CEFET/RJ - CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA
CELSO SUCKOW DA FONSECA

Comunicação entre SMA Embarcados: Uma Arquitetura Baseada em Protocolos da Camada de Aplicação

Joao Pedro Bernardo de Souza

Prof. Orientador:
Carlos Eduardo Pantoja, D.Sc.
Nilson Mori Lazarin, M.Sc.

Rio de Janeiro,
Agosto de 2023

CEFET/RJ - CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA
CELSO SUCKOW DA FONSECA

Comunicação entre SMA Embarcados: Uma Arquitetura Baseada em Protocolos da Camada de Aplicação

Joao Pedro Bernardo de Souza

Projeto final apresentado em cumprimento às normas do Departamento de Educação Superior do Centro Federal de Educação Tecnológica Celso Suckow da Fonseca, CEFET/RJ, como parte dos requisitos para obtenção do título de Bacharel em Sistemas de Informação.

Prof. Orientador:
Carlos Eduardo Pantoja, D.Sc.
Nilson Mori Lazarin, M.Sc.

Rio de Janeiro,
Agosto de 2023

JOÃO PEDRO BERNARDO DE SOUZA

COMUNICAÇÃO ENTRE SISTEMAS MULTIAGENTES EMBARCADOS: UMA ARQUITETURA BASEADA EM PROTOCOLOS DA CAMADA DE ALPICAÇÃO

Trabalho de Conclusão de Curso, apresentado ao Centro Federal de Educação Tecnológica Celso Suckow da Fonseca, como parte das exigências para a obtenção do título de Bacharel em Sistemas de Informação.

Rio de Janeiro, 8 de agosto de 2023.

BANCA EXAMINADORA

_____
Prof. D.Sc. Carlos Eduardo Pantoja
Centro Federal de Educação Tecnológica (CEFET/RJ)

_____
Prof. M.Sc. Nilson Mori Lazarin
Centro Federal de Educação Tecnológica (CEFET/RJ)

_____
Prof. D.Sc. Ivan Miguel Serrano Pires
Instituto Politécnico de Santarém

_____
M.Sc. Tielle da Silva Alexandre
Universidade Federal Fluminense

_____
Prof. D.Sc. Leandro Marques Samyn
Centro Federal de Educação Tecnológica (CEFET/RJ)

AGRADECIMENTOS

Gostaria de expressar minha profunda gratidão a todos que contribuíram para a realização deste trabalho de conclusão de curso. Sem o apoio, incentivo e colaboração de diversas pessoas, este projeto não teria sido possível.

Agradeço primeiramente à minha família, que esteve ao meu lado em todos os momentos, oferecendo amor, compreensão e apoio incondicional. Vocês foram minha fonte de força e motivação durante toda a jornada acadêmica.

Ao meu orientador Carlos Eduardo Pantoja, pelos conhecimentos transmitidos durante toda a graduação, e pelo apoio, dedicação e incentivo em todas as etapas do desenvolvimento deste trabalho. Ao coorientador, Nilson Mori, pela disponibilização de materiais, sugestões, correções e as inúmeras reuniões online, todas elas foram fundamentais para aprimorar o conteúdo deste TCC.

Aos meus amigos de turma, em especial, Antônio, Cristian, Douglas, João Vitor, Leonardo, Maycon, Miriam e Pedro Henrique pelos momentos de descontração, estudos em grupo e encorajamento, principalmente nos momentos mais difíceis do período da pandemia. Nossas trocas de experiências foram enriquecedoras e fundamentais para meu crescimento pessoal e acadêmico.

Por fim, mas não menos importante, gostaria de deixar os agradecimentos aos meus amigos, Clara, Kimberly, Raphael, Nathália, Bruno, Laura, Larissa e Thayná, todos vocês foram uma parte inestimável da minha jornada acadêmica e me apoiaram durante todo o processo de elaboração deste trabalho de conclusão de curso. Cada um de vocês deixou uma marca em minha vida e, sem a presença de vocês, esta conquista não estaria completa.

RESUMO

Sistemas Multiagentes (SMA) são grupos de agentes autônomos que atuam no mesmo ambiente, competindo ou colaborando para alcançarem objetivos individuais ou coletivos. Tais sistemas podem ser integrados em dispositivos embarcados para realizar tarefas que controlam hardware. Alguns dos principais desafios encontrados na comunicação entre SMA Embarcados é a não orientação à conexão, utilizada por alguns *middlewares* IoT para comunicação de agentes e falta de segurança na transferência de mensagens pela rede. Portanto, é fundamental utilizar meios de comunicação eficientes, seguros e confiáveis. Esse trabalho pretende propor uma arquitetura de comunicação orientada à conexão, utilizando canais seguros proporcionados pelos protocolos de rede na camada de aplicação. Para isso, serão explorados os protocolos POP3, SMTP e IMAP. Cada um deles possui características específicas que podem ser adequadas para diferentes cenários de comunicação entre SMA. Este trabalho contribui para o avanço do campo de sistemas multiagentes ao abordar a integração dos protocolos de e-mail e buscar o desenvolvimento de uma arquitetura de agente customizada do *framework* Jason para viabilizar uma comunicação segura e simplificada.

Palavras-chaves: Sistemas Multiagente; Comunicação; SMA Aberto

ABSTRACT

Multi-Agent Systems (MAS) are groups of autonomous agents that act in the same environment, competing or collaborating to achieve individual or collective goals. Such systems can be integrated into embedded devices to perform tasks that control hardware. Some of the main challenges encountered in communication between Embedded MAS are the connectionless protrocols used by some IoT *middlewares* for agent communication and the lack of security in the transfer of messages over the network. Therefore, it is essential to use efficient, safe and reliable means of communication. This work intends to propose a connection-oriented communication architecture, using secure channels provided by network protocols in the application layer. Protocols such as POP3 , SMTP and IMAP will be explored in this paper. Each of them has specific characteristics that may be suitable for different MAS communication scenarios. This work contributes to the advancement of the field of multi-agent systems by addressing the integration of e-mail protocols and seeking the development of a customized agent architecture based on the Jason *framework* to enable secure and simplified communication.

Keywords: Multi-agent Systems; Communication; Open MAS;

Sumário

Lista de Figuras

LISTA DE ABREVIAÇÕES

BDI	*Belief-Desire-Intention*	1
DDS	*Data Distribution Service*	8
IMAP	*Internet Message Access Protocol*	8
IOT	*Internet of Things*	2
KQML	*Knowledge Query and Manipulation Language*	2
MTA	*Mail Transfer Agent*	8
OSI	*Open Systems Interconnection*	7
POP3	*Post Office Protocol*	8
SMA	Sistemas Multiagente	1
SMTP	*Simple Mail Transfer Protocol*	8
SO	*Sistema Operacional*	11
SUMO	*Simulation of Urban MObility*	4
TCP	*Transmission Control Protocol*	7
UDP	*User Datagram Protocol*	1

Capítulo 1

Introdução

Agente é um sistema computacional capaz de perceber o ambiente em que está situado, e, através de sua própria deliberação, baseada em suas convicções e motivações, ele pode atuar nesse ambiente para atingir um propósito. Wooldridge [2009] define um Sistemas Multiagente (SMA) como um conjunto de elementos computacionais, conhecidos como agentes, que interagem entre si. Esses agentes são capazes de tomar decisões autônomas para satisfazer seus objetivos e também são capazes de interagir com outros agentes de forma análoga às interações sociais do dia a dia: cooperando, coordenando, negociando e afins.

Entre os modelos e *frameworks* de desenvolvimento de SMA conhecidos [Bratman, 2009; Bordini et al., 2007], grande maioria utiliza como padrão de arquitetura cognitiva o modelo *Belief-Desire-Intention* (BDI) [Rao and Georgeff, 1991]. Ele propõe uma estrutura para representar a mente de agentes inteligentes, permitindo que eles tomem decisões racionais e ajam de acordo com seus objetivos e conhecimento do ambiente.

No modelo BDI, as crenças representam o conhecimento e a percepção do agente sobre o ambiente. Os desejos são as metas ou objetivos que o agente deseja alcançar. E as intenções são as ações específicas que o agente escolhe realizar com o objetivo de alcançar seus desejos. Um sistema embarcado é um sistema computacional completo embutidos em um dispositivo ou componente físico, permitindo controle e monitoramento do hardware e realização de atividades no ambiente. Ao adicionar inteligencia a esses sistemas embarcados por meio dos SMA, temos dispositivos capazes de tomar decisões autônomas, objetivas e proativas.[Brandão et al., 2021]

1.1 Motivação

Na área de SMA Embarcados, diversos trabalhos utilizam o *middleware* ContextNet [Endler et al., 2011] para realizar a troca de informação entre sistemas distintos [Brandão et al., 2021; de Jesus et al., 2019]. Ele utiliza o protocolo *User Datagram Protocol* (UDP) [Postel, 1980] para transferência de dados, o que não garante a entrega e a privacidade das mensagens [Endler and e Silva, 2018]. Sistemas embarcados também tem limitações de hardware que impossibilitam a

dedicação de muitos recursos para garantir a integridade comunicação. Essa dificuldade pode afetar a eficiência e a confiabilidade da troca de informação entre SMA, o que demanda então uma estratégia de otimização da comunicação entre eles.

O envio de e-mails é uma possibilidade de comunicação dentro de um SMA, entretanto, ele não apresenta o uso de forças ilocucionárias. Essas forças direcionam a comunicação entre os agentes BDI e a tomada de decisões dos agentes. O desenvolvimento de uma arquitetura de comunicação dentro dos padrões de comunicação dos agentes e utilizando os protocolos de rede pode ser uma alternativa aos *middlewares Internet of Things* (IoT) utilizados em outros sistemas.

1.2 Objetivo

Esse trabalho apresenta uma proposta de arquitetura de comunicação entre SMA baseada em troca de e-mails. Para isso serão analisadas as tecnologias de comunicação disponíveis a fim de explorar cenários de troca de mensagens entre SMA, demonstrando sua viabilidade em comparação com outros sistemas.

Para este fim, será desenvolvida uma arquitetura customizável do Jason, incluindo adição de métodos e ações internas que possibilitem a troca de informação via e-mail entre agentes ou diferentes sistemas, respeitando os formatos de mensagem já utilizados pelo *framework*, e utilizando o *Knowledge Query and Manipulation Language* (KQML) [Finin et al., 1994] como padrão comunicação.

Na implantação do serviço de envio de e-mail, serão utilizadas tecnologias Open Source como o sistema operacional Debian GNU/Linux[1], e os pacotes Dovecot[2] e Postfix[3]. Serão realizados testes de envio e recebimento de e-mails através do software Thunderbird e classes de teste em Java para garantir que os serviços estejam operacionais e que as configurações estejam corretas. Será utilizado o método de pesquisa exploratória, com o objetivo de conhecer novas tecnologias de SMA com a finalidade de desenvolver uma alternativa ao *middleware* ContextNet.

[1] `https://www.debian.org/`
[2] `https://www.dovecot.org/`
[3] `https://www.postfix.org/`

1.3 Contribuições

É esperado que o trabalho contribua na área de SMA e SMA Embarcados, ao trazer uma nova possibilidade de comunicação no desenvolvimento desses sistemas. Ao trazer um canal seguro de comunicação através dos protocolos da camada de aplicação, ganha-se em privacidade e proteção dos dados trafegados no sistema, além da integridade de usar um protocolo orientado a conexão.

A comunicação assíncrona também pode trazer vantagens ao possibilitar que cada sistema processe dados ou gere respostas em seu próprio ritmo, sem a necessidade de estarem simultaneamente conectado. Também ganha-se na tolerância a falhas, onde problemas de conectividade não comprometem imediatamente todo o sistema.

1.4 Estrutura do trabalho

O trabalho está estruturado da seguinte forma: na seção 2, são discutidos os trabalhos relacionados; a seção 3, inicia-se com a apresentação do referencial teórico; em seguida, na seção 4, é discorrido sobre a arquitetura customizada do *framework* Jason e sua implementação; na seção 5, uma prova de conceito é demonstrada, e, por fim, na seção 6 está a conclusão.

Capítulo 2

Trabalhos Relacionados

Alguns trabalhos da literatura utilizam o JaCaMo ou umas das suas ferramentas como o Jason, CArtAgO ou Moise para comunicação usando o ContextNet. Em de Jesus et al. [2019], é apresentado um protoloco de comunicação inspirado em relações biológicas utilizando o *middleware* ContexNet com o objetivo de preservar a integridade e os conhecimentos do SMA em caso de falha de hardware transferindo todos os agentes para outro SMA conhecido.

Souza de Castro et al. [2022] discutem a comunicação entre SMA Embarcados através da integração do SMA Embarcado com *Simulation of Urban MObility* (SUMO). Esse artigo usa diversas ferramentas IoT, incluindo o ContextNet, onde podem ser encontrados os mesmos problemas discutidos neste trabalho: dependência de software proprietário, dificuldade de comunicação dependendo da infraestrutura, além da conexão não criptografada.

Além disso, Lazarin et al. [2021] demonstram um protocolo para comunicação entre SMAs embarcados por meio da tecnologia de rádio frequência (RF). Essa é uma proposta de comunicação secundária, não orientada a conexão, possibilitando a comunicação em uma rede de dispositivos RF. Ele se assemelha com essa proposta pois também desenvolve uma customização do Jason para resolver um problema de comunicação.

Capítulo 3

Fundamentação Teórica

Nesta sessão são apresentados os conceitos básicos para a compreensão dos objetivos do trabalho. Primeiramente, são apresentados os fundamentos de agente, sistemas multiagentes, SMA Embarcados, BDI, KQML, e o que são forças ilocucionárias. Em seguida, serão apresentados os conceitos de redes como modelo ISO/OSI e os protocolos de camada de aplicação. Por ultimo, apresentamos o *middleware* ContextNet e quais as suas vantagens e desvantagens.

3.1 Agentes, SMA e SMA Embarcados

Agentes são entidades inteligentes que são capazes de perceber um ambiente onde estão situados e, após um ciclo de raciocínio, são capazes de agir sobre o mesmo ambiente para alcançar objetivos comuns ou conflitantes. Os agentes são entidades autônomas capazes de aprender com suas experiências passadas, interações com outros agentes e percepções do ambiente. E também são capazes de reagir de forma proativa às mudanças dentro do ambiente em que estão situados [Wooldridge, 2000].

Quando um conjunto de agentes autônomos interagem e/ou colaboram para atingir objetivos específicos, onde cada um possui própria capacidade de percepção, tomada de decisão e ação, permitindo-lhe agir de forma independente e cooperativa, temos um Sistema Multiagente. Esses sistemas são projetados para lidar com problemas complexos e distribuídos, onde a distribuição de tarefas, a cooperação entre os agentes e a troca de informações são essenciais para alcançar soluções eficientes [Wooldridge, 2009].

Um SMA pode ser embutido em um dispositivo físico para realizar atividades em um ambiente onde é necessário controle e monitoramento de hardware e também para se comunicar com outros SMA embarcados. Existem diversas ferramentas e *frameworks* que viabilizam o desenvolvimento de SMA embarcados, entre elas o JaCaMo e o Javino.

O JaCaMo [Boissier et al., 2013] é uma ferramenta que une três tecnologias já conhecidas na área de desenvolvimento de SMA: Jason [Bordini et al., 2007], CArtAgO [Ricci et al., 2007] e Moise [Hannoun et al., 2000; Hübner et al., 2007]. O Jason é um *framework* de programação

orientada a agentes que utiliza a linguagem AgentSpeak como base para desenvolver uma sintaxe e semântica que permite programar as crenças, planos, objetivos e intenções de um agente; CArtAgO é uma linguagem de programação de ambientes que permite representar o ambiente de trabalho de um ou mais grupos de agentes. Por fim, o Moise permite programar regras e papéis que alinhem os direitos e deveres do agente dentro do grupo no qual estão inseridos, estabelecendo uma estrutura organizacional para a coordenação e colaboração eficazes entre os agentes.

O Javino [Lazarin and Pantoja, 2015] é um *middleware* que faz a ligação entre agentes e hardware, seu nome vem da junção de Java com Arduíno, ele que viabiliza a comunicação serial entre esse microcontrolador e os agentes.

Sistemas baseados em Crenças, Desejos e Intenções são uma abordagem amplamente utilizada no desenvolvimento de Sistemas Multiagentes (SMA) [Bordini et al., 2007]. A arquitetura BDI é uma estrutura cognitiva que modela a tomada de decisão de agentes autônomos, inspirada pela forma como seres humanos agem e interagem em ambientes complexos. Ela se baseia em três elementos fundamentais: crenças, que representam o conhecimento que um agente possui sobre o mundo; desejos, que são as metas e objetivos que o agente busca alcançar; e intenções, que são as ações planejadas para realizar esses desejos [Bratman, 2009].

A arquitetura BDI permite que os agentes processem informações, avaliem o estado do ambiente, deliberem sobre possíveis ações e selecionem a ação mais apropriada com base em suas crenças, desejos e intenções. O raciocínio prático de agentes BDI é implementado através de ciclos de raciocínio, cujo objetivo é manter o estado mental atualizado. Como consequência deste mecanismo, as principais características dos agentes são autonomia, proatividade, reatividade e habilidades sociais

Os agentes BDI se comunicam por KQML, essa é uma linguagem de comunicação baseada em troca de mensagens utilizada para facilitar a comunicação entre agentes em sistemas multiagentes. Ela foi desenvolvida para permitir que agentes autônomos troquem informações, façam consultas e realizem ações cooperativas em um ambiente distribuído. O KQML define uma sintaxe e um conjunto de protocolos para a troca de mensagens, permitindo que os agentes compartilhem conhecimento e coordenem suas atividades por meio de um canal de comunicação padronizado [Finin et al., 1992].

No KQML, as forças ilocucionárias referem-se aos tipos de ações ou intenções comunicativas que podem ser expressas nas mensagens trocadas entre os agentes. O KQML define um

conjunto de forças ilocucionárias para descrever as intenções subjacentes às mensagens, permitindo que os agentes comuniquem suas necessidades, solicitações, afirmações, perguntas, entre outros, de forma clara e explícita, facilitando a compreensão e a interpretação das mensagens trocadas. Alguns exemplos abordados neste trabalho são:

- **tell**: Usado para transmitir uma crença.
- **achieve**: Usado para transmitir uma intenção.
- **tellHow**: Usado para enviar um plano a outro agente.
- **askHow**: Usado para solicitar um plano a outro agente.
- **askOne**: Usado para solicitar uma crença a outro agente.

Os agentes dentro de um SMA utilizam dessas forças ilocucionárias para trocar mensagens entre si e atingirem seu objetivos. A arquitetura de troca de e-mail proposta nesse trabalho, avança a fronteira da comunicação interna de um único SMA e possibilita a troca de mensagens entre SMA distintos.

3.2 Redes de computadores

No que diz respeitos à redes de computadores, este trabalho utiliza algumas definições do modelo *Open Systems Interconnection* (OSI). Ele é um modelo de referência que descreve como os diferentes sistemas de computador se comunicam e interagem em uma rede de computadores. O OSI é dividido em sete camadas distintas, cada uma responsável por funções específicas relacionadas à comunicação. Nesse trabalho serão destacadas as camadas 4 (Transporte) e 7 (Aplicação).[Kurose and Ross, 2005]

A camada de transporte é responsável por garantir a transferência confiável e eficiente de dados entre os sistemas finais em uma rede de computadores. Nessa camada, são encontrados os protocolos de transporte, que podem ser classificados como orientados a conexão ou não orientados a conexão.

Os protocolos orientados a conexão, como o *Transmission Control Protocol* (TCP) [Kurose and Ross, 2005], estabelecem uma conexão entre remetente e destinatário antes de iniciar a transmissão de dados. Essa abordagem envolve um processo de estabelecimento de conexão,

onde são trocadas mensagens de controle entre as partes, garantindo a confiabilidade e a entrega ordenada dos dados.

Os protocolos não orientados a conexão, como o UDP, não estabelecem uma conexão prévia antes de transmitir os dados. Eles simplesmente empacotam os dados em datagramas e os enviam para o destinatário. Esses protocolos são mais leves e rápidos, uma vez que não possuem a sobrecarga do estabelecimento e término de conexões. No entanto, eles não fornecem os mesmos recursos de confiabilidade do TCP. Os pacotes podem ser perdidos, entregues fora de ordem ou duplicados, e não há confirmação ou retransmissão automática.[Kurose and Ross, 2005]

A camada de aplicação desempenha um papel crucial na troca de informações por meio dos protocolos SMTP [Klensin, 2008], POP3 [Rose and Myers, 1996] e IMAP [Melnikov and Leiba, 2021]. O *Simple Mail Transfer Protocol* (SMTP) é amplamente utilizado para enviar e-mails entre clientes de e-mail e servidores de e-mail. Ele permite que os usuários enviem mensagens eletrônicas, que são transmitidas através de servidores intermediários até chegar ao destinatário final. Já os os protocolos *Post Office Protocol* (POP3) e *Internet Message Access Protocol* (IMAP) são usados pelos clientes de e-mail para recuperar mensagens de um servidor de e-mail. O POP3 realiza o download das mensagens para o dispositivo local do usuário, enquanto o IMAP acessa diretamente ao servidor, permitindo que os usuários visualizem, gerenciem e sincronizem suas caixas de entrada.

Além dos protocolos de e-mail mencionados anteriormente, existem os *Mail Transfer Agent* (MTA) envolvidos na troca de mensagens. Os MTAs são responsáveis pelo roteamento e entrega das mensagens de e-mail entre os servidores de e-mail. Eles atuam como intermediários na transferência de mensagens, encaminhando-as de um servidor para outro com base nas informações de endereço do destinatário. Neste trabalho utilizaremos o Postifx para essa função, entre as opções mais conhecidas ele se destaca pela segurança e performance.

O ContextNet é um *middleware* que utiliza o protocolo UDP e a plataforma OpenSplice para permitir a comunicação entre SMA Embarcados. O protocolo UDP é amplamente utilizado em aplicações que exigem uma transferência de dados rápida e de baixa latência. O OpenSplice é uma plataforma de *middleware* de objetos distribuídos (*Data Distribution Service* (DDS)) de código aberto, projetada para facilitar a comunicação e o compartilhamento de dados em sistemas distribuídos e em tempo real. Uma das principais características dele é a capacidade de lidar com grandes volumes de dados em tempo real.

O ContextNet aproveita a combinação dessas tecnologias para oferecer uma comunicação de alta performance entre os sistemas embarcados. Através do uso do UDP, é possível alcançar uma transferência de dados rápida e minimizando a sobrecarga de comunicação. Enquanto o OpenSplice permite uma comunicação distribuída escalável. Embora o ContextNet seja uma arquitetura de comunicação viável para sistemas multiagentes embarcados, ele também apresenta algumas deficiências que devem ser consideradas.

Entre elas podemos destacar a dependência do protocolo UDP, que, embora ofereça alta velocidade de transferência de dados, não possui mecanismos internos de controle de fluxo e correção de erros, o que pode levar a uma maior probabilidade de perda de pacotes e menor confiabilidade na entrega das mensagens. Além disso, a configuração do ContextNet pode exigir conhecimentos técnicos avançados, e a escalabilidade pode ser limitada por fatores como a largura de banda da rede e a capacidade de processamento dos dispositivos envolvidos.

A utilização de uma comunicação através de e-mails, que utilizam criptografia e são orientados a conexão, em comparação com o *middleware* ContextNet, que implementa o UDP, pode representar uma grande vantagem quando se trata de confiabilidade e confidencialidade das mensagens. A criptografia dos e-mails garante a segurança e a privacidade dos dados, evitando que sejam interceptados ou lidos por pessoas não autorizadas. Ademais, a natureza orientada a conexão dos e-mails permite uma comunicação mais confiável e robusta, garantindo que as mensagens sejam entregues corretamente e na ordem correta.

Capítulo 4

Arquitetura e-mail para Comunicação entre SMA Embarcados

No contexto de sistemas multiagentes embarcados, é necessário haver comunicação entre diferentes protótipos visto a natureza social de tais sistemas e a necessidade de se perceber o ambiente em volta para a tomada de decisões. À medida em que esses sistemas se tornam mais interconectados, surge a necessidade de uma arquitetura de comunicação eficiente, levando em conta variáveis como a segurança, privacidade, escalabilidade e a praticidade da comunicação. Tendo em vista que atuais abordagens utilizando o ContextNet não são suficientes e é preciso ter opções.

Este trabalho propõe uma arquitetura de comunicação entre SMA conforme demonstra a Figura 1. Nessa estrutura, o agente Comunicador é o responsável por receber as mensagens internas oriundas dos outros agentes e então enviá-las para outro sistema multiagente através do servidor de e-mail. No sistema destinatário o agente comunicador será o responsável pela consulta por novas mensagens no servidor.

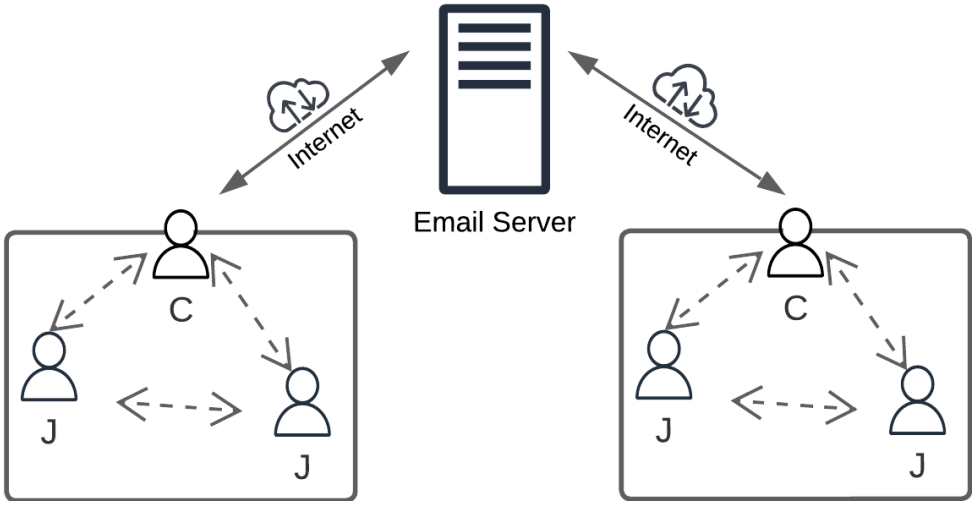

Figura 1: Abstração da comunicação entre SMA

Essa transmissão de mensagens é feita de forma assíncrona, o SMA Embarcado, quando conectado na internet, as envia ou recebe através do agente comunicador. Enquanto offline, o

servidor armazena as mensagens não entregues, fazendo o papel de manter uma fila durante o tempo em que o agente estiver incomunicável.

Quanto a comunicação dos agentes nessa proposta, os parâmetros: destinatário, remetente, força ilocucionária e conteúdo serão associados a estrutura do e-mail conforme mostra a Figura 2 para atingir os objetivos estabelecidos. Em relação as mensagens do Jason, destinatário e remetente tem o mesmo nome e exercem a mesma função que em um e-mail, já as forças serão mapeadas através do assunto do e-mail, e o conteúdo da mensagem será o corpo do e-mail.

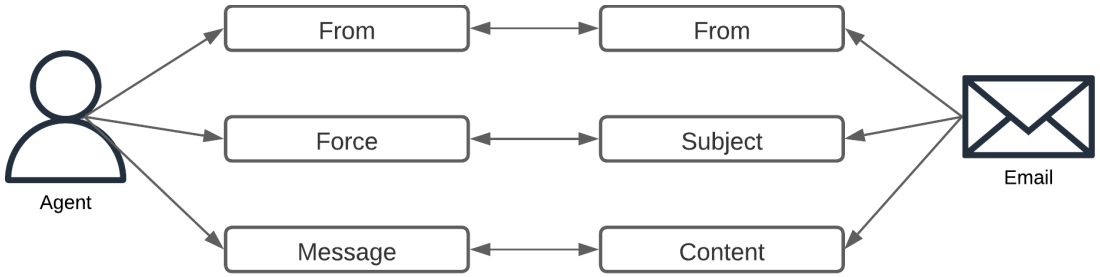

Figura 2: Mapeamento entre uma estrutura comunicação em KQML vs e-mail

4.1 Implementação do Serviço

Para a implementação do servidor de e-mail, foi escolhido o Debian, uma distribuição Linux conhecida por sua estabilidade e segurança. Além disso, esse *Sistema Operacional* (SO) é requisito da ChonIDE, onde programaremos os agentes. O SO foi instalado em um servidor apropriado, garantindo que o hardware atendesse aos requisitos necessários para lidar com o tráfego de e-mails.

Em seguida, deu-se início à instalação dos serviços Postfix e Dovecot. O Postfix é responsável envio e recebimento de e-mails. Durante o processo de instalação, foram realizadas várias etapas para configurar corretamente o servidor. Isso incluiu a definição de parâmetros de rede, como endereço IP e máscara de sub-rede, bem como a configuração dos domínios e endereços de e-mail válidos.

Além disso, medidas de segurança foram implementadas durante a instalação do Postfix. Isso envolveu a configuração de restrições de acesso, como listas de permissões e bloqueios, para evitar abusos e reduzir a probabilidade de envio de spam. Também foram configuradas políticas de autenticação, como a exigência de autenticação para o envio de e-mails, a fim de garantir que apenas usuários autorizados pudessem utilizar o servidor para enviar mensagens.

Depois disso foi configurado o Dovecot como um servidor de e-mail POP3 e IMAP. Ele

permite que os usuários acessem suas caixas de correio remotamente, facilitando a consulta e gerenciamento de e-mails a partir de diferentes dispositivos e locais. Durante a configuração do Dovecot, foram estabelecidos parâmetros adicionais, como opções de autenticação, políticas de segurança e regras de integração com o Postfix. Por exemplo, definiu-se onde as mensagens seriam armazenadas no servidor

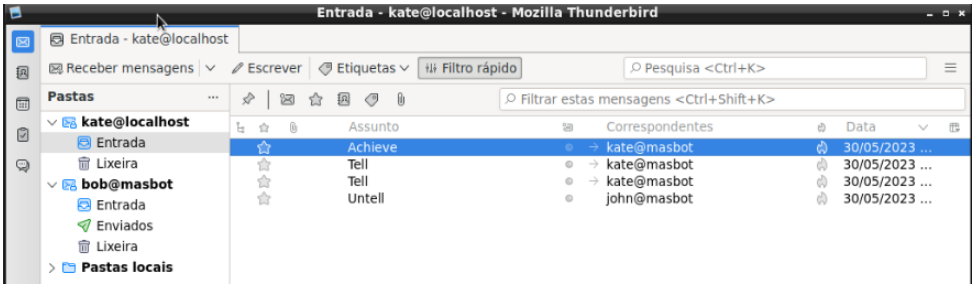

Figura 3: Troca de mensagens pelo cliente Thunderbird

Por fim, para testar a funcionalidade do servidor de e-mail, foi realizada a instalação do cliente de e-mail Thunderbird. O Thunderbird é um cliente de e-mail popular que permite enviar, receber e gerenciar mensagens de forma fácil e intuitiva. Ao configurar o Thunderbird, foi necessário fornecer as informações de conexão corretas, como o endereço do servidor, o tipo de protocolo (POP3 ou IMAP) e as credenciais de autenticação. Com o Thunderbird devidamente configurado, foram realizados testes para verificar o envio e recebimento de e-mails, garantindo que o servidor estivesse funcionando corretamente.

4.2 Extensão do Jason

O Jason [Bordini et al., 2007] é um *framework* de programação orientada a agentes que permite programar as crenças, planos, objetivos e intenções. A sintaxe e semântica do Jason é baseada em AgentSpeak. Para adicionar essa nova proposta de comunicação entre agentes, é necessário desenvolver uma solução que lide com o envio e recebimento de mensagens de e-mail dentro do formato estabelecido no *framework*.

Com o servidor devidamente implementado, foi possível iniciar os testes de envio e recebimento através de funções em Java dentro do padrão definido na sessão anterior. Inicialmente foi necessário configurar as dependências do projeto, incluindo a biblioteca JavaMail.

Em seguida, iniciou-se o desenvolvimento da classe Emailmiddleware, que conterá o métodos de envio e recebimento dos e-mails. No método sendEmail são definidas as propriedades do servidor de e-mail necessárias para o envio da mensagem: o endereço, protocolo, porta de

conexão e métodos de autenticação. Também são passados o destinatário, assunto e conteúdo da mensagem.

```
public void sendMsg(String recipientEmail, String subject, String message) {
    Session session;
    Properties props = sslProps();
    try {
        props.put( k: "mail.store.protocol", Sprotocol);
        props.put( k: "mail." + Sprotocol + ".host", Shost);
        props.put( k: "mail." + Sprotocol + ".port", Sport);
        props.put( k: "mail." + Sprotocol + ".leaveonserver",  v: false);
        session = Session.getInstance(props, getPasswordAuthentication() -> {
                return new PasswordAuthentication(login, password);
        });
    }catch (Exception e){
        System.out.println("Connection error:" + e);
        return;
    }
    try {
        // Create a new message
        Message msg = new MimeMessage(session);
        // Set the recipient, subject, and message content
        msg.setFrom(new InternetAddress(login));
        msg.setRecipients(Message.RecipientType.TO, InternetAddress.parse(recipientEmail));
        msg.setSubject(subject);
        msg.setText(message);
        // Send the message
        Transport.send(msg,login,password);
        System.out.println("Sent successfully!");
    }catch (MessagingException e) {
        System.out.println("Error sending email: " + e.getMessage());
    }
```

Figura 4: Implementação da função de envio

A classe implementa as funções Session e Message da biblioteca JavaMail. Na Session ela estabele a conexão com o servidor SMTP, com a sessão estabelecida então ela cria uma Message e, por fim, envia o e-mail usando a função Transport, que utiliza a sessão e mensagem criadas anteriormente para conectar ao SMTP e encaminhar a mensagem ao destinatário

Para o recebimento, um outro método também utilizando o JavaMail foi implementado. Nele são definidas as propriedades do servidor de e-mail que são necessárias para o recebimento das mensagens: o endereço, protocolo, porta de conexão, métodos de autenticação, login e senha. A classe implementa as funções Session e Store da biblioteca JavaMail. Com a Session ela estabele a conexão com o servidor, e, com essa sessão estabelecida ela então utiliza o Store para abrir a caixa de e-mail, fazer a leitura de cada item, e montar uma lista de mensagens do jason para ser retornada ao final da execução.

As mensagens recebidas pelo agente, devem ser lidas e interpretadas pelo agente durante seu ciclo de raciocínio. Para isso, foi necessário modificar a classe TransitionSystem para verificar se existe uma instancia da classe e-mail*middleware* e a partir disso seguir com a leitura dos

```
public ArrayList<jason.asSemantics.Message> checkEMail() {
    Session session ;
    Properties props = sslProps();
    try {
        props.put( k: "mail.store.protocol", Rprotocol);
        props.put( k: "mail." + Rprotocol + ".host", Rhost);
        props.put( k: "mail." + Rprotocol + ".port", Rport);
        props.put( k: "mail." + Rprotocol + ".leaveonserver", v: false);
        session = Session.getDefaultInstance(props);
    }catch (Exception e){
        System.out.println("Connection error:" + e);
        return null;
    }
    ArrayList<jason.asSemantics.Message> jMsg = new ArrayList<jason.asSemantics.Message>();
    try {
        // Connect to the email server
        Store store = session.getStore();
        store.connect(login, password);
        // Open the inbox folder and get the messages
        Folder inbox = store.getFolder( s: "INBOX");
        inbox.open(Folder.READ_WRITE);
        javax.mail.Message[] messages = inbox.getMessages();
        // Loop through the messages and printing info
        for (Message message : messages) {
            System.out.println("[EMail] New message...");
            //Skip messages marked for deletion
            if (message.getFlags().contains(Flags.Flag.DELETED)) {
                continue;
            }
            jason.asSemantics.Message jasonMsgs = new jason.asSemantics.Message(
                    message.getSubject(),
                    addressToString(message.getFrom()),
                     r: null,
                    message.getContent());
            jMsg.add(jasonMsgs);
            //mark message for deletion
            message.setFlag(Flags.Flag.DELETED,  set: true);
        }
        // Close the folder and store objects
        inbox.close();
        store.close();
    } catch (Exception e) {
        System.out.println("Error: " + e.getMessage());
    }
    return jMsg;
```

Figura 5: Implementação da função de recebimento

e-mails e transformá-los em crenças, planos ou intenções.

Durante a implementação, viu-se que diferentes tipos de servidores, utilizam diferentes propriedades do Javamail [Oracle, 2017]. Então, devido a isso, foi criado um método onde são passados os valores dessas propriedades. São elas:

- **auth**: Boleano - Se verdadeiro, utilizará o comando AUTH na autenticação.

- **starttls**: Booleano - Se verdadeiro, permite o uso do comando STARTTLS para alternar a conexão para uma conexão protegida por TLS antes de emitir qualquer comando de login.
- **sslenable**: Booleano - Se verdadeiro, usa SSL para se conectar e a porta SSL por padrão.
- **ssltrust**: String - Se definido como "*", todos os hosts são confiáveis. Se definido como uma lista de hosts separados por espaços em branco, esses hosts são confiáveis. Caso contrário, a confiança depende do certificado que o servidor apresenta.
- **sslprotocol**: String - Especifica os protocolos SSL que serão habilitados para conexões SSL.

Definidos os metodos de envio, recebimento e definição das propriedades de autenticação. Agora é necessário construir formas dos agentes poderem usar esses metodos, para isso, foram definidas as seguintes ações internas:

- **credentials**: Define as credenciais do agente. Parâmetros:login e senha.
- **receiveEmail**: Busca e-mails no servidor. (opcional)
- **receivingHost**: Define as informações do servidor de recebimento. Parâmetros: host, protocolo e porta
- **receivingProperties**: Define as propriedades de recebimento.
- **sendEmail**: Envia uma mensagem. Parâmetros: destinatário, assunto, mensagem.
- **sendingHost**: Define as informações do servidor de envio. Parâmetros: host, protocolo e porta
- **sendingProperties**: Define as propriedades de envio

As ações internas de configurações precisam ser chamadas somente uma vez, depois de definidas é possível chamar a ação "sendEmail"quantas vezes forem necessárias. Para atingir esse objetivo, é recomendada a criação de um plano inicial onde são definidas as configurações e depois fica a critério do agente quando enviar os e-mails.

Com as ações internas definidas e as interações entre as classes do Jason configuradas, podemos gerar o arquivo executável[1] e executar os testes com o ChonIDE. Um agente (Figura 6) foi programado para exemplificar como seria o envio de e-mail, primeiramente as configurações e depois a execução do plano para enviar.

```
!config.

+!config <-
     .credentials("bobjasonagent@gmx.com", "cyW1PvxrCk");
     .receivingHost("imap.gmx.com","imaps",993);
     .receivingProperties(false,true,false,null,null);
     .sendingHost("mail.gmx.com","smtp",465);
     .sendingProperties(true,true,false,null,null);
     !enviar.

+!enviar <-
     .sendEMail("bobjasonagent@gmx.com",tell,inbox(hello)).


+inbox(X)[source(Y)] <- .print(Y, " disse ", X).
```

Figura 6: Implementação do agente

Após o ciclo de raciocínio, fica comprovado no output do agente que a execução ocorreu corretamente conforme figura abaixo:

```
[ChonOS EmbeddedMAS] Starting the Multi-Agent System.
NOTE: Picked up JDK_JAVA_OPTIONS:  --add-opens=java.base/
--add-opens=java.base/java.util.concurrent=ALL-UNNAMED --
Jason Http Server running on http://127.0.1.1:3272
Sent successfully!
[EMail] New message...
[newAgent] bobjasonagent@gmx.com disse hello
```

Figura 7: Output da execução

[1]Repositório da versão customizada do Jason disponível em: https://github.com/chon-group/jasonEmbedded/tree/SMTP

Capítulo 5

Prova de conceito

Neste capítulo será apresentado um plano para avaliar a performance das ações desenvolvidas nesse trabalho, identificando suas forças e fraquezas com o intuito de garantir o sucesso dos objetivos propostos. Após a execução da Prova de conceito, este capítulo se aprofundará nos resultados obtidos destacando pontos positivos e oportunidades de otimização

Nessa arquitetura, é necessário que o agente instancie as configurações (Figura 8) antes de enviar uma mensagem. Para esses testes, foi escolhido o servidor GMX, um serviço gratuito de e-mails, e que suporta autenticação legado. Clientes modernos, como Gmail ou Exchange, necessitam de configurações adicionais de autenticação que estão fora do escopo desta arquitetura. Nos exemplos apresentados a seguir o método config será suprimido por questões de clareza.

```
!config.

+!config <-
    .credentials("bobjasonagent@gmx.com", "cyW1PvxrCk");
    .receivingHost("imap.gmx.com","imaps",993);
    .receivingProperties(false,true,false,null,null);
    .sendingHost("mail.gmx.com","smtp",465);
    .sendingProperties(true,true,false,null,null).
```

Figura 8: Configuração do agente

5.1 Enviando forças ilocucionárias

- **Crença**: O trecho destacado (Figura 9) mostra o plano enviar, cujo a ação é o envio de uma crença, força da mensagem é definida pelo segundo termo da função sendEmail.

```
+!enviar  <-
.sendEMail("bobjasonagent@gmx.com",tell,msg("Executei um plano recebido!")).
```

Figura 9: Função para envio de uma crença

- **Plano**: O envio de um plano é feito através da força tellHow, no trecho destacado (Figura

10) o plano enviar executa a ação interna e passa texto do plano como um dos termos do sendEmail.

```
+!enviar  <-
.sendEMail("bobjasonagent@gmx.com",tellHow,"+!print: msg(X) <- .print(X).").
```

Figura 10: Função para envio de um plano

- **Intenção**: O código da figura 11 mostra um plano enviar, cujo a ação é o envio de uma intenção "print".

```
+!enviar  <-
.sendEMail("bobjasonagent@gmx.com",achieve,print).
```

Figura 11: Função para envio de uma intenção

- **Lista de crenças**: O envio de uma lista de crenças é feito através da força tell, no trecho destacado (Figura 12) o plano enviar é executado se existir uma crença "msg"e então a envia um dos termos do sendEmail e depois descarta essa crença.

```
msg("Msg 1").
msg("Msg 2").
msg("Msg 3").

+!enviar: msg(X) <-
     .sendEMail("alicejasonagent@gmx.com",tell,msg(X));
     -msg(X);
     .wait(1000);
     !enviar.

-!enviar <-
     .print("Sem mensagens para enviar!").
```

Figura 12: Envio de uma lista de crenças

5.2 Enviar um e-mail para uma lista de agentes

O envio de uma crenças para uma lista de endereços é feito através da força tell, no trecho destacado (Figura 13) o plano enviar é executado se existir uma crença "address"e então ele envia uma crença inbox(hello) para esse address e em seguida descarta a crença do endereço.

```
address("alicejasonagent@gmx.com").
address("bernardo.jpedro@gmail.com").
address("bobjasonagent@aol.com").

+!enviar: address(X) <-
     .sendEMail(X,tell,inbox(hello));
     -address(X);
     .wait(1000);
     !enviar.

-!enviar <-
     .print("Sem destinatários para enviar!").
```

Figura 13: Envio para uma lista de endereços

5.3 Enviar e-mail para um agente offline

As mensagens na arquitetura proposta são feitas de forma assíncrona, como prova dessa comunicação, foram programados dois agentes (Figura 14), um realiza o envio de uma mensagem enquanto o outro está desativado, ao ligar, ele recebe a mensagem e executa um comando.

alice.asl

```
52  !config.
53
54  +!config <-
55       .print("iniciando!");
56       .credentials("alicejasonagent@gmx.com", "cyW1PvxrCk");
57       .receivingHost("imap.gmx.com","imaps",993);
58       .receivingProperties(false,true,false,null,null);
59       .sendingHost("mail.gmx.com","smtp",465);
60       .sendingProperties(true,true,false,null,null);
61       !enviar.
62
63
64  +!enviar <-
65       .sendEMail("bobjasonagent@gmx.com",tell,inbox(hello)).
```

bob.asl

```
1   !config.
2
3   +!config <-
4        .wait(20000);
5        .print("iniciando!");
6        .credentials("bobjasonagent@gmx.com", "cyW1PvxrCk");
7        .receivingHost("imap.gmx.com","imaps",993);
8        .receivingProperties(false,true,false,null,null);
9        .sendingHost("mail.gmx.com","smtp",465);
10       .sendingProperties(true,true,false,null,null).
11
12
13
14  +inbox(X)[source(Y)] <- .print(Y," disse ",X).
```

Figura 14: Código fonte dos agentes bob e alice

5.4 Resultados

5.4.1 Enviando forças ilocucionários

As crenças de um agente podem ser visualizados a partir do Mind Inspector. Após a execução do código apresentado na sessão anterior, é possível ver na mente do agente bob (Figura 15) que a crença enviada pela agente alice aparece listada.

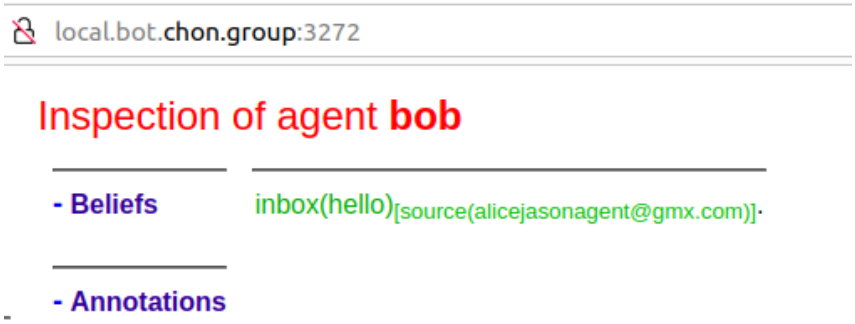

Figura 15: Mente do agente bob

Após a execução do código para enviar um plano na sessão anterior, é possível ver no log da ChonIDE (Figura 16) que a mensagem é recebida mas nada acontece pois um plano só é executado se houver uma intenção.

```
[ChonOS EmbeddedMAS] Starting the Multi-Agent System.
NOTE: Picked up JDK_JAVA_OPTIONS:  --add-opens=java.base/java.
--add-opens=java.base/java.util.concurrent=ALL-UNNAMED --add-o
Jason Http Server running on http://127.0.1.1:3272
[alice] iniciando!
[bob] iniciando!
Sent successfully!
[EMail] New message...
```

Figura 16: Log de recebimento de um plano

Quando é feito o envio de uma intenção após o envio do plano e da crença, então o agente executa o plano recebido anteriormente (Figura 17).

```
[ChonOS EmbeddedMAS] Starting the Multi-Agent System.
NOTE: Picked up JDK_JAVA_OPTIONS:  --add-opens=java.base/java
--add-opens=java.base/java.util.concurrent=ALL-UNNAMED --add-
Jason Http Server running on http://127.0.1.1:3272
[alice] iniciando!
[bob] iniciando!
Sent successfully!
[EMail] New message...
Sent successfully!
[EMail] New message...
Sent successfully!
[EMail] New message...
[bob] Executei um plano recebido!
```

Figura 17: Log de recebimento de uma intenção

Na execução do envio de uma lista de crenças, podemos ver no log de execução (Figura 18) que o agente faz o envio de três mensagens, descarta as crenças referentes à essas mensagens e por ultimo executa o plano de contenção exibindo a mensagem de que não há mais mensagens para enviar. Como prova que a lista de e-mails está sendo recebida, um outro agente (Figura 19) foi codificado para exibir as mensagens obtidas durante seu ciclo de raciocínio.

```
[ChonOS EmbeddedMAS] Starting the Multi-Agent System.
NOTE: Picked up JDK_JAVA_OPTIONS:  --add-opens=java.base/java.
--add-opens=java.base/java.util.concurrent=ALL-UNNAMED --add-
Jason Http Server running on http://127.0.1.1:3273
Sent successfully!
Sent successfully!
Sent successfully!
[alice] Sem mensagens para enviar!
```

Figura 18: Log do envio da lista de crenças

```
[ChonOS EmbeddedMAS] Starting the Multi-Agent System.
NOTE: Picked up JDK_JAVA_OPTIONS:  --add-opens=java.base/java
--add-opens=java.base/java.util.concurrent=ALL-UNNAMED --add-
Jason Http Server running on http://127.0.1.1:3273
[EMail] New message...
[EMail] New message...
[EMail] New message...
[alice] Recebi Msg 1de bobjasonagent@gmx.com
[alice] Recebi Msg 2de bobjasonagent@gmx.com
[alice] Recebi Msg 3de bobjasonagent@gmx.com
```

Figura 19: Log do recebimento da lista de crenças

5.4.2 E-mail para uma lista de agentes

Na execução do envio de uma crença para uma lista de destinatários, podemos ver no log de execução (Figura 20) que o agente faz o envio de três mensagens, descarta as crenças referentes aos destinatários e por ultimo executa o plano de contenção exibindo a mensagem de que não há mais destinatário para envio. A figura 21 mostra o e-mail recebido pelas diferentes plataformas e destinatários.

```
[ChonOS EmbeddedMAS] Starting the Multi-Agent System.
NOTE: Picked up JDK_JAVA_OPTIONS:  --add-opens=java.base/java.lang=
--add-opens=java.base/java.util.concurrent=ALL-UNNAMED --add-opens=
Jason Http Server running on http://127.0.1.1:3273
Sent successfully!
Sent successfully!
Sent successfully!
[alice] Sem destinatários para enviar!
[ChonOS EmbeddedMAS] Stopping the Multi-agent System Gently.
[ChonOS EmbeddedMAS] Multi-Agent System Successfully Terminated!
```

Figura 20: Log da lista de e-mail sendo enviada

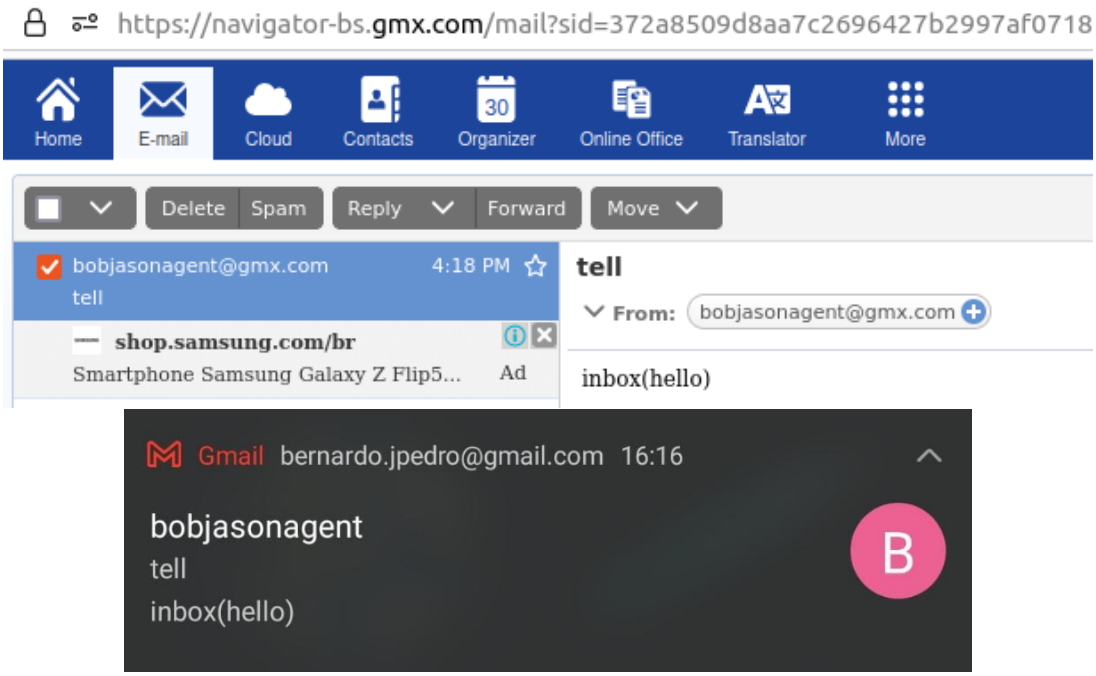

Figura 21: E-mails recebidos em diferentes plataformas

5.4.3 E-mail para um agente offline

Após o teste do envio assíncrono, é possível verificar no log de execução (Figura 14) que o primeiro agente inicia o ciclo e realiza o envio enquanto o outro ainda está desativado. Então, após esse envio o segundo agente inicia seu ciclo, recebe a mensagem enviada pelo primeiro agente e a exibe.

```
[ChonOS EmbeddedMAS] Starting the Multi-Agent System.
NOTE: Picked up JDK_JAVA_OPTIONS:  --add-opens=java.base/java.
--add-opens=java.base/java.util.concurrent=ALL-UNNAMED --add-o
Jason Http Server running on http://127.0.1.1:3273
[alice] iniciando!
Sent successfully!
[bob] iniciando!
[EMail] New message...
[bob] alicejasonagent@gmx.com disse hello
```

Figura 22: Implementação do agente

5.5 Pontos de atenção

Durante a fase de testes foi identificado que em alguns servidores o envio de muitas requisições em um curto intervalo de tempo podem ocasionar um bloqueio temporário, não sendo possível conectar-se ao servidor, e o agente não funcionar corretamente. É necessário então durante o desenvolvimento dos agentes considerar um plano de contenção, como por exemplo um outro servidor ou uma pausa nas requisições, caso isso ocorra.

Capítulo 6

Conclusão

Neste trabalho foi explorado o campo dos sistemas multiagentes embarcados e as formas de comunicação entre eles. A análise revelou algumas deficiências de ferramentas utilizadas na área, como por exemplo, a falta de criptografia e comunicação não orientada a conexão e como solução propôs uma nova arquitetura, utilizando como base a linguagem agent speak e protocolos de e-mails para interligar esses sistemas.

Uma das principais contribuições deste trabalho é viabilizar uma opção simples e segura de comunicação, visto que os protocolos de e-mail não costumam enfrentar bloqueios de rede e também oferecem suporte à criptografia. Além disso, essa proposta oferece uma opção assíncrona de comunicação, sistemas multiagentes embarcados podem utilizar dessa arquitetura para superar limitações da comunicação síncrona como a disponibilidade.

Como trabalhos futuros, um caminho seria se aprofundar no desenvolvimento de uma arquitetura onde seja possível a comunicação via e-mail e pelo ContextNet no mesmo agente Comunicador. Outro ponto relevante é explorar o desenvolvimento das ações AskAll e AskHow a partir da arquitetura via e-mail. Isso poderia proporcionar uma visão mais abrangente e ampliando as aplicações práticas dos SMA.

Referências Bibliográficas

Olivier Boissier, Rafael H. Bordini, Jomi F. Hübner, Alessandro Ricci, and Andrea Santi. Multi-agent oriented programming with JaCaMo. *Science of Computer Programming*, 78(6):747–761, 2013. ISSN 0167-6423. doi: 10.1016/j.scico.2011.10.004. Special section: The Programming Languages track at the 26th ACM Symposium on Applied Computing (SAC 2011) Special section on Agent-oriented Design Methods and Programming Techniques for Distributed Computing in Dynamic and Complex Environments.

Rafael H. Bordini, Jomi Fred Hübner, and Michael Wooldridge. *Programming Multi-Agent Systems in AgentSpeak using Jason*. John Wiley & Sons Ltd, 2007. ISBN 9780470057476.

Fabian Cesar Brandão, Maria Alice Trinta Lima, Carlos Eduardo Pantoja, Jean Zahn, and José Viterbo. Engineering approaches for programming agent-based iot objects using the resource management architecture. *Sensors*, 21(23), 2021. ISSN 1424-8220. doi: 10.3390/s21238110.

Michael E Bratman. Intention, practical rationality, and self-governance. *Ethics*, 119(3):411–443, 2009.

Vinicius Souza de Jesus, Fabian Cesar PB Manoel, and Carlos Eduardo Pantoja. Protocolo de interação entre sma embarcados bio-inspirado na relação de predatismo. In *Anais do XIII Workshop-Escola de Sistemas de Agentes, seus Ambientes e apliCaç oes—WESAAC*, pages 95–106, 2019.

M. Endler, G. Baptista, L. D. Silva, R. Vasconcelos, M. Malcher, V. Pantoja, V. Pinheiro, and J. Viterbo. ContextNet: Context reasoning and sharing middleware for large-scale pervasive collaboration and social networking. In *Proceedings of the Workshop on Posters and Demos Track*, PDT '11, New York, NY, USA, 2011. Association for Computing Machinery. ISBN 9781450310734. doi: 10.1145/2088960.2088962.

Markus Endler and Francisco Silva e Silva. Past, present and future of the ContextNet IoMT middleware. *Open Journal of Internet Of Things (OJIOT)*, 4(1):7–23, 2018. ISSN 2364-7108. URL `http://nbn-resolving.de/urn:nbn:de:101:1-201808051932326762285`7. Special Issue: Proceedings of the International Workshop on Very Large Internet of Things (VLIoT 2018) in conjunction with the VLDB 2018 Conference in Rio de Janeiro, Brazil.

Tim Finin, Don McKay, and Rich Fritzson. An overview of KQML: A knowledge query and manipulation language. 1992.

Tim Finin, Richard Fritzson, Don McKay, and Robin McEntire. KQML as an agent communication language. In *Proceedings of the Third International Conference on Information and Knowledge Management*, CIKM '94, page 456–463, New York, NY, USA, 1994. Association for Computing Machinery. ISBN 0897916743. doi: 10.1145/191246.191322.

Mahdi Hannoun, Olivier Boissier, Jaime S Sichman, and Claudette Sayettat. Moise: An organizational model for multi-agent systems. In *Advances in Artificial Intelligence*, pages 156–165. Springer, 2000.

Jomi F Hübner, Jaime S Sichman, and Olivier Boissier. Developing organised multiagent systems using the moise+ model: programming issues at the system and agent levels. *International Journal of Agent-Oriented Software Engineering*, 1(3-4):370–395, 2007.

Dr. John C. Klensin. Simple Mail Transfer Protocol. RFC 5321, October 2008. URL `https://www.rfc-editor.org/info/rfc5321`.

J Kurose and K Ross. Computer networking: A top-down approach featuring the internet, pearsoneducation, 2005.

Nilson Mori Lazarin and Carlos Eduardo Pantoja. A robotic-agent platform for embedding software agents using raspberry pi and arduino boards. In *Proceedings WESAAC 2015*, pages 13–20, Niteroi, 2015. UFF. URL `http://www2.ic.uff.br/~wesaac2015/Proceedings-WESAAC-2015.pdf`.

Nilson Mori Lazarin, Carlos Eduardo Pantoja, and Vinicius Souza de Jesus. Um Protocolo para Comunicação entre Sistemas Multi-Agentes Embarcados. In *Proceedings of the 15th Workshop-School on Agents, Environments, and Applications (WESAAC 2021)*, pages 166–177, Rio de Janeiro, 2021. doi: https://doi.org/10.5281/zenodo.5774181.

Alexey Melnikov and Barry Leiba. Internet Message Access Protocol (IMAP) - Version 4rev2. RFC 9051, August 2021. URL `https://www.rfc-editor.org/info/rfc9051`.

Oracle. Javamail api. Website, 2017. URL `https://javaee.github.io/javamail/`. Version 1.6.2.

J. Postel. User Datagram Protocol. RFC 768, August 1980. URL https://www.rfc-editor.org/info/rfc768.

Anand S. Rao and Michael P. Georgeff. Modeling rational agents within a BDI-Architecture. In *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning*, KR'91, page 473–484, San Francisco, CA, USA, 1991. Morgan Kaufmann Publishers Inc. ISBN 1558601651.

Alessandro Ricci, Mirko Viroli, and Andrea Omicini. CArtAgO: A framework for prototyping artifact-based environments in mas. In Danny Weyns, H. Van Dyke Parunak, and Fabien Michel, editors, *Environments for Multi-Agent Systems III*, pages 67–86, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg. ISBN 978-3-540-71103-2.

Dr. Marshall T. Rose and John G. Myers. Post Office Protocol - Version 3. RFC 1939, May 1996. URL https://www.rfc-editor.org/info/rfc1939.

Lucas Fernando Souza de Castro, Fabian Cesar Pereira Brandão Manoel, Vinicius Souza de Jesus, Carlos Eduardo Pantoja, Andre Pinz Borges, and Gleifer Vaz Alves. Integrating embedded multiagent systems with urban simulation tools and iot applications. *Revista de Informática Teórica e Aplicada*, 29(1):81–90, Jan. 2022. doi: 10.22456/2175-2745.110837.

Michael Wooldridge. *An introduction to multiagent systems*. John Wiley & Sons, 2009.

Michael J Wooldridge. *Reasoning about rational agents*. MIT press, 2000.