# Towards MASPY Interoperability with Heterogeneous Multi-Agent Platforms

**João P. Lepinsk[1], Gleifer Vaz Alves[1], André Pinz Borges[1], Alexandre L. L. Mellado[1], Rafael C. Cardoso[2]**

[1] Universidade Tecnológica Federal do Paraná (UTFPR)
Ponta Grossa, PR, Brasil

[2]University of Aberdeen
Aberdeen – United Kingdom

{lepinsk,mellado}@alunos.utfpr.edu.br, rafael.cardoso@abdn.ac.uk

{apborges,gleifer}@utfpr.edu.br

***Abstract.*** *The MASPY framework supports the development of BDI multi-agent systems in Python, with internal communication through channels and centralised configuration. However, it does not provide a native mechanism to interoperate with external platforms via Web protocols. This work presents the ExternalChannel, a component that translates external messages into BDI structures, injects them into the agents' reasoning cycle, and forwards outgoing messages through platform-specific adapters. The problem definition was supported by a systematic mapping study, and feasibility was demonstrated in a logistics scenario involving JaCaMo, SPADE, and MASPY, in which MASPY interacted with external platforms through FIPA-ACL-based messages and a REST adapter. Internal injection latency averaged 1.28 ms versus 0.81 ms for native-channel delivery under the same setup, providing an internal reference for the prototype's added cost.*

***Resumo.*** *O framework MASPY oferece suporte ao desenvolvimento de sistemas multiagentes BDI em Python, com comunicação interna por canais e configuração centralizada. No entanto, não oferece mecanismo nativo para interoperar com plataformas externas por protocolos Web. Este trabalho apresenta o ExternalChannel, um componente que traduz mensagens externas em estruturas BDI injetadas no ciclo de raciocínio dos agentes e encaminha mensagens de saída por meio de adaptadores específicos para cada plataforma. A delimitação do problema apoiou-se em um mapeamento sistemático, e a viabilidade foi demonstrada em um cenário de logística com JaCaMo, SPADE e MASPY, no qual o MASPY interagiu com plataformas externas por mensagens baseadas em FIPA-ACL e por adaptador REST. A latência interna de injeção foi de 1,28 ms, frente a 0,81 ms no canal nativo, nas mesmas condições experimentais, como referência interna do custo adicional do protótipo.*

## 1. Introdução

Sistemas Multiagentes (SMA) organizam entidades autônomas capazes de perceber o ambiente, raciocinar sobre objetivos e coordenar ações em problemas distribuídos

[Wooldridge 2009]. Entre as arquiteturas que modelam esse raciocínio, o modelo Belief-Desire-Intention (BDI) representa explicitamente as crenças, desejos e intenções que orientam a deliberação do agente [Bratman 1987]. O desenvolvimento de SMA apoia-se em frameworks que oferecem abstrações para agentes, ambientes e comunicação. O ecossistema atual reúne plataformas heterogêneas quanto à linguagem, ao paradigma e ao protocolo de troca de mensagens. Entre elas estão plataformas como SPADE [Palanca et al. 2020], em Python sobre XMPP, e JaCaMo [Boissier et al. 2013], em Java sobre Jason, CArtAgO e Moise.

O MASPY é um framework em Python para construir SMA baseados em agentes BDI [Mellado et al. 2025]. Sua proposta integra programação declarativa, ciclo de raciocínio, ambiente e comunicação. A troca de mensagens ocorre por canais nomeados, assíncronos e locais, com performativas inspiradas no KQML [Finin et al. 1994]. A validação inicial identificou limitações de desempenho e escalabilidade em configuração centralizada. Em trabalho posterior, foram adicionados recursos de aprendizado por reforço e monitoramento, mantendo a distribuição como direção futura [Mellado et al. 2026].

A interoperabilidade entre SMA e sistemas externos é necessária em aplicações que coordenam agentes de tecnologias distintas, pois permite reaproveitar plataformas especializadas sem reescrever os agentes. Nesse contexto, a ausência de conectividade interplataforma limita o uso do MASPY em fluxos baseados em protocolos da Web. O objetivo deste trabalho é demonstrar que é viável dotar o MASPY de interoperabilidade com plataformas externas por meio da herança da classe de comunicação, sem modificar o núcleo do framework nem a forma nativa de programar os agentes. A hipótese é que a especialização da abstração de canal existente, por herança, é suficiente para converter mensagens externas em estruturas BDI no ciclo de raciocínio, sem recorrer a um mecanismo de comunicação à parte.

Como resposta, este trabalho propõe o *ExternalChannel*, extensão que adiciona ao canal de comunicação do MASPY conectividade HTTP e suporte a mensagens baseadas em FIPA-ACL.

A FIPA define especificações para a comunicação entre agentes em sistemas heterogêneos [Poslad and Charlton 2001]. Sua linguagem ACL estrutura as mensagens como atos comunicativos, sem impor como cada plataforma deve processá-las internamente nem qual infraestrutura de transporte deve ser utilizada. Essa neutralidade permite que mensagens baseadas em FIPA-ACL sejam transmitidas por diferentes protocolos, incluindo HTTP. Neste trabalho, essa possibilidade é explorada por meio de um envelope baseado no FIPA-ACL para a comunicação entre o MASPY e plataformas externas.

Este artigo tem três contribuições principais: (**i**) a arquitetura do *ExternalChannel* como extensão não intrusiva do canal de comunicação; (**ii**) o mecanismo de tradução entre mensagens FIPA-ACL e estruturas BDI nativas do MASPY; e (**iii**) a demonstração por meio de uma prova de conceito, envolvendo os frameworks JaCaMo, SPADE e MASPY, com medições da injeção de mensagens e do protocolo de coordenação.

## 2. Mapeamento Sistemático

A delimitação da lacuna apoiou-se em um mapeamento sistemático sobre interoperabilidade entre sistemas multiagentes e sistemas Web, segundo as diretrizes de Kitchenham e

Charters [Kitchenham and Charters 2007] na ferramenta Parsifal[1]. O estudo orientou-se por quatro questões de pesquisa: (QP1) quais abordagens arquiteturais integram SMA a sistemas Web; (QP2) quais tecnologias e protocolos de comunicação são empregados; (QP3) em que direção ocorre a integração; e (QP4) quais desafios são reportados.

A busca foi realizada em maio de 2026, nas bases ACM Digital Library, IEEE Xplore, ScienceDirect e Scopus, considerando publicações de 2021 a 2026:

```
("multi-agent system" OR "multi-agent systems" OR "multiagent
   system"
 OR "agent-based system") AND (interoperability OR integration)
 AND (HTTP OR REST OR "web service" OR "web API")
```

O uso de expressões com "system" buscou restringir a população a sistemas multiagentes, em consonância com o escopo. Essa escolha pode reduzir a abrangência ao omitir estudos que empreguem apenas *multi-agent*, *agent platform* ou *agent framework*.

Foram incluídos estudos que abordam interoperabilidade ou integração em sistemas multiagentes e que propõem arquiteturas, frameworks ou soluções. Foram excluídos estudos secundários, como revisões e mapeamentos; estudos puramente conceituais, sem proposta técnica; trabalhos indisponíveis; e estudos que não tratam de sistemas multiagentes, de tecnologias Web ou de interoperabilidade. Trabalhos de IA agêntica baseados em LLM foram mantidos por decisão de escopo: embora não adotem BDI nem FIPA, retratam a forma recente de comunicação entre agentes sobre infraestrutura da Web (QP2).

As buscas retornaram 95 estudos, acrescidos de três inclusões manuais: AAMAS 2023 [Vachtsevanou et al. 2023], AAMAS 2024 [Ihejimba and Wenkstern 2024] e um estudo indicado por especialista [Bossa et al. 2026], totalizando 98. Após remover quatro duplicados, 94 seguiram para triagem; 64 foram excluídos por título/resumo e 16 dos 30 avaliados em texto completo, resultando em 14 estudos.

**QP1 – Abordagens arquiteturais.** A maior parte dos trabalhos selecionados distribui-se em quatro grupos principais. O primeiro adota arquiteturas de microsserviços, nas quais os agentes são encapsulados como serviços Web [Ihejimba and Wenkstern 2024, Alves et al. 2022, Zouad and Boufaida 2025]. O segundo organiza a integração por meio de gateways ou mediadores que traduzem formatos e protocolos, como o gateway de integração de serviços Web de [Kouissi et al. 2021]. Essa característica não é exclusiva: parte dos trabalhos sobre microsserviços e orquestração por LLM também emprega um componente de mediação, de modo que os grupos não são mutuamente exclusivos. O terceiro organiza a integração por meio de recursos de hipermídia, RDF ou coordenação no ambiente [Chopra and Singh 2026, Schubotz et al. 2022, Vachtsevanou et al. 2023, Santos et al. 2021]. O quarto, mais recente, explora orquestração por LLMs ou protocolos emergentes entre agentes [Rezaei et al. 2025, Bhosale and Gawande 2025, Louck et al. 2026, Vaziry et al. 2026].

**QP2 – Tecnologias e protocolos.** No plano do transporte, o HTTP predomina, em geral associado ao REST; já a comunicação entre agentes permanece heterogênea, com uso de FIPA-ACL em parte dos trabalhos [Todorov et al. 2025, Kouissi et al. 2021, Bossa et al. 2026] e adoção de protocolos próprios ou de chamadas REST diretas em

[1] `https://parsif.al/` (Acessado em 21 de junho de 2026)

outros.

**QP3 – Direção da integração.** Nove dos 14 estudos são bidirecionais; em três, o SMA consome serviços externos; e em dois, o SMA expõe serviços à Web.

**QP4 – Desafios reportados.** Destacam-se heterogeneidade de plataformas, protocolos e dados [Todorov et al. 2025, Santos et al. 2021, Bossa et al. 2026]; escalabilidade, latência, tolerância a falhas e adaptação [Ihejimba and Wenkstern 2024, Zouad and Boufaida 2025, Rezaei et al. 2025]; segurança, identidade e confiança [Bhosale and Gawande 2025, Louck et al. 2026, Vaziry et al. 2026]; e descoberta, descrição e seleção de agentes ou serviços [Santos et al. 2021, Vachtsevanou et al. 2023, Bhosale and Gawande 2025]. Assim, a interoperabilidade envolve transporte, semântica, descoberta, segurança e operação em ambientes heterogêneos.

A Tabela 1 sintetiza os 14 estudos por plataforma, comunicação e direção. O trabalho mais próximo é o PEAK-ACL [Bossa et al. 2026], que implementa a FIPA-ACL como pacote de comunicação para frameworks Python, com transporte HTTP-MTP. O *ExternalChannel* distingue-se por integrar-se ao MASPY por herança de canal, sem alterar o núcleo, convertendo as mensagens recebidas em estruturas BDI injetadas no ciclo de raciocínio. Nenhum dos trabalhos analisados integra a conectividade Web a um framework BDI em Python por meio da especialização da abstração nativa de comunicação, sem modificar o núcleo. Essa é a lacuna que motiva o componente descrito nas próximas seções.

**Tabela 1. Síntese comparativa dos 14 trabalhos e do *ExternalChannel*.**

| ID | Referência | Plataforma | Comunicação | Direção |
|---|---|---|---|---|
| 1 | [Ihejimba and Wenkstern 2024] | Própria | Não especificada | MAS expõe serviço |
| 2 | [Todorov et al. 2025] | Jadex | FIPA-ACL | Bidirecional |
| 3 | [Kouissi et al. 2021] | JADE | FIPA-ACL | MAS expõe serviço |
| 4 | [Zouad and Boufaida 2025] | SPADE | REST | Bidirecional |
| 5 | [Chopra and Singh 2026] | Própria | Normas sociais | Bidirecional |
| 6 | [Alves et al. 2022] | ActressMAS | Não especificada | Bidirecional |
| 7 | [Rezaei et al. 2025] | LLM/agêntica | Orquestração via LLM | Bidirecional |
| 8 | [Bossa et al. 2026] | JADE, SPADE | FIPA-ACL | Bidirecional |
| 9 | [Bhosale and Gawande 2025] | LLM/agêntica | Orquestração via LLM | MAS consome serviço |
| 10 | [Louck et al. 2026] | LLM/agêntica | Protocolos de interop. | Bidirecional |
| 11 | [Santos et al. 2021] | JADE | Recursos RDF | Bidirecional |
| 12 | [Vachtsevanou et al. 2023] | Jason/JaCaMo | Não especificada | MAS consome serviço |
| 13 | [Schubotz et al. 2022] | Própria | Não especificada | MAS consome serviço |
| 14 | [Vaziry et al. 2026] | LLM/agêntica | API proprietária | Bidirecional |
| EC | Este trabalho | MASPY | FIPA-ACL + REST | Bidirecional |

## 3. Estendendo o Canal de Comunicação no MASPY

O *ExternalChannel* estende o mecanismo de comunicação do MASPY com conectividade HTTP e suporte a mensagens baseadas em FIPA-ACL, reunindo quatro capacidades: servidor HTTP para receber requisições e validar o envelope; fila de prioridade para ordenar as mensagens recebidas; envio não bloqueante; e adaptadores de saída para traduzir a mensagem ao formato de cada plataforma de destino.

### 3.1. Integração não Intrusiva por Herança de Canal

A comunicação no MASPY organiza-se em canais, instâncias da classe *Channel*. A metaclasse *CommsMultiton* garante que cada nome de canal corresponde a uma única instância e registra o canal no *Admin*, componente administrativo do framework. O *ExternalChannel* é uma subclasse de *Channel*; portanto, para o restante do MASPY, ele é tratado como um canal do framework: registra-se no *Admin*, é reconhecido pelo *CommsMultiton* e recebe conexões de agentes pela via usual.

Essa herança impõe uma restrição técnica: o *CommsMultiton* intercepta a instanciação e aceita apenas o nome do canal como argumento, o que impede a passagem da porta de rede diretamente ao construtor. O *ExternalChannel* contorna essa restrição por meio do método de fábrica `create()`, que registra a porta e a eventual chave de autenticação antes da instanciação. Por isso, a criação do canal usa `ExternalChannel.create()`, e não a chamada direta ao construtor.

### 3.2. Comunicação de Entrada: do HTTP ao Ciclo de Raciocínio

O servidor HTTP é assíncrono e recebe requisições externas por meio de rotas registradas. Ao receber uma mensagem, verifica a autenticação quando configurada, valida os campos obrigatórios do envelope FIPA-ACL em JSON, extrai a prioridade e rejeita mensagens malformadas. O servidor também limita o tamanho do corpo das requisições, como medida de proteção contra o consumo excessivo de memória.

A associação entre um *endpoint* e a estrutura BDI a criar é definida pelo método `route()`. Cada rota vincula um caminho a uma crença ou a um objetivo, podendo ainda indicar um agente-alvo e uma função de transformação para extrair, a partir do conteúdo da mensagem, os valores da estrutura. Após a validação, a mensagem não é entregue imediatamente ao agente: é enfileirada e o servidor responde com o status HTTP 202 (Accepted), indicando que a requisição foi aceita para processamento assíncrono. Falhas na função de transformação, como conteúdo sem o formato esperado pela rota, ocorrem antes do enfileiramento e devolvem um erro HTTP ao emissor, sem alcançar a fila.

A fila de prioridade é consumida por uma *thread* dedicada, que processa primeiro as mensagens mais urgentes e preserva a ordem de chegada em caso de empate. Cada item é injetado no agente pelo método nativo de adição; sem agente-alvo, a estrutura é entregue a todos os agentes conectados ao canal. Não são usadas travas externas, pois sincronizar o ciclo a partir de uma *thread* externa bloquearia as fases de percepção e leitura de mensagens. A entrega segue a semântica *at-most-once*, sem persistência ou reenvio. A segurança sob injeções concorrentes é discutida na Seção 5.

### 3.3. Comunicação de Saída: o Formato Base e os Adaptadores

O envio de mensagens a sistemas externos usa o método `notify()`, que segue um padrão de despacho sem espera: a requisição HTTP é submetida a um laço de eventos assíncrono e o ciclo de raciocínio do agente prossegue. Como o MASPY executa cada agente em sua própria *thread*, o *ExternalChannel* mantém um laço de eventos dedicado, em uma *thread* auxiliar, responsável tanto pelas requisições de entrada quanto pelos envios de saída. As conexões de saída reutilizam uma sessão HTTP persistente.

O envelope de mensagem é a *FIPAMessage*, o formato base do componente, inspirado na FIPA-ACL. Não é uma mensagem ACL normativa: reaproveita a estrutura de

atos comunicativos da FIPA-ACL, mas não implementa a linguagem de conteúdo FIPA-SL [Bossa et al. 2026] nem o conjunto completo de parâmetros. O campo `language` declara FIPA-SL nível 0 apenas como rótulo nominal, e o conteúdo trafega como literais. Cada mensagem reúne a performativa, o remetente, o destinatário e o conteúdo, além de campos complementares, como prioridade, ontologia, identificador de conversa e prazo de resposta. As performativas suportadas são oito: inform, request, propose, accept-proposal, reject-proposal, cfp, agree e failure. A construção valida os campos obrigatórios e gera um identificador de conversa quando ausente, permitindo rastrear trocas entre plataformas.

Sobre esse formato base atua a camada de adaptadores de saída. O agente sempre produz uma *FIPAMessage*; no envio, um adaptador traduz essa mensagem para o formato compreendido pela plataforma de destino. Dois adaptadores estão implementados. O primeiro serializa a *FIPAMessage* no formato JSON FIPA-ACL, recebido pela camada HTTP implementada no agente SPADE do experimento. O segundo traduz a mensagem para o formato da camada REST do JaCaMo, encapsulando performativa e conteúdo em um literal compatível com planos Jason. Assim, a mesma mensagem-base pode gerar formatos de saída distintos conforme o destino, e novas plataformas podem ser incorporadas por meio da adição de adaptadores.

A interoperabilidade não decorre de uma correspondência direta entre as performativas internas do MASPY, inspiradas em KQML, e as performativas do FIPA-ACL. O componente realiza uma tradução entre o envelope FIPA-ACL e as estruturas BDI nativas. Na entrada, o conteúdo da mensagem alimenta a função de transformação da rota, que produz a crença ou o objetivo a injetar; na saída, o agente compõe uma *FIPAMessage* e o adaptador a traduz para o formato de destino. O vocabulário padronizado é mantido na fronteira da comunicação, enquanto os agentes operam apenas com base em crenças e objetivos.

### 3.4. Recursos de Robustez e Decisões de Projeto

O *ExternalChannel* incorpora autenticação opcional por *token* portador (Bearer), validação de campos obrigatórios, limitação do tamanho das requisições e liberação da porta ao encerrar o servidor. Sem Transport Layer Security (TLS), o *token* trafega em claro; no protótipo, a cifragem é delegada à infraestrutura de implantação. O Código 1 ilustra a criação do canal, o registro da rota `/delivery`, que gera a crença `delivery_order`, e o envio externo de uma *FIPAMessage* por `notify()`.

```
channel = ExternalChannel.create("delivery_channel", port=9000)
channel.route("/delivery", belief="delivery_order", transform=lambda p: (p["carrier"], p
    ["cep"], p["cost"]))

class DeliveryAgent(Agent):
    @pl(gain, Belief("delivery_order", (Any, Any, Any)))
    def handle_delivery(self, src, data):
        carrier, cep, cost = data
        ch = ExternalChannel.get_channel("delivery_channel")
        ch.notify("http://localhost:5000/inbox",
                  FIPAMessage(performative="inform",
                              sender="agent_1@maspy",
                              receiver="hub@spade",
                              content=f'delivery_done("{carrier}")',
                              priority=0))
```

**Código 1. Criação do canal, registro de rota e resposta a um sistema externo.**

## 4. Prova de Conceito

Este trabalho combina o desenvolvimento do *ExternalChannel* com sua validação por meio de uma prova de conceito. O cenário avaliado integra JaCaMo, SPADE e MASPY em uma aplicação de logística. As métricas observadas são a latência interna de injeção, o número de mensagens internas, a corretude da eleição e a latência total do fluxo. O experimento foi executado dez vezes, sem pretensão de avaliação estatística do desempenho. O experimento integra três plataformas multiagentes heterogêneas. O JaCaMo é baseado em Java, sobre Jason, CArtAgO e Moise; o SPADE é uma plataforma Python apoiada em XMPP; e o MASPY é um framework Python para agentes BDI. O cenário simula um fluxo de comércio eletrônico: um pedido de entrega é submetido a um leilão reverso de frete, e a transportadora vencedora aciona uma frota de veículos autônomos, que se auto-organiza para escolher o veículo que fará a entrega. O experimento exercita duas camadas de coordenação: uma externa, entre plataformas distintas via HTTP, e uma interna, entre veículos, por meio do mecanismo nativo de canais do MASPY.

### 4.1. Arquitetura do Cenário

As três plataformas têm papéis distintos. O JaCaMo atua como um comércio eletrônico, consultando o estado associado ao CEP, conduzindo o leilão reverso e acompanhando a entrega. O SPADE atua como central de transportadoras, calculando custos, escolhendo a transportadora de menor preço e acionando a frota. O MASPY modela três veículos autônomos, que recebem a ordem, executam uma eleição em anel e reportam o resultado. A comunicação entre JaCaMo e SPADE ocorre por meio de requisições HTTP no formato REST do JaCaMo, com mensagens em JSON. A comunicação envolvendo o MASPY ocorre por meio do *ExternalChannel*: com o SPADE, por mensagens baseadas em FIPA-ACL via HTTP; com o JaCaMo, por meio de um adaptador REST. Assim, o mesmo envelope base é usado para alcançar plataformas externas com formatos de saída distintos. Todas as plataformas executam em processos distintos no mesmo host; a heterogeneidade demonstrada é de plataforma, não de distribuição física. As versões empregadas foram Python 3.13.2, MASPY (`maspy-ml`) 2026.5.13, SPADE 4.1.2, Java 17 e JaCaMo 0.10-SNAPSHOT com jacamo-rest 0.5.

### 4.2. Fluxo de Execução

No lado MASPY, a frota é configurada de forma análoga ao Código 1: cria-se o canal externo com `ExternalChannel.create()`, registra-se a rota que converte o conteúdo recebido em crença de ordem de entrega, e conectam-se os três veículos ao canal externo e a um canal interno nativo, responsável pela eleição em anel.

A Figura 1 apresenta o fluxo completo da comunicação entre as três plataformas. O processo inicia no JaCaMo, que obtém o estado a partir de um CEP e emite uma chamada de propostas ao SPADE. O SPADE calcula os custos das transportadoras, com fator regional e variação aleatória, devolve as propostas e aciona a frota do MASPY por meio de um envelope FIPA-ACL com performativa request e prioridade urgente.

Ao receber a ordem, o *ExternalChannel* converte o conteúdo da mensagem em crença que dispara o plano de recepção em cada veículo. Como nenhum veículo é indicado como alvo, a ordem é entregue a todos os agentes conectados. Cada veículo sorteia uma distância até o destino e participa de um protocolo de eleição em anel pelo canal

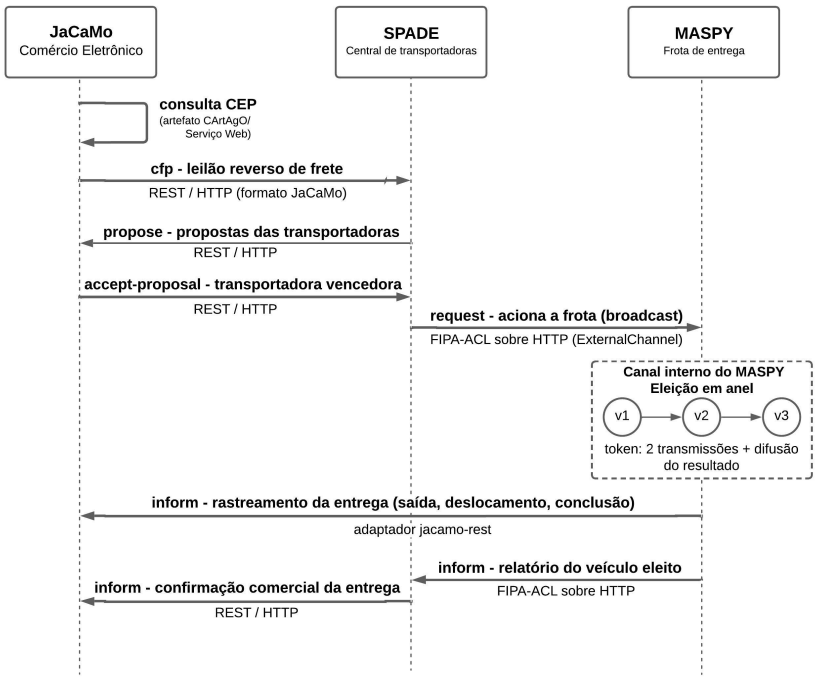


**Figura 1. Diagrama de sequência do fluxo entre JaCaMo, SPADE e MASPY, destacando a troca de mensagens baseadas em FIPA-ACL entre MASPY e SPADE e o rastreamento direto do MASPY ao JaCaMo por adaptador REST.**

interno do MASPY. O *token* percorre os veículos, acumula a menor distância observada e o último nó determina o vencedor e difunde o resultado à frota.

O veículo eleito executa a entrega, com duração proporcional à distância sorteada, e envia eventos de rastreamento diretamente ao JaCaMo por meio do adaptador REST, sem passar pelo SPADE. Ao concluir, reporta ao SPADE um relatório no envelope FIPA-ACL com a performativa inform; os veículos não eleitos reportam sua condição com prioridade normal. O SPADE consolida o relatório do vencedor e confirma a conclusão ao negociador do JaCaMo. O identificador de conversa gerado no envelope de entrada é preservado nas respostas ao SPADE, permitindo rastrear a troca por meio de um único identificador.

### 4.3. Métricas e Resultados

O desempenho foi medido por um coletor de métricas mantido fora do canal, instalado sobre o experimento sem modificar o *ExternalChannel*. O coletor registra marcações temporais ao longo do fluxo e permite medir a latência interna de injeção, o tempo e o número de mensagens do protocolo de anel, a corretude da eleição e a latência total. Neste trabalho, denomina-se latência interna de injeção o intervalo entre o enfileiramento da estrutura BDI pelo canal e o início da execução, pelo agente, do plano que recebe a ordem. As marcações decorrem de chamadas explícitas inseridas nos scripts do experimento, e o custo de cada uma limita-se à leitura do relógio do sistema. O experimento foi executado dez vezes; a primeira execução foi descartada como aquecimento, por apresentar latência de injeção atípica decorrente da inicialização do interpretador, e as métricas reportadas referem-se às nove execuções restantes. A Tabela 2 consolida os resultados.

**Tabela 2. Resultados da prova de conceito em nove execuções.**

| Métrica | Resultado |
|---|---|
| Latência interna de injeção | média 1,28 ms; mín. 1,09 ms; máx. 1,56 ms |
| Latência do canal nativo (*token* A–B) | média 0,81 ms; mín. 0,59 ms; máx. 1,01 ms |
| Tempo do protocolo de anel | média 0,002 s (2 ms) |
| Mensagens internas | 5 por execução |
| Eleição correta | 9/9 execuções |
| Latência total do fluxo | média 1,26 s; mín. 0,61 s; máx. 2,61 s |
| Tempo médio de entrega simulada | 1,06 s |

A latência interna de injeção inclui a travessia da fila e a ativação inicial do plano. O transporte de rede e a desserialização do envelope ocorrem antes desse intervalo e não estão incluídos nesse intervalo. A resposta de aceitação é enviada antes da execução do plano, de modo que parte desse intervalo não é percebida pelo emissor da requisição.

Como referência interna, o canal nativo apresentou 0,81 ms no mesmo ambiente e intervalo, frente a 1,28 ms no *ExternalChannel*, diferença de 0,47 ms. O valor caracteriza o custo adicional do protótipo em relação ao MASPY nativo, mas não constitui comparação com outras soluções.

O protocolo de eleição em anel apresentou um número fixo de mensagens. A métrica de tokens registrou duas transmissões entre veículos em todas as execuções, correspondentes ao percurso pelos demais nós do anel; somadas as três mensagens de difusão do resultado à frota, são cinco mensagens internas por execução, valor determinístico para um anel de três nós, reportado como verificação de conformidade do protocolo, e não como medição de desempenho. A eleição foi correta em todas as nove execuções, com o veículo de menor distância sendo eleito em cada uma delas.

A latência total do fluxo não deve ser interpretada como sobrecarga de comunicação, pois é dominada pelo tempo de entrega simulado. Os resultados indicam que o *ExternalChannel* insere o MASPY em um fluxo com plataformas heterogêneas, alcançando o SPADE por meio de mensagens baseadas em FIPA-ACL e o JaCaMo por meio do adaptador correspondente. O código-fonte, os scripts da prova de conceito e os dados das execuções, com as versões das dependências e as instruções de reprodução, estão disponíveis em `https://github.com/externalchannel/provadeconceito`.

## 5. Discussão e Limitações

Os resultados indicam que o *ExternalChannel* insere agentes do MASPY em um fluxo de coordenação com plataformas heterogêneas, preservando o mecanismo nativo de canais. O protocolo de eleição em anel operou pelo canal interno enquanto o canal externo funcionava em paralelo.

Diante da lacuna identificada no mapeamento da Seção 2, o PEAK-ACL é o trabalho de maior proximidade técnica com a proposta [Bossa et al. 2026], por estabelecer a troca de mensagens baseada em FIPA-ACL via HTTP entre frameworks como SPADE e JADE. O *ExternalChannel* atua em outro ponto: integra-se ao MASPY por herança da classe de comunicação e converte as mensagens recebidas em estruturas BDI injetadas no ciclo de raciocínio. Assim, as propostas são complementares. Com o PEAK-ACL, o agente compõe e manipula objetos de mensagem ACL diretamente em seu código, sobre

um transporte HTTP-MTP simétrico; com o *ExternalChannel*, o agente MASPY opera apenas sobre crenças e objetivos, e a tradução de e para o envelope FIPA-ACL ocorre na fronteira do canal, ponto não coberto por nenhum dos trabalhos analisados.

A validação baseia-se em uma prova de conceito, em um único cenário e em ambiente controlado, com a maior parte dos processos executada localmente. As métricas são consistentes ao longo das nove execuções, mas não abrangem uma avaliação sob carga, com mensagens concorrentes, falhas parciais ou variações de escala. A latência total é dominada pelo tempo de entrega simulado, o que limita o que se pode inferir sobre o comportamento do canal em operação real. Há ainda limitações do componente em si. A entrega segue a semântica *at-most-once*, sem persistência nem reenvio. A segurança da injeção sob chamadas concorrentes não foi verificada por meio de um teste de estresse. A camada de transporte não emprega TLS no protótipo. A comunicação com o JaCaMo usa um adaptador que traduz a *FIPAMessage* para o formato da camada REST da plataforma, e não FIPA-ACL diretamente interpretável, e o conjunto de performativas corresponde a um subconjunto da especificação. Por fim, o experimento é executado em um único host: a heterogeneidade demonstrada é de plataforma, não de distribuição física, e latências de rede reais alterariam as medições. Também existem limitações herdadas do ambiente de execução. O MASPY executa cada agente em uma *thread* sobre o interpretador de referência do Python, cujo Global Interpreter Lock (GIL) impede a execução paralela em múltiplos núcleos. Em configuração centralizada, o tempo de resposta do framework aumenta acentuadamente com o número de agentes [Mellado et al. 2025]. O *External-Channel* não altera esse comportamento, pois opera no mesmo modelo de execução. Em cenários com muitos agentes ou alta taxa de mensagens, esses fatores podem superar o custo de injeção do canal. O componente está em estágio de protótipo: executou o cenário de ponta a ponta, mas sua interface de programação e parte da arquitetura ainda estão sujeitas a refinamento.

## 6. Conclusão

Este trabalho apresentou o *ExternalChannel*, um componente que adiciona ao MASPY conectividade HTTP e suporte a mensagens baseadas em FIPA-ACL por herança da classe de comunicação. O componente traduz mensagens externas em estruturas BDI, encaminha mensagens de saída por meio de adaptadores e preserva o mecanismo nativo de canais. A viabilidade foi demonstrada em uma prova de conceito com JaCaMo, SPADE e MASPY, com latência interna de injeção média de 1,28 ms, frente à média de 0,81 ms do canal nativo nas mesmas condições experimentais.

Como trabalhos futuros, pretende-se ampliar a avaliação com cenários adicionais, execução distribuída, mensagens concorrentes, falhas parciais e um número maior de agentes autônomos por plataforma, com análise estatística e comparação direta com soluções relacionadas; estender a cobertura da FIPA-ACL com novos atos comunicativos e adaptadores, incluindo suporte ao FIPA-MTP para alcançar outras plataformas; e amadurecer o componente como software, com estabilização da interface de programação e possível incorporação à distribuição oficial do MASPY.

**Agradecimentos:** Este trabalho tem financiamento parcial do projeto: 444568/2024-7, CNPq/MCTI/FNDCT Nº 22/2024, *Programa Conhecimento Brasil – Apoio a Projetos em Rede com Pesquisadores Brasileiros no Exterior.*

## Referências


Alves, P., Gomes, D., Rodrigues, C., Carneiro, J., Novais, P., and Marreiros, G. (2022). Grouplanner: A group recommender system for tourism with multi-agent microservices. In *PAAMS 2022*, volume 13616 of *Lecture Notes in Computer Science*, pages 454–460. Springer, Cham.

Bhosale, V. and Gawande, R. (2025). Replacing web interfaces with intelligent multi-agent REST API orchestration. In *2025 IEEE International Conference on Advanced Systems and Emergent Technologies (IC_ASET)*, pages 1–6, Hammamet-Yasmine, Tunisia.

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.

Bossa, S., Ribeiro, B., Santos, G., Gomes, L., and Vale, Z. (2026). PEAK-ACL: A FIPA-compliant agent communication package for python-based multi-agent systems. *SoftwareX*, 34:102647.

Bratman, M. E. (1987). *Intention, Plans, and Practical Reason*. Harvard University Press, Cambridge, MA.

Chopra, A. K. and Singh, M. P. (2026). Fluid: Social norms-based multiagent systems on the web. In *EMAS 2025*, volume 16407 of *Lecture Notes in Computer Science*, pages 62–79. Springer, Cham.

Finin, T., Fritzson, R., McKay, D., and McEntire, R. (1994). KQML as an agent communication language. In *Proceedings of the Third International Conference on Information and Knowledge Management (CIKM '94)*, pages 456–463, New York, NY, USA. ACM.

Ihejimba, C. and Wenkstern, R. Z. (2024). A cloud-based microservices solution for multi-agent traffic control systems. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2024)*, page 889–897, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.

Kitchenham, B. A. and Charters, S. (2007). Guidelines for performing systematic literature reviews in software engineering. Technical Report EBSE-2007-01, Keele University and University of Durham, Keele, UK.

Kouissi, M., En-Naimi, E. M., El Ghouch, N., and Zouhair, A. (2021). Computing platform architecture for management of renewable resources using multi-agents system based on the case based reasoning approach. *Journal of Theoretical and Applied Information Technology*, 99(2).

Louck, Y., Stulman, A., and Dvir, A. (2026). Security analysis of agentic AI communication protocols: A comparative evaluation. *ACM Transactions on AI Security and Privacy*.

Mellado, A. L. L., Borges, A. P., and Alves, G. V. (2025). MASPY: A Python-based framework for developing BDI multi-agent systems. In *Advances in Practical Applications of Agents, Multi-Agent Systems, and Computational Social Science: The PAAMS Collection*, pages 216–227. Springer-Verlag, Berlin, Heidelberg.

Mellado, A. L. L., Neres, G. G., Borges, A. P., Cardoso, R. C., and Alves, G. V. (2026). MASPY: A Python framework for developing BDI agents with reinforcement learning. In *Proceedings of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2026), Demonstration Track*, page 4074–4076, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.

Palanca, J., Terrasa, A., Julian, V., and Carrascosa, C. (2020). SPADE 3: Supporting the new generation of multi-agent systems. *IEEE Access*, 8:182537–182549.

Poslad, S. and Charlton, P. (2001). Standardizing agent interoperability: The FIPA approach. In *Multi-Agent Systems and Applications*, volume 2086 of *Lecture Notes in Artificial Intelligence*, pages 98–117. EASSS, Prague, Czech Republic.

Rezaei, H., Rabhi, F., and Beheshti, A. (2025). GWise: A graph-structured multi-agent framework for service-oriented and generative-ai-enabled financial trading analytics. In *IEEE International Conference on Web Services (ICWS 2025)*, pages 1043–1052, Helsinki, Finland. IEEE.

Santos, G., Canito, A., Carvalho, R., Pinto, T., Vale, Z., Marreiros, G., and Corchado, J. M. (2021). Semantic services catalog for multiagent systems society. In *PAAMS 2021*, volume 12946 of *Lecture Notes in Artificial Intelligence*, pages 229–240. Springer International Publishing, Cham.

Schubotz, R., Spieldenner, T., and Chelli, M. (2022). stigLD: Stigmergic coordination in linked systems. *Mathematics*, 10(7):1041.

Todorov, J., Stoyanov, I., Stoyanov, S., Stoyanova-Doycheva, A., Tabakova-Komsalova, V., and Wassilew, K. (2025). Agent-based air monitoring platform. In *IEEE BdKCSE 2025*, pages 1–8, Bankya, Bulgaria. IEEE.

Vachtsevanou, D., Ciortea, A., Mayer, S., and Lemée, J. (2023). Signifiers as a first-class abstraction in hypermedia multi-agent systems. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2023)*, pages 1200–1208, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.

Vaziry, A., Garzon, S. R., and Küpper, A. (2026). Enhancing the A2A protocol with blockchain-based identities and x402 micropayments for agentic ai. In *Computational Intelligence*, pages 454–472, Cham. Springer Nature Switzerland.

Wooldridge, M. (2009). *An Introduction to MultiAgent Systems*. John Wiley & Sons, Chichester, 2 edition.

Zouad, S. and Boufaida, M. (2025). Enhancing distributed system performance with an intelligent multi-agent microservices architecture. *International Journal of Interactive Mobile Technologies*, 19(24):76–93.