FEDERAL UNIVERSITY OF SANTA CATARINA
TECHNOLOGICAL CENTER
AUTOMATION AND SYSTEMS ENGINEERING POSTGRADUATE PROGRAM

Bruno Souza de Lima

# Action Handling by BDI Agents in Hypermedia Environments

Florianópolis
2024

Bruno Souza de Lima

# Action Handling by BDI Agents in Hypermedia Environments

Thesis submitted to the Automation and Systems Engineering Postgraduate Program from the Federal University of Santa Catarina for obtaining the Master Degree in Automation and Systems Engineering.
Supervisor: Prof. Jomi Fred Hübner, Dr. Eng.

Florianópolis
2024

Ficha catalográfica gerada por meio de sistema automatizado gerenciado pela BU/UFSC.
Dados inseridos pelo próprio autor.

de Lima, Bruno Souza
Action Handling by BDI Agents in Hypermedia Environments / Bruno Souza de Lima ; orientador, Jomi Fred Hübner, 2024.
87 p.

Dissertação (mestrado) - Universidade Federal de Santa Catarina, Centro Tecnológico, Programa de Pós-Graduação em Engenharia de Automação e Sistemas, Florianópolis, 2024.

Inclui referências.

1. Engenharia de Automação e Sistemas. 2. Belief-Desire Intention agents. 3. Practical reasoning. 4. Multi-Agent Systems. 5. Action reasoning. I. Hübner, Jomi Fred. II. Universidade Federal de Santa Catarina. Programa de Pós Graduação em Engenharia de Automação e Sistemas. III. Título.

Bruno Souza de Lima

**Action Handling by BDI Agents in Hypermedia Environments**

O presente trabalho em nível de Mestrado/Doutorado foi avaliado e aprovado, em 11 de dezembro de 2023, pela banca examinadora composta pelos seguintes membros:

Prof. Alison Roberto Panisson, Dr. Eng.
Federal University of Santa Catarina

Prof. Carlos Eduardo Pantoja, Dr. Eng.
Federal Center for Technological Education Celso Suckow da Fonseca

Prof. Cleber Jorge Amaral, Dr. Eng.
Federal Institute of Santa Catarina

Certificamos que esta é a versão original e final do trabalho de conclusão que foi julgado adequado para obtenção do título de Mestre em Engenharia de Automação e Sistemas.


Documento assinado digitalmente
**Julio Elias Normey Rico**
Data: 06/03/2024 09:02:19-0300
CPF: ***.840.859-**
Verifique as assinaturas em https://v.ufsc.br

Prof. Júlio Elias Normey Rico, Dr. Eng.
Coordenador do Programa


Documento assinado digitalmente
**Jomi Fred Hubner**
Data: 05/03/2024 16:17:17-0300
CPF: ***.334.769-**
Verifique as assinaturas em https://v.ufsc.br

Prof. Jomi Fred Hübner, Dr. Eng.
Orientador

Florianópolis, 2024.

I dedicate this Thesis to my parents, Marília and Ricardo, and to my girlfriend, Danielly.

## ACKNOWLEDGMENTS

Finally, I am grateful to my parents, Marília and Ricardo, for providing everything I needed to reach this point. Their motivation and efforts to make me comfortable have greatly contributed to my dedication to this thesis.

I would like to thank my girlfriend, Danielly, for her patience during the many nights I had to dedicate to this work, limiting our time together, and for her unwavering support.

I would like to express my sincere gratitude to my supervisor, Professor Jomi Fred Hübner, for all the support during my master's program. His patience during my difficult times, the motivation he provided, and his vast knowledge have been invaluable to my studies.

I also wish to extend my thanks to Professor Andrei Ciortea and Ph.D. candidate Danai Vachtsevanou from the University of St.Gallen. They were essential in numerous discussions about my thesis topic, their work formed the basis of my thesis, and their substantial support and help were crucial in accomplishing this work. I hope that my work has also been useful for them in some way.

Finally, I would like to thank CAPES for giving financial support for this work.

*I find myself shaking my head and wondering, "Where are all the agents?"*
*(Hendler, 2007)*

## ABSTRACT

The integration of Belief-Desire-Intention (BDI) agents within dynamic and evolving environments is a crucial advancement in multi-agent systems. Conventionally, these agents are designed to operate known artifacts with predefined actions within static environments, a limitation that hinders their applicability in real-world contexts. This study introduces an approach that enhances BDI agents' ability to navigate and act within open, dynamic, and long-lived environments: hypermedia environments. Our solution rests on three foundational pillars: action knowledge, action reasoning, and action execution, each supported by environmental cues known as signifiers in the field of design of things. This approach integrates a mechanism for the resolution of signifiers perceived in run-time, enriching the traditional BDI reasoning cycle. This enhancement empowers agents to adapt their decision-making processes based on available and suitable action possibilities within their environment. To validate our method, we conducted experiments using Jason agents in a hypermedia environment, where they interacted with these signifiers. The results from these experiments demonstrate that our approach improves the agents' ability to dynamically reason about actions. This leads to more effective and efficient goal achievement in environments characterized by their fluidity and unpredictability.


**Keywords**: Belief-Desire-Intention agents. Multi-Agent Systems. Web of Things. Practical reasoning. Action reasoning.

## RESUMO

A integração de agentes *Belief-Desire-Intention* (BDI) em ambientes dinâmicos e em evolução é um avanço crucial nos Sistemas Multiagentes (MAS). Convencionalmente, esses agentes são projetados para operar artefatos conhecidos com ações predefinidas em ambientes estáticos, uma limitação que dificulta sua aplicabilidade em contextos do mundo real. Este trabalho introduz uma abordagem que aprimora a capacidade dos agentes BDI de navegar e atuar em ambientes abertos, dinâmicos e duradouros: Ambientes de Hipermídia (*Hypermedia Environments*). Nossa solução se baseia em três pilares fundamentais: Conhecimento de Ação (*Action Knowledge*), Raciocínio de Ação (*Action Reasoning*) e Execução de Ação (*Action Execution*), cada um apoiado por pistas ambientais conhecidas como *signifiers* no campo de design das coisas. Esta abordagem integra um mecanismo para a resolução de *signifiers* percebidos em tempo de execução, enriquecendo o ciclo de raciocínio BDI tradicional. Este aprimoramento capacita os agentes a adaptar seus processos de tomada de decisão com base nas possibilidades de ação disponíveis e adequadas dentro de seu ambiente. Para validar nossa solução, conduzimos experimentos usando agentes Jason em um Ambiente de Hipermídia, onde interagiram com esses *signifiers*. Os resultados desses experimentos demonstram que nossa abordagem melhora a capacidade dos agentes de raciocinar dinamicamente sobre ações. Isso leva a uma realização de objetivos mais eficaz e eficiente em ambientes caracterizados por sua fluidez e imprevisibilidade.


**Palavras-chave**: Agentes BDI. Sistemas Multi-Agentes. Web das Coisas. Raciocínio prático. Raciocínio sobre ações.

## RESUMO EXPANDIDO

### Introdução

À medida que avançamos nos estudos sobre Sistemas Multiagentes (MAS), a capacidade dos agentes de deliberar e tomar decisões de forma autônoma em ambientes dinâmicos e abertos tornou-se essencial. Utilizando agentes *Belief-Desire-Intention* (BDI), que equilibram comportamentos proativos e reativos, os agentes ajustam suas ações em resposta a mudanças ambientais, como um assistente doméstico inteligente que regula a luminosidade e temperatura para manter o conforto. O desafio aumenta em ambientes complexos, como a Internet das Coisas, onde os agentes precisam interagir com uma variedade de sensores e atuadores desconhecidos até o momento de sua execução. Os Sistemas Multiagentes Hipermedia emergem como uma resposta a essas mudanças, onde os agentes podem descobrir e interagir com artefatos em tempo real. Isso abre novas possibilidades para os agentes se adaptarem e responderem às dinâmicas ambientais em tempo real, como identificar e operar uma lâmpada automaticamente ao entrar em uma sala. Propomos uma abordagem inovadora que melhora a tomada de decisão dos agentes ao permitir a incorporação eficaz de informações ambientais, utilizando ontologias para descrever o ambiente. Isso permite que os agentes alinhem seus planos com o estado atual do ambiente, otimizando a execução de suas ações.

### Objetivos

Este estudo tem como objetivo propor uma abordagem para equipar agentes com a capacidade de perceber, raciocinar e executar ações de forma eficaz em ambientes hipermedia, além de reconhecer quando não existem ações viáveis para alcançar seus objetivos. Assim, o objetivo é encontrar uma solução para três pilares: conhecimento de ação, raciocínio sobre ação e execução de ação. Contudo, assume-se que os ambientes e ações dos agentes podem ser totalmente descritos usando ontologias, o que representa uma limitação diante da natureza dinâmica e imprevisível dos ambientes hipermedia. Essa dependência de ontologias pode restringir a adaptabilidade e eficácia dos agentes em cenários não previstos, destacando a necessidade de pesquisas contínuas para abordagens mais dinâmicas e flexíveis na gestão de ontologias e representação de conhecimento. Além disso, como objetivos secundários buscamos criar uma extensão para AgentSpeak e o incentivar à colaboração comunitária, contribuindo para o avanço da tecnologia de agentes e o engajamento da comunidade de desenvolvedores.

### Metodologia

Na avaliação do nosso estudo, integramos signifiers e um Mecanismo de Resolução de Significados (SRM) ao ciclo de raciocínio dos agentes Jason para testar a eficácia dessa abordagem em um Sistema Multiagente (MAS). Utilizamos agentes diferenciados pela capacidade de integrar e raciocinar com signifiers em cenários variados. A análise foca na adaptação dos agentes para incluir o SRM, utilizando uma biblioteca Java alinhada à ontologia hMAS. Os agentes, denominados A, B, C e D, são avaliados em cenários que vão desde situações estáticas até contextos dinâmicos, exigindo adaptação e raciocínio avançado. O Agente A, sem SRM, serve como controle, operando em um ciclo de raciocínio convencional. O Agente B, com um SRM básico, adapta-se à disponibilidade imediata de ações. O Agente C, incorporando avaliação de contexto ao SRM, alinha suas ações com o contexto ambiental. Por fim, o Agente D utiliza uma versão avançada do SRM, considerando a disponibilidade de ação, contexto e compatibilidade de habilidades.

Avaliamos os agentes sob dois critérios principais: a capacidade de alcançar objetivos definidos internamente (Objetivos do Agente) e a capacidade de atender aos objetivos de design do ambiente (Objetivos do Ambiente). Isso inclui o alinhamento das ações do agente com as intenções do designer do ambiente e a racionalidade das ações considerando o contexto ambiental. Os cenários de avaliação começam com um ambiente estático, onde todos os agentes devem reduzir a iluminância para conservar energia. Progressivamente, introduzimos dinamismo e complexidade, como artefatos que podem se tornar inoperantes, exigindo que os agentes adaptem seus planos. Em cenários mais avançados, os agentes enfrentam divergências entre os objetivos internos e as expectativas do ambiente, como a necessidade de conservar energia sem comprometer o conforto dos ocupantes. No cenário mais complexo, os agentes manipulam braços robóticos com especificações técnicas distintas, desafiando-os a reconhecer e se adaptar às complexidades técnicas e operacionais.

**Resultados e Discussão**

A avaliação realizada em quatro cenários distintos gerou resultados que confirmaram nossas expectativas, evidenciando que a integração de um Mecanismo de Resolução de Significados aprimora significativamente a adaptabilidade e a tomada de decisão dos agentes em ambientes complexos. Todos os agentes conseguiram atender aos objetivos no cenário estático inicial, mas nos cenários subsequentes, que introduziram dinâmicas ambientais e exigências de raciocínio contextual e técnico, os agentes com SRM integrado (B, C e D) se destacaram. O Agente D, com o SRM mais avançado, demonstrou ser excepcionalmente adaptável, superando os outros em todos os cenários de complexidade crescente. Entendemos que em sistemas maiores, com mais agentes e signifiers podem haver desafios como a complexidade na interpretação e aplicação de signifiers, a dependência do design ambiental e da precisão dos signifiers, além de preocupações com o overhead computacional e a coordenação de agentes. Reconhecemos limitações, como a premissa de que os agentes possuem conhecimento prévio dos tipos de parâmetros para ações, e sugerimos pesquisas futuras focadas na validação de tipos de parâmetros em tempo de execução, no armazenamento otimizado de conhecimento sobre signifiers e na análise abrangente de ações em subplanos. Essas áreas de pesquisa futura abrem caminho para avanços significativos no comportamento dos agentes dentro de ambientes abertos e dinâmicos.

**Considerações Finais**

Este trabalho teve como objetivo aprimorar as capacidades de manipulação de ações e tomada de decisão de agentes BDI em ambientes hipermedia dinâmicos, equipando-os com a habilidade de perceber, avaliar e executar eficazmente ações, além de discernir situações onde não existem ações disponíveis para atingir seus objetivos. Focamos no Conhecimento de Ação, Raciocínio sobre Ação e Execução de Ação, com o apoio de signifiers e a implementação do Mecanismo de Resolução de Significados. Essa abordagem melhorou a adaptabilidade dos agentes, permitindo-lhes navegar e responder a ambientes em evolução com capacidades de decisão mais robustas. Nossa implementação demonstrou que agentes com essas capacidades têm mais chances de alcançar com sucesso tanto os objetivos dos agentes quanto os do ambiente. Nosso trabalho também oferece uma contribuição valiosa para a comunidade, com uma extensão de Jason que pode ser utilizada e desenvolvida por pesquisadores e desenvolvedores. Olhando para o futuro, existem várias direções para pesquisas. Será importante abordar as limitações atuais, como a complexidade na interpretação de signifiers e as demandas computacionais do SRM. Trabalhos futuros poderiam explorar a descoberta e validação dinâmica de parâmetros de ação, o geren-

ciamento otimizado de conhecimento sobre signifiers e melhorias na coordenação entre agentes. Além disso, garantir a interoperabilidade e padronização em sistemas diversos e aprimorar o desempenho dos agentes em cenários mais complexos serão áreas chave de foco.

**Palavras-chave**: Agentes BDI. Sistemas Multi-Agentes. Web das Coisas. Raciocínio prático. Raciocínio sobre ações.

## LIST OF FIGURES

Figure 1 – Multi-Agent System dimensions. Source: [16]. . . . . 25
Figure 2 – Distributed hypermedia environment. Source: [7]. . . . . 32
Figure 3 – Process of Action Reasoning. The inputs (Relevant plans, Signifiers and Beliefs from the Belief Base) are passed to an applicable plan selection function embedded with the SRM, returning an applicable plan as output. 45
Figure 4 – Jason Reasoning Cycle. Source: [10]. . . . . 54
Figure 5 – Integration of Tools . . . . . 67

## LIST OF TABLES

Table 1 – Features of the different Signifier Resolution Mechanism (SRM) versions employed by Belief-Desire-Intention (BDI) agents in our evaluation. . . 62
Table 2 – Scenario results. The first mark represents the Agent Objective, and the second mark the Environment Objective. . . . . 74

## LIST OF ABBREVIATIONS AND ACRONYMS

| | |
|---|---|
| AnA | Agents and Artifacts |
| API | Application Programming Interface |
| BDI | Belief-Desire-Intention |
| E4MAS | Environments for Multi-Agent Systems |
| HATEOAS | Hypermedia as the Engine of Application Stat |
| HCTL | Hypermedia Controls Ontology |
| hMAS | Hypermedia Multi-Agent Systems |
| LSMAS | Large-Scale Multi-Agent System |
| LSS | Large-Scale Systems |
| MAS | Multi-Agent Systems |
| OOP | Object-Oriented Programming |
| RDF | Resource Description Framework |
| ROA | Resource-Oriented Architectures |
| SOA | Service-Oriented Architectures |
| SRM | Signifier Resolution Mechanism |
| TD | Thing Description |
| W3C | World Wide Web Consortium |
| WoT | Web of Things |

## CONTENTS


**1 INTRODUCTION . . . . . 17**
1.1 BACKGROUND OF AGENTS ON HYPERMEDIA ENVIRONMENTS . . 17
1.2 PROBLEM STATEMENT . . . . . 19
1.3 OBJECTIVE OF THE STUDY . . . . . 19
1.4 SIGNIFICANCE AND CONTRIBUTION . . . . . 20
1.5 LIMITATIONS . . . . . 21
1.6 STRUCTURE OF THE THESIS . . . . . 21
**2 LITERATURE REVIEW . . . . . 23**
2.1 OVERVIEW OF MULTI-AGENT SYSTEMS . . . . . 23
2.2 OVERVIEW OF MULTI-AGENT SYSTEM'S ENVIRONMENTS . . . . . 26
**2.2.1 Agents and Artifacts Conceptual Model . . . . . 26**
**2.2.2 Artifacts in MAS: Operations, Properties, and Signals . . . . . 27**
**2.2.3 Large-Scale and Open Environments . . . . . 28**
2.3 REPRESENTATION OF ACTIONS . . . . . 29
**2.3.1 Universal Representation of Actions . . . . . 29**
**2.3.2 Actions in the Web of Things . . . . . 30**
2.4 OVERVIEW OF AGENTS ON THE WEB . . . . . 31
2.5 AGENT REASONING ABOUT THE ENVIRONMENT . . . . . 33
**2.5.1 General Reasoning Process . . . . . 33**
**2.5.2 Action reasoning . . . . . 35**
**2.5.3 Action Signifiers for MAS Environment . . . . . 36**
**3 PROPOSAL ACTION HANDLING BY BDI AGENTS . . . . . 38**
3.1 ACTION KNOWLEDGE . . . . . 38
**3.1.1 Acquisition of Action Knowledge . . . . . 39**
**3.1.2 Storage of Action Knowledge . . . . . 41**
3.2 ACTION REASONING . . . . . 42
**3.2.1 Factors influencing action reasoning . . . . . 43**
**3.2.2 Process of Action Reasoning . . . . . 44**
3.3 ACTION EXECUTION . . . . . 49
**3.3.1 Preparation for Action Execution . . . . . 49**
**3.3.2 Process of Action Execution . . . . . 50**
3.4 IMPLEMENTATION AND INTEGRATION WITH JASON/JACAMO . . 50
**3.4.1 Implementation of Action Knowledge . . . . . 50**
3.4.1.1 CArtAgO Artifact for hMAS Resource Profile . . . . . 51
**3.4.2 Implementation of Action Reasoning . . . . . 53**
**3.4.3 Implementation of Action Execution . . . . . 59**
**4 EVALUATION . . . . . 61**

4.1 VARIABLES . . . . . 61
4.2 EVALUATION CRITERIA . . . . . 62
4.3 SCENARIO DESCRIPTIONS AND EXPECTED OUTCOMES . . . . . 64
4.4 SETTING UP EXPERIMENTS . . . . . 65
**4.4.1 Agent-Oriented Programming Language . . . . . 66**
**4.4.2 Hypermedia Environment . . . . . 66**
**4.4.3 Graphics Engine . . . . . 66**
**4.4.4 Application and Deployment Context . . . . . 67**
4.5 RESULTS OVERVIEW . . . . . 68
**5 DISCUSSION . . . . . 75**
5.1 ADVANTAGES OF THE PROPOSED APPROACH . . . . . 75
5.2 CHALLENGES OF THE PROPOSED APPROACH . . . . . 76
5.3 LIMITATIONS AND FUTURE RESEARCH . . . . . 78
**5.3.1 Other recommended themes . . . . . 79**
5.4 CONTRIBUTIONS . . . . . 79
**6 CONCLUSION . . . . . 81**
**Bibliography . . . . . 82**

# 1 INTRODUCTION

As the studies in the field of Multi-Agent Systems (MAS) continue to progress, the capability of agents to deliberate upon goals and make decisions autonomously has become crucial, particularly in dynamic, open, and long-lived environments. Looking at the BDI model for agents as our focus of attention, it operates on a sense-reason-act cycle, striking a balance between proactive and reactive behaviors to achieve their objectives. This is evident in scenarios like an autonomous vacuum cleaner adjusting its cleaning strategy based on real-time environmental changes. However, the intricacy of these tasks escalates in open environments such as the Internet of Things or the Web of Things (WoT) [1], where agents encounter a vast array of sensors and actuators, often external to their own structure and unknown until run-time. This raises significant challenges, as exemplified by a personal assistant agent in a smart home, that, asked to keep the house's comfort level by adjusting luminosity and temperature, necessitates the ability to dynamically acquire and integrate information about available tools to inform and adapt its reasoning and planning processes.

Navigating through such complex and unpredictable environments requires agents to not only sense and act but also to *reason* about actions in a sophisticated manner, considering the dynamic nature of the environment. In this context, the work seeks to propose a novel approach enabling agents to effectively incorporate environmental information within their sense-reason-act cycle, enhancing their decision-making process, assuming ontologies are used to describe the environment. By doing so, agents can make more informed decisions, aligning their internal plans with the real-time state of the environment, and ensuring a smoother and more efficient execution of actions. This approach emphasizes the reasoning aspect of the cycle, ensuring that agents are not just reactive to changes but are also proactively adjusting their plans and actions based on a comprehensive understanding of the environment. This advancement aims to provide a robust framework for autonomous agents, supporting their performance in open, dynamic and long-lived environments.

## 1.1 BACKGROUND OF AGENTS ON HYPERMEDIA ENVIRONMENTS

For more than two decades, there have been numerous studies on how to program and utilize agents on the Web to accomplish a variety of tasks [2–5]. The focus on web agents can be traced back to the inception of the Semantic Web concept proposed by Tim Berners-Lee, a visionary who envisions a pivotal role for agents in the evolution of the Web [6]. Agents, for instance, would have the capability to schedule medical appointments based on data regarding medical facilities and available time slots, since this information would be accessible on the Semantic Web. However, it is worth noting that contrary to Tim Berners-Lee's prognostications, agents, particularly intelligent agents, have not gained

widespread adoption or prominence on the Web [7].

Agents distinguish themselves from both functional programs characterized by well-defined inputs and outputs and reactive programs, such as online banking systems. Intelligent agents, in addition to being reactive, exhibit characteristics of autonomy, proactiveness, and social ability [8]. This means that when assigned a task, an intelligent agent endeavors to autonomously discern the optimal method for achieving its objective. Importantly, this method selection is not mandated to be explicitly specified within the agent's code [9]. While various conceptions exist for constructing intelligent agents, the BDI [10] software paradigm stands as one of the most widely employed. Within this paradigm, an agent's Beliefs represent its informational state, encapsulating its perception of the world and its current knowledge. Desires denote the agent's objectives, articulating its intentions and goals. Lastly, Intentions serve as the agent's commitments, signifying what it has resolved to pursue.

Despite the inherent advantages of employing agents over conventional software systems, their presence on the Web remains notably scarce. In response to this conspicuous absence, Hendler posed the fundamental inquiry, "Where are all the intelligent agents?" [11], a question raised years after the conceptualization of the Semantic Web. The lack of agents on the Web can likely be attributed to the scarcity of scenarios wherein they constitute the optimal solution. Agents continue to await situations wherein they can demonstrate their effectiveness as a solution. Recent transformations in the Web landscape, marked by the development of Application Programming Interface (API)s incorporating the Hypermedia as the Engine of Application Stat (HATEOAS) principle [12], along with initiatives such as the World Wide Web Consortium (W3C) WoT [1] and Linked Data [13], have significantly reshaped the Web in recent years. These evolving characteristics of the Web have the potential to establish dynamic, open, and long-lived systems as the new norm on the Web, creating an environment ideally suited for the deployment of intelligent agents [7].

In light of these developments, a novel form of MAS known as the Hypermedia MAS has emerged [14]. Within this new class of MAS, agents are designed to navigate and operate within hypermedia environments while fulfilling their assigned tasks. Within the context of a hypermedia environment, agents dynamically detect and interact with artifacts during run-time, by parsing an exposed formal representation of the artifact present in the hypermedia environment, such as the Resource Description Framework (RDF). These artifacts represent real or conceptual environment resources within the environment, enabling agents to engage with them [9]. Such interactions may encompass tasks such as identifying properties or executing actions related to the artifact. To illustrate, consider an artifact like a light bulb, with its observable property being its state (i.e., on or off), and the corresponding action being the act of toggling it on or off. In the Hypermedia

[1] `https://www.w3.org/WoT/documentation/`

MAS, the agent should be able to enter the room, and at run-time discover the lightbulb artifact, how to interact with it, and use these new findings in its deliberations. The agent should work even if the developer doesn't know, at design-time, the existence of the artifact.

## 1.2 PROBLEM STATEMENT

Hypermedia environments are characterized by their dynamic and long-lived nature. Given these attributes, it is difficult for a developer to base the programming of an agent on an unknown and open environment. The agent must possess the capability to identify and reason about the available actions it can perform in this environment, exhibiting an intrinsic adaptive capacity. In the context of these environments, actions may spontaneously be available or not during run-time, necessitating a flexible, adaptive design in the agent's architecture.

In the current landscape, the majority of MASs are engineered for a known environment, and most efforts have focused on solving specific problems with specific solutions [15]. Such systems, while beneficial in more stable settings, may not adequately address the dynamic and evolving parameters of hypermedia environments. Consequently, a strategic reconsideration of the underlying approach is needed to adapt to these dynamic contexts more effectively.

## 1.3 OBJECTIVE OF THE STUDY

The primary objective of this study is to propose an innovative approach wherein agents are equipped with the capability to perceive, reason about, and effectively execute available actions within a hypermedia environment. Additionally, the study aims to empower agents with the ability to discern instances where no viable actions exist to fulfill their intended objective.

Due to the open and dynamic nature of the hypermedia environments, several developers can work with various agents and create, modify, and delete multiple objects made in the environment at any moment. In this kind of environment, to deal with actions, an agent must undertake several processes, which may be characterized by three pillars:

- **Action Knowledge:** How the agent becomes aware of the available actions and how it stores this information.
- **Action Reasoning:** How the agent determines the most appropriate action to its goals.
- **Action Execution:** How the agent executes the action properly.

It is crucial to assess the efficacy and practicality of the approach. To this end, an evaluation will be conducted through the deployment of a system where JaCaMo [16] agents are tasked to enter unknown environments, discover, reason upon, and perform actions to fulfill their goals. JaCaMo was chosen as the agent and environment language for some reasons: MAS implementations basing this work are built upon JaCaMo; it easily integrates Agents and Environments; the authors have more familiarity with the tool.

Furthermore, an additional objective is to implement an extension of Jason [10] for the community. This extension will enable developers to immediately test the agents developed in this work. In this manner, individuals may contribute to the expansion of the community by working on these new features, offering enhancements, or discovering bugs. This collaborative approach aims to foster a thriving community of developers and researchers in the field of intelligent agents and hypermedia environments.

By addressing these challenges, this study seeks to contribute to the development of intelligent agents capable of thriving in the intricate and ever-changing landscape of hypermedia environments.

## 1.4 SIGNIFICANCE AND CONTRIBUTION

This study holds significant importance within the domain of intelligent agents and hypermedia environments for some compelling reasons.

The study addresses an issue in the current agent technology landscape. As the Web continues to evolve, hypermedia environments have become increasingly prevalent, offering dynamic and long-lived spaces where intelligent agents can operate. However, the ability of these agents to effectively adapt, reason about available actions, and make informed decisions in such environments remains a critical challenge. By proposing an approach that equips agents with the capacity to perceive, evaluate, and execute actions within hypermedia environments, this research contributes to the advancement of intelligent agent technologies.

Furthermore, the study extends its significance to the broader community of developers, researchers, and practitioners. Through the implementation of an extension for AgentSpeak [17] (an agent-oriented language based on logic programming and the BDI framework that allowed the creation of other agent-oriented-languages such as Jason), this work not only introduces a novel approach but also provides a practical tool for immediate testing and experimentation. Developers can readily explore and utilize the additional features offered by this research, accelerating the integration of intelligent agents into hypermedia environments. The open-source nature of this extension encourages community involvement, fostering collaboration, enhancements, and the identification of potential issues or bugs. In doing so, it promotes knowledge sharing and contributes to the growth of a vibrant and engaged community dedicated to advancing the capabilities of intelligent agents in dynamic online spaces.

Moreover, this study aligns with the evolving landscape of Web technologies, which has witnessed the emergence of principles such as HATEOAS and initiatives like the WoT and Linked Data. These developments have reshaped the Web, emphasizing dynamic, open, and long-lived systems. Consequently, the proposed approach, suited for the challenges posed by these transformations, is well-timed and pertinent. It offers an opportunity to harness the full potential of intelligent agents in environments characterized by dynamicity and complexity.

In summary, this research not only contributes to the academic understanding of intelligent agents and hypermedia environments but also offers a practical solution that can be immediately adopted by the developer community. By addressing the challenges of action perception, reasoning, and execution in dynamic environments, this study paves the way for more effective and adaptable intelligent agents, propelling them into the forefront of modern Web applications and services.

## 1.5 LIMITATIONS

This study assumes environments and agent actions can be fully described using ontologies, with an existing mechanism for ontology alignment. This presents a limitation due to the dynamic and unpredictable nature of hypermedia environments, where capturing every potential interaction accurately is challenging. The reliance on ontologies for describing environments and actions imposes constraints on the adaptability and responsiveness of agents, potentially hindering their effectiveness in unforeseen scenarios.

Addressing these limitations requires ongoing research into more dynamic and flexible approaches for ontology management, alignment mechanisms, and knowledge representation.

## 1.6 STRUCTURE OF THE THESIS

In Chapter 2, we undertake an extensive exploration of the existing literature, exploring MASs, environments, and action representation. We delve into foundational concepts that underpin our study. Additionally, we pinpoint the gaps in the current research landscape that serve as the driving force behind our proposed approach.

Chapter 3 marks the inception of our proposal, "Action Handling by BDI Agents." In this pivotal chapter, we introduce our approach, which equips agents to proficiently handle actions within dynamic hypermedia environments. We lay the groundwork by elucidating the essential components of action knowledge, action reasoning, and action execution. Furthermore, we examine the integration of environmental cues, called signifiers, within the action pillars and the practical implementation within the JaCaMo framework.

With the groundwork firmly established, Chapter 4 ushers us into the evaluation. We delineate the methodology employed to assess the efficacy of our proposed approach.

The chapter unfolds as we define the variables, scenarios, and evaluation criteria that support our experimental endeavors. Detailed insights into the experimental setup, expected outcomes, and scenario specifics are revealed, culminating in an enlightening overview of the results obtained.

Chapter 5 engages us in a discussion. Here, we elucidate the advantages and strengths of our proposed approach, while candidly addressing the inevitable challenges and limitations that emerged during the study. We direct our attention towards the prospects of future research, proffering insightful recommendations and directions. Moreover, we proudly present our contributions to the broader research and developer community.

Finally, in Chapter 6, we revisit the study's original objectives, summarizing the key findings that have emerged through our exploration. We underscore the profound significance of our research within the technology landscape.

# 2 LITERATURE REVIEW

This chapter provides a literature review to establish an academic foundation on the subject. Section 3.1 introduces MASs, presenting their core principles and functionalities. Section 3.2 then focuses on the specific environments in which these agents are situated, delineating the conditions and parameters that influence agent behavior.

To delve deeper, Section 3.2.1 introduces the Agents and Artifacts conceptual model, providing clarity on the critical components that constitute MASs. Section 3.2.2 shifts the focus to artifacts in MASs, exploring their operations, properties, and signals. In Section 3.2.3, attention is directed toward large-scale and open environments, emphasizing the complexity and scale of the contexts in which agents function.

Section 3.3 explores the representation of actions within the literature, offering insight into how agents translate intentions into tangible outcomes. Section 3.4 navigates through the digital expanse of the Web, highlighting the unique attributes and challenges faced by agents operating in this space.

The chapter then turns to an examination of agent reasoning about the environment in Section 3.5. This section is further divided into subsections, where Section 3.5.1 delineates the general reasoning process of agents, Section 3.5.2 zooms in on action reasoning, and Section 3.5.3 identifies challenges and gaps in the current landscape, underscoring areas in need of further academic scrutiny.

## 2.1 OVERVIEW OF MULTI-AGENT SYSTEMS

Multi-Agent Systems have their historical origins anchored in artificial intelligence research, with roots extending back to influential works from the early 70s [18] at the Stanford Research Institute and gaining momentum in the mid-80s [19]. In AI, an agent is depicted as a computer system situated within a particular environment, endowed with the capability for autonomous and flexible action to meet its designated objectives [20]. Central to this depiction is the concept of autonomy — an agent's innate ability to operate without needing direct human or other agent oversight. The agent functions within an environment that it perceives using sensors and influences using actuators. In distributed AI, a MAS consists of agents co-located in a shared environment, interacting collectively to fulfill their goals.

The MAS paradigm offers a versatile blueprint for modeling and engineering complex systems, combining methodologies, techniques, concepts, and technologies. Through MAS, complex systems are portrayed as organized groupings of autonomous agents operating and interacting within specified environments. These agents are decision-making components, autonomously driven to achieve particular objectives. A vivid example is a smart city scenario where autonomous vehicles are modeled as agents, with tasks such as safely reaching destinations while considering user preferences. These agents operate

within an environment, which could be physical elements like roads, or digital constructs like virtual message boards displaying pertinent information [16].

In principle, any programming technology could be used to implement a MAS, provided there's a clear description of the model, like an Object-Oriented Programming (OOP) language. The challenge lies in the risk of overly focusing on the agent's perspective, potentially neglecting the broader environmental and organizational contexts. Although MAS and OOP share attributes like information encapsulation and the centrality of interactions, they differ in several facets. Agents are proactive and goal-driven entities, distinct from passive objects. Unlike actors, which are reactive, agents initiate actions in line with their objectives regardless of message reception [16]. Another key differentiation in agent-oriented modeling is the prime role of the environment. While objects dominate object-oriented worlds, in agent-oriented systems, agents engage with both peers and an encompassing environment through actions and perceptions [16].

Adopting programming languages with first-class programming abstractions helps ensure consistent abstraction levels from design to development and even run-time.

Multi-Agent-oriented programming, an approach targeting MAS, emphasizes three crucial MAS dimensions, as illustrated by Figure 1: the agent, environment, and organization. Each dimension encapsulates specific concepts and programming abstractions [16]:

- **Agent Dimension:** This dimension revolves around defining and programming the agents in the system. Agents are decision-making entities, each having a distinct logical control thread, allowing them to engage autonomously with the environment, fellow agents, and overarching organizational systems.

  The hallmark trait of agents within this paradigm is their autonomy. An agent's autonomy is rooted in its capability to determine its objectives and strategize on their realization considering the prevailing system conditions. Key facets of this autonomy include [10]:

  - **Proactiveness:** Taking initiative concerning actions geared towards goal attainment.
  - **Reactivity:** Swift adaptation to environmental events.
  - **Social Ability:** Effective communication and collaboration with other agents.

- **Environment Dimension:** The environment goes beyond serving as merely a background for agents; it's a vital dimension in its own right. It presents concepts and first-class abstractions for defining and programming the distributed resources and their links to the real world, all of which are shared among the agents. While agents embody autonomous, goal-oriented entities, the environment, when treated as a first-class abstraction, represents any element that agents might use or control to

fulfill their objectives. This environment abstraction renders agents situated, logically placing them in a context. This context not only empowers agents with a range of actions to influence the environment but also exposes a discernible state and events that agents can perceive and react to.

- **Organization Dimension:** This dimension encapsulates concepts essential for defining and programming agent relationships, joint tasks, and policies within a shared environment. At its core is the concept of organization, which delineates the structuring, coordination, and regulation of agents collaborating in a system. Given the agent dimension's emphasis on autonomy and the dynamic nature of the environment dimension, the organization dimension's defining features are coordination and regulation.
  - **Coordination** addresses the intricacies of enabling multiple agents to work in tandem. These agents, while autonomous, might depend on each other to realize their individual or collective objectives.
  - **Regulation** concerns itself with modulating agent autonomy. This is typically achieved through norms — societal rules that agents are anticipated to adhere to. Non-compliance can have repercussions. Incorporating the organization dimension facilitates agents in reasoning about their relationships, collaborative tasks, and policies. It equips them to make informed decisions about adapting to these guidelines and ensures compliance with the constraints set forth.

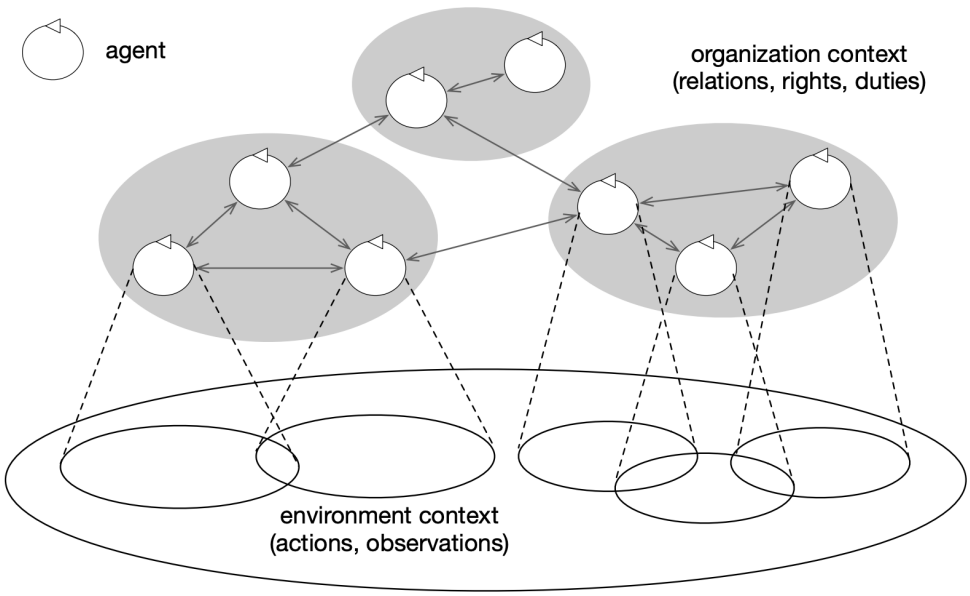


Figure 1 – Multi-Agent System dimensions. Source: [16].

The evolving landscape of MAS underscores the necessity of more than just agent-centric dimensions. Notable communities like Environments for Multi-Agent Systems

(E4MAS) [15] emphasize the importance of the environment and organizational dimensions in MAS design. Acknowledging the environment as a pillar facilitates the conceptual union of MAS and the Web, transforming the latter from a mere communication channel to a global, agent-centric operational platform. This external orientation ensures agents' autonomous operations are compatible with the Web's expansive and intricate nature [7].

However, challenges persist. One obstacle is the suboptimal integration of MAS technologies into mainstream software engineering. While the MAS community is actively addressing this, other sectors stand to benefit from MAS research outcomes, presenting applications for MAS, especially within the WoT[1] and Linked Data[2] systems contexts [7].

## 2.2 OVERVIEW OF MULTI-AGENT SYSTEM'S ENVIRONMENTS

In the MAS context, environments embody an important role, presenting themselves as abstractions that encapsulate resources and tools for the agents to interact and improve their performance in their goals. To navigate through the complexity of environments in MAS, an analogy to human work can be employed. In the same way that humans don't only communicate but also utilize a variety of tools and resources in their workspaces to accomplish tasks, agents in MAS interact with artifacts within environments that are instrumental in determining the effectiveness of their actions [16]. These environments are more than mere passive spaces, actively influencing the outcomes of agents' actions. In a practical scenario, consider a team of robotic agents tasked with assembling products in a factory. Artifacts in this environment could include assembly tools, sensor systems, and communication modules. The robots (agents) interact with these artifacts to perform tasks, such as picking up tools or receiving sensory information about the product's state. By invoking operations on these artifacts, the agents can enhance their efficiency and precision in the assembly process.

### 2.2.1 Agents and Artifacts Conceptual Model

The Agents and Artifacts (AnA) conceptual model stands as a significant framework for interpreting the complexities of agent-environment interactions [16]. Within this model, environments are conceptualized as workspaces, places where agents can seamlessly engage, share, and collaborate with artifacts. Artifacts, in this paradigm, are not passive entities but are active elements that structure and organize the environment, serving as non-autonomous, function-oriented counterpoints to the autonomous agents. The principles of the AnA model find extensive application across various domains. In intelligent traffic management systems, agents represent different entities like traffic lights and cameras, while artifacts could be the traffic control algorithms and sensor data processors. Here, the

[1] https://www.w3.org/WoT/WG/
[2] https://www.w3.org/DesignIssues/LinkedData.html

interaction between agents and artifacts ensures optimal traffic flow and reduces congestion [16].

From the vantage point of MAS designers, artifacts emerge as fundamental building blocks, essential first-class abstractions for the engineering of environments. Agents, on the other hand, can create and interact with these artifacts as first-class entities of their world, utilizing them to fulfill their activities and achieve their objectives.

The environment in MAS encapsulates services and resources that are not aptly modeled as cognitive agents, given their inherent lack of autonomy and proactivity. Examples abound, ranging from blackboards to databases and shared knowledge bases, all integral components of the agent's environment.

### 2.2.2 Artifacts in MAS: Operations, Properties, and Signals

The AnA model introduces three critical concepts to articulate the functionality of artifacts within the environment: signals, operations, and observable properties.

- **Operations:** Artifacts encapsulate their functionality through operations, which agents can invoke to perform tasks. These operations can either be atomic or involve a sequence of computational steps, providing a modular approach to structuring artifact functionalities. From the agent's perspective, operations represent the actions available to them, each defined by a unique label and a set of input parameters.
- **Observable Properties:** These properties delineate the observable state of an artifact, which agents can perceive and respond to. Observable properties serve as a transparent window into the internal state of an artifact, allowing agents to base their decisions and actions on the current state of their environment.
- **Signals:** Artifacts can generate signals, observable events that serve as a communication mechanism between the artifact and the agents. Signals can represent a wide array of situations, conditions, or messages, providing a flexible tool for artifacts to communicate with agents, independent of the observable properties.

For instance, observable properties of an artifact in a smart home environment could include the current temperature and humidity levels, which are sensed by thermostat agents. Agents can perceive these properties and respond accordingly, perhaps by adjusting the air conditioning to maintain a comfortable living environment. Similarly, an artifact might send a signal to alert agents when a significant temperature change is detected, prompting immediate action.

Despite its strengths, the AnA model is not without challenges. One of the potential limitations lies in the complexity of managing interactions as the number of agents and artifacts in the environment grows. Ensuring that agents can efficiently discover and

interact with the relevant artifacts, particularly in dynamic environments, remains an area for ongoing research and development.

### 2.2.3 Large-Scale and Open Environments

Over the past decades, there has been a growing consensus within the MAS community that the environment is a crucial component of any MAS. Workshop for E4MAS [21] and various publications have significantly contributed to this understanding, highlighting the importance of integrating the environment as a primary abstraction in MAS models and tools. Two important kinds of environments present on this discussion are Large-Scale Multi-Agent System (LSMAS) and Open Environments for Multi-Agent Systems.

Large-Scale Systems (LSS) are characterized by their high dimensionality, vast size, and stochastic nature [22], demanding a decentralized control approach [23]. LSMAS, a subset of LSSs, are particularly challenging to engineer and deploy due to the massive number of interactions and the need for real-time response [22]s. The key requirements for LSMAS are [15]:

- Scalable Structure: The environment must be distributed to handle the vast number of agents and data.
- Access to Resources: With heterogeneous agents, the environment must regulate resource access while ensuring trust and security.
- Scalable Communication: The environment should support decentralized communication to avoid bottlenecks.
- Interaction Model: Efficient perception, action, and interaction mechanisms are crucial, requiring high-level primitives for agents.

While scalability was not a central focus in past E4MAS efforts, the community has made strides in addressing these challenges. Decentralizing the environment's structure and managing resource access have been identified as key principles for scalability. Nature-inspired mechanisms, such as digital pheromones, have emerged as potential solutions for scalable communication and interaction models [15].

On the other hand, open environments represent a paradigm shift, acknowledging the fluidity and dynamism in agent interactions and system structures. The text categorizes openness into three levels [24]:

- Off-line Openness: The addition of new agents is possible only when the system is offline.
- Static Openness: Agents can be added without restarting the system, but require prior notification or a pre-existing list of potential new agents.

- Dynamic Openness: Agents can enter or leave the system in real-time, without the need for explicit global notification.

The focus here is on dynamic openness, highlighting its relevance in environments characterized by uncertainty and constant change. Historically, the emphasis on openness in environments has been centered around engineering challenges, architectural needs, and the provision of suitable abstractions for agent interaction [15]. Questions have been raised about the responsibilities of the environment and the services it can offer to enhance openness to heterogeneous agents.

The need for open environments has only intensified with the increase in dynamic and uncertain operating conditions of software-intensive systems. As such, the challenges ahead lie in adapting to these changes, ensuring that environments are capable of accommodating a diverse and constantly changing agent population [15].

## 2.3 REPRESENTATION OF ACTIONS

This section unfolds across two distinct subsections, each delving into the nuances of action representation from different perspectives. In the initial subsection, see the action representation in a broad, universal context. Following this, the second subsection transitions to focus exclusively on the unique domain of the WoT, offering a tailored and in-depth analysis of action representation within this specific sphere.

### 2.3.1 Universal Representation of Actions

Although there is a lack of studies combining agents and action representation, we can find works on the latter. In his thesis, Trypuz brings several definitions for actions [25], some of them are: (*i*) action is an event, namely, the event that is carried out by an agent; (*ii*) action is an event done by an agent with the intention to do this action; (*iii*) action is an event done by an agent for a reason; (*iv*) action is an instance of the relation of bringing about (or making happen), whose terms are agent and event (or the consequence of this event); (*v*) action is an event, which is caused by an agent; (*vi*) action is an event (bodily movement), which is under the control of the agent. The majority of definitions emphasize the importance of the agent in the action. They also present the causal relationship between the agent and the event.

Kemke [26], in his study, attempts to create a general description of actions. He identifies numerous parameters for an action, such as (*i*) agent, the one who performs the action; (*ii*) motive, the reason for the action; (*iii*) patiens, entity impacted by the action; (*iv)* duration, start and end time of the action; (*v*) the main effect, results, and side-effects of the action. In addition, he defines an action as a controlled, volitional, or programmed doing conducted by an agent.

Kemke defines three levels of abstraction of actions:

- The realization level: The action is described on a physical level. If the action is to turn on the lights, the realization level would be "move finger to the light switch, position finger in the correct location on the light switch, and press." It incorporates basic vocabulary as if you were describing the action to a toddler or programming a robot using terms such as "move-to" and "push."
- The semantic level: The action is told in a more natural manner, in an environmental context, using ordinary conceptual categories, such as "switch on" and "light".
- The pragmatic or motivational level: The action is expressed in terms of its motive, such as "brighten the room".

Equivalencies of lower levels can be assumed if a higher-level action is stated. For instance, if the pragmatic action is to brighten the room, there are several ways to do it on the semantic level, whether switching on lights or opening the window blinds if it is a sunny day. These semantic-level actions could be in this case considered equivalent.

In another paper, Kemke intends to develop a representational framework that may serve as a conceptual and ontological basis for the semantic interpretation of natural language inputs and, concurrently, the description of relevant actions to be carried out by an artificial agent system [27]. When formalizing the ontology he states that the actions have a pre-condition and effect, and that the actions change a feature (the change-feature-value concept) value or the role value (the change-role-feature concept) of an object. An example of the generic change-role-feature concept can be the change-state concept for changing the state of a light bulb. The pre-condition is for the light bulb to have an initial state, and the effect would be the changing of the state. The object is the light bulb and the feature it has is the state (on or off).

Furthermore, certain ontologies, like MARIO Ontology Network [28] and DOLCE [29], have definitions of action entities. Although they include concepts as agents and events, they lack key parameters presented in earlier publications, such as causes, outcomes, and side effects.

### 2.3.2 Actions in the Web of Things

Hypermedia environments naturally lend themselves to the dynamic identification and utilization of actions, leveraging hypermedia formats to facilitate the invocation, reasoning, and execution of actions. This is related to how HTML pages present actions through hyperlinks and forms, providing human users with interactive and dynamic controls that are rich in semantic information. These controls guide users in their decision-making process while navigating the intricate web, simultaneously managing lower-level implementation details, such as the generation of HTTP requests based on the hypermedia controls.

For software agents, this paradigm is mirrored in the development of diverse interaction models and machine-readable hypermedia formats. A typical example is Hydra [30], introducing a specialized vocabulary designed to represent hypermedia actions made available by Web APIs, thereby aiding agents in discovering essential implementation details for interacting with Web resources.

Building upon this groundwork, the WoT initiative [31] marks a significant advancement, integrating additional concepts and a higher level of abstraction into the descriptions of actions. According to the W3C WoT Thing Description (TD) Recommendation [32], interactions are conceptualized as TD Interaction Affordances, enriched with hypermedia controls defined using the Hypermedia Controls Ontology (HCTL) vocabulary [33]. This enhances the agents' capabilities, enabling them to discover action metadata and interact with a wider array of entities, extending beyond Web resources and incorporating various application-layer protocols.

Moreover, the W3C WoT TD Recommendation introduces a minimal vocabulary for delineating higher-level action types, enabling interactions with WoT Things through actions, properties, and events. This is exemplified by the TD Action Affordance, which delineates metadata crucial for affecting the state of a WoT Thing, along with references to input schemas. Importantly, this metadata can be augmented with domain-specific semantics, enriching the agents' capacity for informed interactions across a variety of applications, ranging from manufacturing to smart home environments.

## 2.4 OVERVIEW OF AGENTS ON THE WEB

Attempts to implement Multi-Agent Systems on the Web stretch back to the early 2000s. These early efforts were significantly impacted by Service-Oriented Architectures (SOA) based on WS-* specifications (SOAP, WSDL, UDDI, etc.) [2, 3, 34, 35]. Nonetheless, Web design has changed significantly during the previous decade. It is now well-known that WS-* services only utilize the Web as a transport layer [36].

Following the same idea, the Foundation for Intelligent Physical Agents (FIPA) proposed a specification using HTTP as a transport protocol for exchanging messages between agents[3]. Several systems that comply with FIPA have implemented this standard [4, 37, 38]. Nevertheless, there is an issue with these techniques. Web architecture states that HTTP is not a transport protocol, but rather a protocol for executing actions on resources and altering the representation and state of those resources. Therefore, using the Web just as a Web transport layer is a limited view of the Web architecture. Therefore, these systems make little use of the Web infrastructure and do not inherit the characteristics that make the Web an open and dynamic system, such as scalability and loose coupling.

[3] `www.fipa.org/specs/fipa00084/SC00084F.html`

Years later, some Web-based MAS works started to be no longer based on SOA, but rather on Resource-Oriented Architectures (ROA) based on REST services [5, 39]. In contrast to WS-* services, which are often developed in terms of operations, services are now created in terms of resources. A limited set of actions with well-defined semantics are used by clients to communicate with services, such as those described by the HTTP protocol, such as GET, POST, PUT, and DELETE. These services use the Web not as a transport layer, but as an application layer, which is more consistent with the design of the Web. However, these services do not use hypermedia or HATEOAS, resulting in a significant degree of coupling.

The most recent approach to Multi-Agent Systems on the Web is the Hypermedia Multi-Agent System. This strategy is quite similar to the preceding strategy, with the addition of HATEOAS. In a hypermedia Multi-Agent System, agents are located in a distributed hypermedia environment where they may explore and use the environment's attributes, like hyperlinks, to achieve their objectives (Figure 2). To construct a Multi-Agent Hypermedia System, it is required to adhere to three Web architecture-based principles to guarantee the right usage of hypermedia as a mechanism for uniform interaction within the system [14, 40].

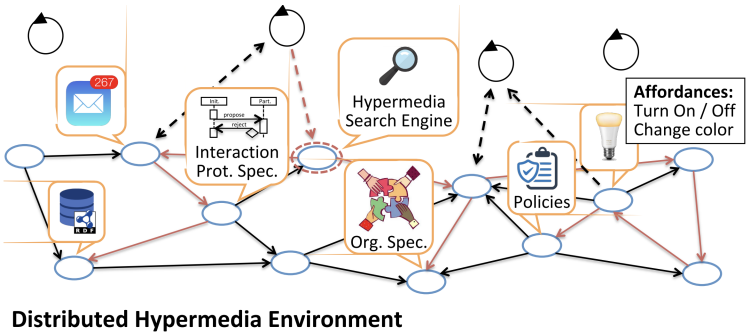


Figure 2 – Distributed hypermedia environment. Source: [7].

The first principle states that all entities in a system and their relationships must be represented in the hypermedia environment in a uniform and resource-oriented manner. This allows agents to interpret, reason, and interact with the system by consuming and producing hypermedia. An agent wishing, for example, to turn on a light bulb can manipulate the state of a resource representing the light bulb in this hypermedia environment. Additionally, all resources must be represented by IRIs.

The second principle is the single entry point principle. According to this principle, given an entry point to the environment, an agent must be able to discover all the necessary knowledge to participate in the system by navigating through hypermedia. In other words, agents must possess a minimal standardized prior knowledge to interact with the system, and any other required knowledge should be discoverable. In the example of the light bulb, the agent's a priori knowledge would be the RDF standards, a general model of the environment, and an OWL ontology describing light bulbs. Knowledge of the light bulb's

identifier and the HTTP request specification to turn on the light should be discovered. This prevents specific information from needing to be in the agent's code, so the agent is not tied to a specific type of light bulb.

The third and final principle states that any resource of interest to the agent must be observable. While the first two principles ensure the discovery and navigation of the system, constant navigation to keep up with the evolution of a huge system would be inefficient. The third principle seeks mechanisms for the agent to observe only what is relevant to it. Consequently, this principle improves the scalability of the system, as agents can handle larger environments. With these three principles, the hypermedia Multi-Agent System ensures system dynamism and scalability.

## 2.5 AGENT REASONING ABOUT THE ENVIRONMENT

In Multi-Agent Systems, the interaction between agents and their environment, as well as their reasoning about actions, are two critical aspects that warrant a comprehensive exploration. Agents are situated within an environment, where they perceive their surroundings through sensors and respond accordingly. This interaction results in a distributed perception, with each agent having information about a part of the environment, that needs to be integrated [41].

The decision-making process of agents is influenced by their perceptions of the system. Each agent, whether an individual or an organization, pursues its own goals, and its existence may directly or indirectly impact the decisions and welfare of other agents [42].

In complex environments, rational agents strive to achieve their goals by selecting the action with the optimal expected outcome. To navigate environmental uncertainties, agents need to exhibit autonomy and adaptability. Furthermore, agents can evaluate the accuracy of model hypotheses by comparing predicted actions with actual actions and analyzing action frequencies and causal dependencies [43].

This interplay between agents and their environment in Multi-Agent Systems underscores the importance of understanding these dynamics for effective system design and operation. Further research in this area promises to yield valuable insights into optimizing agent behavior within these systems.

### 2.5.1 General Reasoning Process

The reasoning process in a MAS involves several aspects:

- **Environment Interaction:** Agents interact with their environment. The environment's properties such as accessibility (whether it is possible to gather complete information about the environment), determinism (whether an action causes a definite effect), dynamics (how many entities influence the environment at the moment),

discreteness (whether the number of possible actions in the environment is finite), episodicity (whether agent actions in certain periods influence other periods) and dimensionality (whether spatial characteristics are important factors of the environment and the agent considers space in its decision making) influence the agent's decision-making process [44].

- **Middleware Mediation:** Agent actions are typically mediated via an appropriate middleware. This middleware offers a first-class design abstraction for Multi-Agent Systems, providing means to govern resource access and agent coordination [45].
- **Real-Time Compliance:** In many real-world applications like healthcare and automotive, it's crucial for MAS to respect strict timing constraints. Unfortunately, current AI/MAS theories and applications only reason "about time" and are incapable of acting "in time" guaranteeing any timing predictability. In [46], the authors postulate a formal definition and mathematical model of real-time Multi-Agent Systems (RT-MAS).

The reasoning in a MAS occurs within the internal architecture of each agent [9, 47–49]. The internal architecture of an agent in a MAS can be viewed as a special case of the container-component architecture. In this case, the components are the individual modules within the agent that perform specific tasks, and the container is the environment that provides discovery and communication services to its components [47].

The internal architecture of an agent typically includes several key components [47]:

- **Perception Module:** This module is responsible for interpreting sensory data from the environment.
- **Reasoning Module:** This module uses the data from the perception module to make decisions. It can include a variety of AI techniques, such as machine learning algorithms, rule-based systems, or planning algorithms.
- **Action Module:** This module carries out actions in the environment based on the decisions made by the reasoning module.
- **Communication Module:** This module enables the agent to communicate with other agents. It can include protocols for sending and receiving messages, as well as mechanisms for understanding and generating language.

For example, in a MAS used for disaster response, each rescue robot (agent) would have sensors to perceive its environment (perception module), algorithms to decide where to go and what to do (reasoning module), actuators to move around and perform tasks

(action module), and wireless communication capabilities to coordinate with other robots (communication module) [50].

In another example, in a MAS used for online trading, each trader (agent) would have software to gather and interpret stock market data (perception module), algorithms to decide when to buy or sell stocks (reasoning module), software to execute trades (action module), and internet communication capabilities to coordinate with other traders or receive instructions from a human operator (communication module) [51].

### 2.5.2 Action reasoning

As defined by Winikoff [52], actions are paramount for agents to interact with their environment effectively. In practice, actions in BDI frameworks (software models for programming intelligent agents, characterized by the implementation of an agent's beliefs, desires, and intentions) are typically implemented as functions embedded within more complex structures, such as plans. The invocation of these actions is usually concisely represented using a function identifier (e.g., `move`, `toggle`) along with optional arguments that are instantiated at run-time (e.g., `move(100, Y)`).

For an agent to successfully invoke an action, it needs to be aware of the action identifier and the data types of its arguments. These requirements form the foundational layer of discoverable interaction cues for agents. Moreover, the semantics of action and argument identifiers should be transparent, enhancing intuitive programming for action invocation. That is why it is important for the environment, artifacts and actions to be represented by ontologies and to have an ontology-alignment mechanism to connect the ontologies. As stated in the introduction, this work assumes this mechanism exists.

Beyond the necessity of action invocation, it is crucial for interaction cues to support agents in reasoning about actions. This reasoning typically hinges on domain-specific logical conditions, ensuring that actions are contextually appropriate. Within the BDI architecture, an agent's actions are influenced by its current intentions, which are themselves guided by the conditions for adopting or retaining specific goals and plans [53]. For instance, in Jason, intentions are represented as plans that the agent is committed to executing [9, 16]. The selection of a plan, and subsequently the initiation of an action, depends on the current system state's alignment with the plan's application context, reflecting the agent's beliefs.

BDI frameworks offer various constructs to support reasoning about actions, including application contexts in Jason plans, conditional operators in 3APL plans [54], guards in GWENDOLEN plans [55], and contexts in GOAL modules [56]. These constructs enable agents to determine the contextual relevance of actions, guiding their invocation process.

Some BDI frameworks introduce additional conditions for more advanced reasoning about actions, their invocation, success, and subsequent belief revision. Although these conditions go beyond the core requirements of BDI architecture, they enrich the agents' rea-

soning capabilities. Action descriptions often encompass preconditions and postconditions, defining the necessary conditions for an action's execution and the expected state of the system post-execution, respectively. However, the semantics and application of these conditions vary across different frameworks. For instance, while 3APL and GOAL utilize action preconditions to verify an action's feasibility based on the agent's beliefs, GWENDOLEN employs them for plan reconfiguration and evaluating action success post-execution.

Finally, the process of reasoning about and invoking actions culminates in the execution of code that embodies the action, contingent on the deployment context [45]. These implementation details are generally beyond the scope of agent programming languages and are presumed to be stable and known during design-time. This assumption holds for applications in static environments but falls short in open, dynamic settings where artifacts are deployed at run-time, and the designers of these artifacts may be different from the agent designers.

In such open environments, which are increasingly prevalent and discussed in recent MAS literature [7, 57–61], it becomes vital for BDI agents to operate independently of specific implementation details of artifacts. This underscores the necessity for interaction cues and reasoning mechanisms that accommodate the dynamism and unpredictability of open environments, ensuring that agents remain versatile, context-aware, and effective in their actions.

### 2.5.3 Action Signifiers for MAS Environment

The traditional approach to interaction cues has been challenged by the dynamics and openness of Web-based MAS, necessitating innovations for more flexible and adaptive mechanisms. In response to this challenge, the concept of signifiers has been introduced as a paradigm shift in agent-environment interaction.

In the context of design, as explained by Don Norman, signifiers are elements that signal or indicate what actions are possible and how they should be done [62].

Signifiers must be perceivable, or else they fail to function. For instance, in a digital context, links are often blue or underlined to signify that the text is clickable [62]. Similarly, the words "click here" on a user interface button communicate to the user that the button is actionable.

Signifiers can also be any kind of signal like signs, labels, and drawings. They are properties that help us determine where the action will take place. Perceived affordances (the actions that are possible for a person to take with a designed object) often act as signifiers, but they can be ambiguous [62].

In summary, signifiers in design are crucial as they guide users on how to interact with different elements in their environment.

In the MAS context, signifiers serve as a first-class abstraction in Hypermedia MAS, enabling agents to use the hypermedia-driven exploitation of affordances on the

Web while capturing the agent-environment complementarity essential for exploiting affordances [58]. They come with mechanisms for dynamically adjusting signifier exposure based on the agent-environment situation, facilitating affordance discovery and exploitation in affordance-rich and open Web-based MAS. These advancements are crucial in ensuring that BDI agents remain versatile and effective in their actions, even in the face of uncertainty and change [58]. They can be used to help the agents sense and reason about the action.

# 3 PROPOSAL ACTION HANDLING BY BDI AGENTS

Given the inherently open and dynamic behavior of hypermedia environments, a large number of developers concurrently engage in the creation, modification, and deletion of a diverse array of objects within these settings. This activity introduces a layer of complexity, as agents operating within these environments are required to navigate and interact with a perpetually evolving landscape. To adeptly manage actions in such a challenging context, agents must undergo a comprehensive and systematic process, encompassing three pivotal stages: Action Knowledge, Action Reasoning, and Action Execution. The following sections will elucidate how our proposed approach systematically addresses each of these critical phases, showcasing its robustness and adaptability in dynamic hypermedia environments.

Within the scope of Action Knowledge, we introduce the innovative use of signifiers as environmental cues, offering agents critical insights into the nature of available actions, their contexts, and associated abilities. This information is stored within the agent's belief base, consolidating the knowledge about the actions. This storage can be made using annotations or namespaces to identify the signifiers.

Moving to Action Reasoning, we build the SRM algorithm — a pillar of our proposed framework. This mechanism embarks on a strategic evaluation of the agent's plans, scrutinizing the encompassed actions to verify their feasibility. It examines the compatibility of actions with the current context and the agent's capabilities, thereby determining their viability.

Lastly, for Action Execution, our approach maintains continuity with conventional execution methods, necessitating minimal alterations. We ensure that agents retain their inherent ability to execute actions, now augmented with the capacity to make HTTP requests. This facilitates a seamless integration into existing hypermedia environments, enabling agents to actuate their decisions with efficacy and precision.

In the subsequent sections, we will delve into each of these stages in greater detail, detailing our approach and demonstrating its effectiveness in hypermedia environments.

## 3.1 ACTION KNOWLEDGE

Action knowledge in the context of BDI agents can be defined as the step to acquire information about the various actions they can perform to interact with their environment [63]. This knowledge encompasses the action itself, the artifact that is affording the action, the recommended prerequisites to perform the action, and the inputs for the action. Action knowledge is crucial for BDI agents as it enables them to deliberate on their options, make informed decisions, and act in a manner that is consistent with their intentions and the current state of the world.

### 3.1.1 Acquisition of Action Knowledge

BDI agents can acquire action knowledge through various means, including:

- **Programming:** The agent's action knowledge can be hardcoded by its developers, providing a foundational layer of reliable and consistent actions that the agent is capable of performing. This method ensures that the agent has immediate access to a predefined set of actions, facilitating swift and efficient responses in familiar scenarios.

  Nevertheless, the limitation of this approach becomes apparent in open and dynamic environments where unpredictability and complexity may be faced. In such contexts, the developer might not possess comprehensive knowledge of all possible situations and actions required for optimal agent performance. Our goal is to enable the agent to discover actions at run-time, and not at design-time.

- **Communication with Other Agents:** In MASs, individual agents can leverage inter-agent communication to acquire action knowledge that might not be readily available through their direct interactions with the environment. By exchanging information, insights, and experiences, agents can collectively enhance their understanding of the environment, uncovering different facets and potential actions that may otherwise remain obscured. This collaborative approach facilitates a more comprehensive and enriched knowledge base, enabling agents to tap into the collective intelligence of the system.

  However, it is crucial to note that our proposal is designed to prioritize and emphasize the agent's direct interaction with its environment as the primary means of action discovery and knowledge acquisition. The goal is to ensure that even in solitary settings, without the presence of other agents to communicate with, the agent is fully capable of autonomously exploring, learning, and adapting to its surroundings. While communication with other agents can indeed accelerate the process of action discovery and knowledge acquisition, promoting a quicker and potentially more efficient learning experience, our approach is not based on this inter-agent interaction. We aim to give BDI agents the capability can navigate, learn, and thrive independently, ensuring that their performance and adaptability are not compromised in the absence of peer agents. Thus, while agent-to-agent communication is recognized as a valuable asset and supplementary resource, our primary focus and commitment remain on fortifying the agent's intrinsic capabilities to interact with, learn from, and adapt to its immediate environment.

- **Perception of Artifact Signifiers:** BDI agents can also augment their action knowledge by perceiving and interpreting signifiers present in their environment. Signifiers, in this context, refer to cues or symbols associated with artifacts (objects

or entities) in the environment that indicate the potential actions that can be performed with or upon them. For instance, a button on a user interface might have a label or an icon that signifies its function, guiding the agent on the possible action of pressing it to achieve a specific outcome.

By being aligned to these signifiers, an agent can derive valuable insights about the affordances of different artifacts, essentially understanding what actions are possible just by observing the environment. This capability is crucial, especially in unfamiliar or dynamically changing settings, as it enables the agent to rapidly assimilate and expand its action knowledge by focusing its attention on the relevant artifacts, without the need for extensive prior programming or reliance on inter-agent communication,

However, it is important to note that the effectiveness of this means of action knowledge acquisition is dependent on the agent's ability to accurately perceive and correctly interpret the signifiers present in the environment. This necessitates a well-designed environment where actions are intuitively signified by artifacts. By mastering this capability, a BDI agent enhances its autonomy and adaptability, becoming better equipped to navigate and interact with diverse and evolving environments.

Incorporating the perception of artifact signifiers into a BDI agent's repertoire of action knowledge acquisition means ensures a more robust approach, enabling the agent to effectively learn and adapt in real-time.

We chose to prioritize the **Perception of Artifact Signifiers** as the central means for BDI agents to acquire action knowledge, recognizing its substantial benefits for enhancing an agent's autonomy and adaptability.

Recognizing the substantial contributions of relevant works in this domain [58], we draw upon these insights to empower agents with the ability to perceive and interpret environmental cues autonomously. These studies have laid the groundwork for agents to build and update their repertoire of actions through direct observation, abstaining from the need for detailed preprogramming or exhaustive instruction sets.

In this work, signifiers will be treated as observable properties of an artifact. This observable property will indicate 3 cues of how the agent can interact with the artifact: an action, recommended contexts for this action, and recommended abilities to perform the action. Contexts and abilities are represented by beliefs.

This self-sufficient learning mechanism is particularly advantageous in environments that are complex and subject to change, as it enables agents to adapt by learning from novel stimuli encountered in real-time. Therefore, it is the flexibility, efficiency, and inherent intuitiveness of the Perception of Artifact Signifiers that support our decision to adopt it as the preferred means of action knowledge acquisition for BDI agents.

### 3.1.2 Storage of Action Knowledge

BDI agents typically store their action knowledge in a structured format that allows for efficient retrieval. The information is organized in a way that makes it easy for the agent to search for actions that are relevant to its current goals and beliefs.

- **Belief Base:** The belief base in a BDI agent plays a pivotal role in storing information about the agent's environment, itself, and other agents. This information directly influences the agent's decision-making processes and action selection. In terms of action knowledge.

  The belief base holds information about the current state of the environment and the agent itself. This context is vital when deciding which action to take, as different actions may be more appropriate or feasible in different situations.

  As the agent interacts with its environment and performs actions, it constantly receives new information that it uses to update its belief base, ensuring that its knowledge stays relevant and accurate.

- **Storing Information in an Artifact:** Agents can also store and retrieve action knowledge from artifacts within their environment. Artifacts in this context refer to objects in the agent's environment that it can interact with. These artifacts can be designed to hold information about actions, serving as external knowledge bases for the agent.

  Storing action knowledge in artifacts can facilitate knowledge sharing in MASs. When one agent discovers new action knowledge, it can store this information in an artifact, making it accessible to other agents. This approach allows for the localization of action knowledge, tying specific pieces of knowledge to the parts of the environment where they are most relevant.

  By offloading some of the storage of action knowledge to the environment itself, agents can reduce their own cognitive load, potentially leading to more efficient operation.

- **Remote ontologies repositories:** In addition to their belief base environmental artifacts, BDI agents may also utilize remote ontologies for storing and accessing action knowledge. Remote ontologies repositories are centralized knowledge structures that reside on external servers or cloud platforms, offering a broader scope of knowledge that can be shared across different agents and systems [64].

  Remote ontologies repositories provide agents with access to a vast and extended repository of action knowledge that may not be feasible to store locally due to resource constraints. These ontologies often adhere to standardized formats, facilitating interoperability among diverse agents and systems. This standardization

ensures that when multiple agents refer to an action or concept within the ontology, they have a shared understanding of its meaning.

Remote ontologies repositories can be dynamically updated by experts or automated systems with the latest information, which can then be propagated to all agents that utilize them. This ensures that the action knowledge is current and reflects the latest understanding of the domain. By storing action knowledge in a central location, redundancy can be minimized. Agents don't need to store all the knowledge locally but can query the remote ontology as needed.

Our chosen method for storing action knowledge in this work is the **belief base**. The belief base is a core component of the BDI architecture, where the agent's knowledge about the world, itself, and other agents is dynamically maintained. This approach offers simplicity and immediacy, which are crucial at the early stages of development. It allows for a rapid update cycle of the agent's knowledge as it interacts with and learns from its environment, especially through the perception of artifact signifiers.

Incorporating the belief base for action knowledge storage aligns with our focus on empowering the agent to understand and utilize environmental signifiers independently. It enables the agent to swiftly adapt to changes and make decisions based on the latest information without the complexities of external data management systems. This integrated approach facilitates quick iterations and refinements, speeding up the prototyping process.

## 3.2 ACTION REASONING

We define Action reasoning as the process through which a BDI agent evaluates its available actions and determines which one to execute. This process is critical for ensuring that the agent's behavior goal-directed, and appropriate for the current context.

In the BDI model, plans are instructions that an agent can follow to address an event given some beliefs. A plan becomes relevant when the event it is designed to address occurs, and it becomes applicable if the beliefs required for its execution hold true in the agent's belief base. In our approach, the applicability of a plan in the BDI model is determined not only by the beliefs but also by the availability of actions (the action must be afforded by an artifact at the moment). This means that a plan becomes applicable if both the beliefs required for its execution hold true in the agent's belief base and the actions required for the plan are available. This adds an extra layer of realism to the model, as it takes into account the fact that actions may not always be available due to various constraints in the real world.

For example, consider a BDI agent that controls a smart thermostat. It has a plan to turn on the heating when the temperature drops below a certain threshold. This plan becomes relevant when the temperature drops (the event), and it is applicable if the agent

believes that the heating system is functioning correctly (the belief). If the thermostat is broken, the action of changing the temperature is not available. Even though the plan to turn on the heating when the temperature drops becomes relevant (since the event has occurred), it is not applicable because the required action cannot be performed due to the broken thermostat.

### 3.2.1 Factors influencing action reasoning

In our approach to action reasoning within BDI agents, we focus on three key points to ensure actions are not only conceived but also optimally executed.

- **Action Availability:** The first point is whether an action is available in the agent's environment. This is determined through the agent's perception of signifiers. These signifiers imply the presence of certain actions. In essence, they serve as indicators that certain actions can be undertaken by the agent. For an action to be considered available, the agent must detect its corresponding signifier within the environment. The signifier exposes the action described using a certain ontology. And the assumed ontology alignment mechanism will make the agent understand it.
- **Recommended Context:** Actions may be linked with recommended contexts; these are specific conditions or states of the environment where an action is most suitable to be executed. Signifiers do not merely suggest actions; they may also encode information about the ideal circumstances for these actions to be taken. An action signifier guides the agent not just on what can be done, but under what environmental or situational conditions it would be best to perform the action. Recommended contexts are also defined by an ontology. In this work, context are beliefs, the context is how the agent believes the system state is at a specific moment.
- **Recommended Abilities:** Furthermore, action signifiers can hint at required or recommended abilities that the agent should possess to perform the action effectively. These abilities could range from physical capabilities to cognitive skills like problem-solving or specific knowledge domains. The agent assesses its own capabilities against the recommended abilities to decide whether it can execute the action proficiently. Recommended abilities are also defined by an ontology. In this work, abilities are beliefs, the agent can believe it has an ability or not.

To illustrate these three points, consider a robot arm situated in a hypermedia environment, in this case, a laboratory, and the robot contains various signifiers. A BDI agent enters this laboratory, with the goal of catching a ball using the robot arm. The agent focuses on the service robot and perceives a signifier indicating the action "Move arm", meaning that this action is available. Also, this signifier has an associated recommended context: "Nobody is in the room." The agent processes this information and determines

that the action of catching the ball should only be initiated when the room is empty. This context recommendation is likely in place to ensure safety, as the action of catching a ball might entail rapid movements that could pose risks to humans in proximity.

Additionally, the signifier suggests a recommended ability: "Operate with Cartesian Coordinates." This means the robot must have the technical know-how to perform tasks based on a Cartesian coordinate system, which contrasts with a robot that operates based on joint coordinates. If the robot's programming and hardware are aligned with Cartesian coordinates operation, then it assesses that it has the competency to perform the action; otherwise, it may refrain from engaging in the ball-catching task to avoid malfunction or suboptimal performance. The agent could then acquire the ability through artifacts.

In the given scenario, the agent possesses the operational proficiency required for Cartesian Coordinate manipulation. Upon acquiring environmental data, the agent ensures that the room is indeed empty. Consequently, the agent deduces that the conditions are favorable, and it is thus appropriately positioned to execute the action, in other words, the **means for the action are available**.

This illustrative example showcases how our approach equips BDI agents with the ability to reason about actions dynamically, considering not only the possibility of actions but also the contextual and capability-related appropriateness of engaging in those actions.

### 3.2.2 Process of Action Reasoning

For the agent to reason about the actions as explained in Section 3.2.1, we built a mechanism called SRM. This mechanism is part of the BDI agents' means-end reasoning process, not the whole reasoning system. As shown in Figure 3, in addition to the beliefs gathered from the signifiers in the Action Knowledge step, the SRM also depends on the agent having a list of plans and the plans having actions to be reasoned about. Moreover, it performs better if the agent has already selected the relevant plans for the given goal, so the SRM only needs to be applied to a smaller set of plans. The mechanism allows the transformation of available abstract actions in plans into concrete actions that are ready for execution when required.

Listing 3.1 – A plan that represents the knowledge about an abstract action provided at design-time.

```
1 +!decreased_illuminance(room) : true
2          <- invokeAction("saref:ToggleCommand", ["saref:OnOffState"], [0]) ;
3             ?decreased_illuminance(room) .
```

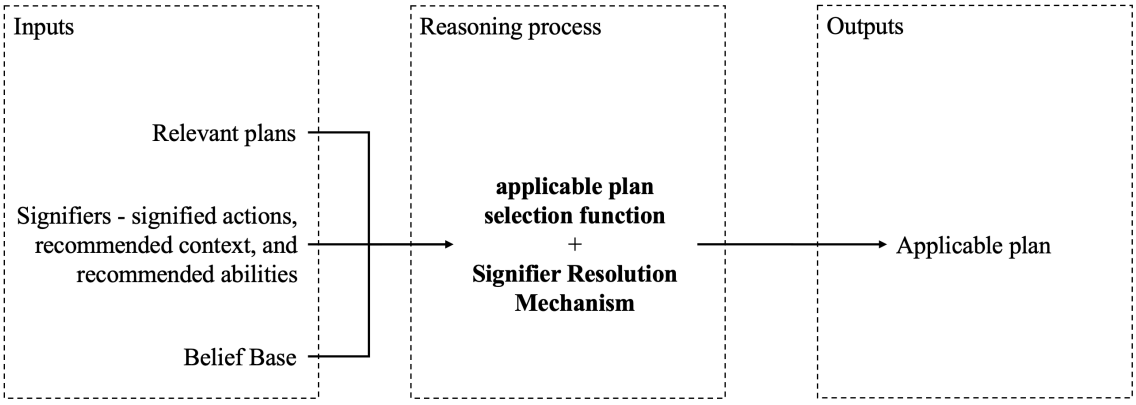


Figure 3 – Process of Action Reasoning. The inputs (Relevant plans, Signifiers and Beliefs from the Belief Base) are passed to an applicable plan selection function embedded with the SRM, returning an applicable plan as output.

Given the diversity of frameworks for BDI agents, SRMs may vary: The variation may stem from framework-specific sub-processes for means-end reasoning, their formalization using specialized concepts or other factors. In what follows, we introduce SRM functionality based on concepts inspired by Jason and AgentSpeak [9], but this does not restrict the generality of our approach — and, where applicable, we draw parallels to concepts used in other BDI models and frameworks.

We consider that abstract actions are defined within *plans*, which represent means that enable agents to execute a simple or complex series of actions. In Listing 3.1, we use a simple plan structure to show how the SRM works. Plans, alongside analogous concepts like action rules or activities, play a fundamental role in reasoning about the best approach to pursue an intention.

The intention relevant to the plan presented in Listing 3.1 is to decrease room illuminance (see Line 1, where `+!` denotes the addition of this achievement goal event in AgentSpeak [17]). This is a single-action plan whose *application context* subsumes any conditions that must hold at run-time to invoke the action (see `true` in Line 1). Finally, to address any conditions that are expected to hold post-action, the plan is declaratively programmed by adding a test goal after action invocation (see Line 3, denoted by the AgentSpeak test operator `?`). This caters to approaches where action postconditions should be verified against the agent's beliefs to evaluate action success (via the test goal addition). Finally, the plan involves the invocation of a command that is compatible to `ToggleCommand` with an `OnOffState` in the SAREF ontology on a source of illumination (see Line 2).

Listing 3.2 – Representation of a signifier which signifies a toggling action in a lamp with a generic BDI vocabulary.

```
1  @base <http://ex.org/wksp/1/arts/1>.
2  ...
11
12 <> a hmas:ResourceProfile ;
13    hmas:isProfileOf <#artifact> ;
14    hmas:exposesSignifier <#togglable> .
15
16 <#artifact> a hmas:Artifact, saref:Lamp .
17
18 <#togglable> a hmas:Signifier ;
19   hmas:recommendsAbility [ a bdi:LogicBDIReasoner ] ;
20   hmas:recommendsContext <#empty-room-constraint> ;
21   hmas:signifies <#toggle-spec> .
22
23 <#toggle-spec> a sh:NodeShape;
24    sh:class hmas:ActionExecution, saref:ToggleCommand ;
25    sh:property [
26     sh:path prov:used ;
27     sh:minCount 1;
28     sh:maxCount 1 ;
29     sh:hasValue <#form>
30    ] ;
31    sh:property [
32      sh:path hmas:hasInput;
33      sh:qualifiedValueShape <#input-constraint> ;
34      sh:qualifiedMinCount 1 ;
35      sh:qualifiedMaxCount 1
36    ] .
37
38 <#input-constraint> a sh:NodeShape ;
39   sh:class saref:OnOffState ;
40   sh:property [ sh:path saref:hasValue ;
41     sh:minCount 1 ;
42     sh:maxCount 1 ;
43     sh:hasValue 0 ;
44     sh:name "lampState" ] .
45
46 <#form> a hctl:Form ;
47   hctl:hasTarget <https://lamp.example.org/state> ;
48   htv:methodName "PUT" .
49
50  <#empty-room-constraint> a sh:NodeShape ;
51   sh:class hmas:ResourceProfile ;
52   sh:property [
53     sh:path hmas:isProfileOf ;
54     sh:qualifiedValueShape <#agent-constraint> ;
55     sh:qualifiedMinCount "1"^^xs:int;
56     sh:qualifiedMaxCount "1"^^xs:int
57   ] .
58
59 <#empty-room-constraint> a sh:NodeShape ;
60   sh:class hmas:Agent ;
61   sh:property [
62     sh:path bdi:hasBelief ;
63     sh:minCount "1"^^xs:int;
64     sh:maxCount "1"^^xs:int;
65     sh:hasValue "empty(room)"
66   ] .
```

Additionally, consider the signifier described in Listing 3.2 adapted from [65]. Expressed in RDF syntax, this signifier characterizes the resource profile for a Lamp Artifact, as indicated on line 16 of the Listing. The resource profile reveals a `togglable` signifier, as specified on line 14. This signifier:

- Signifies the action `saref:ToggleCommand` (lines 21, 23-36).
- Recommends a context `empty(room)` (lines 20, 50 - 66)
- Recommends the ability `bdi:LogicReasoner` (line 19).

Algorithm 1 outlines the SRM that is integrated into the reasoning cycle of a BDI agent to enhance its means selection function. The goal of the SRM is to enable the agent to determine if a plan can be executed by checking the availability of actions within that plan against the signifiers perceived in the environment.

**Algorithm 1** Algorithm for the SRM that extends the BDI reasoning cycle's means selection function.

$Sig \leftarrow$ perceived signifiers, $P \leftarrow$ relevant plans, $B \leftarrow$ beliefs, $Ab \leftarrow$ set of agent's abilities
$A_{Sig} \leftarrow$ set of actions signified by $Sig$
**for each** $\pi \in P$ **do**
  $A_\pi \leftarrow$ set of actions of $\pi$
  **if** $A_\pi \subset A_{Sig}$ **then**
    $MeansAvailable \leftarrow$ true
    **for each** $\alpha \in A_\pi$ **do**
      $C_{Sig} \leftarrow$ recommended context of the signifier that signifies $\alpha$
      **if** $B \not\models C_{Sig}$ **then**
        $MeansAvailable \leftarrow$ false
        **break**
      **end if**
      $Ab_{Sig} \leftarrow$ set of recommended abilities of the signifier that signifies $\alpha$
      **if** $Ab_{Sig} \not\subseteq Ab$ **then**
        $MeansAvailable \leftarrow$ false
        **break**
      **end if**
    **end for**
    **if** $MeansAvailable$ **then**
      **return** $\pi$
    **end if**
  **end if**
**end for**
**return** null

1. **Initialization:** The algorithm initializes by collecting key components:
   - $Sig$: The signifiers perceived by the agent.
   - $P$: The list of relevant plans available to the agent.
   - $B$: The current beliefs of the agent.
   - $Ab$: The set of abilities possessed by the agent.
   - $A_{Sig}$: The set of actions signified by $Sig$.
2. **Plan Evaluation Loop:** The agent iterates through each plan ($\pi$) in $P$.
3. **Means Availability Check for Actions:**
   - First, the algorithm assumes that the means (actions) required for the plan are available ($MeansAvailable$ is set to true).
   - $A_\pi$: It extracts the set of actions required for plan $\pi$.
   - The algorithm then checks whether the set of actions required for the plan ($A_\pi$) is a subset of the actions signified by the signifiers ($A_{Sig}$). If not, it implies that

at least one action required for the plan is not available in the environment, leading to setting $MeansAvailable$ to false and breaking out of the loop.

4. **Context Evaluation Loop:**

   - For each action in the plan ($\alpha \in A_\pi$), the algorithm retrieves $C_{Sig}$, the recommended context of the signifier that signifies the action $\alpha$.
   - It checks if the recommended context $C_{Sig}$ holds in the agent's current beliefs $B$. If the context does not hold, $MeansAvailable$ is set to false, and the loop breaks.

5. **Abilities Evaluation Loop:**

   - For each action in the plan ($\alpha \in A_\pi$), the algorithm retrieves $Ab_{Sig}$, the set of recommended abilities of the signifier that signifies $\alpha$.
   - It then checks if the set of recommended abilities $Ab_{Sig}$ is a subset of the agent's abilities $Ab$. If not, $MeansAvailable$ is set to false, and the loop breaks.

6. **Plan Selection:**

   - If, after evaluating all actions in a plan, $MeansAvailable$ remains true, the plan $\pi$ is returned as the chosen plan for execution.
   - If none of the plans have their means available after the evaluation (i.e., $MeansAvailable$ is false for all plans), the algorithm returns null, indicating that no applicable plan was found.

The SRM algorithm essentially extends the BDI agent's ability to adapt to the environment by incorporating a more dynamic action selection process that utilizes both environmental cues (signifiers) and the agent's internal state (beliefs and abilities) to make decisions about which plans are executable.

Referring to Listings 3.1 and 3.2, consider the scenario where an agent's objective is to reduce room illumination. To achieve this, it identifies the plan denoted as `+!decreased_illuminance(room)` from Listing 3.1 as the pertinent course of action. The SRM then steps in to evaluate this plan against available signifiers—in this instance, specifically the `togglable` signifier.

The SRM systematically reviews each action specified within the plan—here, it would be the `saref:ToggleCommand`. It verifies the action's presence, confirmed by the signifier, thus satisfying the initial requirement. Subsequently, the SRM assesses whether the context indicated by the signifier aligns with the agent's current beliefs and abilities. Given the agent's belief that the `empty(room)`, given by environmental perception, and its capability as a `bdi:LogicBDIReasoner`, the action is deemed executable, prompting the SRM to confirm its availability.

Should the plan encompass additional actions, the SRM would scrutinize them similarly. Conversely, if the context were `full(room)` or the agent lacked necessary abilities, the action would be classified as inaccessible, leading the SRM to either consider alternative signifiers or proceed to evaluate subsequent plans. The robustness of this mechanism ensures that the agent progresses logically and efficiently towards fulfilling its designated goal.

## 3.3 ACTION EXECUTION

Action execution refers to the process through which a BDI agent carries out a chosen action in order to affect its environment or achieve a specific goal.

### 3.3.1 Preparation for Action Execution

In the phase of preparation for action execution, the process by which an agent interfaces with the actions outlined by signifiers is intricate, particularly due to specified input constraints associated with these actions. Such constraints are multifaceted, often entailing a specified semantic type and data structure, which could be an integer, boolean, string, etc. Additionally, these constraints might stipulate certain boundary conditions, such as maximum or minimum values, and maximum or minimum quantity. These detailed stipulations are thoroughly documented within the resource profile representation, an example of which can be observed in Listing 3.2, specifically within lines 38 to 44.

The complexity of dynamically interpreting these constraints and ensuring that the agent provides the correct type of input for the action is a significant challenge. It requires the agent to not only discern the correct data type but also to adhere to the specified range or format as dictated by the signifier's profile. This level of dynamic interpretation is a complex computational problem that this work does not tackle directly. Instead, we operate under the assumption that if the agent possesses the capability to execute the action, it—or by extension, the developer—has prior knowledge of the appropriate input types for that action. This presupposition allows for a streamlined focus on the agent's decision-making process, without delving into the intricacies of dynamically parsing and applying input constraints in real-time.

Let us consider the action specified in Listing 3.2, the `saref:ToggleCommand`. This particular command accepts a discrete input of either 0 or 1, which semantically corresponds to the "Off" and "On" states, respectively. Such a binary input scheme simplifies the decision-making process for the agent, as the action requires no complex data types or ranges—just a straightforward selection between two states.

In accordance with this simplification, when we examine the agent's plan delineated in Listing 3.1, we observe that the action invocation method—`invokeAction`—is pre-configured with the argument value 0. This indicates that the action to be performed

is predetermined to switch the state to "Off". By hardcoding the argument within the plan, the agent bypasses the need for real-time decision-making regarding the input for this action. Such an approach, while limiting the agent's flexibility, ensures efficiency and reduces the potential for errors in action execution, presuming that the circumstances under which the plan is activated always warrant turning the toggleable device off. In future works, the challenge of dynamically reasoning about action inputs in real-time can be explored, potentially enhancing the agent's adaptability and decision-making capabilities in varying contexts. This can be helped by the ontology alignment mechanism mentioned in the introduction.

### 3.3.2 Process of Action Execution

In the execution phase of the agent's reasoning cycle within a MAS, no significant alterations are made to the traditional process. However, a critical prerequisite is introduced: the agent must be equipped with the capacity to send HTTP requests or leverage an enhanced CArtAgO artifact that handles this functionality. This capability is essential for interacting with the target endpoints, which are specified within the action descriptions (see lines 46-49 in Listing 3.2). Each action's description details not only the URL of the HTTP endpoint but also the required HTTP method—such as POST or PUT—to be used when making the request. This approach enables agents to effectively communicate and perform actions within the hypermedia environment by leveraging standard web protocols.

## 3.4 IMPLEMENTATION AND INTEGRATION WITH JASON/JACAMO

In this section, we will elucidate the implementation of action knowledge, action reasoning, and action execution within the context of JaCaMo [16], focusing on three key components: Signifiers, SRM), and HTTP communication. Each subsection will delve into the specificities of the implementation, detailing how these components interact with the BDI reasoning cycle to enhance the capabilities of agents in a MAS.

### 3.4.1 Implementation of Action Knowledge

The focus of this subsection is on implementing action knowledge, which centers around the incorporation of signifiers and the mechanisms through which agents perceive them. Signifiers [65] are the cornerstone of our approach, allowing actions to be defined as machine-readable information resources, such as in RDF format.

The hypermedia MAS structure enables two crucial capabilities:

- It allows the independent publication and modification of signifiers by various designers, and potentially by software agents themselves.

- It enables the autonomous agents to discover and utilize these signifiers at run-time. This can occur through several mechanisms, including crawling the hypermedia environment, accessing signifier repositories, exchanging signifiers among agents, or utilizing hypermedia search engines, as exemplified in [66].

To enable such functionalities, we leverage concepts from the signifier model introduced in [58] and articulate them utilizing an ontology for Hypermedia Multi-Agent Systems (hMAS) detailed in [14, 67]. The intent behind adopting these concepts is to generalize the descriptions of action possibilities within hypermedia environments and to tailor these descriptions to support the representation, discovery, and application of signifiers by BDI agents.

Initially, we assume that signifiers can be publicized and identified within a *resource profile*, as suggested in [58]. For instance, a W3C WoT TD can serve as a resource profile. In scenarios where an artifact can be characterized as a WoT Thing, its associated signifiers can be made public within a resource profile shaped as a WoT TD [32]. Within these TDs, Action Affordances delineate the methodologies through which agents may alter the artifact's condition. This metadata is vital as it encapsulates all necessary details for the definition of an action's invocation when devising an agent's plan.

A practical example of a *resource profile* containing a signifier was presented in Listing 3.2. This example demonstrates how signifiers are structured and how they encapsulate actionable metadata which agents can interpret and act upon.

#### 3.4.1.1 CArtAgO Artifact for hMAS Resource Profile

In our implementation, the intricacies of signifiers are embodied within the CArtAgO artifact [68][1]. Conceptually, signifiers reveal information about the discoverable affordances in a hypermedia-driven MAS, and though they may not be naturally akin to artifact's observable properties, this approach facilitates their integration into the agent's perceptual framework.

Observable properties within CArtAgO artifacts represent the visible state aspects, enabling agents to perceive changes through generated events. In our adaptation, signifiers, as defined by the framework in [65], are initialized through an hMAS resource profile. The artifact, upon reading and parsing this profile, translates the signifiers into observable properties within the agent's environment. As a direct consequence of this design choice, the agents, upon focusing on the CArtAgO artifact, effortlessly incorporate the signifiers into their belief base as beliefs.

The code snippet presented in Listing 3.3 provides insight into the implementation of the resource artifact. The initialization of the artifact is orchestrated by the `init()`

[1] https://github.com/Interactions-HSG/jacamo-hypermedia/blob/feature/resource-artifact/src/env/hmas/ResourceArtifact.java

function shown in line 7, which requires a resource profile file as its input parameter. Subsequent to the initialization, the `ArtifactProfileGraphReader.readFromFile()` function, invoked in line 11, processes the resource profile file, parsing its content and storing the resulting information into the `profile` variable.

Extracting signifiers is the next step, performed by `profile.getExposedSignifiers()` in line 13. This operation collates the signifiers detailed within the profile, storing them in the `signifiers` collection.

Following the extraction, the `handleExposedSignifiers()` method, which is initiated in line 15 and elaborated upon in line 21, extracts pivotal data from each signifier. Specifically, it retrieves the action types, the suggested contexts, and the requisite abilities, which are then utilized within the `informSRM()` method, called in line 44.

Within the `informSRM` method, delineated starting in line 49, the establishment of an observable property is performed in line 53. This property is crucial as it renders the signifier's details perceptible within the agent's environment, thus seamlessly integrating into the agent's perceptive framework and facilitating the observation of the signifier within the system's environment.

While conceptually signifiers extend beyond mere observable states, representing a broader spectrum of interaction possibilities with the hypermedia environment, our implementation pragmatically aligns them with observable properties for seamless integration into the agents' perception mechanisms. This decision was made to ensure that agents can become aware of and act upon the signifiers' associated actions effectively.

Listing 3.3 – Code snippet of the Resource Artifact implementation in CArtAgO.

```
1  package hmas;
2  // imports
3
4  public class ResourceArtifact extends Artifact {
5    protected ArtifactProfile profile;
6    protected Set<Signifier> signifiers;
7    public void init(String location) {
8      try {
9        // Retrieve the resource profile (here, from a local location)
10       String absoluteLocation = getAbsoluteLocation(location);
11       this.profile = ArtifactProfileGraphReader.readFromFile(absoluteLocation);
12       // Retrieve the exposed signifiers
13       this.signifiers = profile.getExposedSignifiers();
14       // Handle the exposed signifiers to inform the SRM
15       handleExposedSignifiers();
16     } catch (IOException | URISyntaxException e) {
17       failed(e.getMessage());
18     }
19   }
20   ...
21     private void handleExposedSignifiers() {
22     // Iterate over all the signifiers to inform the SRM
23     for (Signifier signifier : signifiers) {
24       // Retrieve the action types
25       Set<String> actionTypes = signifier.getActionSpecification().
```

```
            getRequiredSemanticTypes();
26        // Retrieve the recommended context
27        Set<Context> recommendedContexts = signifier.getRecommendedContexts();
28        // Retrieve the recommended abilities
29        Set<Ability> recommendedAbilities = signifier.getRecommendedAbilities();
30        List<String> recommendedBDIContexts = new ArrayList<>();
31        // Check for BDI support
32        boolean BDISupport = recommendedAbilities
33          .stream()
34          .anyMatch(ability -> ability.getSemanticTypes().contains("http://example.org/
              bdi/BDIReasoner"));
35        // Handle the context if it is compatible with BDI agents
36        if (BDISupport) {
37          recommendedBDIContexts = extractRecommendedBeliefs(recommendedContexts);
38        }
39        // Handle the abilities by retrieve all of their types
40        List<String> recommendedAbilityTypes = recommendedAbilities.stream()
41          .flatMap(ability -> ability.getSemanticTypes().stream())
42          .collect(Collectors.toList());
43        // Inform the SRM about the exposed signifiers
44        informSRM(actionTypes, recommendedBDIContexts, recommendedAbilityTypes);
45      }
46    }
47    ...
48
49      private void informSRM(Set<String> actionTypes, List<String> recommendedContext,
            List<String> recommendedAbilities) {
50        for (String actionType : actionTypes) {
51          Object[] contextArray = recommendedContext.toArray(new Object[
                recommendedContext.size()]);
52          Object[] abilitiesArray = recommendedAbilities.toArray(new Object[
                recommendedAbilities.size()]);
53          ObsProperty pr = defineObsProperty("signifier", actionType, contextArray,
                abilitiesArray);
54          log("- New observable property: " + Arrays.toString(pr.getValues()));
55        }
56      }
57    ...
58
59    }
```

By drawing on the principles of Hypermedia MAS and the concept of signifiers, we empower agents with the capability to dynamically interact with their environment, interpreting the signifiers as actionable items that inform their decision-making processes. Our pragmatic approach to implementing signifiers provides a viable and effective means of enhancing agent autonomy and responsiveness within the given MAS framework.

### 3.4.2 Implementation of Action Reasoning

In the pursuit of imbuing the BDI agents with the capability of sophisticated action reasoning, we have integrated Jason agents [10] with the SRM as detailed in Algorithm 1. This SRM has been implemented within the agents' architecture, thus embedding it deep within the Jason agents' deliberative processes. To elucidate how our agents engage in action reasoning, we provide an overview of the default Jason agent's reasoning cycle,

illustrated in Figure 4:

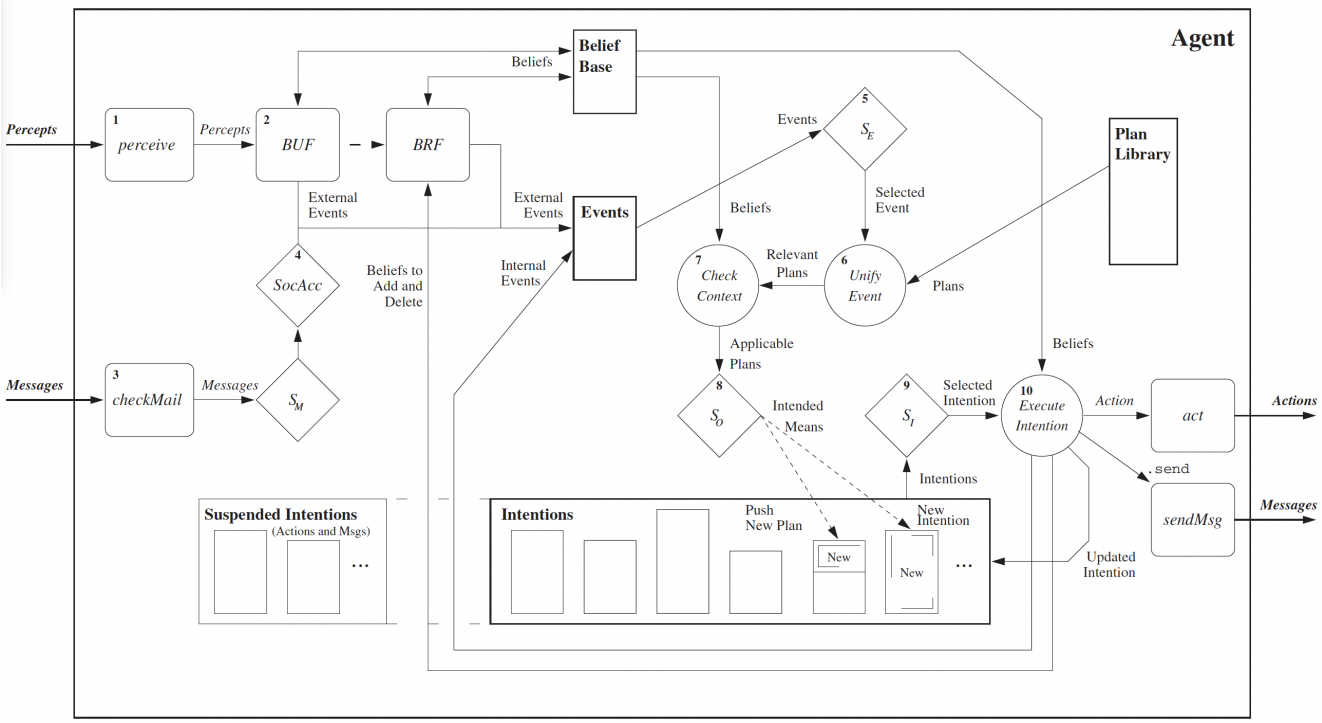


Figure 4 – Jason Reasoning Cycle. Source: [10].

1. **Perceiving the Environment:** Agents commence their reasoning cycle by perceiving the environment, updating their internal representation of the world with a set of literals that symbolize the current state.

2. **Updating the Belief Base:** Following perception, agents integrate these new literals into their belief base, reflecting changes in the environment using the *belief update function* ($BUF$).

3. **Receiving Communication from Other Agents:** Agents then check their "mailbox" for messages from their peers, potentially gaining new information that could influence subsequent reasoning steps.

4. **Selecting 'Socially Acceptable' Messages:** Received messages undergo a screening process, where a social acceptance function (`SocAcc`) determines their admissibility based on the agent's social criteria.

5. **Selecting an Event:** With potential updates to their belief base and new messages, agents select an event to address from their list of pending events using an event selection function ($S_\varepsilon$).

6. **Retrieving all Relevant Plans:** Agents consult their Plan Library to find all plans relevant to the selected event, enabling them to act in a manner that addresses the event.

7. **Determining the Applicable Plans:** Not all relevant plans are actionable. Agents must discern which ones are applicable by matching the plans' context with their current beliefs.

8. **Selecting One Applicable Plan:** From the subset of applicable plans, the agent chooses one plan to commit to, which becomes its intention for addressing the event. This is done by the *option selection function or applicable plan selection function* ($S_{\mathcal{O}}$).

9. **Selecting an Intention for Further Execution:** Agents have multiple intentions, but they will select and execute only one step of one intention during a reasoning cycle.

10. **Executing One Step of an Intention:** The agent acts upon the chosen intention, potentially modifying the environment or its internal state.

11. **Final Stage before Restarting the Cycle**: Finally, before the cycle recommences, the agent checks for any feedback or message replies that might influence its intentions, readying them for selection in the upcoming cycle. Thus, the reasoning cycle is a continuous loop of perception, deliberation, and action, enabling the Jason agent to act effectively within its environment.

Within the Jason agent's reasoning cycle, the strategic insertion of the SRM aligns with Step 8: "Selecting One Applicable Plan". This phase of the cycle is where the SRM's core functionality naturally converges, as it is inherently focused on the selection of a plan appropriate for the current context and goals.

In the default behavior, the plan selection function ($S_{\mathcal{O}}$) chooses the first plan from the roster of applicable ones. We refine this function by embedding the SRM into the process, ensuring that the plan it selects is the most aligned with the broader objectives and environmental context, as previously detailed in the discussion on action availability, recommended context, and agent abilities.

Listing 3.4 delineates the Jason implementation of Algorithm 1. The methods `getExposedActions()`, and `getExposedAbilities()` are tasked with listing the actions available in the environment and the agent's abilities, respectively. The method `getPlanActions` identifies the actions within a given plan. Additionally, `getPlanContext` and `getPlanAbilities` meticulously retrieve the recommended context and abilities associated with an action, drawing from the properties of the signifiers. The redefined `selectOption` function, essential in our implementation, receives a list of options (each representing an applicable plan) and executes the steps defined in the Algorithm 1 utilizing the methods outlined above.

Listing 3.4 – Implementation of the SRM in Jason

```
1  package signifiers;
2  //imports
3
4  public class SignifierResolutionMechanism extends Agent {
5      /* Get exposed actions from the beliefs */
6      public Set<Action> getExposedActions() {
7          PredicateIndicator predicate_indicator = new PredicateIndicator("signifier",
               3);
8          Iterator<Literal> candidate_beliefs = getBB().getCandidateBeliefs(
               predicate_indicator);
9          Set<Action> exposed_actions = new HashSet<Action>();
10         if(candidate_beliefs == null){
11             return exposed_actions;
12         }
13         while (candidate_beliefs.hasNext()){
14             Literal candidate_belief = candidate_beliefs.next();
15             Term action_name = candidate_belief.getTerm(0);
16             Term action_resource = candidate_belief.getAnnot("artifact_name").getTerm
                   (0);
17             exposed_actions.add(new Action(action_name, action_resource));
18         }
19         return exposed_actions;
20     }
21     /* Get agent abilities from the beliefs */
22     public Set<Ability> getAgentAbilities() {
23         PredicateIndicator predicate_indicator = new PredicateIndicator("ability", 1)
               ;
24         Iterator<Literal> candidate_beliefs = getBB().getCandidateBeliefs(
               predicate_indicator);
25         Set<Ability> abilities = new HashSet<Ability>();
26         if(candidate_beliefs == null){
27             return abilities;
28         }
29         while (candidate_beliefs.hasNext()){
30             Literal candidate_belief = candidate_beliefs.next();
31             Term ability_name = candidate_belief.getTerm(0);
32             abilities.add(new Ability(ability_name));
33         }
34         return abilities;
35     }
36     /* Get actions from the plan body */
37     public Set<Action> getPlanActions(Option option){
38         Plan plan = option.getPlan();
39         Set<Action> plan_actions_list = new HashSet<Action>();
40         PlanBody plan_body = plan.getBody();
41         while(plan_body != null){
42             if(plan_body.getBodyType() == BodyType.action){
43                 Literal body_term = (Literal)plan_body.getBodyTerm();
44                 if(body_term.getFunctor().equals("invokeAction")){
45                     Term action_resource = body_term.getAnnot("artifact_name").
                           getTerm(0);
46                     if(option.getUnifier().get(action_resource.toString()) != null){
47                         action_resource = option.getUnifier().get(action_resource.
                               toString());
48                     }
49                     Term action_name = body_term.getTerm(0);
50                     plan_actions_list.add(new Action(action_name, action_resource));
51                 }
52             }
53             plan_body = plan_body.getBodyNext();
```

```
54          }
55          return plan_actions_list;
56      }
57      /* Get context from an action */
58      public List<LogicalFormula> getPlanContext(Action plan_action){
59          PredicateIndicator predicate_indicator = new PredicateIndicator("signifier",
                3);
60          Iterator<Literal> candidate_beliefs = getBB().getCandidateBeliefs(
                predicate_indicator);
61          List<LogicalFormula> plan_context_list = new ArrayList<LogicalFormula>();
62          if(candidate_beliefs == null){
63              return plan_context_list;
64          }
65          while (candidate_beliefs.hasNext()){
66              Literal candidate_belief = candidate_beliefs.next();
67              Term action_name = candidate_belief.getTerm(0);
68              Term action_resource = candidate_belief.getAnnot("artifact_name").getTerm
                    (0);
69              Action exposed_action = new Action(action_name, action_resource);
70              if (plan_action.equals(exposed_action)){
71                  List<Term> context_list = (List<Term>) candidate_belief.getTerm(1);
72                  for(Term context: context_list){
73                      try {
74                          plan_context_list.add(
75                              parseFormula(
76                                  (
77                                    (StringTerm) context
78                                  ).getString()
79                                 )
80                          );
81                      } catch (ParseException e) {
82                          e.printStackTrace();
83                          return null;
84                      }
85                  }
86              }
87          }
88          return plan_context_list;
89      }
90      /* Get ability from an action */
91      public List<Ability> getPlanAbilities(Action plan_action){
92          PredicateIndicator predicate_indicator = new PredicateIndicator("signifier",
                3);
93          Iterator<Literal> candidate_beliefs = getBB().getCandidateBeliefs(
                predicate_indicator);
94          List<Ability> plan_abilities_list = new ArrayList<Ability>();
95          if(candidate_beliefs == null){
96              return plan_abilities_list;
97          }
98          while (candidate_beliefs.hasNext()){
99              Literal candidate_belief = candidate_beliefs.next();
100             Term action_name = candidate_belief.getTerm(0);
101             Term action_resource = candidate_belief.getAnnot("artifact_name").getTerm
                    (0);
102             Action exposed_action = new Action(action_name, action_resource);
103             if (plan_action.equals(exposed_action)){
104                 List<Term> abilities_list = (List<Term>) candidate_belief.getTerm(2);
105                 for (Term ability: abilities_list){
106                     plan_abilities_list.add(new Ability(ability));
107                 }
```

```
108                 return plan_abilities_list;
109             }
110         }
111         return plan_abilities_list;
112     }
113     /* Override selectOption function */
114     @Override
115     public Option selectOption(List<Option> options) {
116         if (options != null && !options.isEmpty()) {
117             /* Creating a set of exposed actions, context and abilities */
118             Set<Action> exposed_actions = getExposedActions();
119             System.out.println("\nExposed actions: "+ exposed_actions);
120             Set<Ability> agent_abilities = getAgentAbilities();
121             System.out.println("Agent abilities: "+ agent_abilities);
122             for (Option option: options) {
123                 boolean is_applicable = true;
124                 /* Creating a set of plan actions and checking if it is a subset of
                        exposed actions */
125                 Set<Action> plan_actions = getPlanActions(option);
126                 System.out.println("\nActions required for " + option.getPlan().
                        getTrigger() +": " + plan_actions);
127                 if (!exposed_actions.containsAll(plan_actions)){
128                     System.out.println("One or more actions from " + plan_actions +"
                            are not available, skipping plan...");
129                     is_applicable = false;
130                 } else{
131                     /* Creating a set of plan context and checking if it is a subset
                            of exposed contexts */
132                     for (Action action: plan_actions){
133                         List<LogicalFormula> recommended_context_list =
                                getPlanContext(action);
134                         System.out.println("Recommended contexts for " + action +": "
                                + recommended_context_list);
135                         for(LogicalFormula recommended_context:
                                recommended_context_list){
136                             Boolean contextExists = this.believes(recommended_context
                                    , new Unifier());
137                             if(!contextExists){
138                                 is_applicable = false;
139                                 System.out.println("The recommended context " +
                                        recommended_context + " is not available,
                                        skipping plan...");
140                                 break;
141                             }
142                         }
143                     }
144                     /* Creating a set of plan abilities and checking if it is a
                            subset of exposed abilities */
145                     for (Action action: plan_actions){
146                         List<Ability> recommended_abilities_list = getPlanAbilities(
                                action);
147                         System.out.println("Recommended abilities for " + action +":
                                " + recommended_abilities_list);
148                         for(Ability recommended_ability: recommended_abilities_list){
149                             if(!agent_abilities.contains(recommended_ability)){
150                                 System.out.println("The recommended ability " +
                                        recommended_ability + " is not available,
                                        skipping plan...\n");
151                                 is_applicable = false;
152                                 break;
```

```
153                             }
154                         }
155                     }
156                     if (is_applicable){
157                         System.out.println(plan_actions);
158                         return option;
159                     }
160                 }
161             }
162         }
163         return null;
164     }
165 }
```

This algorithm operates under a set of assumptions for its execution:

- The agent's abilities must be delineated within its belief base, articulated as literals in the form of `ability(ability_name)`.
- Contexts recommended by signifiers should be articulated in the observable properties as a logical formula. For example, a conjunction such as `room(empty) && temperature(high)` would symbolize two interlinked contexts prescribed by an AND logical operator.
- The actions subjected to SRM's deliberations are those invoked through the Jason function `invokeAction()`, which will be discussed in Section 3.4.3.

To finish the integration of the enhanced $S_{\mathcal{O}}$ into the agent's operational paradigm, it is necessary to make a specification within the JaCaMo Configuration Model (`.jcm`) file. This entails setting the `ag-class` parameter for the agent to correspond with the Java class as defined in Listing 3.4. An example of the configuration within the `.jcm` file to instantiate the agent with the customized selection mechanism might look as follows:

Listing 3.5 – JaCaMo Configuration for SRM Agent Class

```
1 mas jacamo_hypermedia {
2     agent bob: hmas_agent.asl {
3         ag-class: "signifiers.SignifierResolutionMechanism"
4     }
5 }
```

The result of this integration is an agent with elevated cognitive capabilities, better prepared to act in a manner that is not only reactive to the present state but also proactive in consideration of future implications, as guided by the structured reasoning the SRM provides.

### 3.4.3 Implementation of Action Execution

As discussed in Section 3.3, the implementation of this phase necessitates that the agent possesses the capability to communicate via HTTP protocols. This capacity is

critical for the agent to send requests to the target endpoints defined by the signifiers.

This functionality has been incorporated into the hMAS library, which was also utilized in the implementation of the action knowledge step. Specifically, the HTTP communication capability is embedded within the CArtAgO artifact, through the `invokeAction` function.

The `invokeAction` function has been designed to take the action as input, from where it can retrieve the target URL and the required HTTP method. It then formulates an appropriate HTTP request which is sent to the specified endpoint. The artifact ensures that the requests are well-formed and comply with the HTTP standards required by the action's signifier. By leveraging CArtAgO's capabilities and the existing hMAS library, agents can be imbued with the ability to perform actions within a hypermedia-driven environment with minimal additional development overhead.

# 4 EVALUATION

In this chapter, we present the empirical evaluation of our integration of signifiers and a SRM into the Jason reasoning cycle [10]. This assessment is crucial to validating the practicality and effectiveness of our approach within a MAS. To achieve this, we have enhanced the Jason agents' options selection function to incorporate Algorithm 0, ensuring a seamless blend of SRM into the agents' decision-making. Handling and processing of the signifiers were streamlined through a Java library that adheres to the hMAS ontology, with the agents being an integral part of a JaCaMo application. Our evaluation framework centers around distinct variables and scenarios, designed to test and differentiate the capabilities of four BDI agents—Agent A, Agent B, Agent C, and Agent D. Each agent, varying in SRM integration and reasoning abilities, offers insights into the role of signifiers in enhancing decision-making in diverse environments.

The evaluation methodology extends over various scenarios, each progressively increasing in complexity to challenge the agents' decision-making and environmental interaction capabilities. From basic plan applicability tests to advanced scenarios demanding contextual reasoning and technical proficiency, the scenarios aim to comprehensively assess the agents' performances. These tests not only shed light on the effectiveness of SRM integration in dynamic environments but also pave the way for future advancements in autonomous agents and MASs. In the following sections, we delve into the evaluation criteria, describe each scenario with its expected outcomes, and provide an overview of the results, painting a detailed picture of the operational efficacy of our SRM-integrated agents.

## 4.1 VARIABLES

Our evaluation framework is designed around a set of variables and scenarios that put the four BDI agents – Agent A, Agent B, Agent C, and Agent D – through their paces. The agents differ in their reasoning capabilities, particularly concerning how they integrate and utilize signifiers within their decision-making processes. This design allows us to assess the efficacy of signifier resolution in diverse run-time environments. Here is a breakdown of the differences among the agents, also illustrated in Table 1:

- **Agent A** stands as our control variable, functioning without the aid of an SRM. It relies on the conventional Jason reasoning cycle, executing predefined actions irrespective of their current applicability or effectiveness. The performance of Agent A is expected to highlight the limitations of a static plan selection approach in dynamic environments.
- **Agent B** represents our first experimental variable. Equipped with a basic SRM, it is capable of evaluating the immediate availability of actions. When faced with

Table 1 – Features of the different SRM versions employed by BDI agents in our evaluation.

| *SRM Features* | *A* | *B* | *C* | *D* |
|---|---|---|---|---|
| `Signified Actions` | ✗ | ✓ | ✓ | ✓ |
| `Recommended Context` | ✗ | ✗ | ✓ | ✓ |
| `Recommended Abilities` | ✗ | ✗ | ✗ | ✓ |

an action that is not available, Agent B can dynamically select an alternative plan. This flexibility is anticipated to reduce execution errors and improve the agent's responsiveness to environmental changes.

- **Agent C** adds an additional layer of complexity over Agent B. Not only does it consider action availability, but it also incorporates context assessment into its SRM. By aligning its actions with the recommended contexts signified within the environment, Agent C aims to demonstrate improved decision-making that is more attuned to the nuances of the environment it is situated in.
- **Agent D** represents the most advanced version of our SRM integration. This agent embodies a full realization of our SRM, considering not just action availability and context, but also the compatibility of its own abilities with the signifier-recommended abilities.

## 4.2 EVALUATION CRITERIA

The comprehensive evaluation of agent performance within our scenarios is grounded on two pivotal types of objectives: *Agent Objectives* and the *Environment Objective*. These objectives provide a dual perspective of assessment, considering both the agent's internal rationale and the external intentions of the environment designer.

### Agent Objectives

Agent Objectives refer to the agent's internal decision-making process and its efficacy in achieving the goals it was programmed to fulfill. This criterion scrutinizes the agent's actions and plans in terms of their rationality and effectiveness from the agent's own perspective.

- **Rational Behavior:** The agent's behavior is deemed rational if it selects and executes plans that directly contribute to achieving its predefined goals.
- **Goal Fulfillment:** The ultimate measure of success for an agent is the fulfillment of its objectives within the scenario. If an agent enacts a plan that culminates in the desired outcome, it is said to have met its Agent's Objective.

### Environment Objectives

In contrast to Agent Objectives, the Environment Objective offers an evaluation from the standpoint of the agent's role as envisioned by the environment designer. It assesses whether the agent's behavior aligns with the broader goals set for the environment it operates within.

- **Alignment with Environment Designer's Intention:** This metric evaluates how the agent's actions correspond with the environment designer's overarching vision for the agent's function and behavior within the scenario.
- **External Rationality:** An agent is also evaluated on its ability to act rationally from an external viewpoint. This involves considering the environmental context and the environment's design goals when making decisions.

To further illustrate the distinction between Agent Objectives and Environment Objectives, consider an autonomous agent responsible for energy management in a smart building. The agent is programmed with the Agent Objective of conserving energy by managing the building's lighting conditions.

- **Agent Objective Example:** The agent's strategy to achieve its energy conservation goal involves reducing the illuminance levels in the building. When it detects what it considers an underutilized room, it turns off the lights to save energy, fulfilling its Agent Objective when the plan is successfully carried out.
- **Environment Objective Example:** Now, let's imagine the agent enters a new environment: Another smart building designed with human-centric Environment Objectives, prioritizing occupant comfort over energy savings. The lights in occupied rooms are intended to remain on to ensure a comfortable and productive environment for the inhabitants. This agent was designed to conserve energy, but it is not what the designer of the new smart building intended.

In a scenario where the agent turns off the lights in a room that, despite being full of people, appears underutilized due to low activity, it satisfies its Agent Objective of energy conservation. Yet, it violates the Environment Objective, which is to maintain a comfortable environment for the building's occupants. This action, though well-intentioned from an energy-saving perspective, disregards the comfort and needs of the people, highlighting a potential oversight in the agent's operational parameters.

This dichotomy between Agent and Environment Objectives serves as a pillar for our evaluation methodology. It allows for a nuanced analysis of agent behavior, assessing not only the achievement of internal goals but also the adherence to the systemic design principles.

## 4.3 SCENARIO DESCRIPTIONS AND EXPECTED OUTCOMES

The evaluation of the autonomous agents A, B, C, and D unfolds across four distinct scenarios. These scenarios are designed to progressively increase in complexity, examining the agents' capabilities to achieve both Agent Objectives and Environment Objectives within a dynamic environment featuring various interactive artifacts.

### Scenario 1: Basic Plan Applicability

Scenario 1 establishes the baseline for agent performance. The environment is deliberately static, ensuring the availability of artifacts like a lamp and window blinds, which are integral to the agents' plans to reduce room illuminance. This scenario is crucial for ascertaining that each agent is equipped with the foundational ability to achieve the shared Agent Objective and Environment Objective of energy conservation through illuminance reduction. Since the artifacts' availability is guaranteed, the successful execution of either plan results in the attainment of the objectives, thereby providing a controlled setup to validate the agents' basic functional capabilities.

### Scenario 2: Plan Applicability based on Action Availability

The second scenario introduces dynamic elements to the environment. Artifacts may become unavailable due to a variety of factors, adding a layer of unpredictability. For instance, the lamp may be inoperative, necessitating the agents to adapt their plans accordingly. Agents B, C, and D, which incorporate an SRM, can pivot to alternative strategies, such as using the window blinds to adjust the room's illuminance. Agent A, without the benefit of an SRM, is unable to adapt to the change and persistently attempts to toggle the unavailable lamp, thereby failing to meet its objectives. This scenario tests the agents' resilience and flexibility in the face of changing environmental conditions.

### Scenario 3: Plan Applicability based on Context Recommendation

Building on the complexity, Scenario 3 introduces a Environment Objective that diverges from the Agent Objective. The Environment Objective mandates energy conservation without compromising occupant comfort, implying that actions such as reducing illuminance should only be taken when the room is unoccupied. Agents C and D, with more advanced SRMs, are capable of contextual reasoning and can withhold action if they assess the room to be occupied. Agent B, with a basic SRM, misses the mark on the Environment Objective by lowering the blinds regardless of occupancy. Agent A's limitations are once again exposed, as it cannot discern the context, repeating its errors from the previous scenario.

### Scenario 4: Plan Applicability based on Ability Recommendation

In our most complex scenario, the focus shifts to manipulating robotic arms within the room, guided by carefully designed signifiers. These signifiers outline a series of actions necessary for the accurate positioning of the robotic arms, a task that is notably more intricate than the simpler actions of light adjustment or blind operation previously encountered. Each signifier directs the same general action – repositioning of the robotic arms. However, a key variation lies in the nature of the control interfaces: some arms are maneuvered through Cartesian coordinate systems, while others operate via joint-based control systems. This divergence in operational parameters is meticulously detailed in the signifiers to ensure precise and appropriate interaction. In addition, an agent's proficiency in navigating either Cartesian or joint space is highlighted as a desirable skill, thus aiding in the identification of suitable signifiers and enhancing complex decision-making processes.

The primary objective for all agents in this scenario is to reposition the robotic arms to designated locations. Agent A consistently encounters difficulties, primarily due to its inability to adapt to robots that are either engaged elsewhere or technically incompatible, leading to a failure in meeting the objectives. Agent B also encounters similar challenges, as it does not account for the operational status of the robotic arms, resulting in unachieved objectives. Agent C shows potential for aligning with both the Agent and Environment Objectives through its context-based reasoning abilities. However, it misinterprets the required input specifications, erroneously applying Cartesian parameters to robots that are configured for joint space control. Contrastingly, Agent D demonstrates superior adaptability and technical understanding. It effectively determines whether it possesses the necessary skills to operate a given robotic arm and, if not, seeks assistance from other agents. This advanced level of reasoning enables Agent D to be the sole agent consistently achieving the intended objectives.

These scenarios collectively assess each agent's ability to meet specified objectives under varying conditions of complexity and uncertainty. They highlight the crucial role of signifiers and the SRM in equipping agents with the necessary adaptability and contextual awareness to navigate dynamic environments and fulfill both their own objectives and those envisioned by system designers.

## 4.4 SETTING UP EXPERIMENTS

For setting up the experiments, we employed a suite of technologies, each playing a crucial role in the development and execution of our MAS scenarios. This section provides an overview of these technologies and how they contribute to the experimental framework.

### 4.4.1 Agent-Oriented Programming Language

Central to our experiments is the use of an Agent-oriented programming (AOP) language, which is essential for developing sophisticated agents. Compared to traditional object-oriented programming, AOP languages offer enhanced support for defining agent-specific behaviors such as communication, belief management, commitment strategies, and decision-making processes. For our experiments, we focused on BDI agents, necessitating a language that supports the intricacies of this agent model. Jason stands out as a robust choice, being an open-source interpreter for an extended version of AgentSpeak, and offering a comprehensive set of features for practical MAS development [10]. Its compatibility with the JaCaMo framework [16] allows for intricate interactions with CArtAgO artifacts, a capability we leverage using the hmas-java library[1]. This library includes the implementation of the CArtAgO artifact that is responsible for handling the exposure of signifiers and managing the HTTP communication required for the agents' interactions within the environment.

### 4.4.2 Hypermedia Environment

To create the hypermedia environment for our experiments, we leverage the Yggdrasil open-source framework developed by researchers at the University of St.Gallen [14]. Yggdrasil stands out as a pioneering framework, enabling the development and deployment of hypermedia environments specifically tailored for autonomous agents.

For the efficient handling of these RESTful interactions, we have chosen Flask[2] as our primary framework for API development. Flask's appeal is in its simplicity and lightweight architecture, making it an excellent choice for quick development cycles and streamlined integration. Despite its minimalistic approach, Flask offers a robust set of features essential for our experimental needs. It excels at creating REST APIs, which are vital for managing the interactions between the agents and the hypermedia environment. This ability of Flask to seamlessly handle HTTP requests, along with its flexibility in routing and response handling, makes it particularly well-suited for our purposes.

### 4.4.3 Graphics Engine

A graphic environment will be created to make the tests more visible, then it will be feasible to see how things would behave in reality. Unity[3] was selected as the game engine for this purpose. Unity was chosen for the graphics because it is widely used, has high-quality assets, can be readily linked with the API using C# scripts, and has a large community that offers excellent support and a wealth of educational resources.

[1] `https://github.com/danaivach/hmas-java`
[2] `https://flask.palletsprojects.com/en/2.2.x/`
[3] `https://unity.com/`

Additionally, it contains many free packages including modeled furniture and electronics, so it is not necessary to create them from scratch.

### 4.4.4 Application and Deployment Context

For the purpose of our experiments, we utilize two separate computers to simulate a remote hypermedia environment and a local MAS environment. Both systems will be interconnected via the same WLAN network, ensuring seamless communication.

In this setup, the Jason agent is programmed to access the remote hypermedia environment hosted on the Yggdrasil node. The agent has the capability to initiate actions within this environment, which is a crucial part of our experimental interaction. These actions involve making REST requests to the Flask API, integrated with the Yggdrasil node. These requests are instrumental in modifying the state of the API endpoints, which, in turn, are reflected in the Unity simulation. The Unity simulation is designed to continuously fetch data from these endpoints, allowing it to update its graphical representation in real-time based on the evolving state of the environment.

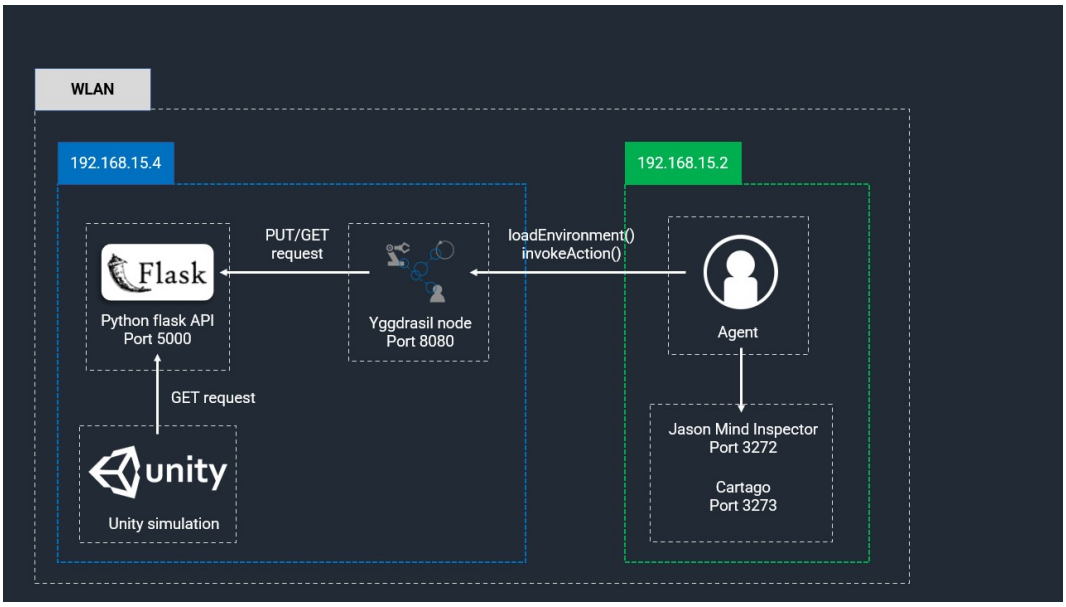


Figure 5 – Integration of Tools

The workflow of the application is illustrated in Image 5 can be delineated as follows:

1. The Yggdrasil node is initiated on port 8080 of the remote machine.
2. Resource Artifacts are uploaded to the Yggdrasil node using REST requests.
3. The Flask API is launched on the remote machine.
4. The Unity simulation starts, with its objects programmed to periodically send GET requests to the Flask API and respond to the received data.

5. The MAS is initiated, leading to the creation of the agent. The agent's Mind Inspector and the CArtAgO Inspector are made available through ports 3272 and 3273, respectively.

6. The agent employs the loadEnvironment() method to load the environment from the Yggdrasil node, which results in the creation of corresponding workspaces and artifacts.

7. To execute a desired action, the agent uses the invokeAction() function. This results in a REST request being sent to the API at the action's targeted endpoint.

8. Any change in the API's data triggers a corresponding reaction in the Unity simulation objects.

9. The process loops back to step 7, as required.

## 4.5 RESULTS OVERVIEW

The comprehensive evaluation conducted across the four scenarios provided insightful results that are in line with our projected outcomes. Table 2 summarizes the performance of each agent against both the Agent and Environment Objectives, offering a clear depiction of their capabilities and adaptability across the different scenarios.

### Performance in Basic Plan Applicability (Scenario 1)

As indicated in Table 2, Scenario 1 saw all agents, including the control Agent A, successfully meeting both the Agent and Environment Objectives. This scenario, being static in nature, did not pose a significant challenge, affirming the foundational functional capabilities of all agents.

#### Agent A

```
1 Enters the environment.
2 Focus on the artifacts.
3 Turns the lightbulb off.
```

#### Agent B

```
1 Enters the environment.
2 Focus on the artifacts.
3 Checking action availability...
4     Available action to turn the lightbulb off.
5 Turns the lightbulb off.
```

Agent C

```
1 Enters the environment.
2 Focus on the artifacts.
3 Checking action availability...
4     Available action to turn the lightbulb off.
5 Checking recommended context...
6     No recommended context.
7 Turns the lightbulb off.
```

Agent D

```
1 Enters the environment.
2 Focus on the artifacts.
3 Checking action availability...
4     Available action to turn the lightbulb off.
5 Checking recommended context...
6     No recommended context.
7 Checking recommended abilities.
8     No recommended abilities.
9 Turns the lightbulb off.
```

### Adaptability to Dynamic Environments (Scenario 2)

In Scenario 2, the dynamic nature of the environment accentuated the utility of SRM integration. Agents B, C, and D adeptly navigated the unforeseen unavailability of the lamp, as seen in their dual success marks in Table 2. Agent A, lacking an SRM, was unable to adapt, as evidenced by its double failure marks, highlighting the shortcomings of static planning in dynamic settings.

Agent A

```
1 Enters the environment.
2 Focus on the artifacts.
3 Error: Failed to turn the lightbulb off.
```

Agent B

```
1 Enters the environment.
2 Focus on the artifacts.
3 Checking action availability...
4     Unavailable action to turn the lightbulb off.
5 Checking action availability...
```

```
6     Available action to close the window blind.
7 Closes the window blind.
```

### Agent C

```
1 Enters the environment.
2 Focus on the artifacts.
3 Checking action availability...
4     Unavailable action to turn the lightbulb off.
5 Checking action availability...
6     Available action to close the window blind.
7 Checking recommended context...
8     No recommended context.
9 Closes the window blind.
```

### Agent D

```
1 Enters the environment.
2 Focus on the artifacts.
3 Checking action availability...
4     Unavailable action to turn the lightbulb off.
5 Checking action availability...
6     Available action to close the window blind.
7 Checking recommended context...
8     No recommended context.
9 Checking recommended abilities...
10    No recommended abilities.
11 Closes the window blind.
```

## Contextual Decision-Making (Scenario 3)

Table 2 clearly shows the differentiation in performance in Scenario 3. Agents C and D, with their advanced SRMs, could align their actions with the Environment Objective, as reflected in their successful completion of both objectives. Agent B, while achieving the Agent Objective, failed to consider the Environment Objective. Agent A continued to demonstrate its inability to adapt to changing conditions.

### Agent A

```
1 Enters the environment.
2 Focus on the artifacts.
3 Error: Failed to turn the lightbulb off.
```

### Agent B

```
1 Enters the environment.
2 Focus on the artifacts.
3 Checking action availability...
4     Unavailable action to turn the lightbulb off.
5 Checking action availability...
6     Available action to close the window blind.
7 Closes the window blind.
8 # Environment's objective failed, the illuminance should not be
    decreased under not recommended context.
```

### Agent C

```
1 Enters the environment.
2 Focus on the artifacts.
3 Checking action availability...
4     Unavailable action to turn the lightbulb off.
5 Checking action availability...
6     Available action to close the window blind.
7 Checking recommended context...
8     The recommended context is room(empty) is false.
9 No action performed.
10 # People leave the room
11 Checking action availability...
12     Unavailable action to turn the lightbulb off.
13 Checking action availability...
14     Available action to close the window blind.
15 Checking recommended context...
16     The recommended context is room(empty) is true
17 Closes the window blind.
```

### Agent D

```
1 Enters the environment.
2 Focus on the artifacts.
3 Checking action availability...
4     Unavailable action to turn the lightbulb off.
5 Checking action availability...
6     Available action to close the window blind.
7 Checking recommended context...
8     The recommended context is room(empty) is false
9 No action performed.
10 # People leave the room
11 Checking action availability...
```

```
12     Unavailable action to turn the lightbulb off.
13 Checking action availability...
14     Available action to close the window blind.
15 Checking recommended context...
16     The recommended context is room(empty) is true
17 Checking recommended abilities...
18     No recommended ability.
19 Closes the window blind.
```

### Advanced Reasoning and Technical Proficiency (Scenario 4)

The complexity of Scenario 4 brought the agents' abilities into sharp focus. Agent D's advanced SRM enabled it to effectively handle the technical complexities of the robotic arms, as shown by its successful marks in the table. In contrast, Agent C, despite its potential, and Agents A and B, struggled to meet the demanding requirements of this scenario.

#### Agent A

```
1 Enters the environment.
2 Focus on the artifacts.
3 Error: Failed to manipulate robot arm 1.
```

#### Agent B

```
1 Enters the environment.
2 Focus on the artifacts.
3 Checking action availability...
4     Unavailable action to operate robot arm 1.
5 Checking action availability...
6     Available action to operate robot arm 2.
7 Operates robot arm 2.
8 Error: The agent should wait the robot to be free (no operator using
      it) to be operated.
```

#### Agent C

```
1 Enters the environment.
2 Focus on the artifacts.
3 Checking action availability...
4     Unavailable action to operate robot arm 1.
5 Checking action availability...
6     Available action to operate robot arm 2.
```

```
Checking recommended context...
    The recommended context is robot_arm_2(free) is false
No action performed.
# wait robot arm to be free.
Checking action availability...
    Unavailable action to operate robot arm 1.
Checking action availability...
    Available action to operate robot arm 2.
Checking recommended context...
    The recommended context is robot_arm_2(free) is true
Operates robot arm 2.
Error: The robot arm should not be operated with Cartesian parameters
    .
```

### Agent D

```
Enters the environment.
Focus on the artifacts.
Checking action availability...
    Unavailable action to operate robot arm 1.
Checking action availability...
    Available action to operate robot arm 2.
Checking recommended context...
    The recommended context is robot_arm_2(free) is false
No action performed.
# wait robot arm to be free.
Checking action availability...
    Unavailable action to operate robot arm 1.
Checking action availability...
    Available action to operate robot arm 2.
Checking recommended context...
    The recommended context is robot_arm_2(free) is true
Checking abilities...
    The agent does not possess the ability operate(cartesian
        coordinates).
No action performed.
# agent learns the ability operate(cartesian coordinates).
Checking action availability...
    Unavailable action to operate robot arm 1.
Checking action availability...
    Available action to operate robot arm 2.
Checking recommended context...
    The recommended context is robot_arm_2(free) is true
Checking abilities...
    The agent possesses the ability operate(cartesian coordinates).
Operates robot arm 2.
```

Table 2 – Scenario results. The first mark represents the Agent Objective, and the second mark the Environment Objective.

| *Scenario* | *A* | *B* | *C* | *D* |
|---|---|---|---|---|
| 1 | ✓✓ | ✓✓ | ✓✓ | ✓✓ |
| 2 | ✗✗ | ✓✓ | ✓✓ | ✓✓ |
| 3 | ✗✗ | ✓✗ | ✓✓ | ✓✓ |
| 4 | ✗✗ | ✓✗ | ✓✗ | ✓✓ |

### Summary of Results

The results, as summarized in Table 2, validate the hypothesis that integrating a sophisticated SRM significantly enhances an agent's adaptability and decision-making in complex environments. The gradual increase in complexity across the scenarios demonstrated the growing necessity for and effectiveness of SRM capabilities in MASs. Agent D, equipped with the most advanced SRM, consistently outperformed the others, showcasing its ability to navigate intricate environments and meet both internal and external objectives.

These findings not only highlight the practical benefits of SRM integration in BDI agents but also pave the way for future research into more context-aware and adaptive agent systems. The insights gained underscore the importance of developing intelligent agents that can respond effectively to the complexities and unpredictability of real-world environments.

# 5 DISCUSSION

In this chapter, we undertake a discussion of our research structured around three core themes: firstly, we evaluate the "Advantages of the Proposed Approach", highlighting how our methodology improves agent adaptability and decision-making. Then, we address the "Challenges of the Proposed Approach", identifying the practical and technical hurdles encountered. Finally, we look toward future possibilities in "Limitations and Future Research", acknowledging the current constraints and suggesting directions for ongoing advancements in the field.

## 5.1 ADVANTAGES OF THE PROPOSED APPROACH

As we progress toward the conclusion of this work, it becomes important to highlight the advantages of our enhanced BDI agents over traditional agents, particularly in the context of dynamic hypermedia environments. Traditional agents, while effective in static and predictable settings, often struggle in the face of the ever-changing landscapes of hypermedia environments. Our approach, which integrates dynamic action reasoning capabilities, stands in contrast to these conventional methods, offering benefits that are especially pertinent in these challenging environments.

### Enhanced Responsiveness to Dynamic Environments

One of the limitations of traditional agents is their rigidity in action execution. These agents, typically programmed with a fixed set of actions, lack the ability to adapt to unforeseen changes in their environment. In contrast, our enhanced agents are equipped with the capacity to perceive and interpret environmental *signifiers*, enabling them to dynamically update their action repertoire in response to real-time changes.

### Context-Aware Decision-Making

Traditional agents often follow a predetermined plan without considering the current state of the environment, leading to inefficiencies or even failure in dynamic settings. Our approach addresses this shortcoming through the SRM, which enables agents to make decisions based on the context and suitability of actions. This advanced decision-making process ensures that actions are not only feasible but also the most appropriate given the specific circumstances, leading to more efficient and effective outcomes.

### Improved Adaptability and Evolvability

The ability to learn and adapt is a crucial requirement in hypermedia environments. Agents with static action sets are not well equipped for evolvability from the environment. Our enhanced agents, on the other hand, continuously update their knowledge base with

information about new actions and environmental changes. This continuous evolvability process ensures that the agents remain relevant and effective even as the environment evolves.

#### Contribution to Future-Proofing Agents

As hypermedia environments continue to evolve, the ability of agents to adapt and respond to new challenges becomes increasingly critical. Our approach addresses the needs of these environments laying the groundwork for agents that can evolve alongside future technological advancements, such like in initiatives as the WoT. This future-proofing aspect ensures the long-term relevance and effectiveness of our enhanced agents.

#### Prioritizing Designer's Vision in Dynamic Environments

A key distinction of our enhanced agents is their ability to align their actions with the broader Environment Objectives, a feature starkly absent in traditional agents. Traditional agents operate under a fixed set of instructions, often disregarding the evolving context or the overarching environment objectives set by the environment designer. Our enhanced agents, through the integration of SRM, demonstrate a nuanced understanding of the environment, enabling them to make decisions that not only fulfill their immediate goals but also resonate with the environment designer's long-term vision.

### 5.2 CHALLENGES OF THE PROPOSED APPROACH

Despite the advantages of this study, it is important to recognize and address the challenges and limitations of our approach. The integration of the SRM and the focus on dynamic action handling certainly augment agent capabilities, yet they also bring to light several issues that warrant attention.

#### Complexity in Signifier Interpretation and Application

One of the primary challenges lies in the complexity of interpreting and applying signifiers. While signifiers enable agents to adapt to dynamic environments, their utilization depends heavily on the agent's ability to accurately interpret these cues. Misinterpretations or oversights in understanding the signifiers can lead to inappropriate actions, ruining the purpose of dynamic adaptation.

#### Reliance on Environment Design and Signifier Accuracy

The efficacy of our approach relies upon the environment's design and the accuracy of the signifiers within it. If the environment lacks well-defined signifiers or if the signifiers are inaccurately represented, agents may struggle to make informed decisions. This

dependency places a responsibility on the environment designers to ensure that signifiers are both present and precise.

### Computational Overhead and Performance Concerns

The integration of SRM into the agents' reasoning cycle introduces additional computational overhead. Processing signifiers, evaluating contexts, and matching actions with abilities demand more from the agents' computational resources. In scenarios with a high density of signifiers and complex action dynamics, we don't know if it could lead to performance bottlenecks, impacting the agents' efficiency.

### Challenges in Multi-Agent Coordination

In MASs, coordinating actions among agents becomes more complex with the introduction of SRM. Ensuring that agents not only operate efficiently on an individual level but also collaborate effectively in a shared environment with dynamic action possibilities is a challenging endeavor. The risk of conflicting actions or misaligned objectives among agents is amplified in such settings.

### Adaptability to Highly Dynamic Environments

While our approach enhances adaptability, extremely dynamic environments where signifiers and actions change rapidly can still pose challenges. The agents' ability to continuously update their beliefs and plans in such rapidly shifting contexts is a critical aspect that needs further exploration and optimization.

### Scalability and Practical Deployment Issues

Scaling this approach to larger, more complex systems or different domains may reveal additional challenges. Practical deployment issues, such as integrating with existing systems, handling legacy environments, or adapting to various technological standards and protocols, need to be addressed for broader applicability.

### Limitations in Current Implementation

Our current implementation, while effective in demonstrating the proposed approach, has its limitations. For instance, the assumption that agents have prior knowledge of appropriate parameter types for actions may not hold in all scenarios. The challenge of dynamically reasoning about action parameters in run-time remains an area for future development.

## 5.3 LIMITATIONS AND FUTURE RESEARCH

While our current approach shows promise, specific limitations highlight areas that could be explored in future research. Addressing these areas can lead to significant advancements for this work and for the behavior of agents within dynamic, complex environments.

### Dynamic Discovery and Validation of Action Parameters

**Limitation:** The assumption of prior knowledge of action parameters limits the agents' flexibility and adaptability in dynamic environments.

**Future Research Recommendations:**

- **Run-time Parameter Type Validation:** Explore techniques for run-time discovery and validation of parameter types required for actions, enabling agents to handle a wider range of scenarios effectively.
- **Developing Adaptive Learning Mechanisms:** Investigate machine learning algorithms that allow agents to dynamically infer or learn the requirements for action parameters based on environmental cues or historical interactions.

### Optimized Storage and Management of Signifier Knowledge

**Limitation:** Storing signifier knowledge in the belief base can lead to inefficiencies and scalability issues.

**Future Research Recommendations:**

- **Decentralized Knowledge Management:** Develop specialized artifacts or dedicated databases to store signifier information, easing the cognitive load on agents and improving system performance.
- **Utilizing Remote Ontologies:** Research the integration of remote ontologies for a more structured, centralized approach to managing signifier knowledge, facilitating quicker updates and better scalability.

### Comprehensive Analysis of Actions in Subplans

**Limitation:** Current models overlook actions contained within subplans, which can lead to incomplete action reasoning and execution.

**Future Research Recommendations:**

- **Recursive Subplan Analysis:** Enhance the SRM algorithm to include recursive analysis of subplans, ensuring a comprehensive evaluation of all plan components.

### 5.3.1 Other recommended themes

In addition to addressing these specific limitations, future research could also consider the following themes:

- **Interoperability and Standardization:** As agents increasingly interact within diverse and open systems, research should focus on developing standardized protocols and frameworks to ensure seamless interoperability across different platforms and environments.
- **Performance Optimization:** As systems grow in complexity, optimizing the performance of agents in terms of computational resources and response times remains a key challenge.

## 5.4 CONTRIBUTIONS

Our work presents a blend of conceptual and practical tooling enhancements, each contributing to the Multi-Agents Systems field. Our approach not only advances theoretical understanding but also provides tangible tools for developers and researchers in MAS.

### Conceptual Contribution

At the core of our conceptual contribution is the approach to enhancing BDI agents' ability to operate within dynamic and open hypermedia environments. Our methodology emphasizes three critical aspects: action knowledge, action reasoning, and action execution, each intricately linked to environmental cues known as signifiers. This approach deviates from traditional static action definitions, allowing agents to dynamically recognize and reason about potential actions in their environment. By integrating signifiers into the agents' belief bases, we have enriched the conventional BDI reasoning cycle, enabling agents to adapt their decision-making processes in run-time based on the evolving landscape of their operational environment. This paradigm shift not only enhances the adaptability and efficacy of BDI agents but also aligns with the ever-evolving nature of technologies like the WoT.

### Tooling Contribution

Complementing our conceptual advancements, our tooling contributions are centered around the development of the SRM, an algorithm integrated into the Jason agents' reasoning cycle (https://github.com/bruno-szdl/signifier-resolution-mechanism). The SRM allows agents to assess the viability of actions based on their availability, the recommended context, and the agents' own abilities. This mechanism transforms abstract plans into concrete actions, ready for execution in run-time, thereby enhancing the agents' ability to interact with and adapt to their environment.

Moreover, we have extended the functionality of CArtAgO artifacts to encompass the handling of signifiers, enabling the agents to perceive and interact with environmental cues seamlessly. This integration facilitates a more intuitive and dynamic interaction between agents and their operational environments. Additionally, the inclusion of HTTP communication capabilities within the CArtAgO artifacts empowers agents to execute actions in hypermedia environments effectively, bridging the gap between theoretical models and practical applications.

# 6 CONCLUSION

This work aimed to enhance the action-handling and decision-making capabilities of BDI agents operating in dynamic hypermedia environments. Our focus was centered on equipping these agents with the capability to perceive, evaluate, and effectively execute actions within such environments, and to discern situations where actions to fulfill their goals were not available. This objective was achieved through a focus on Action Knowledge, Action Reasoning, and Action Execution, supported by the use of signifiers and the implementation of the SRM. This approach has improved the agents' adaptability, allowing them to navigate and respond to evolving environments with more powerful decision-making capabilities. Our implementation showed that agents with these capabilities are more likely to successfully achieve both the agents' and the environment's objectives.

The importance of this study lies in its contribution to the field of MASs and hypermedia environments. By addressing the challenges of dynamic action handling, we have laid the groundwork for future-proofing BDI agents, ensuring their continued relevance and effectiveness as technologies evolve. Our work also provides a valuable contribution to the community, with an extension of Jason that can be utilized and further developed by researchers and developers alike.

Looking ahead, there are several directions for future research. Addressing the current limitations, such as the complexity of signifier interpretation and the computational demands of SRM, will be crucial. Future work could explore the dynamic discovery and validation of action parameters, optimized management of signifier knowledge, and improvements in Multi-Agent coordination. Additionally, ensuring interoperability and standardization across diverse systems and enhancing agent performance in more complex scenarios will be key areas to focus on. Advances in BDI agents in dynamic environments are moving forward, and this work is a significant step in this direction.

# BIBLIOGRAPHY


1. Guinard, D., Trifa, V., Mattern, F. & Wilde, E. in *Architecting the Internet of Things* 97–129 (Springer Berlin Heidelberg, 2011).

2. Gibbins, N., Harris, S. & Shadbolt, N. Agent-based Semantic Web Services. *Journal of Web Semantics* **1,** 141–154 (Feb. 2004).

3. Huhns, M. Agents as Web services. *IEEE Internet Computing* **6,** 93–95 (July 2002).

4. Gregori, M. E., Cámara, J. P. & Bada, G. A. *A jabber-based multi-agent system platform* in *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems - AAMAS '06* (ACM Press, 2006).

5. Mitrović, D., Ivanović, M., Budimac, Z. & Vidaković, M. Radigost: Interoperable web-based multi-agent platform. *Journal of Systems and Software* **90,** 167–178 (Apr. 2014).

6. Berners-Lee, T., Hendler, J. & Lassila, O. The Semantic Web: A New Form of Web Content That is Meaningful to Computers Will Unleash a Revolution of New Possibilities. *ScientificAmerican.com* (May 2001).

7. Ciortea, A. *et al. A Decade in Hindsight: The Missing Bridge Between Multi-Agent Systems and the World Wide Web* in *AAMAS 2019 - 18th International Conference on Autonomous Agents and Multiagent Systems* (Montréal, Canada, May 2019), 5.

8. Wooldridge, M. & Jennings, N. R. Intelligent agents: theory and practice. *The Knowledge Engineering Review* **10,** 115–152 (June 1995).

9. Bordini, R. H., Wooldridge, M. & Hubner, J. F. *Programming Multi-Agent Systems in AgentSpeak using Jason* en (Wiley-Blackwell, Hoboken, NJ, Oct. 2007).

10. Bordini, R. H. & Hübner, J. F. in *Lecture Notes in Computer Science* 143–164 (Springer Berlin Heidelberg, 2006).

11. Hendler, J. Where Are All the Intelligent Agents? *IEEE Intelligent Systems* **22,** 2–3 (May 2007).

12. Fielding, R. T. *REST: Architectural Styles and the Design of Network-based Software Architectures* Doctoral dissertation (University of California, Irvine, 2000).

13. Bizer, C., Heath, T. & Berners-Lee, T. Linked Data - The Story So Far. *International Journal on Semantic Web and Information Systems* **5,** 1–22 (July 2009).

14. Ciortea, A., Boissier, O. & Ricci, A. in *Engineering Multi-Agent Systems* 285–301 (Springer International Publishing, 2019).

15. Weyns, D. *et al. Agent Environments for Multi-agent Systems – A Research Roadmap* (Nov. 2015).

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

17. Rao, A. S. in *Lecture Notes in Computer Science* 42–55 (Springer Berlin Heidelberg, 1996).

18. Fikes, R. E. & Nilsson, N. J. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* **2,** 189–208 (Dec. 1971).

19. Georgeff, M. P. & Lansky, A. L. *Reactive Reasoning and Planning* in *Proceedings of the Sixth National Conference on Artificial Intelligence - Volume 2* (AAAI Press, Seattle, Washington, 1987), 677–682.

20. Weiss, G. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence* (CogNet, 1999).

21. E4MAS 2004 (2004 : New York, N. Y. ) *Environments for multi-agent systems* 2005th ed. en (eds Weyns, D., Van Dyke Parunak, H & Michel, F.) (Springer, Berlin, Germany, Feb. 2005).

22. Scerri, P., Vincent, R. & Mailler, R. in *Coordination of Large-Scale Multiagent Systems* 53–71 (Springer-Verlag).

23. Jamshidi, M. *Large scale systems* en (Elsevier Science, London, England, Feb. 1983).

24. Shehory, O. in *Agent-Oriented Software Engineering: First International Workshop, AOSE 2000 Limerick, Ireland, June 10, 2000 Revised Papers* 77–90 (Springer-Verlag, Berlin, Heidelberg, 2009).

25. Trypuz, R. *Formal Ontology of Action: a unifying approach* PhD thesis (Jan. 2007).

26. Kemke, C. *About the ontology of actions* tech. rep. (Technical Report MCCS-01-328, Computing Research Laboratory, New Mexico ..., 2001).

27. Kemke, C. An Action Ontology Framework for Natural Language Interfaces to Agent Systems An Action Ontology Framework for Natural Language Interfaces to Agent Systems. *Artificial Intelligence Review, Springer* (2007).

28. Asprino, L. *et al. An Ontology Design Pattern for Supporting Behaviour Arbitration in Cognitive Agents* in *WOP@ISWC* (2016).

29. Masolo, C., Borgo, S., Gangemi, A., Guarino, N. & Oltramari, A. *WonderWeb Deliverable D18 Ontology Library* in (2003).

30. Lanthaler, M. *Hydra Core Vocabulary: A Vocabulary for Hypermedia-Driven Web APIs* Unofficial Draft. http://www.hydra-cg.com/spec/latest/core/ (Hydra W3C Community Group, July 2021).

31. Wilde, E. *Putting Things to REST* tech. rep. UCB iSchool Report 2007-015 (University of California at Berkeley, 2007).

32. Käbisch, S., Korkan, E. & McCool, M. *Web of Things (WoT) Thing Description 1.1* W3C Proposed Reccommendation. https://www.w3.org/TR/2023/PR-wot-thing-description11-20230711/ (W3C, July 2023).

33. Charpenay, V. & Kovatsch, M. *Hypermedia Controls Ontology* W3C Internal Document. https://www.w3.org/2019/wot/hypermedia (W3C, 2023).

34. Huhns, M. & Singh, M. Service-oriented computing: key concepts and principles. *IEEE Internet Computing* **9,** 75–81 (Jan. 2005).

35. Singh, M. P. & Huhns, M. N. *Service-oriented computing* en (John Wiley & Sons, Nashville, TN, Feb. 2006).

36. Gandon, F. *Distributed Artificial Intelligence And Knowledge Management: Ontologies And Multi-Agent Systems For A Corporate Semantic Web* tech. rep. (Université Nice Sophia Antipolis, 2022).

37. Dikenelli, O., Erdur, R. C. & Gümüs, Ö. *SEAGENT: a platform for developing semantic web based multi agent systems* in *AAMAS '05* (2005).

38. Exposito, J., Ametller, J., & Roble, S. *Configuring the JADE HTTP MTP.* ttp://jade.tilab.com/documentation/tutorials-guides/configuring-the-jade-http-mtp/. 2010.

39. Gouaich, A. & Bergeret, M. *REST-A: An agent virtual machine based on REST framework* in *PAAMS'10: 8th International Conference on Practical Applications of Agents and Multi-Agent Systems* (University of Salamanca, Spain, Apr. 2010), N/A.

40. O'Neill, E., Lillis, D., O'Hare, G. M. P. & Collier, R. W. in *Engineering Multi-Agent Systems* 1–20 (Springer International Publishing, 2020).

41. Groen, F. C. A., Spaan, M. T. J., Kok, J. R. & Pavlin, G. in *Logic, Language, and Computation* 154–165 (Springer Berlin Heidelberg, 2007).

42. Sasaki, Y. in *Handbook of Systems Sciences* 337–352 (Springer Singapore, 2021).

43. Albrecht, S. V. & Stone, P. Autonomous agents modelling other agents: A comprehensive survey and open problems. *Artificial Intelligence* **258,** 66–95 (May 2018).

44. Salamon, T. *Design of agent-based models* (Bruckner Tomas. Repin, Repin, Czech Republic, Sept. 2011).

45. Weyns, D., Omicini, A. & Odell, J. Environment as a first class abstraction in multiagent systems. *Autonomous Agents and Multi-Agent Systems* **14,** 5–30 (July 2006).

46. Calvaresi, D. *et al.* Real-time multi-agent systems: rationality, formal model, and empirical results. *Autonomous Agents and Multi-Agent Systems* **35** (Feb. 2021).

47. Shehory, O. & Sturm, A. in *Agent-Oriented Software Engineering* 57–78 (Springer Berlin Heidelberg, 2014).

48. Palau, A. S., Dhada, M. H. & Parlikad, A. K. Multi-agent system architectures for collaborative prognostics. *Journal of Intelligent Manufacturing* **30,** 2999–3013 (June 2019).

49. Tianfield, H., Tian, J. & Yao, X. On the architectures of complex multi-agent systems, 195–206 (Nov. 2003).

50. Schurr, N. *et al. The Future of Disaster Response: Humans Working with Multiagent Teams using DEFACTO* in *AAAI Spring Symposium: AI Technologies for Homeland Security* (2005).

51. Rogers, A., David, E., Jennings, N. R. & Schiff, J. The effects of proxy bidding and minimum bid increments within eBay auctions. *ACM Transactions on the Web* **1,** 9 (Aug. 2007).

52. Winikoff, M. & Padgham, L. in *Multiagent Systems, Second Edition* (ed Weiss, G.) chap. 15 (MIT Press, 2013).

53. Wooldridge, M. in *Multiagent Systems, Second Edition* (ed Weiss, G.) chap. 1 (MIT Press, 2013).

54. Dastani, M., van Birna Riemsdijk, M. & Meyer, J.-J. C. Programming Multi-Agent Systems in 3APL. *Multi-agent programming: Languages, platforms and applications* (2005).

55. Dennis, L. A. & Farwer, B. *GWENDOLEN: A BDI Language for Verifiable Agents* in *Proceedings of the AISB 2008 Symposium on Logic and the Simulation of Interaction and Reasoning, Society for the Study of Artificial Intelligence and Simulation of Behaviour* (2008).

56. Hindriks, K. V. in *Multi-agent programming: Languages, tools and applications* (Springer, 2009).

57. W. Collier, R., O'Neill, E., Lillis, D. & O'Hare, G. *MAMS: Multi-Agent MicroServices* in *Companion Proceedings of the 2019 World Wide Web Conference* (Association for Computing Machinery, 2019).

58. Vachtsevanou, D., Ciortea, A., Mayer, S. & Lemée, J. *Signifiers as a First-Class Abstraction in Hypermedia Multi-Agent Systems* in *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems* (International Foundation for Autonomous Agents and Multiagent Systems, 2023).

59. Charpenay, V., Käfer, T. & Harth, A. *A Unifying Framework for Agency in Hypermedia Environments* in *International Workshop on Engineering Multi-Agent Systems* (2021).

60. Antakli, A. *et al. AJAN: An Engineering Framework for Semantic Web-Enabled Agents and Multi-Agent Systems* in *International Conference on Practical Applications of Agents and Multi-Agent Systems* (2023).

61. Sorici, A. & Florea, A. M. *Towards Context-based Authorizations for Interactions in Hypermedia-Driven Agent Environments-The CASHMERE framework* in *International Workshop on Engineering Multi-Agent Systems* (2023).

62. Norman, D. *The design of everyday things* 2nd ed. en (Basic Books, London, England, Nov. 2013).

63. Archibald, B., Calder, M., Sevegnani, M. & Xu, M. in *Lecture Notes in Computer Science* 262–281 (Springer International Publishing, 2021).

64. Demarchi, F. *Utilização de agentes com ontologias remotas para a produção de conteúdo significante a partir de informações disponíveis na Web Semântica* MA thesis (Universidade Federal de Santa Catarina, Centro Tecnológico, Programa de Pós-Graduação em Ciência da Computação, 2018).

65. Vachtsevanou, D. *et al.* Enabling BDI Agents to Reason on a Dynamic Action Repertoire in Hypermedia Environments. *AAMAS2024.* (Under submission) (2023).

66. Bienz, S., Ciortea, A., Mayer, S., Gandon, F. & Corby, O. *Escaping the Streetlight Effect: Semantic Hypermedia Search Enhances Autonomous Behavior in the Web of*

*Things* in *Proceedings of the 9th International Conference on the Internet of Things* (Association for Computing Machinery, 2019).

67. Ciortea, A. *et al. Hypermedia MAS Core Ontology* tech. rep. https://purl.org/hmas/core (HyperAgents, Nov. 2021).

68. Ricci, A., Piunti, M., Viroli, M. & Omicini, A. in *Multi-Agent Programming* 259–288 (Springer US, 2009).