Reengenharia de uma Arquitetura de Gerenciamento de Recursos para Agentes Utilizado Golang

Maria Alice Trinta[1], **Fabian C. B. Manoel**[1], **Carlos Eduardo Pantoja**[1]

[1]Centro Federal de Educação Tecnológica (CEFET–RJ)
20785-220 – Rio de Janeiro – RJ – Brasil

maria.trinta@aluno.cefet-rj.br,fabiancpbm@gmail.com

pantoja@cefet-rj.br

Abstract. *The Resource Management Architecture (RMA) is an architecture that allows the management of multi-agent systems in an IoT network. However, the architecture still cannot be applied on a large scale as it supports a low number of concurrent connections. Therefore, this work presents the reengineering of RMA using a Golang programming language to allow a more significant number of references.*

Resumo. *A Arquitetura de Gerenciamento de Recursos (RMA) é uma arquitetura que permite o gerenciamento de sistemas multiagente em uma rede IoT. Entretanto, a arquitetura ainda não consegue ser aplicada em larga escala pois suporta um baixo numero de conexões simultâneas. Sendo assim, este trabalho apresenta uma reengenharia da RMA utilizando a linguagem de programação Golang, com o objetivo de permitir um maior numero de conexões.*

1. Introdução

A cada dia novas tecnologias surgem afim de criar sistemas que possibilitem o compartilhamento e o acesso a informações de qualquer lugar e momento. Entretanto, muitas enfrentam problemas de escalabilidade e performance, o que pode implicar em limitações quanto ao número de conexões e lentidão. Sendo assim, para resolver estes problemas, é necessário utilizar de meios que possibilitem a otimização do sistema, para que então seja possível a aplicação destes em larga escala, como a interligação de equipamentos em grandes hospitais e hotéis. Pode ser atribuído à um sistema inteligente a capacidade de controlar de maneira autônoma os recursos e a execução de ações em diferentes domínios. Dentre as abordagens para o desenvolvimento de tais sistemas, está a utilização de agentes, que contribuem com um certo nível cognitivo a partir de uma análise de dados para tomada de decisões. Pode-se definir um Agente Inteligente como uma entidade autônoma que pode analisar um ambiente e agir sobre ele. Logo, um Sistema Multiagente (SMA) seria um grupo de agentes inteligentes, organizados para a cooperação ou competição findados a atingir certo objetivo [Wooldridge 2009].

Os sistemas inteligentes podem ser associados à tecnologias de gerenciamento, compartilhamento de informações ou *middlewares* a fim da possibilitar sua utilização em larga escala. Neste caso, a Internet das Coisas (*Internet of Things* - IoT) pode ser aplicada a fim da criação de uma rede que interligue o sistema em diferentes pontos possibilitando a cobertura de grandes áreas, pois, ao estender a internet ao mundo físico, torna

possível o gerenciamento remoto de qualquer dispositivo, o seu rastreamento através da internet, e o compartilhamento de informações [Zhang et al. 2012]. A Arquitetura de Gerenciamento de Recursos (RMA) [Pantoja et al. 2019] é uma arquitetura que possibilita organizar sistemas multiagentes em uma rede IoT afim de permitir o controle remoto e o compartilhamento de informações em larga escala. Entretanto, a RMA possui limitações quanto ao número de dispositivos em conexões simultâneas, o tempo de processamento das mensagens e no tempo de resposta de conexão. Isto faz com que o sistema gere *delays* indesejados, impossibilitando novas conexões e dificultando a ampliação do seu campo aplicação. Atualmente, a arquitetura perde performance ao empregar em torno de 50 objetos inteligentes gerenciados por SMA embarcados.

Portanto, o objetivo deste trabalho é refatorar a RMA para que o número atual de conexões suportadas aumente, assim como sua performance. Tendo em vista que a RMA foi construída utilizando a linguagem de programação Java [Byous 1998], é proposta que ao utilizar a linguagem Go [Donovan and Kernighan 2015], tais problemas podem ser minimizados, uma vez que esta linguagem pode ser compilada três vezes mais rápido que java e possui resultados melhores em testes comparativos entre as duas linguagens no que diz a respeito à performance de concorrência [Togashi and Klyuev 2014]. Com a troca de linguagens na estrutura da RMA, é necessário repensar a maneira como é estabelecida a comunicação entre a IoT e o SMA Embarcado utilizando o JaCaMo, uma vez que anteriormente utilizava-se o *middleware ContextNet* [Endler et al. 2011] que possui suporte somente para a linguagem Java e Lua. Para tal, foi escolhido o NATS, que é um *middleware* orientado a mensagens que possui suporte para as linguagens mais utilizadas incluindo Go, Java, Rubi, Python, entre outras. Além disso, o *middleware*, que possui como princípios a escalabilidade, a performance e a simplicidade, consegue enviar mais de sete milhões de mensagens por segundo [Quevedo 2018]. Além disso, o agente Comunicador responsável pela comunicação, que estava preparado para utilizar o *ContextNet*, deverá ter sua estrutura reorganizada para ser capaz de se comunicar usando o NATS.

Para comprovar o desempenho da RMA-GO, um cenário de testes foi criado para analisar o desempenho do envio de uma mensagem, desde o emissor até o banco de dados que armazena informações dos recursos virtualizados. Os testes foram realizados com múltiplos dispositivos simulados, afim de testar também a capacidade de conexões simultâneas.

Este trabalho está organizado da seguinte forma: na seção 2 é apresentado o referencial teórico, em seguida, na seção 3 é apresentada a metodologia para a construção da RMA-GO. A avaliação experimental será apresentada na seção 4. Em seguida, serão discutidos alguns trabalhos relacionados e por fim, a conclusão.

2. Referencial Teórico

Nesta seção serão apresentados os referenciais teóricos necessários para um entendimento acerca das tecnologias utilizadas durante o desenvolvimento do trabalho. A primeira delas é a Internet das Coisas (*Internet of Things* - IoT) [Bandyopadhyay et al. 2011] que pode ser definida como uma rede que interliga objetos físicos, possibilitando o controle e o compartilhamento de informações a distância. Para estabelecer esta rede é necessário utilizar um *middleware* para IoT, uma vez que este possibilita a comunicação entre meios heterogêneos, nesse caso, entre camadas diferentes em uma mesma arquitetura.

Ambos NATS e *ContextNet* são *middlewares* para IoT que possibilitam a comunicação à distância, porém o *ContextNet* possui suporte apenas para Java e Lua, enquanto o NATS pode ser implementado através de diferentes linguagens de programação. Além disso, o NATS por ser um *middleware* orientado a mensagens, permite a comunicação assíncrona pois possui um fraco acoplamento e permite um número reduzido de conexões [Nilsson and Pregén 2020]. A alteração do middleware vem como uma alternativa para permitir que o processamento dos pedidos de conexão, reconexão, de compartilhamento de informações e de ações a serem executadas em hardware sejam feitas de forma mais eficiente.

A RMA é responsável pela virtualização de componentes para serem consumidos e acessados por clientes de forma remota em uma rede IoT. A RMA é composta de três camadas que se interconectam através da troca de mensagens. Em uma visão *top-down*, na camada de clientes estão localizadas as aplicações que consomem de forma remota as informações de recursos provenientes de dispositivos conectados e virtualizados na Camada de Gerenciamento de Recursos (*Resource Management Layer* - RML), além de ser responsável por enviar ações para serem executadas por algum recurso de dispositivo (um atuador). A RML tem a responsabilidade de gerenciar os pedidos de conexão, o processamento e o armazenamento das informações dos recursos de dispositivos em determinado ambiente configurado pelo usuário. Na camada de dispositivos, encontram-se todos os dispositivos que possuem um sistema embarcado e recursos (sensores e atuadores) para sensoriamento e atuação em determinado ambiente real. Tais dispositivos inteligentes podem empregar SMA embarcados utilizando o JaCaMo [Boissier et al. 2013].

Compreende-se também que um Agente Inteligente é uma entidade física ou virtual capaz de deliberar e tomar decisões que influenciem ou não sobre o ambiente em que estejam inseridos. Assim, um conjunto de agentes é denominado um SMA, onde estes podem estar organizados para atuar e exercer influência sobre um ambiente simulado ou físico para atingir determinado objetivo. Neste trabalho, dispositivos inteligentes utilizam um SMA embarcado para permitir o controle de sensores e atuadores conectados a um microcontrolador, que possa se conectar em uma rede IoT onde a RML esteja disponível, e que possa ser ao mesmo tempo autônomo e possa receber pedidos de ações enviados por clientes que podem ser atendidos ou não (dependendo da deliberação através do conjunto de planos disponibilizado pelo projetista do dispositivo).

Este SMA embarcado traz consigo todo o necessário para a execução de tarefas autônomas, uma vez que pode, através de sensores e atuadores, recolher dados sobre o ambiente, analisá-los e devolver uma resposta por meio de uma atuação sem depender de enviar informações para um servidor ou aplicação centralizadora. Desta forma, é possível aumentar o nível de autonomia e cognição na borda de sistemas (*Edge Computing*) [Khan et al. 2019] através da utilização de agentes que adotam o modelo *Belief-Desire-Intention* (BDI).

3. RMA-GO

Nesta seção será apresentada a metodologia utilizada durante o desenvolvimento da RMA-GO, descrevendo como estão organizadas as camadas após a reformulação, explicando o que está sendo utilizado e como manter a integração entre elas. A começar pela camada de cliente, em seguida a camada de gerenciamento de recursos e, por último,

a camada de dispositivos, que interagem entre si através da troca de mensagens e formam a Arquitetura de Gerenciamento de Recursos, como pode-se ver através da Figura 1.

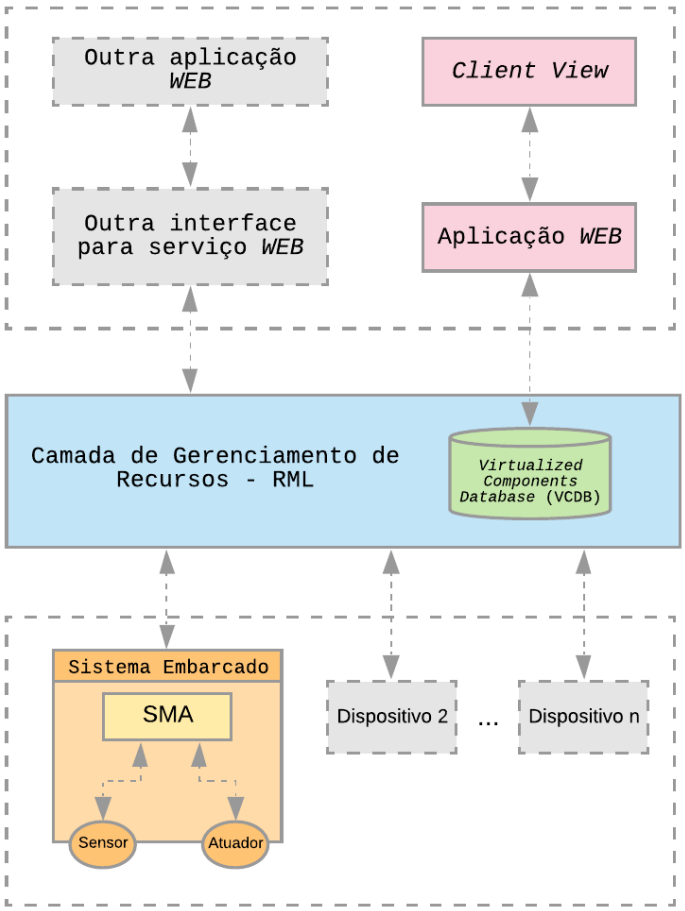

Figura 1. Arquitetura de Gerenciamento de Recursos

3.1. Camada de cliente

Na Camada de cliente estão todas as aplicações capazes de consumir dados dos dispositivos e exibi-los aos usuário além de permitir uma interface para envio de comandos para serem executados nos atuadores dos dispositivos. As possibilidades de implementação nesta camada podem ser desde aplicativos móveis à aplicações *WEB*, utilizando qualquer diferentes tipos de tecnologias — incluindo SMA — necessitando somente fazer com que tal aplicação interaja com a Camada de Gerenciamento de Recursos, que é responsável pelo tratamento dos dados que trafegam entre as camadas dos dispositivos e de cliente.

Nesta arquitetura, uma aplicação *WEB* consume os dados gerados pela camada de dispositivos e também é capaz de enviar comandos aos dispositivos. Na Figura 2 é possível ver a organização da arquitetura. Nota-se que a interação com a RML se dá através do banco de dados (*Virtualized Components Database* - VCDB) utilizado pela RML, refatorada em Golang, para armazenar os dados que fluem pela arquitetura. O

VCDB foi implementado utilizando o MongoDB, um banco de dados orientado a documentos que utiliza da formatação JSON. Dessa forma, a aplicação *WEB*, também desenvolvida utilizando Golang, consome os dados armazenados no VCDB, e os disponibiliza ao usuário através da *view* que foi implementada utilizando HTML e Bootstrap.

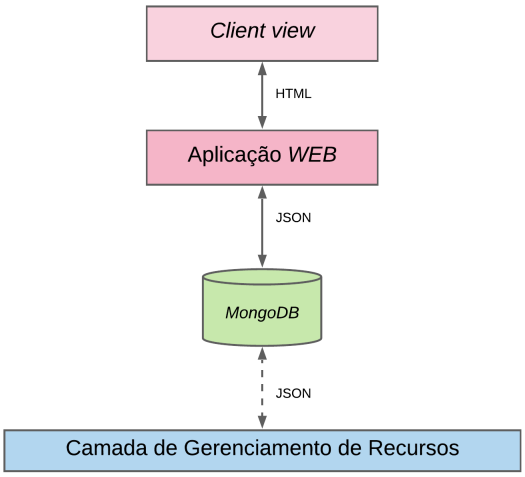

Figura 2. Arquitetura de Gerenciamento de Recursos

3.2. Camada de Gerenciamento de recursos

A Camada de Gerenciamento de Recursos é a camada intermediária responsável por gerenciar os dados gerados e consumidos dentro da arquitetura. A RML é responsável por cadastrar os dispositivos, definir seu número de identificação, controlar seu *status* indicando se este está ativo ou inativo, receber todos os dados que os dispositivos conectados enviam e armazená-los no VCDB, e por fim, designar a determinado dispositivo as ordens que o cliente envia através das aplicações, seja para desligar ou ligar um atuador ou o próprio dispositivo. A RML recebe dos dispositivos que querem se cadastrar um pedido de conexão e as informações para o cadastro e, em ambos os casos, responde com o status de conectado, caso tudo tenha ocorrido dentro do esperado. Após a conexão do dispositivo, este dispositivo envia os dados de seus recursos (e.g., dados dos sensores) para serem armazenados no VCDB e posteriormente consumidos por aplicações clientes. Caso existam ações para serem executadas, a RML repassa tais ações para os dispositivos específicos. Para exemplificar, a Figura 3 demonstra a RML desempenhando suas diversas funções.

Nesta versão refatorada, a RML foi implementada utilizando a linguagem Golang, para estabelecer a conectividade entre as camadas foi utilizado o NATS, um *middleware* de comunicação que funciona a partir de um aplicação servidora em conjunto da implementação de suas funções em código da aplicação específica (cliente NATS). O NATS é um *middleware* orientado a mensagem, que dentre suas funcionalidades, possibilita a criação de canais para a troca de mensagens através do modelo *publish-subscribe*, e também onde uma mensagem enviada possa ser respondida, através do modelo denominado *reply-response*. Desse modo, foram implementados canais de comunicação para que a RML compreenda que tipo de dado está chegando.

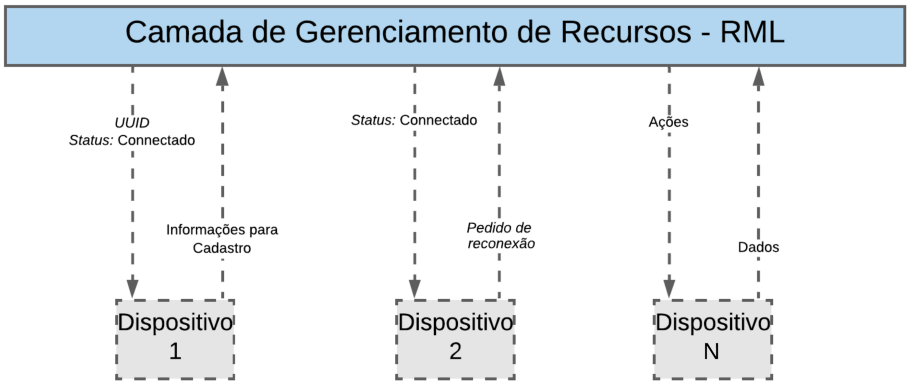

Figura 3. Exemplo de funcionamento da RML

3.3. Camada de dispositivos

A camada de dispositivos representa os possíveis dispositivos e suas tecnologias inseridos em um ambiente real que podem se conectar a RML para deixarem seus recursos virtualizáveis às aplicações clientes. Um dispositivo usado na arquitetura da RMA é composto de sensores e atuadores ligados a um controlador e gerenciado por um sistema embarcado. Neste trabalho, um dispositivo é embarcado com um SMA, que é capaz de interagir com o ambiente podendo obter percepções, processa-las e enviar tais informações e conclusões para RML ou então, executar uma ação na ponta de um sistema (*edge computing*), sem necessidade de encaminhar tais dados para tomada de decisão em outro local.

O *framework* adotado para a programação do SMA foi o JaCaMo. Para a integração com o hardware e com o *middleware* é necessário empregar os agentes Argo [Pantoja et al. 2016], que recolhe as informações do ambiente, e o agente Comunicador [Pantoja et al. 2018], que foi adaptado neste trabalho para ser um instância cliente do NATS para realizar a comunicação com a RML refatorada em Golang. Para isso, o Agente Comunicador possui duas ações internas adaptadas para realizar os modelos de comunicação *publish-subscribe* e *reply-response* utilizando o NATS com o objetivo de se conectar e enviar as informações para a RML: *connectToRml* e a *sendToRML*.

4. Avaliação experimental

Para avaliar a performance de conexão da RMA-GO, os mesmos testes realizados para avaliar a performance da RMA [Pantoja et al. 2019] em Java foram replicados. Para isso, foram criados 50 dispositivos simulados que enviam um pedido de conexão a RML e, uma vez conectados, começam a enviar mensagens com latência de 1 segundo. Então, será medido o tempo de conexão desde o pedido da requisição até a recebimento da confirmação para avaliar se a RML tem uma performance melhor que anterior conforme dispositivos são conectados e começam a enviar informações. Os resultados dos testes podem ser vistos na Figura 4.

Na RMA Java, conforme a quantidade de dispositivos aumenta, o fluxo de dados cresce na mesma proporção e portanto, o tempo de conexão cresce. Porém, na RMA-GO, pode-se concluir que cerca de 50 dispositivos ainda não influenciam negativamente

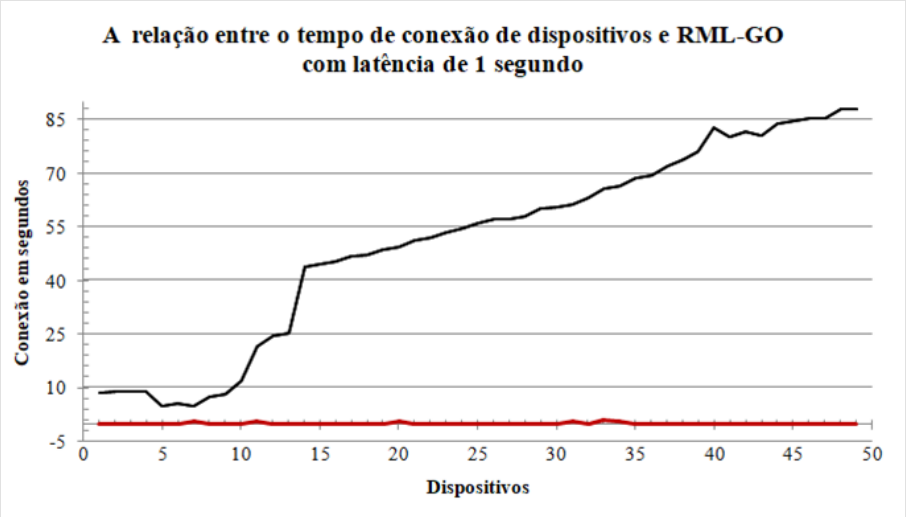

Figura 4. Teste de performance do tempo de conexão da RML-GO.

no tempo de conexão de novos IoT Objects. Este resultado ajuda a reduzir uma das preocupações em implementações de SMA em uma rede IoT. Mesmo assim, ainda é necessário avaliar o tempo de conexões em um cenário onde a quantidade de dispositivos conectados é maior do que 100.

5. Trabalhos Relacionados

Diversos trabalhos focam no desenvolvimento de plataformas de gerenciamento de recursos para IoT integrando SMA ou agentes. Muitas dessas plataformas focam na centralização de agentes [Chaouche et al. 2014][Zschörnig et al. 2019] ou em soluções ad-hoc [Sánchez-Pi et al. 2010][Villarrubia et al. 2014], o que de fato não demonstra o potencial da capacidade distributiva de agentes e também limitam ou dificultam a utilização em domínios diferentes. Além disso, existe uma lacuna nos testes e validação em larga escala de tais soluções. A RMA inicial [Pantoja et al. 2019] demonstra alguns resultados iniciais com uma quantidade de dispositivos que demonstram que sua utilização se limitam a poucos dispositivos por solução (entre 50 e 100), o que pode limitar a sua utilização em grandes escalas e em soluções maiores. Apesar da RMA ser genérica considerando o domínio e ser capaz de ser replicada de forma individual, a intercomunicação entre essas réplicas não seria possível, tendo assim que uma aplicação cliente estar conectada a diversos servidores. Neste trabalho, foi adotada a RMA refatorada para a virtualização de dispositivos embarcados com SMA visando a ampliação da capacidade de processamento de mensagens e conexões de dispositivos como uma indicação de que seja possível unificar diversas soluções em um único servidor ou então permitir o aumento da capacidade de dispositivos empregados em uma única solução.

6. Conclusão

Este trabalho apresentou uma refatoração da RMA para virtualização de recursos de dispositivos embarcados com SMA. Os recursos dos dispositivos são consumidos por aplicações clientes, que acessam seus dados através da RML. Para permitir que mais números de conexões e de mensagens processadas pudessem ser enviadas e tratadas, a RML foi refatorada utilizando a Golang, por ser uma linguagem com menos custo de

processamento que a anterior utilizada, o Java. Dessa forma, também foi modificada o *middleware* utilizado na comunicação de dados entre as camadas para o NATS. Testes iniciais de performance indicaram que a refatoração demonstrou resultados satisfatórios e promissores.

Apesar dos testes iniciais, a RMA-GO ainda precisa de ajustes e de uma bateria de testes mais apropriados que estressem todos os pontos possíveis da arquitetura. Além disso, também é necessário aplicar em um domínio ou estudo de caso para garantir que o sistema como um todo responderá adequadamente. Portanto, como trabalhos futuros, a RMA-GO será aplicada em um cenário mais específico utilizando diversos dispositivos com SMA embarcados onde estes devem se organizar para coletivamente gerenciarem um ambiente baseado no que perceberem, e lidar com as ações que as aplicações clientes precisam que seja executadas em ambiente e que sejam conflitantes com as intenções dos agentes.

Referências

Bandyopadhyay, S., Sengupta, M., Maiti, S., and Dutta, S. (2011). A survey of middleware for internet of things. In *Recent trends in wireless and mobile networks*, pages 288–296. Springer.

Boissier, O., Bordini, R. H., Hübner, J. F., Ricci, A., and Santi, A. (2013). Multi-agent oriented programming with JaCaMo. *Science of Computer Programming*, 78(6):747–761.

Byous, J. (1998). Java technology: an early history. *URL: http://java. sun. com/features/1998/05/birthday. html, Artigo pesquisado em 07 de Junho de 2002*.

Chaouche, A.-C., Seghrouchni, A. E. F., Ilié, J.-M., and Saïdouni, D. E. (2014). A higher-order agent model with contextual planning management for ambient systems. In *Transactions on Computational Collective Intelligence XVI*, pages 146–169. Springer.

Donovan, A. A. and Kernighan, B. W. (2015). *The Go programming language*. Addison-Wesley Professional.

Endler, M., Baptista, G., Silva, L., Vasconcelos, R., Malcher, M., Pantoja, V., Pinheiro, V., and Viterbo, J. (2011). Contextnet: context reasoning and sharing middleware for large-scale pervasive collaboration and social networking. In *Proceedings of the Workshop on Posters and Demos Track*, page 2. ACM.

Khan, W. Z., Ahmed, E., Hakak, S., Yaqoob, I., and Ahmed, A. (2019). Edge computing: A survey. *Future Generation Computer Systems*, 97:219–235.

Nilsson, E. and Pregén, V. (2020). Performance evaluation of message-oriented middleware.

Pantoja, C. E., Jesus, V. S., Manoel, F. C. P. B., and Viterbo, J. (2018). A heterogeneous architecture for integrating multi-agent systems in ami systems. *The Thirtieth International Conference on Software Engineering and Knowledge Engineering (SEKE 2018)*.

Pantoja, C. E., Soares, H. D., Viterbo, J., Alexandre, T., Seghrouchni, A. E.-F., and Casals, A. (2019). Exposing iot objects in the internet using the resource management architecture. *International Journal of Software Engineering and Knowledge Engineering*, 29(11n12):1703–1725.

Pantoja, C. E., Stabile, M. F., Lazarin, N. M., and Sichman, J. S. (2016). Argo: An extended jason architecture that facilitates embedded robotic agents programming. In *International Workshop on Engineering Multi-Agent Systems*, pages 136–155. Springer.

Quevedo, W. (2018). Introduction to nats. In *Practical NATS*, pages 1–18. Springer.

Sánchez-Pi, N., Mangina, E., Carbó, J., and Molina, J. M. (2010). *Multi-agent System (MAS) Applications in Ambient Intelligence (AmI) Environments*, pages 493–500. Springer Berlin Heidelberg, Berlin, Heidelberg.

Togashi, N. and Klyuev, V. (2014). Concurrency in go and java: performance analysis. In *2014 4th IEEE International Conference on Information Science and Technology*, pages 213–216. IEEE.

Villarrubia, G., De Paz, J. F., Bajo, J., and Corchado, J. M. (2014). Ambient agents: Embedded agents for remote control and monitoring using the PANGEA platform. *Sensors*, 14(8):13955–13979.

Wooldridge, M. (2009). *An Introduction to Multi-Agent Systems*. Wiley.

Zhang, D., Ning, H., Xu, K. S., Lin, F., and Yang, L. T. (2012). Internet of things j. ucs special issue. *Journal of Universal Computer Science*, 18(9):1069–1071.

Zschörnig, T., Wehlitz, R., and Franczyk, B. (2019). A fog-enabled smart home analytics platform. In *ICEIS (1)*, pages 616–622.