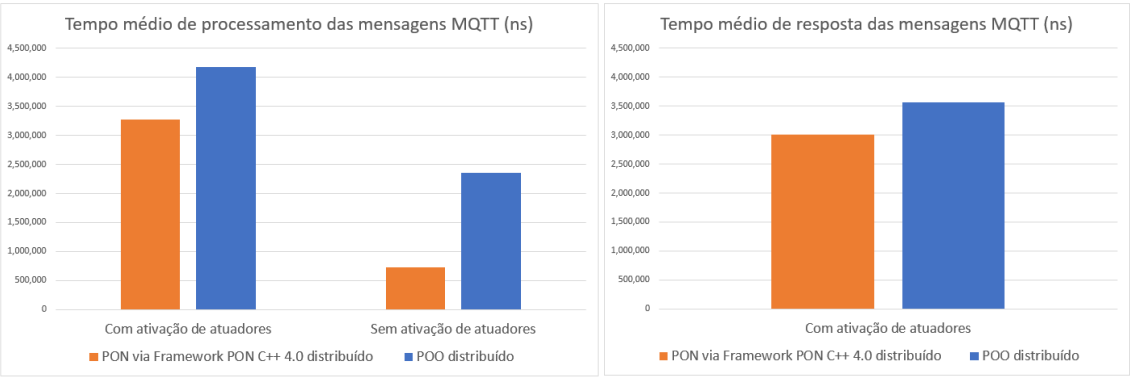


Figura 3. Tempo médio de execução.

mecanismo de notificações pontuais e concisas do PON reduz as redundâncias fazendo com que o processamento ocorra somente nas partes que tiveram seu estado alterado.



(a) Tempo médio de processamento. **(b)** Tempo médio de resposta.

Figura 4. Tempo médio de processamento e de resposta.

O uso de memória RAM da aplicação em POO e POO distribuído foi de 14,4 MB e 25,9 MB, respectivamente. Nas aplicações com PON e PON distribuído o uso máximo de memória RAM foi de 411,0 MB e 424,8 MB, respectivamente. O uso de processador em todas as implementações foi de aproximadamente 13%. Observa-se que as implementações em PON via *Framework* PON C++ 4.0 utilizaram cerca de 20 vezes mais memória RAM do que as aplicações em POO. Além disso, as implementações distribuídas utilizaram mais memória RAM que as implementações não distribuídas. Nesse contexto, o maior uso de memória RAM pelas aplicações em PON via *frameworks* pode ser considerado dentro do esperado por conta da estrutura adicional do próprio *framework*. O maior uso de memória do PON distribuído se justifica pelas estruturas adicionais necessárias para distribuição das entidades do PON.

6. CONCLUSÕES

Este trabalho apresentou, de forma inovadora, o PON em um ambiente distribuído por meio de uma arquitetura Publicação/Assinatura com o protocolo MQTT, mostrando a adaptabilidade e boa aderência do paradigma em arquiteturas desse tipo.

Os resultados mostraram que, em cenários com pouca variação dos estados do sistema, a implementação em PON via *Framework* PON C++ 4.0 é superior às implementações desenvolvidas com POO em termos de tempos de execução. Em relação ao tempo de processamento, o PON distribuído foi mais eficiente que o POO distribuído em ambos os cenários, apresentando um tempo de processamento 22% menor no cenário com ativação de atuadores e 69% menor no cenário sem a ativação de atuadores. Em relação ao tempo de resposta da aplicação, considerando a ativação dos atuadores, no PON distribuído esse tempo foi, em média, 16% melhor que o POO distribuído. Observa-se, porém, que nas aplicações em PON (distribuído e não distribuído), devido ao uso de *frameworks* (nomeadamente do *Framework* PON C++ 4.0), o uso de memória RAM é cerca de 20 vezes maior que as aplicações em POO, o que pode dificultar o seu uso em dispositivos com poucos recursos de memória disponíveis.

Como trabalhos futuros, sugere-se os aprimoramentos da tecnologia LingPON com foco na geração de código contemplando elementos distribuídos via rede e do *Framework* PON C++ 4.0 em sua versão distribuída apresentada neste artigo, com foco na otimização do uso dos recursos de memória e processamento.

Referências

MQTT Version 5.0. OASIS Standard. Disponível em: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html. Acesso em 23 de março de 2022.

Al-Masri, E., Kalyanam, K. R., Batts, J., Kim, J., Singh, S., Vo, T., and Yan, C. (2020). Investigating messaging protocols for the internet of things (iot). *IEEE Access*, 8:94880–94911.

Banaszewski, R. F. (2009). Paradigma orientado a notificações: avanços e comparações. Master's thesis. CPGEI / UTFPR. Curitiba, Brasil.

Bautsch, M. (2007). Cycles of software crises. ENISA Quarterly on Secure Software, vol. 3, no. 4, p. 3-5.

Light, R. A. (2017). Mosquitto: server and client implementation of the mqtt protocol. *Journal of Open Source Software*, 2(13):265.

Neves, F. d. S., R., L. R., and Simão, J. M. (2021). Application of generic programming for the development of a c++ framework for the notification oriented paradigm. pages 44–49. *11th International Conference on Information Society and Technology.*

Oshiro, L., Ronszcka, A., Fabro, J., and Simão, J. (2021). Linguagem e compilador para o paradigma orientado a notificações: Uma solução performante orientada a regras. In *Anais da XII Escola Regional de Alto Desempenho de São Paulo. p61–64*, Brasil. SBC.

Ronszcka, A. F. (2019). *Método para a criação de linguagens de programação e compiladores para o paradigma orientado a notificações em plataformas distintas.* PhD thesis, CPGEI / UTFPR, Curitiba.

Simão, J. M. and Stadzisz, P. C. (2008). Paradigma orientado a notificações (pon)–uma técnica de composição e execução de software orientado a notificações. PI08055181, data de depósito: 26/11/2008. Universidade Tecnológica Federal do Paraná - UTFPR.