UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CURSO DE SISTEMAS DE INFORMAÇÃO

Flávio Silvino

Modelo para embasamento simbólico fundamentado em teoria de Agentes BDI e Redes Neurais

Florianópolis
2021

Flávio Silvino

Modelo para embasamento simbólico fundamentado em teoria de Agentes BDI e Redes Neurais

Trabalho Conclusão do Curso de Graduação em Sistemas de Informação do Centro Tecnológico da Universidade Federal de Santa Catarina como requisito para a obtenção do título de Bacharel em Sistemas de Informação
Orientador: Prof. Elder Rizzon Santos, Dr..

Florianópolis
2021

Flávio Silvino

Modelo para embasamento simbólico fundamentado em teoria de Agentes BDI e Redes Neurais

Este Trabalho de Conclusão de Curso foi julgado adequado para obtenção do Título de "Bacharel em Sistemas de Informação" e aprovado em sua forma final pelo Departamento de Informática e Estatística da Universidade Federal de Santa Catarina.

_____

Prof. Dr. Cristian Koliver
Coordenador do Curso

Banca Examinadora:

_____

Prof. Dr. Elder Rizzon Santos
Orientador

_____

Prof. Dr. Thiago Ângelo Gelaim

_____

Prof. Me. Rodrigo Rodrigues Pires de Mello

AGRADECIMENTOS

Aos meus pais, pela dedicação, carinho e amor dedicados a mim e por estarem sempre ao meu lado, a meu irmão e a toda a minha família.

Especialmente à minha mãe, que enfrenta uma batalha contra a leucemia, diagnosticada em 2020 e que, mesmo com todas as dificuldades, se preocupou em todos os momentos com a continuidade dos meus estudos.

À minha esposa, que sempre esteve ao meu lado, que me deu todo o suporte e abdicou de diversos momentos de lazer, além de adiar a realização de alguns planos de vida para que eu pudesse me dedicar aos estudos e ao trabalho.

Ao meu orientador, que me deu todo o apoio, me guiou durante todo o desenvolvimento deste trabalho de uma maneira incrível, compartilhou todo o seu conhecimento e, por muitas vezes, fez muito além do seu papel como orientador.

RESUMO

A incapacidade dos computadores de relacionar os símbolos com os seus referentes no mundo real caracteriza o Problema do Embasamento Simbólico, definido por Harnad, em 1990. Desde a sua definição, diversas tentativas de soluções foram propostas para o problema, sendo que as mesmas podem ser classificadas em abordagens representacionalistas, semi-representacionalistas e não-representacionalistas. Inspirado nas abordagens representacionalistas, este trabalho propõe um modelo para o embasamento simbólico fundamentado em teoria de agentes BDI e modelos conexionistas para reconhecimento de objetos. Para tal, é analisado o estado da arte em modelos para embasamento simbólico, detecção de objetos e agentes BDI. Um modelo que integra partes do conhecimento do agente a uma respectiva representação conexionista é proposto. Um protótipo deste modelo é apresentado e um experimento é definido para a avaliação do mesmo. Por fim, é realizada a análise dos resultados do experimento e são apresentadas as conclusões. Apesar de contribuir para a tarefa de embasamento de símbolos do agente BDI, conectando-os aos referentes no mundo real, através da disponibilização de uma biblioteca com três modelos para embasamento de crenças de um agente BDI que possui uma arquitetura de fácil utilização e extensão, este trabalho não resolve o problema do embasamento simbólico, visto que a rede neural de detecção de objetos precisa ser treinada com dados que são fornecidos de forma extrínseca. Portanto, o Problema do Embasamento Simbólico segue em aberto para a exploração de trabalhos futuros, sendo que há a possibilidade de evolução do modelo proposto neste trabalho.

Palavras-chave: Agentes. Detecção de Objetos. Symbol Grounding Problem. Inteligência Artificial.

ABSTRACT

The inability of computers to relate symbols to their referents in the real world characterizes the Symbol Grounding Problem, defined by Harnad, in 1990. Since its definition, several solutions attempts have been proposed for the problem, and they can be classified into representationalist, semi-representationalist and non-representationalist approaches. Inspired by representationalist approaches, this work proposes a model for the symbol grounding based on the theory of BDI agents and connectionist models for object detection. To this end, the state of the art in models for symbol grounding, object detection and BDI agents is analyzed. A model that integrates parts of the agent's knowledge with a respective connectionist representation is proposed. A prototype of the model is presented and an experiment is defined for its evaluation. Finally, the results of the experiment are analyzed and the conclusions are presented. Despite contributing to the task of grounding symbols of the BDI agent, connecting them to referents in the real world, through the provision of a library with three models for grounding beliefs of a BDI agent that has an architecture that is easy to use and extend, this work does not solve the Symbol Grounding Problem, since the neural network of object detection needs to be trained with data that are provided extrinsically. Therefore, the Symbol Grounding Problem remains open for the exploration of future works, with the possibility of evolution of the model proposed in this work.

Keywords: Agents. Object Detection. Symbol Grounding Problem. Artificial Intelligence.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

A Inteligência Artificial (IA) abrange as mais diversas áreas, desde aprendizado e percepção até jogos de xadrez ou diagnóstico de doenças. Por automatizar as tarefas intelectuais, a IA possui grande relevância para qualquer área da atividade intelectual humana (NORVIG; RUSSEL, 2004). Segundo Luger (2009), a Inteligência Artificial pode ser definida como um ramo da ciência da computação que se preocupa com a automação do comportamento inteligente. Porém, a IA possui várias definições, visto que o tema é abordado de forma diferente por diferentes autores.

As definições de IA, em linhas gerais, dividem-se entre as que estão relacionadas a processos de pensamento e raciocínio e as que se referem ao comportamento. Além disso, as definições medem o sucesso de maneiras diferentes, podendo fazê-lo em termos da fidelidade ao desempenho humano ou comparando-o a um conceito ideal de inteligência (NORVIG; RUSSEL, 2004).

Independentemente da definição, o estudo da Inteligência Artificial envolve a busca pelo entendimento do funcionamento do pensamento humano e, para tanto, conta com contribuições de diversas áreas como a Filosofia e a Psicologia. Segundo Searle (1980), a IA pode ser dividida em dois grandes grupos: a IA fraca e a IA forte. A primeira, caracteriza-se pela simulação do comportamento inteligente, ou seja, ela permite que sejam testadas hipóteses de como a mente funciona, mas não pretende alcançar a inteligência humana. Já a IA forte afirma que os programas, se desenvolvidos corretamente, são capazes de entender e de possuir estados cognitivos, assim como as mentes humanas. Um dos principais desafios é entender como a mente humana atribui significado aos símbolos e como funciona o processo de cognição.

Símbolos normalmente são uma convenção formal de notação utilizada pelos usuários do sistema simbólico ao qual fazem parte e não estão ligados aos seus respectivos significados (HARNAD, 1990). Em 1990, Harnad formulou o que chamou de “Symbol Grounding Problem” (Problema do Embasamento Simbólico). Ele coloca o problema em duas perguntas principais: “Como a interpretação semântica de um sistema formal de símbolos pode ser intrínseca ao sistema, em vez de apenas parasitária nos significados de nossas cabeças? Como os significados de símbolos sem sentido, manipulados exclusivamente com base em suas formas (arbitrárias), podem ser fundamentados em qualquer coisa, exceto em outros símbolos sem significado?”. Em resumo, o Problema do Embasamento Simbólico questiona como desenvolver um sistema inteligente que seja capaz de atribuir significado a um símbolo de maneira autônoma, sem qualquer influência externa.

As propostas de soluções para o Problema do Embasamento Simbólico são organizadas em três abordagens principais: as representacionalistas, as semi-representacionalistas e as não representacionalistas (TADDEO; FLORIDI, 2005). As abordagens representacionalistas, em especial, buscam resolver o problema através da conexão das percepções do agente com os símbolos. Tratando-se de processamento de percepções, as abordagens de inteligência artificial conexionista, especialmente as Redes Neurais, possuem grandes vantagens em relação às abordagens simbólicas.

Com base nesta abordagem, o presente trabalho tem o objetivo de propor um modelo que aplica a abordagem representacionalista, realizando o embasamento simbólico das crenças de um Agente BDI através do uso de Redes Neurais que farão parte das percepções do agente.

1.1 OBJETIVOS

1.1.1 Objetivo Geral

Propor um modelo para o embasamento simbólico fundamentado em teoria de agentes BDI e modelos conexionistas para reconhecimento de objetos.

1.1.2 Objetivos Específicos

1. Analisar o estado da arte em modelos para embasamento simbólico, detecção de objetos e agentes BDI (representação de conhecimento e ações).
2. Propor um modelo que integre partes do conhecimento do agente a uma respectiva representação conexionista.
3. Desenvolver um protótipo do modelo utilizando-se de um ou mais modelos para reconhecimento de objetos e uma arquitetura para agentes BDI.
4. Definir e desenvolver um experimento para avaliar o modelo proposto.
5. Analisar os resultados práticos do experimento com relação ao modelo proposto.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresentará os principais conceitos relacionados com o Problema do Embasamento Simbólico, bem como suas principais abordagens e soluções propostas. Serão

abordados também os conceitos e técnicas na tarefa de detecção de objetos. Além disso, serão conceituados os tópicos referentes à agentes, suas teorias e arquiteturas. O capítulo está organizada da seguinte forma: na seção 2.1 são apresentados os conteúdos relacionados com o problema do embasamento simbólico, na seção 2.2 os conceitos relacionados com redes neurais e detecção de objetos são apresentados e, na seção 2.3, são abordados os conceitos relacionados à teoria de agentes e a arquitetura de agentes BDI.

2.1 PROBLEMA DO EMBASAMENTO SIMBÓLICO

Definido por Harnad, em 1990, o Symbol Grounding Problem (Problema do Embasamento Simbólico) questiona como os símbolos são conectados com os seus respectivos significados, de forma que não estejam embasados apenas em outros símbolos sem embasamento. Em outras palavras, o problema questiona como a interpretação semântica de um sistema formal de símbolos pode se tornar intrínseca ao sistema e não depender dos significados em nossas cabeças.

Segundo o modelo simbólico da mente, a mente é um sistema simbólico e a cognição é a manipulação de símbolos. Para refutar tal afirmação, Harnad (1990) utiliza dois exemplos que demonstram falhas neste raciocínio e também explicitam o problema do embasamento simbólico. O primeiro exemplo é o que ficou conhecido como o Argumento do Quarto Chinês, proposto por Searle (1980):

> "Suponha que eu esteja trancado em uma sala e receba um grande lote de chinês, escrito ou falado, e que eu não sou confiante nem mesmo de que poderia reconhecer escrita chinesa como escrita chinesa, distinta de, digamos escrita japonesa ou rabiscos sem sentido. Para mim, a escrita chinesa não passa de muitos rabiscos sem sentido. Agora suponha que, após este primeiro lote de escrita chinesa, eu receba um segundo lote de escrita chinesa junto com um conjunto de regras para correlacionar o segundo lote com o primeiro. As regras estão em inglês e eu entendo estas regras tão bem quanto qualquer outro falante nativo de inglês. Elas me permitem correlacionar um conjunto de símbolos formais com outro conjunto de símbolos formais, e tudo que "formal" significa aqui é que eu posso identificar os símbolos inteiramente por suas formas. Agora suponha também que me é fornecido um terceiro lote de símbolos chineses junto com algumas instruções, novamente em inglês, que me permitem correlacionar elementos do terceiro lote com os dois primeiros lotes, e essas regras me instruem como devolver certos símbolos chineses com certos tipos de formas em resposta a certos tipos de formas que me foram fornecidas no terceiro lote. Sem que eu saiba, as pessoas que estão me fornecendo todos estes símbolos chamam o primeiro lote de 'script', chamam o segundo lote de 'história', e o chamam o terceiro lote de 'perguntas'. Além disso, eles chamam os símbolos que devolvi em resposta ao terceiro lote de 'respostas às perguntas', e o conjunto de regras em inglês que me forneceram, eles chamam de 'o programa'."

Searle segue com a reflexão, supondo que receba destas pessoas histórias em inglês, que em seguida receba perguntas sobre estas histórias e que ele responda às perguntas em inglês. Supondo que ele domine as instruções para a manipulação dos símbolos em chinês e que produza respostas corretas, para um observador externo, as respostas dadas por ele podem ser indistinguíveis de um falante chinês nativo. Porém, é notório que o processo cognitivo realizado para as respostas dadas em chinês é completamente diferente daquele realizado para as respostas em inglês. Apesar de responder às perguntas em chinês, Searle não entende o significado dos símbolos e por isso desconhece a história, diferentemente do caso da estória fornecida em inglês a qual foi interpretada por ele e respondida de acordo com o seu entendimento sobre a mesma. Ou seja, no caso dos símbolos em chinês, Searle se comporta como um computador, apenas realizando operações computacionais (manipulação dos símbolos), seguindo as regras sintáticas fornecidas em uma linguagem que ele conhece.

O segundo exemplo apresentado por Harnad possui duas versões. A primeira versão, considerada difícil por ele, questiona a possibilidade de se aprender chinês como segunda língua tendo apenas um dicionário de chinês/chinês como fonte de informação, visto que haveria uma viagem sem fim de um símbolo sem sentido a outro, já que as definições também estariam em chinês e seria necessário buscar o significado daqueles símbolos usados para explicar o primeiro. A segunda versão, impossível para Harnad, considera a tarefa de aprender chinês como a primeira língua, utilizando apenas um dicionário de chinês/chinês. Esta está mais próxima da realidade enfrentada por modelos da mente puramente simbólicos, pois não há como sair da associação de símbolos com outros símbolos e embasar tais símbolos com outra coisa sem que sejam outros símbolos sem sentido (HARNAD, 1990). Em resumo, a incapacidade dos computadores de relacionar os símbolos com os seus referentes no mundo real caracteriza o Problema do Embasamento Simbólico.

Ao definir o Problema do Embasamento Simbólico, Harnad (1990) também propõe um caminho para uma possível solução. Visando beneficiar-se do melhor dos dois mundos, Harnad propõe a utilização de um modelo híbrido de Inteligência Artificial (IA), combinando a IA Simbólica com a IA Conexionista. A primeira possui a capacidade de manipulação de símbolos e simulação de raciocínio, e a segunda está mais próxima daquilo que se conhece sobre o funcionamento do cérebro humano e possui a capacidade de aprendizado de padrões, podendo ser utilizada para a extração de características invariantes das suas projeções sensoriais. Através do relacionamento daquilo que é percebido através de sensores com seus referentes simbólicos, os símbolos estariam então embasados com seus referentes no mundo real.

Taddeo e Floridi (2005) definiram um critério, com base no problema apresentado por Harnad, para que uma proposta de solução do Problema do Embasamento Simbólico seja válida. Eles o chamaram de *Zero Semantical Commitment Condition* (Condição de Compromisso Semântico Zero). O critério estabelece o seguinte: 1) nenhuma forma de inatismo é permitida, ou seja, nenhum recurso semântico pode estar pré-instalado no agente artificial; 2) nenhuma forma de externalismo é permitida, ou seja, nenhum recurso semântico pode ser carregado de fora do agente artificial e 3) o agente artificial deve possuir capacidades e recursos próprios para ser capaz de embasar seus símbolos.

Taddeo e Floridi (2005) classificaram as soluções propostas para o Problema do Embasamento Simbólico em três abordagens: representacionalista, semi-representacionalista e não-representacionalista. A abordagem representacionalista, segundo os autores, procura resolver o problema fundamentando os símbolos de um Agente Artificial com base nas representações extraídas dos dados perceptivos do agente. Já a abordagem semi-representacionalista, ainda possui como base a abordagem representacionalista, porém baseia-se nos princípios da robótica baseada em comportamento para realizar o aprendizado e o embasamento dos símbolos. Por fim, a abordagem não-representacionalista defende que não é necessária uma representação simbólica, pois o comportamento inteligente pode ser resultado de interações entre um Agente Artificial corporificado e situado em seu ambiente, bastando acoplamentos sensório-motores.

O modelo híbrido proposto por Harnad, que se encaixa na abordagem representacionalista, não satisfaz a Condição de Compromisso Semântico Zero, pois para que a rede neural que realizará a análise e extração das características dos dados sensoriais precisará ser treinada primeiro e, consequentemente, seu embasamento é extrínseco.

2.2 REDES NEURAIS E DETECÇÃO DE OBJETOS

Os neurônios artificiais, que compõem as Redes Neurais, são um modelo matemático inspirado nos neurônios biológicos presentes no cérebro humano. Basicamente eles são ativados quando uma combinação linear de suas entradas excede algum limiar, sendo que a esta soma é aplicada uma função de ativação. As Redes Neurais Artificiais consistem na combinação de vários neurônios artificiais (NORVIG; RUSSEL, 2004). Segundo Luger (2009), um dos principais pontos fortes das redes neurais é que, por serem geralmente treinadas ou condicionadas, ao invés de programadas explicitamente, as redes neurais são capazes de

capturar invariâncias no mundo se possuírem uma arquitetura de rede e um algoritmo de aprendizagem apropriadamente projetados.

Com a evolução dos estudos sobre as redes neurais e com o aumento da disponibilidade de poder computacional, diversas arquiteturas e algoritmos de aprendizagem foram desenvolvidos e o campo de aplicação das mesmas foi extensamente ampliado. Uma importante aplicação das redes neurais, na área de visão computacional, é a detecção de objetos.

JIAO et. al. (2019), afirmam que:

> "A detecção de objetos é uma tecnologia computacional relacionada à visão computacional e ao processamento de imagens que trata da detecção de instâncias de objetos semânticos de uma determinada classe (como humanos, edifícios ou carros) em imagens e vídeos digitais. Domínios bem pesquisados de detecção de objetos incluem detecção de multicategorias, detecção de bordas, detecção de objetos salientes, detecção de pose, detecção de texto de cena, detecção de rosto e detecção de pedestres, etc. Como uma parte importante da compreensão de cena, a detecção de objeto tem sido amplamente utilizada em muitos campos da vida moderna, como campo de segurança, campo militar, campo de transporte, campo médico e campo de vida."

O reconhecimento de objetos, como também é conhecida a detecção de objetos, geralmente consiste em dois tipos de tarefas: o *reconhecimento de instância de objeto* onde o objetivo é a identificação de instâncias de objetos já vistas anteriormente, ou seja, pretende encontrar um determinado objeto em uma imagem, e o *reconhecimento de classe de objeto* que visa reconhecer instâncias (nunca vistas antes) de algumas categorias de objetos pré-definidas (ZHANG, 2013).

Segundo JIAO et.al (2019), os detetores de objetos podem ser divididos em duas categorias, sendo uma delas a dos detectores de dois estágios (tendo o Faster R-CNN como o mais representativo), que possuem uma alta acurácia na localização e reconhecimento dos objetos, e a segunda categoria que compreende os detectores de um único estágio (como por exemplo o YOLO), os quais possuem alta performance de inferência.

O uso de Redes Neurais Convolucionais (Convolutional Neural Networks CNN, em inglês) na tarefa de detecção de objetos proporcionou um grande avanço na área e diversos detectores foram propostos desde então. As redes neurais convolucionais iniciam a detecção a partir de pequenas regiões nas imagens, identificando features simples e realizando composições das mesmas para então chegar a uma classificação mais abstrata da classe que está

sendo detectada. Por exemplo, para detectar um quadrado, uma CNN poderia identificá-lo como uma composição de quatro retas em uma determinada disposição. Por tratarem de pequenas regiões da imagem e manterem os pesos para os vizinhos da próxima camada, as CNNs reduzem drasticamente a complexidade da rede, diminuindo o número de conexões necessárias entre os neurônios e, além disso, permitem detectar os objetos independentemente da sua localização na imagem (ALBAWI; MOHAMMED; AL-ZAWI, 2017).

Durante anos de pesquisas, os detectores de objetos foram aprimorados muitas vezes a partir de detectores propostos anteriormente. O detector de objetos Faster R-CNN, proposto por Ren et. al. (2017), introduziu as chamadas Region Proposal Networks (RPN), que são formadas por uma rede totalmente convolucional e que compartilham as camadas convolucionais com um detector de objetos Fast R-CNN. Estas redes possuem o papel de gerar as regiões nas quais o detector de objetos realizará a busca pelos objetos, atribuindo um score a cada região. Através do compartilhamento das camadas convolucionais, os autores conseguiram aumentar a performance final do processo de detecção (REN et. al., 2017).

2.3 AGENTES BDI

A definição de agentes inteligentes não é única, pois muitas delas baseiam-se no objetivo de sua aplicação. Segundo Wooldridge (2002), um agente "é um sistema de computador que está situado em algum ambiente, e que é capaz de ação autônoma nesse ambiente a fim de atender aos seus objetivos de projeto". Para Luger (2009), um sistema multi-agente

> "é um programa de computador com solucionadores de problemas situados em ambientes interativos, cada um capaz de ações flexíveis, autônomas, mas socialmente organizadas, que podem, mas não precisam ser, direcionadas a objetivos ou metas pré-determinadas. Assim, os quatro critérios para um sistema de agente inteligente incluem softwares solucionadores de problemas situados, autônomos, flexíveis e sociais."

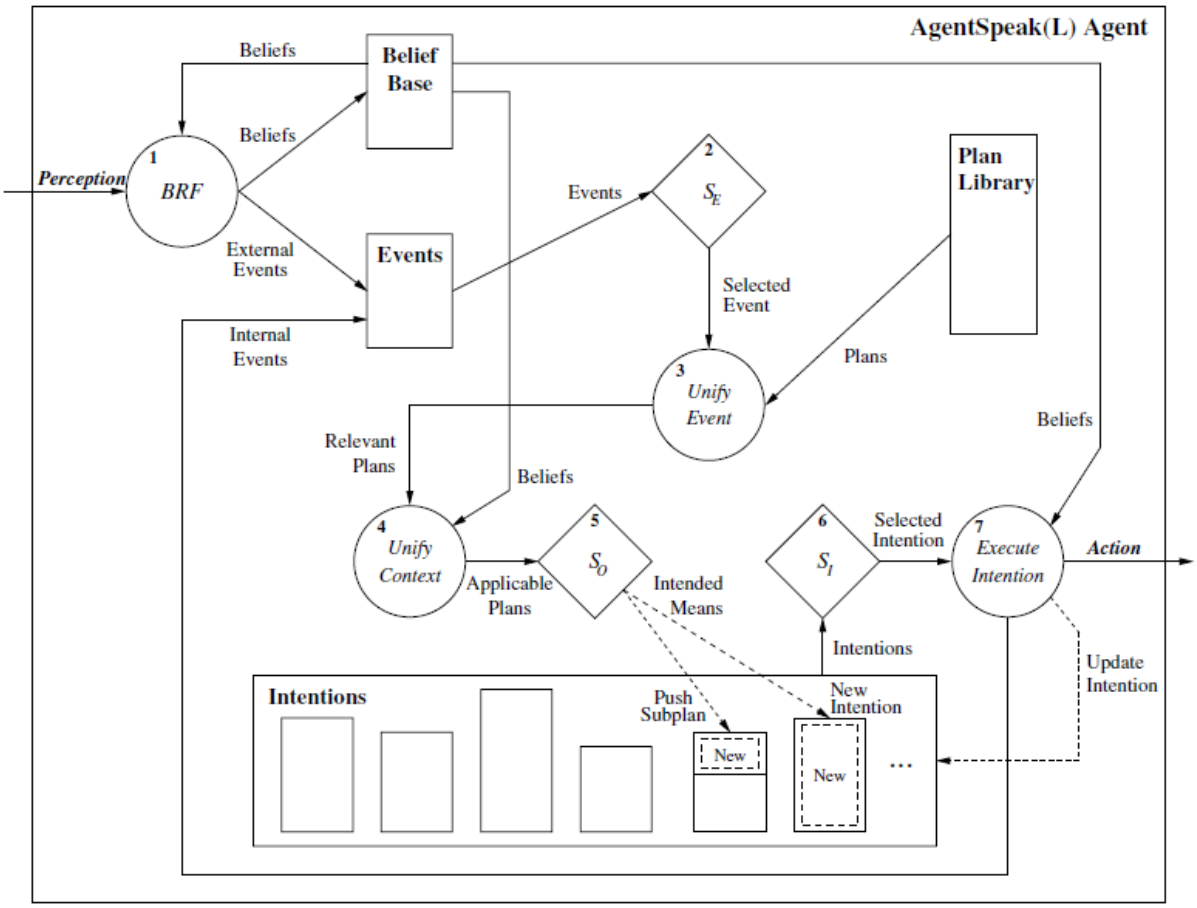
Segundo Wooldridge e Jennings (1995), apesar das diversas definições para agentes, algumas propriedades devem estar presentes nos mesmos, sendo elas: 1) autonomia: o agente possuir algum tipo de controle sobre suas ações e seu estado interno e deve operar sem intervenção direta de humanos ou outros agentes; 2) habilidade social: ele deve comunicar-se com outros agentes e, possivelmente, humanos; 3) reatividade: o agente deve perceber o ambiente em que se encontra e reagir a tempo às mudanças; e 4) proatividade: o agente não pode apenas reagir às mudanças, mas também deve tomar iniciativas para alcançar seus objetivos.

A arquitetura de agentes BDI (Belief-Desire-Intention) foi inspirada no trabalho de Bratman (1987 apud Silva, Meneguzzi, Logan, 2020), que afirmou que o raciocínio prático pode ser visto como o ato de pesar múltiplas considerações conflitantes a favor ou contra escolhas conflitantes, à luz do que o agente acredita, valoriza e se preocupa. O raciocínio pode ser dividido em duas etapas, sendo a primeira a deliberação, onde o agente decide qual estado deseja alcançar com base em seus desejos, e a segunda etapa, conhecida como raciocínio meios-fins, onde o agente decide como irá fazer para alcançar tal estado, ou seja, ele escolhe um plano de ação (SILVA; MENEGUZZI; LOGAN, 2020).

As crenças (*beliefs*) do agente são as informações sobre o ambiente que o agente possui e são atualizadas por meio de ações de detecção. É através delas que o agente se torna capaz de identificar a possibilidade ou não de realizar ações para alcançar seus objetivos e entender os efeitos causados pelas mesmas. Os desejos (*desires*) do agente são os objetivos a serem alcançados e estão associados a prioridades ou recompensas. Para alcançar seus objetivos, o agente precisa analisar o ambiente para identificar se possui condições de realizar determinadas ações em busca dos mesmos. Porém, por se tratar de questões não-determinísticas, é possível que imediatamente antes ou durante uma ação do agente ocorra algum evento que altere o ambiente, e que as condições impeçam a realização da ação. Por isso, é necessário que o agente possua uma forma de verificar tal situação sem que passe todo o tempo apenas verificando-as e nunca realizando suas ações. Por este motivo, as intenções (*intentions*) representam a escolha atual da ação a ser tomada pelo agente, que é baseada em seus desejos (RAO; GEORGEFF, 1995).

Bordini e Hübner (2006) desenvolveram uma plataforma de desenvolvimento de agentes BDI, chamada *Jason*, baseada na linguagem *AgentSpeak* em que forneceram diversas extensões que permitem o desenvolvimento prático de sistemas multi-agentes, visto que a linguagem *AgentSpeak* é apenas abstrata, realizando também algumas adaptações na linguagem. A Figura 1 ilustra um ciclo de interpretação de um programa *AgentSpeak* no Jason.

Figura 1 - Um ciclo de interpretação de um programa AgentSpeak



Fonte: Bordini e Hübner (2006).

Jason é open source, foi desenvolvido em Java e possui diversas features. Dentre elas: comunicação interagente baseada em atos de fala, anotações em rótulos de planos, possibilidade de executar um sistema multi-agente distribuído em rede e funções de seleção totalmente personalizadas.

2.4 DISCUSSÃO

Como visto anteriormente, uma das abordagens para lidar com o problema do embasamento simbólico compreende o desenvolvimento de sistemas híbridos, que combinam sistemas de IA Conexionista e de IA Simbólica, visando o embasamento dos símbolos manipulados pela IA Simbólica através da conexão dos mesmos com percepções processadas pela IA Conexionista. Este trabalho irá utilizar a detecção de objetos em conjunto com agentes BDI para propor um modelo de embasamento simbólico. A detecção de objetos será realizada utilizando um detector de objetos Faster R-CNN com o extrator de features Inception v2 e o agente BDI será desenvolvido utilizando a plataforma Jason.

3 TRABALHOS RELACIONADOS

Este capítulo apresentará os trabalhos relacionados que possuem características relevantes para o desenvolvimento deste estudo. Após a apresentação dos trabalhos serão discutidos os principais aspectos analisados nos mesmos e uma tabela comparativa ilustra as características de cada estudo. Por fim são indicados os estudos considerados mais relevantes para o desenvolvimento deste trabalho, bem como são apresentados os seus diferenciais em relação aos trabalhos analisados.

3.1 EFFECTIVE INTEGRATION OF SYMBOLIC AND CONNECTIONIST APPROACHES THROUGH A HYBRID REPRESENTATION

Existem diversas abordagens que tratam da integração neural-simbólico e Hilario (1995 apud Moreno et. al., 2019) realizou uma separação em dois tipos de estratégias: unificada (que busca combinar redes neurais com recursos simbólicos em uma estrutura única) e híbrida (que realiza a combinação sinérgica dos modelos). Ainda de acordo com Hilario (1995 apud Moreno et. al., 2019), a abordagem híbrida pode ser dividida em translacional, onde as redes neurais atuam como processadores e há traduções entre as estruturas simbólicas e as redes neurais, e funcional que compreende as estruturas simbólicas, as redes neurais e seus processadores correspondentes e pode ser organizada de quatro maneiras diferentes: processamento em cadeia, subprocessamento, metaprocessamento e coprocessamento. Garcez et. al. (2002, apud Moreno et. al., 2019), também discutiram sobre estas abordagens, porém ainda existem algumas desvantagens, principalmente relacionadas à falta de dinamicidade das soluções. Isto porque estas abordagens não são capazes de alternar entre modelos ou utilizar apenas uma parte dele. Por este motivo, os autores defendem que um mecanismo de execução capaz de entender e executar descrições de integração deve atuar como um middleware para promover a dinamicidade, manipulando ambos os modelos (simbólico e conexionista) quando apropriado.

Os autores defendem a utilização de um modelo híbrido de representação que promova uma integração efetiva entre os subsistemas simbólico e conexionista e seja capaz de manipulá-los de maneiras diferentes em um sentido mais amplo, fornecendo uma abstração que permite tal manipulação definindo a relação entre as estruturas simbólicas, as redes neurais e seus processadores correspondentes. Esta representação híbrida foi apresentada por Moreno et. al.

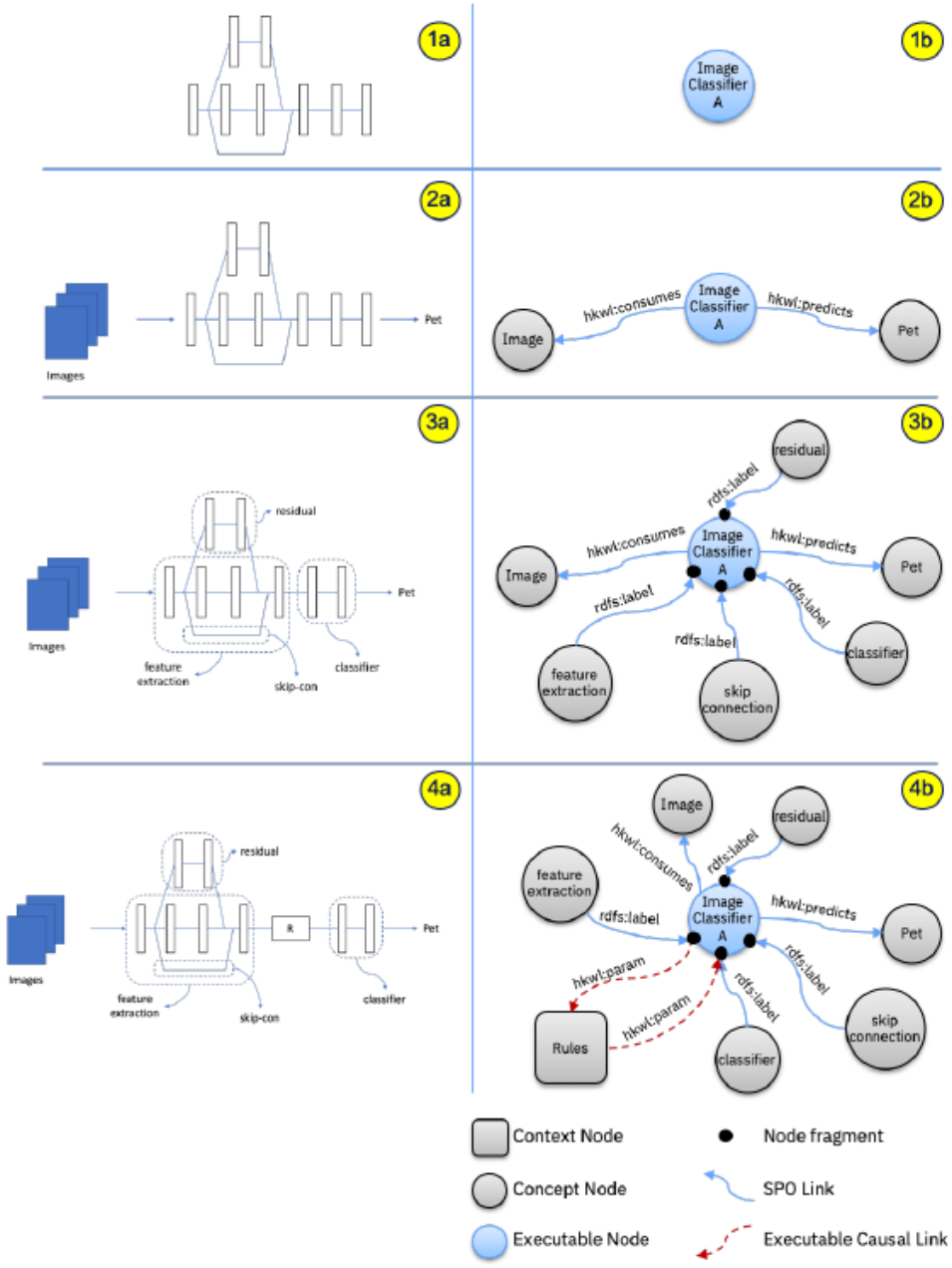
(2017 apud Moreno et. al., 2019) porém com foco em como os fundamentos da hipermídia poderiam aumentar a expressividade das representações do conhecimento.

Visando resolver estas limitações, os autores propõem uma representação híbrida capaz de combinar características específicas dos modelos, baseada nos fundamentos de nós e ligações. O foco do estudo foi em ligações SPO (sujeito-predicado-objeto) que são capazes de especificar fatos como em RDF (Resource Description Framework) e ligações causais onde há condições e ações envolvidas na relação. As ligações causais definem a execução da representação e as ligações SPO permitem o raciocínio sobre as ligações simbólicas e subsimbólicas.

Basicamente, o trabalho realizado consiste em descrever os componentes de uma rede neural através de uma representação simbólica, utilizando os conceitos discutidos anteriormente. A Figura 2 mostra detalhes do processo de modelagem da integração neural-simbólica, apresentando na coluna da esquerda uma rede neural de classificação de imagens e na coluna da direita a representação utilizando nós e ligações.

A topologia da rede neural foi dividida em partes menores e tais partes foram transformadas em nós. Para vincular um conjunto de regras a um modelo neural, utilizam-se ligações executáveis causais, onde por exemplo, pode-se especificar uma condição de espera de uma outra ligação para realizar-se uma ação e também para enviar um outro sinal a uma outra ligação. Isto acontece no exemplo, na computação das regras que disparam o outro link executável causal que envia os resultados desta computação para o classificador. Estes tipos de ligações causais também podem especificar um conjunto de condições e um conjunto de ações para serem disparadas quando as condições são satisfeitas.

Figura 2 - Modelando a integração neural-simbólica



Fonte: Moreno et. al. (2019).

A representação híbrida apresentada pelos autores se destaca pela capacidade de representar modelos de inteligência artificial em geral, descrevendo além dos modelos (conexionista e simbólico), as integrações entre os modelos e os seus respectivos processadores. Além disso, a representação é capaz de fornecer meios de rastreabilidade das alterações do modelo, permitindo o controle do ciclo de vida do mesmo.

Figura 3 - Exemplos de consultas possíveis na modelagem do caso de uso apresentado

1. SELECT (context) node WHERE node IS dataset
2. SELECT node WHERE node IS dataset AND dataset CONTAINS cat images
3. SELECT DISTINCT dataset WHERE image FROM dataset AND image HAS cat
4. SELECT code WHERE architecture HAS code AND code HAS convolutional_layer
5. SELECT DISTINCT trained_model WHERE training_orchestrator TRAINS trained_model AND training_orchestrator CONSUMES dataset AND image HAS pet AND image FROM dataset AND training_orchestrator PRODUCES testing_metrics AND test FROM testing_metrics AND test.accuracy >= 0.9

Fonte: Moreno et. al. (2019).

Para demonstrar o funcionamento da representação proposta, os autores modelaram um caso de uso. A modelagem foi feita para uma rede neural de classificação de imagens de cães e gatos. Na representação, foram incluídos nós representando os datasets, métricas de teste, o classificador, o orquestrador de treinamento, etc. Através desta modelagem, é possível executar consultas no modelo, como demonstrado na Figura 3.

Devido aos resultados apresentados durante o estudo, os autores pretendem seguir no desenvolvimento da representação, trabalhando especialmente na evolução do mecanismo para permitir a criação automática de modelos a partir de fragmentos.

3.2 SYMBOL GROUNDING IN MULTIMODAL SEQUENCES USING RECURRENT NEURAL NETWORKS

Estudos apontam que as crianças, durante o seu desenvolvimento, baseiam os conceitos semânticos em suas entradas sensoriais. Além disso, verificou-se que há uma correspondência entre a aquisição de vocabulário (áudio) e o reconhecimento de objetos (visual). Com base nesses estudos e com o propósito de endereçar o problema do embasamento simbólico, os autores propõem a utilização de redes neurais recorrentes, mais especificamente as redes LSTM (Long Short-Term Memory). Eles utilizaram estas redes para promover a associação entre entradas multimodais (neste caso entradas visuais e de áudio) que contém a mesma sequência

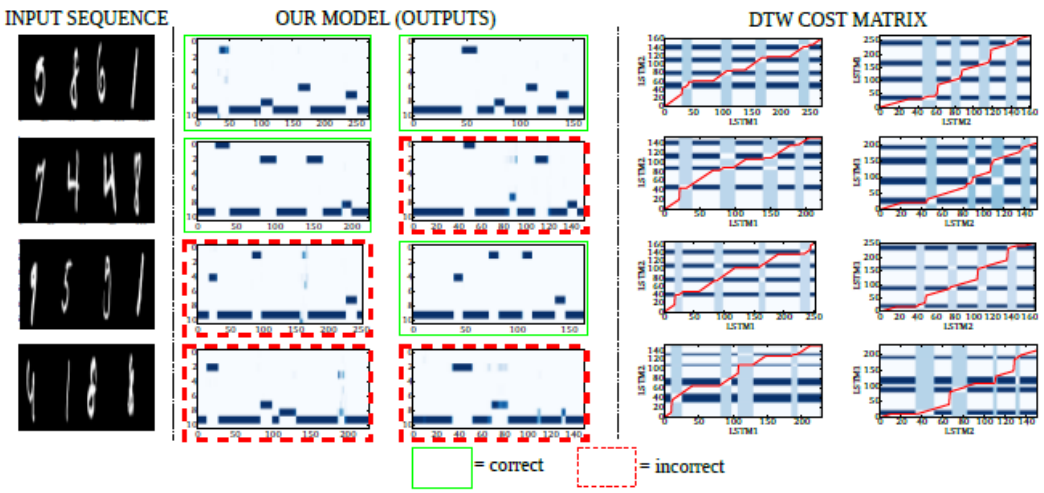
semântica, porém sem qualquer segmentação prévia (diferente de outros trabalhos que utilizam entradas segmentadas).

As redes neurais LSTM têm como saída a probabilidade da classe a cada etapa de tempo. Através da combinação de outra técnica, denominada Connectionist Temporal Classification (CTC), que adiciona uma classe em branco à sequência alvo, durante o processo de aprendizagem de onde inserir corretamente as classes em branco é possível, consequentemente, ao aprender a segmentação, além da classificação dada pela LSTM.

Com base nesses conceitos, os autores propuseram um cenário simplificado, onde são fornecidas duas entradas, uma visual e outra auditiva. Além disso, definem duas suposições básicas para a realização do estudo: 1) existe uma representação simbólica e 2) duas entradas diferentes representam uma mesma estrutura simbólica (relação entre um conceito semântico, uma entrada visual/auditiva e um conjunto de características simbólicas).

Para realizar a associação entre conceitos semânticos da entrada visual e os da entrada auditiva, os autores utilizaram duas redes LSTM (uma para cada tipo de entrada). Cada rede LSTM recebe uma das modalidades e as suas saídas, e os conceitos semânticos são processados por uma restrição estatística, que seleciona a relação mais provável entre os conceitos semânticos e as características simbólicas. Na etapa seguinte é realizada a sincronização das duas redes, utilizando Dynamic Time Warping, que é uma técnica que permite o mapeamento não linear entre duas séries temporais. A Figura 4 mostra exemplos de matrizes de custo do DTW.

Figura 4 - Exemplos de matrizes de custo do DTW



Fonte: Raue (2015).

Notas: A linha vermelha mostra o caminho passando por nove regiões. Essas regiões representam a classe em branco e os conceitos semânticos

Os autores realizaram os testes do framework utilizando datasets para três cenários diferentes: reconhecimento de dígitos escritos, reconhecimento de letras impressas e reconhecimento de palavras. Cada dataset possuía um componente visual e outro de áudio.

Após realizarem os testes, os autores concluíram que é possível realizar a aprendizagem de representações simbólicas de entradas sensoriais não segmentadas um mínimo de suposições. Apesar de ser limitado a uma única dimensão, existem muitas aplicações neste contexto, como por exemplo a combinação de rastreamento de olhos com áudio. Como desenvolvimento futuro do trabalho, os autores sinalizam a evolução do estudo para cenários mais realísticos, como por exemplo tratar conceitos faltantes em um dos componentes ou em ambos.

3.3 PROBLEMA DE EMBASAMENTO DE SÍMBOLOS EM UM SISTEMA MULTI-CONTEXTO

Segundo o autor, a Inteligência Artificial (IA), apesar de ser um tema tratado pela ciência da computação, está fortemente conectado com outras áreas como as ciências cognitivas, a filosofia e a psicologia, por exemplo. Isto porque, para o desenvolvimento de sistemas inteligentes, é necessário compreender como funciona a mente humana. Em 1976, Newell e Simon formularam a hipótese de que um sistema de símbolos físicos têm os meios necessários e suficientes para uma ação inteligente geral. Tal hipótese foi contestada por Searle, em 1980, através do argumento do quarto chinês. Searle demonstra com o experimento que a simples manipulação de símbolos não corresponde ao pensamento, pois os seres humanos associam significados aos símbolos, o que não acontece com o computador.

O autor explica que, em 1990, Harnad formulou o Problema do Embasamento Simbólico, que questiona como os símbolos adquirem significado sem que este não se baseie em outro símbolo também sem significado. Tal problema motivou diversos estudos e segue em aberto, sendo analisado por diversas perspectivas, por diversos autores. O trabalho de Eichstaedt (2019) pretende demonstrar que o problema do embasamento simbólico possui consequências práticas no desenvolvimento de inteligências artificiais, bem como propor um modelo que lide com tal problema integrado a um framework de sistemas multi-contexto.

Segundo o autor, dentro do campo da IA Conexionista existem duas grandes categorias de aprendizagem: a supervisionada e a não supervisionada. Na primeira, os algoritmos são alimentados com saídas esperadas e aprendem a reduzir o erro com base nestes valores

fornecidos. Já no aprendizado não supervisionado, os algoritmos realizam agrupamentos e não há interferência externa.

Os agentes fazem parte da IA Simbólica e são definidos por Norvig e Russell (2013) como entidades autônomas que são capazes de agir sobre o ambiente em que se encontram, com base nas informações que recebem através de sensores. Os agentes podem ser categorizados com base na sua arquitetura, podendo ser reativos (não possuem representação do ambiente e não realizam raciocínios lógicos complexos, apenas selecionam ações com base no seu estado atual), deliberativos (possuem um modelo simbólico explícito sobre o mundo real e realizam raciocínios lógicos mais complexos, com base em suas intenções e crenças, podendo criar e escolher ações e planos), e híbridos que combinam características das duas categorias anteriores (EICHSTAEDT, 2019).

Segundo o autor, os agentes BDI são agentes deliberativos, que tiveram como base para a sua definição o raciocínio prático e consideram três estados mentais para a descrição do processamento interno: crenças, desejos e intenções. As crenças representam a visão do agente sobre o ambiente, outros agentes, interação com outros agentes e até mesmo sobre suas próprias crenças. Os desejos são os estados desejados pelo agente, que motivam as suas ações. Os desejos são selecionados com base na sua viabilidade e viram intenções. As intenções representam o plano escolhido pelo agente para executar suas ações.

Outro conceito apresentado no trabalho é o de sistemas multi-contexto. Estes sistemas consistem em um conjunto de contextos, compostos por uma lógica e um conjunto de fórmulas escritas nessa lógica, além de um conjunto de regras de ponte para transferir informações entre os contextos. Mais especificamente, um sistema multi-contexto possui unidades, linguagens declarativas, teorias e regras de ponte. Esta composição permite que o sistema multi-contexto possa ser entendido como um sistema que possui módulos.

Retornando ao problema do embasamento simbólico, o qual está relacionado a como os símbolos adquirem significado, o trabalho explica que pode-se classificar as soluções em três abordagens: representacionalistas, semi-representacionalistas e não-representacionalistas. Todas as abordagens buscam embasar os símbolos utilizando as capacidades sensório-motoras do agente, porém diferem quanto aos métodos utilizados para elaborar os dados obtidos por tais experiências, ou seja, no processo de atribuição do significado aos símbolos.

As abordagens representacionalistas, segundo o autor, procuram resolver o problema através da associação dos símbolos aos seus referentes dados perceptivos dos agentes. No modelo híbrido proposto por Harnad, o embasamento é realizado através da associação de uma IA conexionista e uma IA simbólica, buscando aproveitar o melhor dos dois mundos. Utilizando

uma rede neural para a obtenção de dados do ambiente e a IA simbólica para a manipulação de símbolos. Para Harnad, os símbolos devem ser embasados em três etapas: iconização, discriminação e identificação.

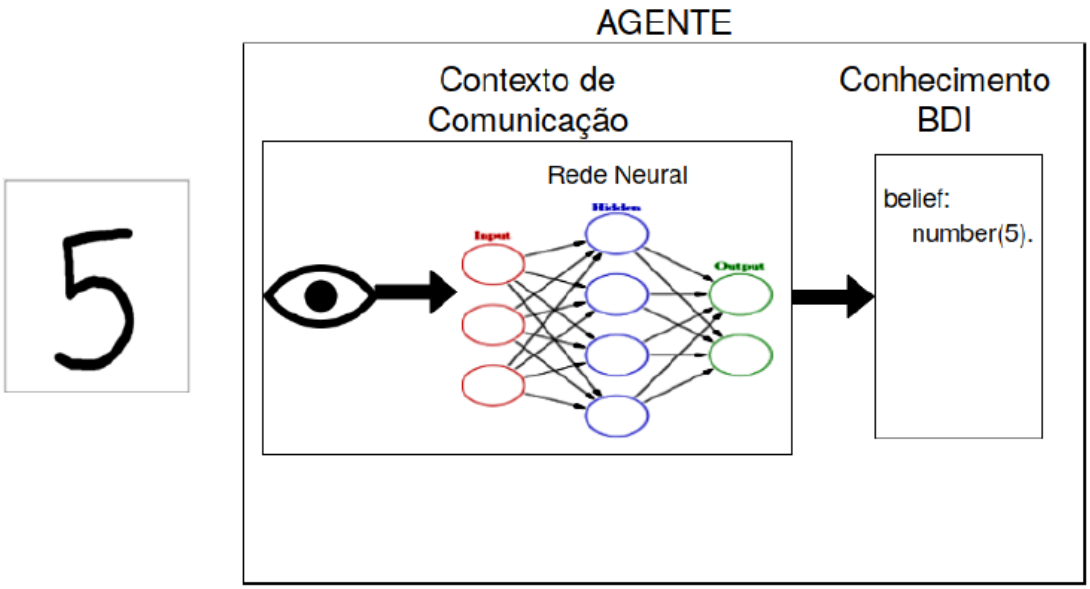
De acordo com o autor, as abordagens semi-representacionalistas mantêm a natureza representacionalista, porém se diferenciam por utilizar princípios de robôs baseados em comportamento como base para as representações do agente.

Por fim, segundo Eichstaedt (2019), as abordagens não-representacionalistas defendem que as representações simbólicas não são necessárias, pois segundo elas o comportamento inteligente se dá pela interação entre agentes incorporados e situados em seu ambiente. Como não existem símbolos, não há o que ser embasado e, consequentemente, o problema do embasamento simbólico deixa de existir.

A proposta do autor utiliza a linguagem Sigon, que foi construída com o propósito de operacionalizar agentes como sistemas multi-contexto. Além disso, o trabalho utilizou o framework desenvolvido por Arnhold (2018 apud EICHSTAEDT, 2019) que permite o desenvolvimento de agentes BDI como sistemas multi-contexto, utilizando a linguagem Sigon.

Com o intuito de resolver a dificuldade do agente computacional em associar um estímulo visual ao seu correspondente simbólico interno, o autor propõe um modelo híbrido e uma função que mapeia a entrada visual (imagem) com o representante simbólico interno (token). Para realizar tal mapeamento, o autor propõe a utilização de uma regra de ponte que obtém a informação percebida pelo sensor no contexto de comunicação e transfere a informação para o contexto de crenças do agente.

Figura 5 - Arquitetura proposta por Vailatti para o sistema multi-contexto

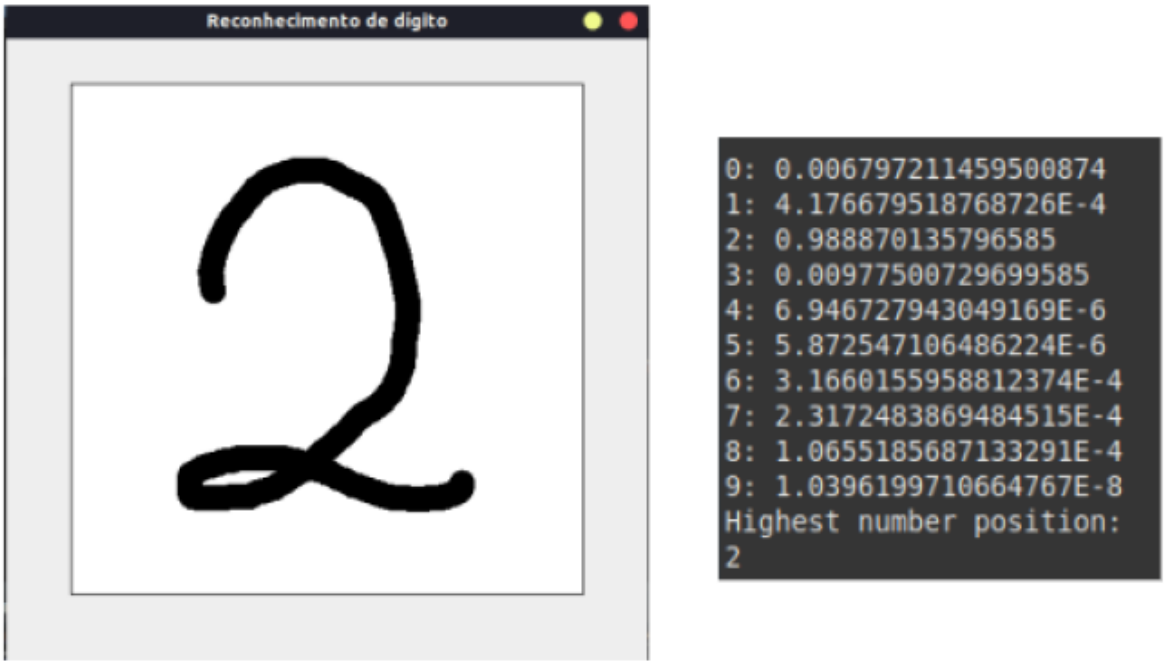


Fonte: Eichstaedt (2009)

Como a rede neural é treinada com um conjunto de dados pré-rotulado, a proposta viola a condição do problema do embasamento simbólico de que o agente precisa ser autônomo, pois o conhecimento prévio de qual deveria ser a classificação das imagens utilizadas no treinamento da rede neural veio de um humano. Porém, o trabalho segue relevante pois simula a aprendizagem das categorias no caso das crianças, quando adquirem a linguagem, visto que elas recebem informações dos pais e de outras pessoas com as quais se relacionam.

Uma rede neural foi treinada utilizando o dataset MNIST, que contém diversas imagens de números manuscritos, de 28 x 28 pixels, sendo 60 mil destinadas para o treinamento e 10 mil para o teste. As imagens de dígitos foram utilizadas para simular as percepções do agente quanto ao ambiente. Foi utilizada uma rede neural simples, de 3 camadas. A camada de entrada possui 784 neurônios (28 x 28, ou seja, um neurônio por pixel). Para a camada oculta foram testados diversos tamanhos e foi escolhido o número de 30 neurônios que permitiu um acerto de 93% das classificações. A camada de saída contém 10 neurônios, pois cada neurônio denota o dígito identificado pela rede.

Figura 6 - Exemplo de classificação do dígito



Fonte: Eichstaedt (2009)

Para receber a entrada sensorial do agente, foi utilizado um canvas no qual o usuário desenha um número que será identificado pela rede neural. Esta interface foi desenvolvida em Java. No momento em que o usuário termina de desenhar o número na tela, o sistema normaliza a imagem para o padrão das imagens do MNIST, ou seja, o tamanho é ajustado para 28 x 28 e as cores são invertidas (para fundo preto e número branco). Após a normalização, a imagem é passada para a rede neural que realiza a classificação do dígito. Este dígito é passado para o agente na forma de uma crença. E com isso, resumidamente, o agente atinge o desejo de ter o número.

Através desta implementação o autor obteve uma função que mapeia uma representação visual em um símbolo, embasando o mesmo. Apesar das limitações mencionadas anteriormente, foi possível demonstrar a utilização da abordagem proposta por Harnad em um ambiente de sistemas multi-contexto, na forma de Agentes BDI, utilizando uma rede neural como a parte conexionista da solução.

3.4 OUTROS TRABALHOS

Spangenberg e Henrich (2016) realizam o embasamento simbólico utilizando um dicionário que conecta os símbolos, as informações subsimbólicas e os sensores/componentes

físicos do robô, que são utilizados para extrair informações ou executar ações. Eles usam o que chama de "verbalized physical effects" para extrair e executar um conjunto de ações a partir de um comando falado pelo usuário. Os autores realizaram um teste com um robô que utilizava um Microsoft Kinect para o reconhecimento de objetos.

Ma (2020) trata da tarefa de Navegação com visão e linguagem (Vision-and-Language Navigation) com uma estratégia de embasar o entendimento do ambiente por parte do agente através do embasamento de objetos que compõem a cena e suas correlações. Além disso, o autor realiza o que chama de "coembasamento", em que ambas entradas, textual e visual, são embasadas ao mesmo tempo através do uso de uma rede neural LSTM e depois são utilizadas para a seleção da ação a ser realizada pelo agente.

3.5 DISCUSSÃO

Através da análise dos trabalhos relacionados apresentados neste capítulo, extraiu-se características relevantes e que estão relacionadas com o objetivo deste trabalho, sendo elas:

- Técnica conexionista: é importante entender qual tipo de técnica conexionista foi escolhida para que então se entenda como tal técnica foi utilizada para embasar os símbolos.
- Aplicação: informa a tarefa em que o trabalho foi aplicado. Na maioria dos trabalhos ela representa a interação do agente inteligente com o ambiente (salvo os trabalhos sem aplicação prática).
- Dataset: na eventual disponibilidade do dataset do trabalho, é interessante entender como está distribuído, que tipos de imagens são analisadas e entender a complexidade tratada pela técnica conexionista.
- Representação simbólica: da mesma forma que é importante conhecer a técnica conexionista para então compreender sua ligação com os símbolos, é importante entender como a representação simbólica é feita, pois ela faz parte desta ligação. O entendimento da representação simbólica também auxilia na compreensão da utilidade do embasamento para a aplicação.
- Abordagem para fundamentação: esta característica revela como o embasamento propriamente dito foi realizado, ou seja, como o que é percebido do mundo externo é conectado com os símbolos.

- Raciocínio/inferências: aqui é apresentado o que é feito com o embasamento, como ele ajuda a aplicação em seu objetivo.

Dentre os trabalhos aqui apresentados, considerou-se como os mais relevantes para este trabalho, o de Raue *et. al.* (2015), por tratar duas entradas sensoriais paralelamente e realizando também um "coembasamento" das percepções visuais e auditivas, e o de Eichstaedt (2019) o qual este trabalho visa dar continuidade, contribuindo através da proposta de um modelo para conectar as crenças do agente BDI com suas respectivas percepções do mundo real, realizada através da detecção de objetos pela rede neural. A Tabela 1 mostra o comparativo dos trabalhos analisados.

Este trabalho visa propor um modelo para embasamento simbólico que conecte as crenças do agente BDI com os seus referentes no mundo real. Através de uma arquitetura bem definida, este trabalho visa proporcionar uma biblioteca de fácil utilização e extensão. Esta biblioteca poderá ser utilizada em projetos de diferentes contextos, sejam eles novos ou preexistentes, diferentemente dos trabalhos analisados, que possuem contextos específicos. Além disso, ao possuir suas crenças embasadas, o agente BDI pode realizar a tomada de decisões com base em informações do mundo real, aumentando o seu conhecimento em relação ao ambiente em que está situado.

Tabela 1 - Comparativo das soluções para o embasamento simbólico

Trabalho	Técnica Conexionista	Aplicação	Dataset	Avaliação	Representação Simbólica	Abordagem para Fundamentação	Raciocínio / Inferências
Effective Integration of Symbolic and Connectionist Approaches through a Hybrid Representation	Rede Neural	Classificação de imagens	não informado		RDF	A representação simbólica é a descrição da rede neural, seu funcionamento e seu estado	Extração de conhecimento da rede neural através de consultas na representação simbólica
Symbol Grounding in Multimodal Sequences using Recurrent Neural Networks	Rede Neural Recorrente do tipo LSTM	Classificação e segmentação de imagens e áudios	Derivado do MNIST + Áudios gerado pelo Festival Toolkit + Áudios do GRID	Label Error Rate (LER)	A saída da RN é considerada a representação simbólica e o conceito semântico é conhecido previamente (por exemplo, o conceito "gato")	É realizado através da combinação dos conceitos semânticos (conhecidos previamente) e as saídas das redes neurais, com o uso de uma função estatística	Não explicitado no estudo, porém permite associar as duas representações (visual e auditiva), podendo "descobrir" como se fala uma palavra por exemplo tendo uma entrada visual da mesma
Problema de embasamento de símbolos em um sistema multi-contexto	MLP	Classificação de imagens	MNIST		Crenças do agente	Representacionalista, conectando os símbolos do agente BDI à classificação feita pela RN	Agente BDI
Symbol grounding for symbolic robot commands based on physical properties	Os autores utilizam o Microsoft Kinect para o reconhecimento de objetos	Solicitar tarefas a um robô por voz	não informado	Experimento com um robô	Dicionário físico	Semi-representacionalista, conectando os dados simbólicos com experiências físicas do robô	Utilizado para entender os comandos e relacioná-los com os objetos e com a tarefa a ser executada
Toward Grounded Spatio-temporal Reasoning	Rede Neural Recorrente do tipo LSTM	Navegação com visão e linguagem	R2R Dataset	Navigation Error; Success Rate; Oracle Success Rate e Success rate weighted by (normalized inverse) Path Length	O texto da instrução para o agente	É realizado através da combinação do texto da instrução com o reconhecimento do vídeo	São feitas a partir do progresso atual em relação à instrução sendo realizada, sendo que o progresso depende do embasamento
Este trabalho	Faster R-CNN with Inception ResNet	Embasamento de crenças de um agente BDI	Rede pré-treinada com Open Images Dataset	nº de linhas de cód. alter.; nº de planos do agente alter.; nº de arq. alter.	Crenças do agente	Representacionalista	Agente BDI

Fonte: Elaborado pelo autor.

4 DESENVOLVIMENTO

Neste capítulo serão apresentadas as etapas do desenvolvimento deste trabalho. A seção 4.1 apresenta as análises e os testes preliminares realizados. Foram desenvolvidos um agente experimental para testes e também um módulo de detecção de objetos, que são especificados nas subseções 4.1.1 e 4.1.2. A seção 4.1.3 aborda o modelo inicial para embasamento simbólico. Em seguida foram implementados os modelos desenvolvidos neste trabalho, os quais são apresentados na seção 4.2. Após a implementação dos modelos, foi realizada a definição e execução dos experimentos para analisar e validar os modelos propostos. A seção 4.3 descreve os experimentos realizados e seus resultados, bem como discute sobre os mesmos.

4.1 ANÁLISES E TESTES PRELIMINARES

Os primeiros passos do desenvolvimento do trabalho se deram através da pesquisa de redes neurais pré-treinadas para detecção de objetos, visando a sua utilização em testes dos modelos. Como o objetivo do trabalho não está relacionado com a tarefa de detecção, mas sim com a utilização do resultado desta para o embasamento de símbolos, optou-se por esta estratégia de implementação, focada na integração das IAs conexionista e simbólica.

Através da utilização de um código de exemplo, disponibilizado no repositório do GitHub do projeto Tensor Flow, foram testadas algumas redes neurais de detecção de objetos pré-treinadas, disponíveis também no repositório do projeto, sendo alguns exemplos: ssd_mobiledet_cpu_coco, ssd_resnet_101_fpn_oidv4 e faster_rcnn_inception_resnet_v2_atrous_oidv4. Por se tratarem de redes pré-treinadas, os objetos a serem detectados dependem fortemente do dataset utilizado no treino, e isso acarreta em uma restrição do contexto em que o agente BDI poderá ser desenvolvido. Devido à intenção de se utilizar uma rede Faster R-CNN, bem como de utilizar imagens de objetos comuns, optou-se pelo uso da rede faster_rcnn_inception_resnet_v2_atrous_oidv4, que foi treinada com o dataset Open Images, que possui uma grande variedade de objetos.

A partir da escolha da rede a ser utilizada, foi desenvolvido o módulo de detecção de objetos a ser utilizado no trabalho com base no código de exemplo mencionado anteriormente. Inicialmente, o código fonte foi alterado para realizar a leitura de imagens em um diretório e realizar a detecção dos objetos nestas imagens. Além disso, junto com a imagem de resultado da detecção de objetos (com as caixas de detecção desenhadas, junto com os nomes de objetos

detectados) , foi programado para que fosse salvo também em disco um arquivo JSON com os detalhes do resultado da detecção dos objetos. O arquivo foi gravado para ser a fonte de informações para o agente BDI. Com esta parte desenvolvida, a rede foi testada com várias imagens.

4.1.1 Desenvolvimento de um agente BDI para testes iniciais

Durante a implementação inicial do agente BDI, procurou-se realizar testes de possíveis abordagens para a elaboração do modelo para embasamento simbólico. Com isso, inicialmente foi desenvolvido um agente BDI com o objetivo de cortar objetos. O agente então possui planos para realizar esta tarefa e também para tentar descobrir se é possível realizar o corte de determinado objeto utilizando outro objeto disponível. Para isso, o agente executa ações para solicitar ao ambiente mais informações sobre os objetos detectados, como, por exemplo, saber se determinado objeto é cortável ou cortante, e também se determinado objeto cortável pode ser cortado por um objeto cortante disponível. O código fonte deste agente está disponível no Apêndice A.

Para que o agente recebesse a informação dos objetos disponíveis, o ambiente Jason realizava periodicamente a leitura do arquivo JSON gerado pela detecção de objetos, assumindo que a mesma estivesse sempre atualizada (nesta fase inicial, a detecção de objetos era executada manualmente, através de uma linha de comando).

As informações extras sobre os objetos, solicitadas pelo agente, eram respondidas pelo ambiente e foram programadas para tal, sendo respostas totalmente limitadas e pré-definidas pelo autor, visto que o foco do teste estava no desenvolvimento do modelo.

Após a finalização deste teste, foi desenvolvido um outro agente que seguiu a mesma proposta do anterior, porém possuía outro objetivo e também considerava o resultado da sua ação, tentando perceber no ambiente se sua ação foi efetiva. Este segundo agente BDI foi modelado para simular um robô que serve café e inicia com duas crenças básicas: a) é possível servir com uma chaleira; b) é possível servir em uma xícara. Assim que é iniciado, o agente realiza a ação externa de perceber o ambiente, que sinaliza para o ambiente Jason a necessidade de ler o arquivo JSON. O ambiente Jason por sua vez, realiza a leitura deste arquivo JSON gerado pelo módulo de detecção de objetos e atualiza as crenças do agente, adicionando novos objetos detectados e removendo objetos que deixaram de ser detectados.

Figura 7 - Imagem do resultado da detecção de objetos em uma imagem simulada

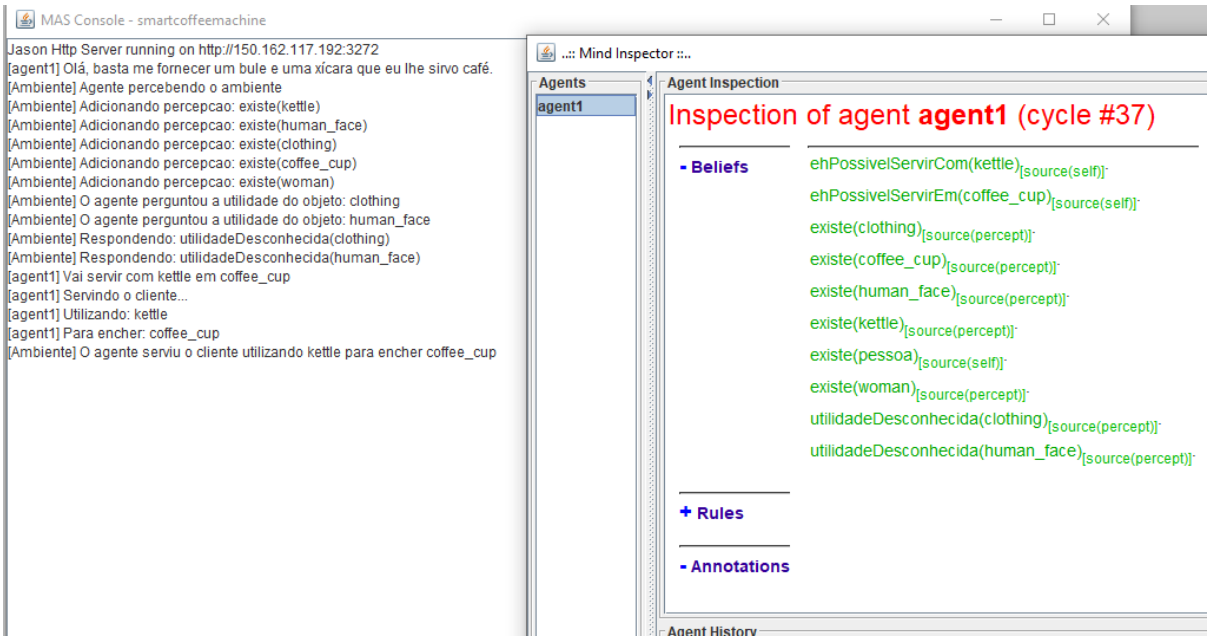
Fonte: Elaborado pelo autor.

Figura 8. Trecho do JSON gerado pelo módulo de detecção de objetos


```
[
    {
        "score": 0.9849456548690796,
        "class": {
            "id": 259,
            "name": "Kettle"
        },
        "box": [
            0.09529615193605423,
            0.5756107568740845,
            0.7043643593788147,
            0.9246169924736023
        ]
    },
```


Fonte: Elaborado pelo autor.

Figura 9 - Teste de execução do agente BDI



Fonte: Elaborado pelo autor.

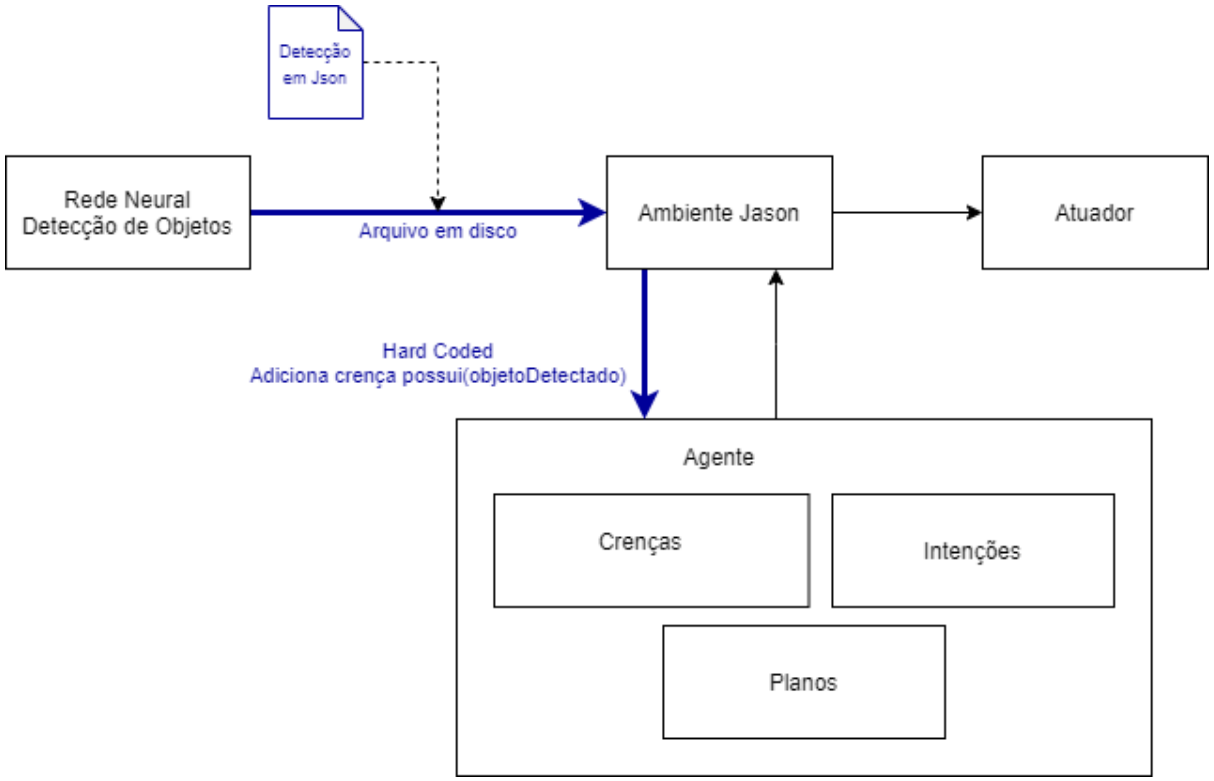
Ao perceber que há uma pessoa no ambiente, o agente espera perceber que possui os objetos necessários para realizar sua função. Ao perceber um objeto do qual o agente não possui conhecimento se é possível de ser utilizado para servir o café, o agente pergunta ao ambiente qual a utilidade daquele objeto (se é possível servir com ou nele).

Quando o agente identifica, através de suas crenças, que possui um objeto com o qual pode servir e outro objeto no o qual pode servir o café, ele executa a ação de servir e então aguarda alguns segundos, verifica novamente o ambiente (comunicando-se com o módulo de detecção de objetos) e analisa se a sua ação foi ou não bem sucedida.

4.1.2 Modelo para embasamento simbólico

A partir dos testes iniciais realizados tanto com a parte conexionista, quanto com a parte simbólica a ser utilizada no modelo, procurou-se esquematizar possibilidades de modelos a serem desenvolvidos, considerando diversas possibilidades. O esquema inicial elaborado está representado na Figura 10 e possui basicamente a transcrição de forma gráfica daquilo que foi implementado inicialmente.

Figura 10 - Opção para o modelo para embasamento simbólico



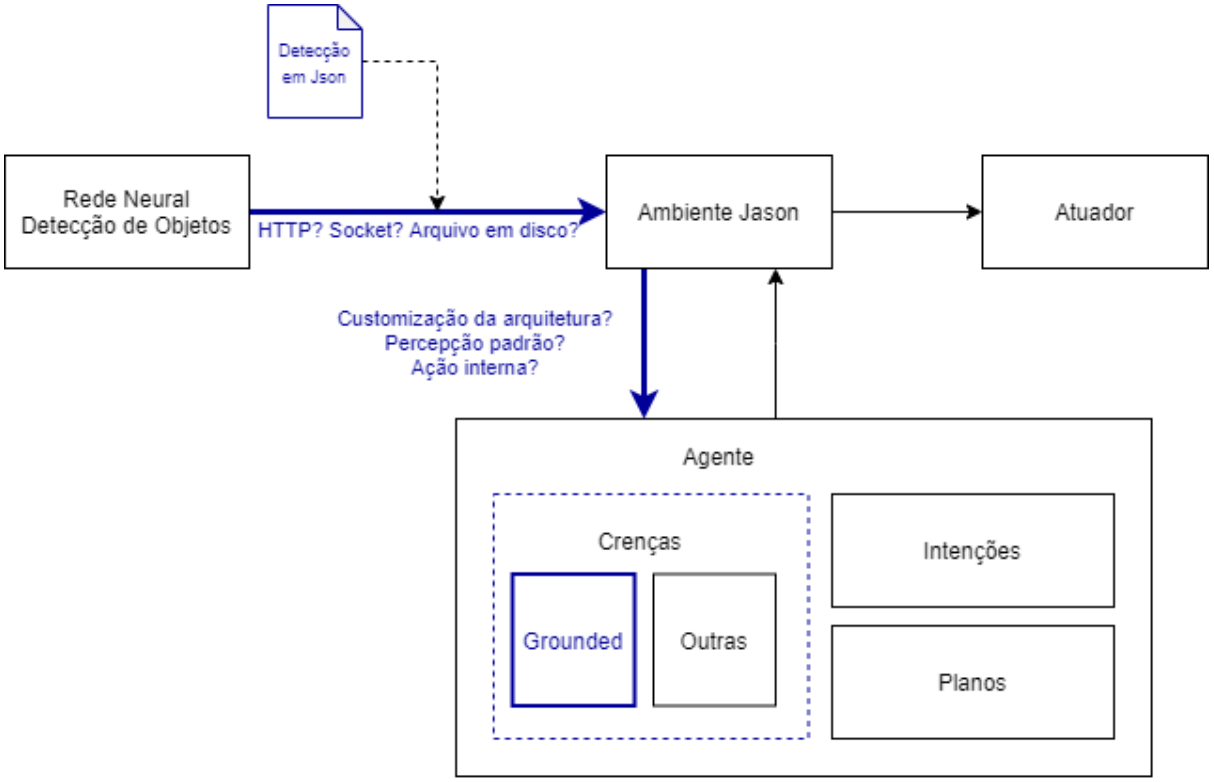
Fonte: Elaborado pelo autor.

Para a definição do modelo, foram consideradas algumas possibilidades, conforme a descrição abaixo:

1. Para a detecção de objetos:
    a. Tratar a Rede Neural como parte do ambiente e realizar a manipulação da detecção diretamente no ambiente Jason.
    b. A detecção gerada pela rede neural de detecção de objetos é convertida para um formato JSON pré-estabelecido. Este JSON precisa ser transmitido para o ambiente Jason e, para isso, foram consideradas algumas possibilidades:
        i. A disponibilização do arquivo em um diretório configurável, do qual o ambiente faz a leitura periodicamente para atualizar as percepções do agente.
        ii. Disponibilização do JSON via API REST, para a qual o ambiente faz requisições periodicamente para atualizar as percepções do agente.
        iii. Estabelecimento de uma conexão direta entre o ambiente e o sistema de detecção da rede neural utilizando sockets, o que permite uma interação bidirecional e em tempo real entre as partes.
2. Para o agente:

a. Desenvolver uma ação interna que faça o embasamento de uma crença.
b. Desenvolver um plano de embasamento de uma percepção.
c. Personalizar a arquitetura do agente para embasar uma percepção quando esta for advinda da detecção de objetos.
d. Personalizar funções internas da arquitetura do agente de forma a considerar crenças embasadas.

Figura 11 - Evolução da opção do modelo para embasamento simbólico



Fonte: Elaborado pelo autor.

Após a análise inicial das possibilidades de implementação e abordagens para o desenvolvimento do modelo, estabeleceu-se que seria mantida a estratégia de geração de um arquivo JSON que seria gravado em um diretório pelo módulo de detecção de objetos (módulo conexionista) e lido pelo agente BDI (módulo simbólico). Para fins de simplificação, foi estabelecido que a comunicação entre os módulos seria feita de maneira simples, através de trocas de mensagens utilizando sockets. Com isso, ambos os módulos devem de alguma forma conhecer a priori a localização do arquivo JSON. Tais pressupostos foram estabelecidos

exclusivamente com o intuito de simplificar a implementação daquilo que não faz parte dos objetivos do trabalho.

Além disso, algumas abordagens para o desenvolvimento foram esquematizadas e analisadas quanto às suas vantagens e desvantagens e as informações relativas a esta análise estão na Tabela 2.

Tabela 2 - Análise de possíveis abordagens para o desenvolvimento do modelo

Estratégia	Vantagem	Desvantagem
Criar anotação automática de crenças [grounded]	Flexibilidade para o desenvolvedor	Precisa ter alguma forma de definir quando uma crença pode ser anotada como grounded e envolve a alteração da engine do Jason também
Ação interna que verifique se determinada crença está embasada ou não	Permite ao desenvolvedor utilizar o mecanismo na medida de sua necessidade	Vai demandar o desenvolvimento da conexão com a detecção de objetos
Classe de agente + Classe de Ambiente + Ação interna	Cria um modelo completo para realizar a comunicação com a detecção de objetos, trata a informação repassada pelo ambiente customizado (sobre o embasamento) e permite a verificação por parte do desenvolvedor através da ação interna	Obriga o desenvolvedor a utilizar uma classe de agente específica

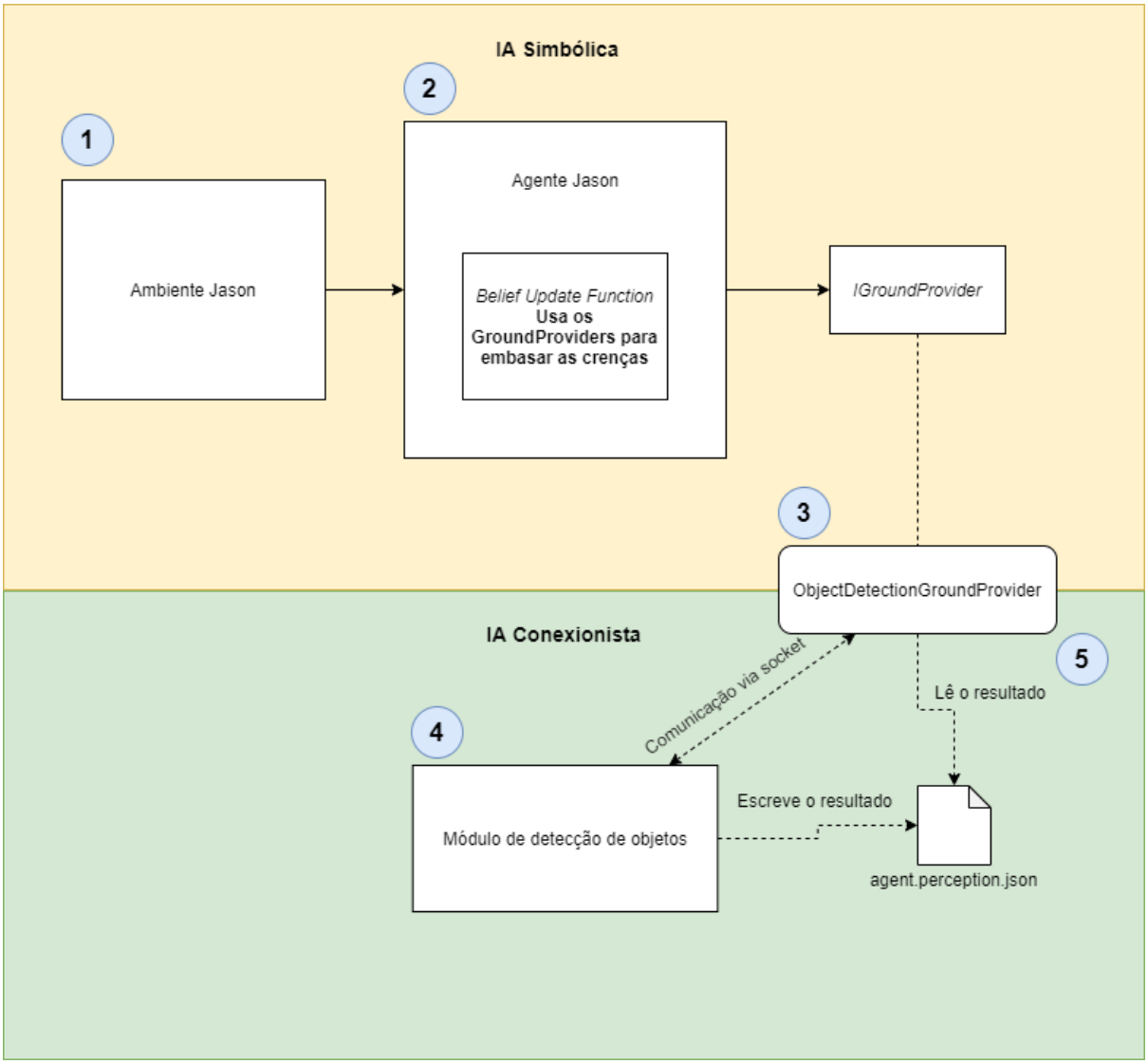
Fonte: Elaborado pelo autor.

4.2 IMPLEMENTAÇÃO DOS MODELOS

4.2.1 Personalização da classe Agent e introdução de provedores de embasamento

Esta implementação visou analisar a possibilidade de personalização da classe Agent do Jason, com o objetivo de utilizar a função de atualização de crenças (*buf*) como ponto de partida para a conexão com a detecção de objetos através de Provedores de Embasamento. Tais provedores foram definidos de forma que podem ser implementados pelas mais diversas fontes de embasamento possíveis. Eles consistem em uma classe que fica responsável por receber uma lista de crenças e retorná-la com as devidas crenças anotadas. Neste trabalho, a implementação do provedor deu-se através da conexão, de forma limitada e simplificada, com o módulo de detecção de objetos. A Figura 12 ilustra, de maneira simplificada, o fluxo executado pelo agente, que e consiste em:

Figura 12 - Modelo utilizando provedores de embasamento



Fonte: Elaborado pelo autor.

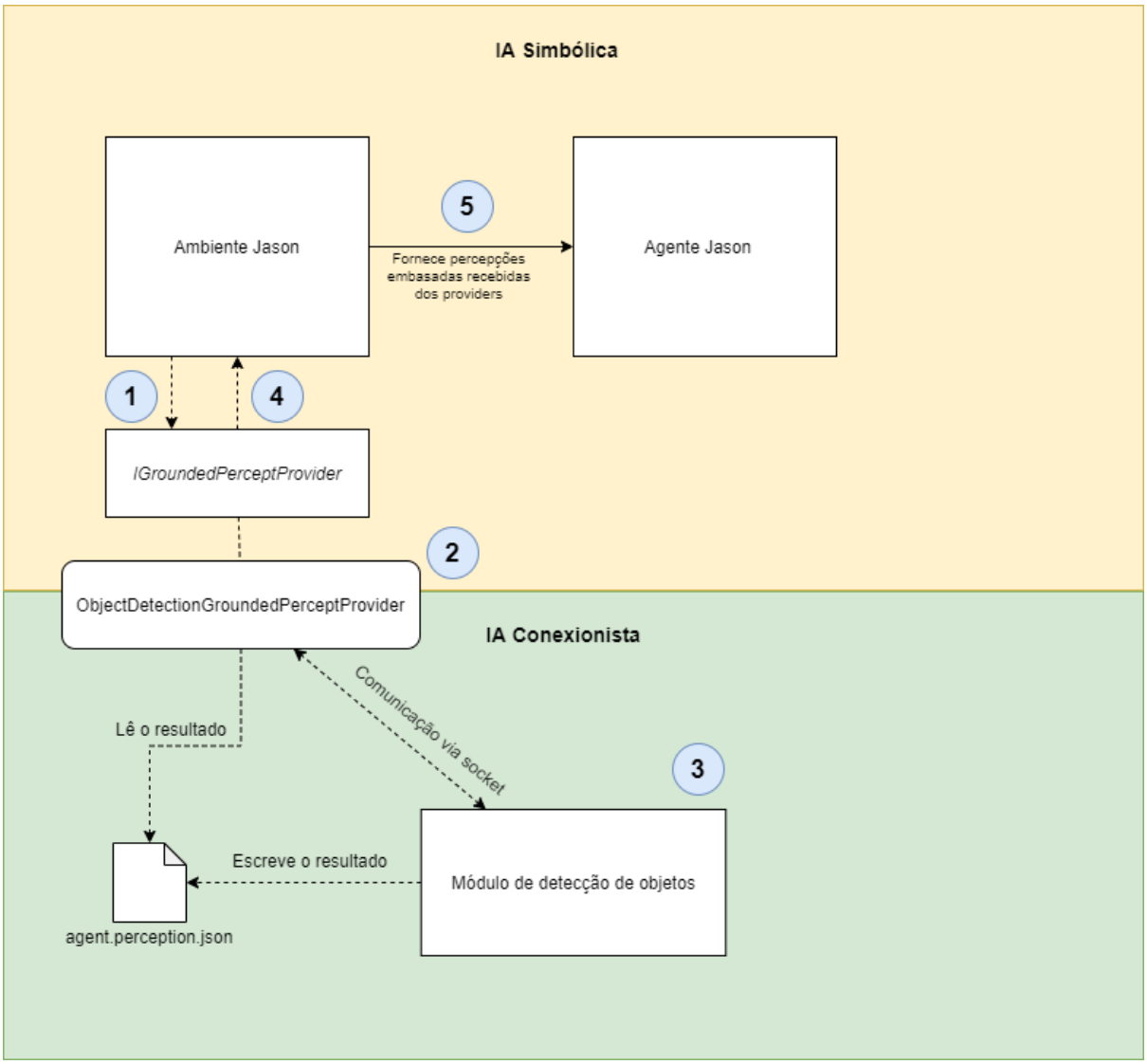
1. O ambiente Jason envia para o agente uma ou mais percepções.
2. Ao executar a função de atualização de crenças, o agente personalizado solicita a todos os provedores de embasamento que realizem o embasamento das crenças que foram recém recebidas pelo agente. Desta forma, o provedor de embasamento está livre para inserir novas crenças, além de embasar as crenças fornecidas pelo agente.
3. No caso do provedor de embasamento implementado neste trabalho, tal função é feita através da comunicação com o módulo de detecção de objetos, que foi desenvolvido em Python. O provedor de embasamento abre uma conexão via socket e envia um comando para o módulo de detecção de objetos para que este execute a detecção.

4. Após terminar a detecção, o módulo de detecção de objetos grava, em um arquivo JSON, o resultado da detecção e envia uma resposta para o provedor de embasamento, para que este possa ler o arquivo de resultado. É de se destacar aqui que o foco deste trabalho não está na integração entre as tecnologias, mas no modelo para o embasamento simbólico em si. Por este motivo, esta integração foi simplificada e não controla erros de comunicação e também assume que tanto o provedor quanto o módulo de detecção de objetos conhecem o local do arquivo a ser gerado com o resultado.
5. Ao receber a resposta do módulo de detecção de objetos, o provedor de embasamento lê o arquivo de resultado da detecção e analisa quais percepções podem ser embasadas a partir das detecções realizadas pelo módulo correspondente. Em seguida, o provedor adiciona anotações nas crenças que podem ser embasadas. No caso deste provedor, optou-se por incluir duas anotações: *[grounded]*, que indica que aquela crença está embasada (e tal anotação foi definida como um padrão para o embasamento) e, além desta, a anotação *[visual]* para indicar que o embasamento daquela crença está sendo feito com base em uma percepção visual.

4.2.2 Personalização da classe Environment e introdução de provedores de percepções embasadas

Outra abordagem experimentada foi a da personalização da classe Environment do Jason, permitindo o uso de Provedores de Percepções Embasadas. Neste modelo, ao invés de ser o agente o responsável pela busca do embasamento de suas crenças, o ambiente é quem recebe as percepções embasadas e as repassa aos agentes. Desta forma, os Provedores de Percepções Embasadas possuem um papel ativo, diferente do modelo anterior em que eram requisitados pelo agente. A Figura 13 ilustra o modelo, que consiste em:

Figura 13 - Modelo utilizando provedores de percepções embasadas



Fonte: Elaborado pelo autor.

1. O ambiente Jason cria seus Provedores de Percepções Embasadas e os inicia.
2. O Provedor de Percepção Embasada, ao ser iniciado pelo ambiente, começa a realizar seu processamento. No caso deste trabalho, foi realizada a implementação de um provedor de percepções através da detecção de objetos. O provedor realiza a comunicação com o módulo de detecção de objetos periodicamente, enviando um comando através de uma conexão via socket, para que o módulo de detecção execute a mesma.
3. Ao receber o comando para realizar a detecção, o módulo de detecção de objetos realiza sua tarefa, grava o resultado da detecção no arquivo JSON e responde ao provedor de percepções embasadas, para que este realize a leitura do resultado.

4. Ao receber a resposta do módulo de detecção de objetos, o provedor de percepções embasadas realiza a leitura do resultado e envia as percepções para o ambiente Jason, devidamente anotadas (com *[grounded]* para indicar o embasamento e *[visual]* para indicar que é uma percepção visual).
5. Ao receber as percepções embasadas dos provedores de percepções embasadas, o ambiente Jason as encaminha para o agente, que pode tratá-las da forma que for conveniente, utilizando as anotações para a tomada de decisões.

4.2.3 Ação interna para embasamento de símbolos

Para o desenvolvimento do modelo de embasamento através da utilização de ação interna foi necessário realizar alguns testes para definir de qual a seria a funcionalidade da ação, quais seriam seus parâmetros e como ela poderia ser utilizada.

Inicialmente considerou-se a criação de uma ação interna *sglib.isGrounded(s)*, onde *s* seria o parâmetro com o símbolo a ser embasado, e a ação retornaria *true* ou *false*. Porém, ao realizar os testes iniciais, constatou-se que o Jason considera que a ação interna falhou caso seu retorno seja *false*, o que inviabilizou esta implementação.

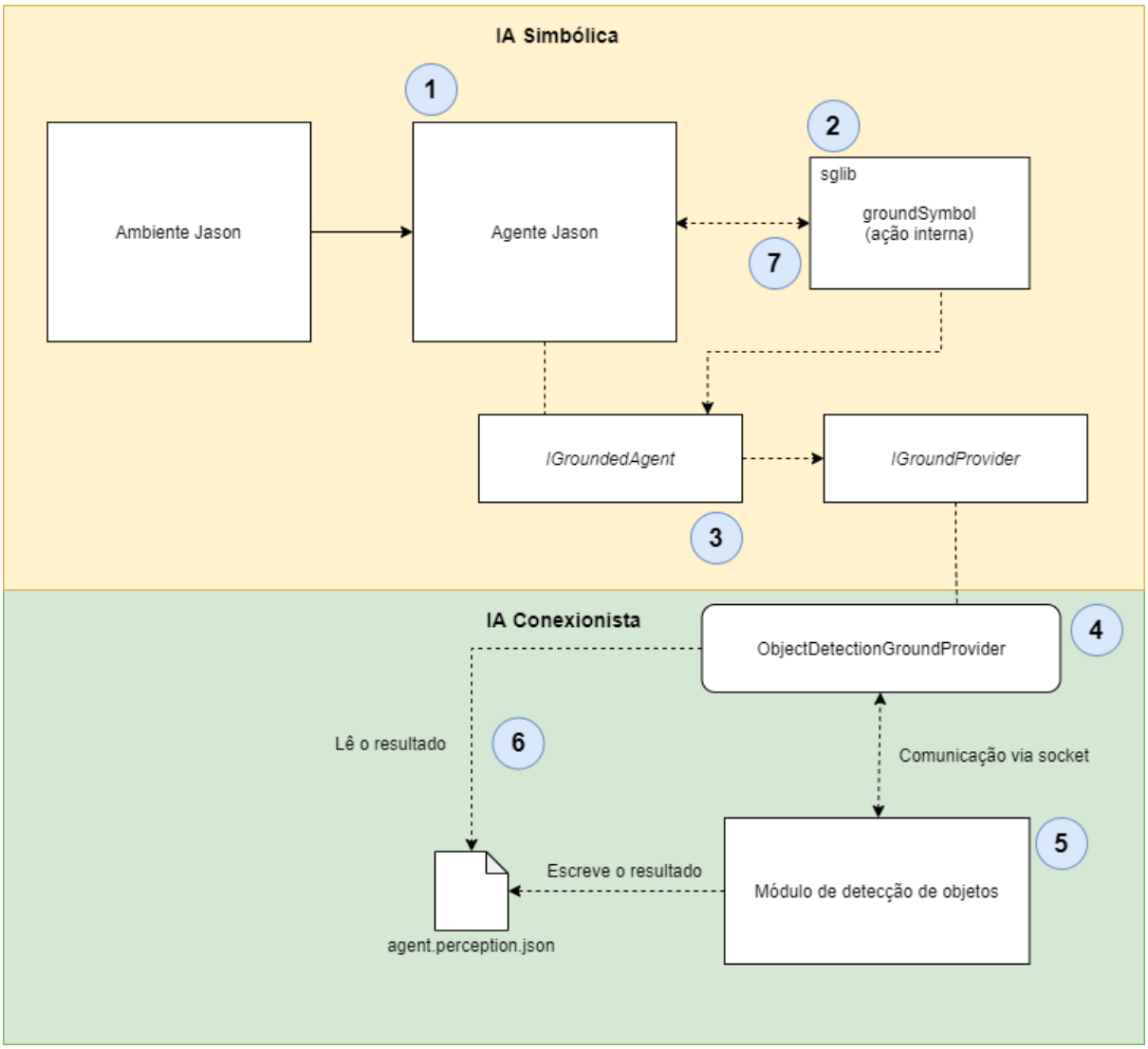
Uma segunda possibilidade analisada foi a de retornar o símbolo passado por parâmetro para a ação interna com as anotações sobre o embasamento, no mesmo padrão utilizado nas abordagens anteriores, ou seja, uma anotação *grounded* e outra *visual* para o caso da detecção de objetos. Porém, esta implementação tornaria complexa a utilização dentro do código do agente e por este motivo foi também descartada.

Por fim, a implementação escolhida foi a de uma ação interna chamada *sglib.groundSymbol(s,G)*, onde *s* é o símbolo a ser embasado e *G* é a variável que receberá a lista de embasamentos daquele símbolo. Caso o símbolo não esteja embasado, a lista estará vazia. Dessa forma, ao receber a lista em uma variável, o desenvolvedor possui a flexibilidade de utilizar os dados da lista conforme sua necessidade. Caso a intenção seja apenas verificar se determinado símbolo está embasado, basta verificar se a lista não está vazia. Além disso, se for necessário tomar decisões com base em qual tipo de embasamento está disponível, o desenvolvedor pode procurar na lista o tipo desejado, como por exemplo '*visual'*.

O desenvolvimento desta ação interna foi realizado em duas iterações. Isto porque para realizar o embasamento do símbolo recebido por parâmetro, a ação interna utiliza provedores de embasamento. Como o objetivo é deixar o modelo extensível, na primeira iteração o detentor dos provedores de embasamento era o agente, que deveria implementar uma interface que

permitia à ação interna solicitar o embasamento dos símbolos. Com isso, o desenvolvedor poderia configurar, através da classe do agente, quais provedores de embasamento seriam utilizados pelo mesmo. A Figura 14 ilustra o modelo e o funcionamento nesta primeira iteração, que consistia em:

Figura 14. Primeira versão do modelo com ação interna de embasamento de símbolos



Fonte: Elaborado pelo autor.

1. O agente executa a ação interna, passando o símbolo a ser embasado e a variável de retorno.
2. A ação interna verifica que seu agente implementa a interface *IGroundedAgent* e chama o método *groundSymbol* da interface, passando o símbolo como parâmetro.
3. O agente executa a operação em cada um dos *IGroundProviders* configurados.
4. No caso do provider implementado neste trabalho, este se comunica com o módulo de detecção de objetos via socket, solicitando a detecção.

5. Ao receber o comando, o módulo de detecção de objetos realiza sua tarefa, grava o resultado no arquivo JSON e notifica o provedor que terminou a operação.
6. O provedor lê o resultado da detecção de objetos e verifica se o símbolo solicitado está embasado e retorna os embasamentos para o agente.
7. A ação interna retorna o resultado do embasamento para a variável recebida por parâmetro.

Esta iteração serviu como prova de conceito para o uso da ação interna e permitiu a sua validação com um fluxo completo de execução. Porém, ela apresenta alguns pontos negativos em relação à generalização do modelo e também delega mais responsabilidades para o desenvolvedor, o que se deseja reduzir. O principal ponto negativo é a necessidade de implementação da interface *IGroundedAgent* na classe de todos os agentes que visem utilizar a ação interna. Isto torna o modelo menos flexível e demanda um trabalho inicial maior para a sua utilização.

Para resolver estes problemas, foram analisadas possíveis soluções para realizar a configuração dos provedores a serem utilizados pela ação interna, de tal modo que esta fosse capaz de realizar o embasamento sem depender diretamente do agente que a está executando. Uma possibilidade considerada seria a realização da configuração durante a inicialização do sistema, através do ambiente, por exemplo.

Contudo, é possível que se deseje utilizar a ação interna de embasamento de símbolos em agentes com propósitos diferentes e que necessitem utilizar um conjunto diferente de provedores de embasamento. Com isso, identificou-se que a configuração deveria também poder ser direcionada ao agente para que, ao ser executada, a ação interna pudesse utilizar os provedores corretos.

Ao utilizar o ambiente para a realização da configuração dos provedores de embasamento, o modelo estaria também impondo ao desenvolvedor a utilização de um ambiente conhecido pela ação interna para que esta pudesse buscar as configurações. Visando evitar tal necessidade, buscou-se outras formas de realizar a configuração.

As ações internas no Jason são implementadas através da criação de uma classe que herda de *DefaultInternalAction*. Cada agente, ao executar a ação pela primeira vez, cria uma instância da classe da ação interna e utiliza esta única instância durante todo o seu ciclo de vida. Isto permite que a ação possua um estado por agente, ou seja, é possível guardar dados relativos à sua execução naquele agente ou qualquer outra informação que seja conveniente. Considerando esta característica, identificou-se a possibilidade de guardar na instância da ação

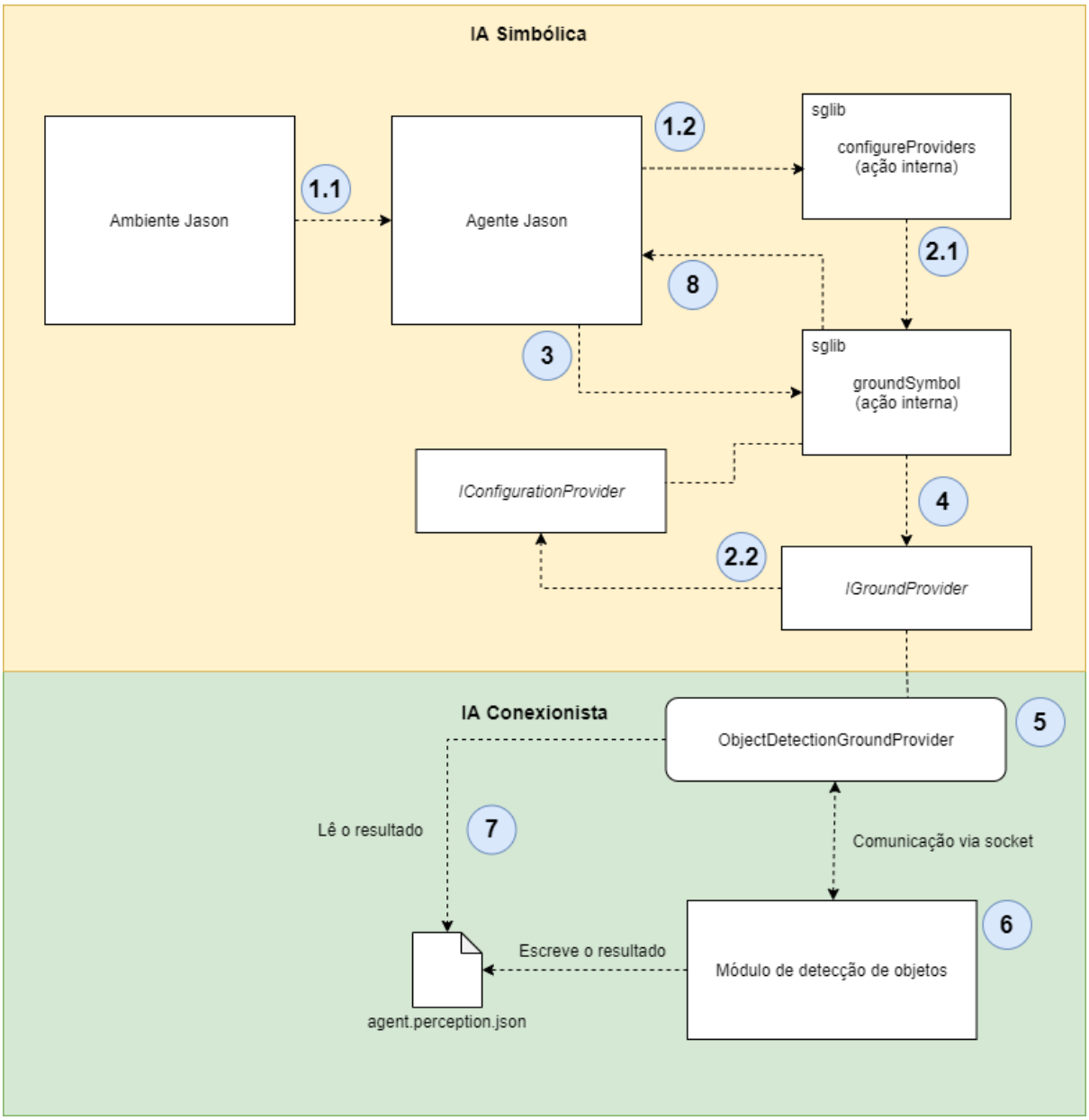
interna as configurações dos provedores de embasamento e demais configurações necessárias para seu funcionamento.

Visando enviar para o agente a configuração da ação interna *sglib.groundSymbol(s,G)*, foi desenvolvida uma segunda ação interna chamada *sglib.configureGrounders(j)*, onde *j* é uma string contendo um JSON com as configurações que informam quais os provedores de embasamento devem ser utilizados por aquele agente e quais são as configurações dos módulos dos quais os providers selecionados dependem.

A função da ação interna *sglib.configureGrounders(j)* é enviar para a instância da ação interna *sglib.groundSymbol(s,G)* a configuração informada. Para isto, a ação interna de configuração obtém a instância da ação de embasamento acessando o agente e enviando o JSON de configuração através de um método disponível na classe da ação interna de embasamento de símbolos. Após executar a ação de configuração, o agente está pronto para utilizar a ação interna de embasamento de símbolos, independentemente da sua classe ou do ambiente no qual está situado.

A Figura 15 ilustra a segunda versão do modelo de embasamento através da utilização de ação interna, que consiste em:

Figura 15 - Segunda versão do modelo com ação interna de embasamento de símbolos



Fonte: Elaborado pelo autor.

1. A primeira etapa necessária para a utilização deste modelo pode ser executada de duas maneiras, sendo de livre escolha do desenvolvedor:
   1.1. Através do ambiente, utilizando um código fonte em Java, o desenvolvedor gera o JSON com as configurações e, através da adição de uma percepção, envia a configuração para o agente, que por sua vez deve possuir um plano para o evento de adição desta crença que irá executar a ação interna *sglib.configureGrounders(j)*. A geração do JSON pode ser feita manualmente, o arquivo pode ser lido do disco, ou ainda é possível utilizar um construtor de configurações que foi desenvolvido para facilitar o uso da biblioteca. A classe

SGLibConfigurationBuilder pode ser instanciada e o desenvolvedor pode adicionar os nomes dos provedores a serem utilizados, bem como as configurações dos módulos, instanciando objetos do tipo *SGLibModule* ou tipos derivados deste, como por exemplo o *SGLibObjectDetectionModule* que já possui métodos para facilitar a configuração do módulo de detecção de objetos já desenvolvido neste trabalho.

1.2. Este passo deve ser obrigatoriamente executado, ele pode ser programado para ser executado diretamente pelo agente, através do código *asl*, passando o JSON diretamente para a ação interna de configuração, ou através do evento gerado pelo passo 1.1. Aqui o agente executa a ação interna de configuração e passa o JSON.

2. A ação interna de configuração envia para a ação interna de embasamento de símbolos o JSON de configuração.

2.1. A ação interna por sua vez lê as configurações e inicia os providers e módulos necessários para o seu funcionamento. A ação interna *sglib.groundSymbol(s,G)* implementa a interface *IConfigurationProvider* para que os provedores de embasamento possam buscar as configurações relevantes para o seu funcionamento.

2.2. Os provedores de embasamento então buscam na ação interna, através da interface *IConfigurationProvider*, as configurações que são necessárias para o seu funcionamento.

3. Após realizar a configuração, o agente está livre para executar a ação interna de embasamento de símbolos. A ação por sua vez executa todos os provedores de embasamentos configurados para serem utilizados e realiza o embasamento propriamente dito.

4. Neste trabalho, a ação interna irá executar o provedor de embasamento de detecção de objetos.

5. O provedor de embasamento de detecção de objetos realiza a comunicação com o módulo de detecção de objetos através de uma conexão via socket, enviando um comando para realizar a detecção.

6. O módulo de detecção de objetos realiza a sua tarefa e grava o resultado da detecção no arquivo de resultados. Depois deste processo, notifica o provedor para que este leia o resultado no arquivo.

7. O provedor de embasamento lê o resultado da detecção de objetos e realiza o embasamento, caso seja possível.
8. A ação interna retorna a lista de embasamentos do símbolo para a variável passada por parâmetro. Caso não haja embasamento, é retornada uma lista vazia.

4.3 EXPERIMENTOS

4.3.1 Objetivo

Visando desenvolver um protótipo do modelo proposto, utilizando-se de um modelo para reconhecimento de objetos e uma arquitetura para agentes BDI, os experimentos a serem apresentados implementaram os modelos propostos neste trabalho para que sua utilização pudesse ser analisada. A utilização experimental dos três modelos apresentados anteriormente permite a análise das suas vantagens e desvantagens em relação ao contexto desenvolvido para o experimento. Além disso, permite que seja analisada a capacidade dos modelos de fundamentar símbolos e de conectar os componentes de IA Simbólica e IA Conexionista.

4.3.2 Métricas de avaliação

Devido à natureza do problema que o modelo pretende tratar, a maior parte da sua avaliação se torna qualitativa, visto que este trabalho não visa melhoria de performance de quaisquer partes envolvidas, como por exemplo a acurácia das detecções realizadas pela rede neural de detecção de objetos, ou ainda do seu tempo de processamento. O foco deste trabalho está no problema do embasamento simbólico e como os modelos propostos auxiliam no embasamento de símbolos. Além disso, deseja-se avaliar o quanto os modelos facilitam a tarefa de embasamento de símbolos. A Tabela 3 especifica as métricas estabelecidas para a avaliação dos experimentos.

Uma versão deste sistema foi desenvolvida como *baseline* para a comparação dos experimentos de cada um dos modelos propostos neste trabalho. Esta versão não realiza nenhuma consideração a respeito no embasamento das crenças do agente. As mudanças no ambiente são simuladas através da classe ambiente do Jason e o agente reage às mudanças conforme elas são simuladas. O mesmo comportamento de simulação das mudanças no ambiente foi implementado nos demais experimentos, com os devidos ajustes para se encaixar nos requisitos de cada modelo.

A partir da versão de base do experimento, foram então criadas novas versões para a implementação da utilização de cada um dos modelos propostos neste trabalho. Com isto, foi possível identificar quais seriam as alterações e adaptações necessárias a serem realizadas para a utilização de cada modelo.

Tabela 3 - Métricas de avaliação dos experimentos

Tipo	**Métrica**
Qualitativa	O modelo é capaz de embasar símbolos?
	O quanto o modelo facilita a tarefa de fundamentação de símbolos?
	O quão adequada foi esta implementação para o contexto do experimento?
	Qual é o nível de complexidade de utilização do modelo?
Quantitativa	Número de linhas de código alteradas
	Número de arquivos alterados
	Número de planos alterados ou incluídos no agente

Fonte: Elaborado pelo autor.

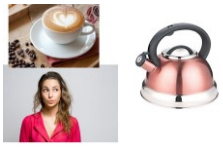
Para o desenvolvimento dos experimentos foi definido um cenário hipotético de um agente que simula um robô de café inteligente, que utiliza objetos para servir o café, conforme foi descrito na seção 4.2. O agente espera possuir um objeto que será utilizado para servir o café ao cliente, como por exemplo uma xícara, e outro objeto que será utilizado para despejar o café no primeiro objeto, como por exemplo uma chaleira. Ao perceber que há uma pessoa no ambiente e que estão disponíveis os objetos necessários para a realização da tarefa, o agente serve o café e verifica se sua ação foi bem sucedida.

Para a execução do experimento, foram simuladas de forma simples algumas situações que o robô poderia vivenciar no mundo real para que o mesmo realizasse suas tarefas. Para simplificar o experimento e, ao mesmo tempo, determinar a eficiência dos modelos, foram preparadas cinco situações. A Tabela 4 resume as situações utilizadas durante os experimentos.

Devido ao fato de que a rede neural utilizada detecta a imagem simulada com a classe woman, os agentes foram programados para adicionarem uma crença detected(person) ao receber a crença detected(woman). Isto foi feito com o intuito de facilitar os testes.

O módulo de detecção de objetos possui suas classes já pré-definidas e a saída da rede neural respeita tais classes. Porém, visto que o Jason possui algumas restrições sintáticas, foi necessário realizar um tratamento no nome das classes recebidas da detecção de objetos para que não houvesse um problema de compatibilidade com o Jason. Visto que a limitação é imposta pelo Jason, optou-se por realizar tal tratamento nos provedores de embasamento, já que estes estão ligados à arquitetura do Jason. Na Tabela 4, as percepções que devem ser embasadas por estarem conectadas aos seus respectivos objetos detectados pela rede neural estão destacados.

Tabela 4 - Situações de ambiente simuladas nos experimentos

Situação	**Percepções simuladas**	**Imagem**	**Detecções**	**Percepção gerada pela detecção (transformada)**
Nº 1	detected(kettle)		Human face Clothing Woman Girl	detected(human_face) detected(clothing) detected(woman) detected(girl)
Nº 2	detected(person) detected(coffee_cup)		Kettle Human face Clothing Woman Human hair	detected(kettle) detected(human_face) detected(clothing) detected(woman) detected(human_hair)
Nº 3	detected(person) detected(kettle) detected(cup)		Kettle Human face Clothing Coffee cup Woman Saucer	detected(kettle) detected(human_face) detected(clothing) detected(coffee_cup) detected(woman) detected(saucer)
Nº 4	detected(person) detected(kettle) detected(coffee_cup)		Kettle Human face Clothing Coffee cup Woman Saucer	detected(kettle) detected(human_face) detected(clothing) detected(coffee_cup) detected(woman) detected(saucer)
Nº 5	detected(person) detected(kettle) detected(coffee_cup) detected(coffee)		Kettle Human face Coffee Saucer Clothing Coffee cup Woman Girl	detected(kettle) detected(human_face) detected(coffee) detected(saucer) detected(clothing) detected(coffee_cup) detected(woman) detected(girl)

Fonte: Elaborado pelo autor.

Cada situação visa demonstrar possíveis etapas de percepção do agente, bem como seus embasamentos. Na primeira situação há apenas a percepção de detecção de uma chaleira simulada pelo ambiente, porém esta percepção não está embasada, visto que na detecção de objetos a imagem não mostra este objeto. A segunda situação apresenta a percepção de detecção de uma pessoa e de uma xícara de café. A primeira percepção deve ser embasada, pois existe uma pessoa na imagem, já a xícara não. Nas situações 3 e 4 a imagem utilizada é a mesma, porém o conjunto de percepções simuladas pelo ambiente é diferente, visando demonstrar a diferença nas decisões do agente com base no embasamento de suas crenças. Por fim, a quinta situação é a que simula o término da tarefa de servir o cliente, utilizando uma imagem que possui o café na xícara, bem como as detecções correspondentes simuladas pelo ambiente.

4.3.3 Execução

Para a execução dos experimentos foram criadas imagens para simular as situações do ambiente, conforme demonstrado na coluna imagem, da Tabela 4. Nenhum dataset de treinamento foi utilizado, pois a rede neural utilizada já se encontrava pré-treinada. Para a utilização do TensorFlow com a placa de vídeo disponível no notebook utilizado foi necessária a instalação do CUDA Toolkit. A Tabela 5 mostra um resumo das configurações do ambiente e das ferramentas utilizadas para a execução dos experimentos.

Tabela 5 - Configuração do ambiente e ferramentas utilizadas nos experimentos

Ambiente
Windows 10 Home 64 bits Processador Intel Core i5 2.20GHz Placa de vídeo NVidia GeForce 820M Memória RAM 8GB DDR3
Detecção de objetos
TensorFlow 2.1.0 Python 3.7.5 CUDA v10.1 TensorFlow open images-trained model: faster_rcnn_inception_resnet_v2_atrous_oidv4
Agente BDI
Eclipse IDE for Enterprise Java Developers - 2020-06 Jason Eclipse plug-in

```
Java(TM) SE Runtime Environment (build 1.8.0_281-b09)
Jason 2.5
Json library 20200518.0.0
```

Fonte: Elaborado pelo autor.

Para facilitar a execução e a instalação das bibliotecas do Python, foi criado um ambiente virtual de execução, que é normalmente chamado de *venv*. Este procedimento não é obrigatório para a reprodução dos experimentos, porém facilita a manutenção do ambiente corretamente configurado, de maneira isolada. As imagens de teste criadas foram colocadas em um diretório e foram criados arquivos *.bat* que são executados pelo ambiente Jason para simular as mudanças nas detecções de objetos. Estes arquivos *.bat* mudam a imagem que será detectada, excluindo o arquivo atual e substituindo pelo desejado para a situação em questão.

O módulo de detecção de objetos, desenvolvido em Python a partir do código de exemplo disponibilizado pelo TensorFlow, consiste em dois arquivos:

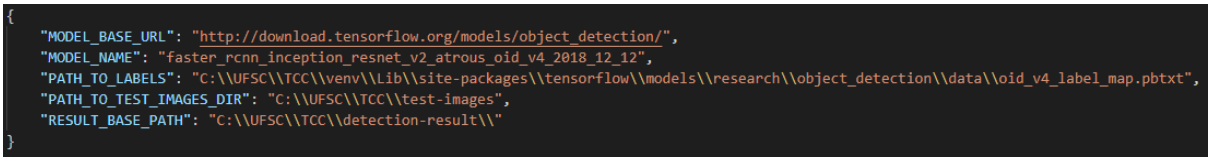
detection.py: Neste arquivo está a lógica da detecção de objetos propriamente dita. A classe ObjectDetection realiza toda a tarefa de download da rede neural, descoberta da imagem a ser detectada, execução da detecção e persistência do resultado em arquivo JSON.

server.py: Aqui está definido o servidor socket que é utilizado para comunicação com o agente BDI. Ele executa a detecção a partir do comando recebido e responde após a finalização.

Além disso, o módulo de detecção de objetos possui um arquivo de configuração no formato JSON, que precisa estar conter as seguintes configurações:

- MODEL_BASE_URL: Esta é a URL base de onde o módulo irá fazer o download da rede neural pré-treinada disponibilizada no site do TensorFlow.
- MODEL_NAME: Este é o nome do arquivo da rede neural pré-treinada, sem a extensão, para que seja feito o download.
- PATH_TO_LABELS: Deve conter o caminho para o arquivo *.pbtxt* com o mapeamento das classes dos objetos para o dataset utilizado no treinamento da rede neural escolhida.
- PATH_TO_TEST_IMAGES_DIR: Este é o caminho do diretório que contém a imagem que será utilizada para realizar a detecção de objetos. É neste diretório também que deverá existir os arquivos *.bat* que realizarão a troca das imagens de teste, os o módulo do agente BDI irá executar.

Figura 16 - Arquivo de configuração do módulo de detecção de objetos utilizado nos experimentos



```
{
    "MODEL_BASE_URL": "http://download.tensorflow.org/models/object_detection/",
    "MODEL_NAME": "faster_rcnn_inception_resnet_v2_atrous_oid_v4_2018_12_12",
    "PATH_TO_LABELS": "C:\\UFSC\\TCC\\venv\\Lib\\site-packages\\tensorflow\\models\\research\\object_detection\\data\\oid_v4_label_map.pbtxt",
    "PATH_TO_TEST_IMAGES_DIR": "C:\\UFSC\\TCC\\test-images",
    "RESULT_BASE_PATH": "C:\\UFSC\\TCC\\detection-result\\"
}
```

Fonte: Elaborado pelo autor.

O módulo do agente BDI, por sua vez, foi estruturado em projetos, onde cada projeto possui um papel específico para o desenvolvimento deste trabalho, bem como para a execução e avaliação dos experimentos. Os projetos desenvolvidos são:

sglib: O nome deste projeto deriva de *Symbol Grounding Library* e é a biblioteca que implementa os modelos propostos. É neste projeto que se encontram todas as interfaces e classes desenvolvidas para os modelos. Aqui estão os provedores de embasamento, os provedores de percepções embasadas, o agente e o ambiente personalizados, a ação interna, etc.

sgmodel_baseline: Este projeto realiza a implementação do cenário dos experimentos sem qualquer utilização de embasamento simbólico. Ele serve como base para a comparação com os demais projetos.

sgmodel_v1: Implementa o experimento do modelo de personalização da classe Agent e uso de provedores de embasamento.

sgmodel_v2: Implementa o experimento do modelo de personalização da classe Environment e uso de provedores de percepções embasadas.

sgmodel_v3: Implementa o experimento do modelo de ação interna para embasamento de símbolos.

4.3.3.1 Execução do projeto sgmodel_baseline

A execução do projeto *sgmodel_baseline* demonstra o funcionamento simples do experimento, sem qualquer utilização de embasamento simbólico. O ambiente simula as mudanças de percepções e, através do MAS Console, é possível verificar o funcionamento do agente.

Ao utilizar o Mind Inspector é possível verificar as crenças do agente e com isso constata-se que o mesmo possui corretamente as crenças esperadas. Não há anotações de embasamento nas mesmas, visto que este projeto não as considera.

4.3.3.2 Execução do módulo de detecção de objetos

Para a execução dos demais experimentos, a primeira etapa é iniciar o módulo de detecção de objetos. A Figura 17 mostra um exemplo de execução do módulo, que inicia em uma linha de comando a ativação do ambiente virtual de execução do Python e, em seguida, executa o arquivo *server.py*.

Figura 17 - Exemplo de execução do módulo de detecção de objetos

```
C:\>c:\ufsc\tcc\venv\scripts\activate

(venv) C:\>py d:\GitHub\tcc\object_detection\server.py
2021-04-05 09:49:18.772934: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic
library cudart64_101.dll
ObjectDetection init started
Loading configuration from d:\GitHub\tcc\object_detection/config.json
PATH_TO_LABELS: C:\UFSC\TCC\venv\Lib\site-packages\tensorflow\models\research\object_detection\data\oid_v4_label_map.pbt
xt
PATH_TO_TEST_IMAGES_DIR: C:\UFSC\TCC\test-images
MODEL_NAME: faster_rcnn_inception_resnet_v2_atrous_oid_v4_2018_12_12
loading category index...
category index loaded
loading model...
BASE_URL: http://download.tensorflow.org/models/object_detection/
Loading model from: http://download.tensorflow.org/models/object_detection/faster_rcnn_inception_resnet_v2_atrous_oid_v4
_2018_12_12.tar.gz
2021-04-05 09:49:51.555403: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic
library nvcuda.dll
```

Fonte: Elaborado pelo autor.

Ao iniciar, o módulo de detecção de objetos já executa a detecção caso exista alguma imagem no diretório configurado, o que facilita a primeira execução dos testes. Com este módulo executando corretamente, é possível passar para a próxima etapa da execução dos experimentos.

4.3.3.3 Compilação do projeto sglib e preparação dos demais projetos

Para o desenvolvimento e execução dos projetos Jason, foi utilizada a IDE Eclipse, juntamente com o Jason Eclipse plug-in. A primeira etapa consiste em compilar a biblioteca sglib. Após realizar a compilação, deve-se copiar a pasta *bin/sglib* (que contém os arquivos .class) para a raiz dos demais projetos (com exceção do projeto *sgmodel_baseline*, que não utiliza a biblioteca), resultando por exemplo no caminho *sgmodel_v1/sglib*.

Após realizar a cópia da biblioteca compilada, já é possível compilar e executar os projetos dos experimentos. Para o acompanhamento do funcionamento dos agentes é necessário

visualizar os logs gerados no MAS Console. Tanto a *sglib* quanto os projetos dos experimentos utilizam o logger para imprimir detalhes de sua execução no console, a fim de demonstrar o funcionamento do agente. Além disso, é possível utilizar o Mind Inspector para analisar o estado do agente, visualizando por exemplo suas crenças em cada etapa de sua execução.

4.3.3.4 Execução do projeto sgmodel_v1

O projeto *sgmodel_v1* implementa, como descrito anteriormente, a utilização da classe Agent personalizada. Com isso, ao executar a função buf, o agente executa o embasamento através dos provedores que estão configurados para serem utilizados. Neste caso, o *ObjectDetectionGroundProvider* é o único a ser utilizado e é responsável pela conexão com o módulo de detecção de objetos. Através da leitura do log de execução no MAS Console é possível verificar o seu funcionamento, desde a conexão e envio do comando, até o recebimento da resposta e leitura do resultado da detecção para o efetivo embasamento das crenças.

Ao receber uma percepção de um objeto que poderia ser utilizado para servir o café, o agente só o considera válido se for possível embasá-lo. Ou seja, mesmo que o ambiente envie uma percepção *detected(coffee_cup)*, por exemplo, a percepção da detecção da xícara de café só será considerada válida para o agente caso ele consiga embasar esta detecção através do módulo de detecção de objetos.

A partir do momento em que o agente possui todos os objetos necessários, devidamente embasados e, além disso, há uma pessoa aguardando, o agente inicia sua tarefa de servir café. Após realizar a tarefa, o agente aguarda alguns segundos e então verifica se sua tarefa foi bem sucedida.

Utilizando o Mind Inspector é possível analisar as crenças do agente, sendo que agora algumas delas estão embasadas, contendo as anotações *grounded* e *visual*. Com esta análise é possível perceber que este projeto também realiza corretamente a sua tarefa e cumpre o papel de embasar os símbolos através do uso experimental do primeiro modelo proposto.

4.3.3.5 Execução do projeto sgmodel_v2

O modelo de provedores de percepções embasadas foi implementado no projeto *sgmodel_v2*. Aqui são configurados os provedores na classe do ambiente, que por sua vez serve de receptor das percepções embasadas. Nele os provedores são iniciados e começam seu trabalho de prover as percepções. No caso do *ObjectDetectionGroundedPerceptProvider*, um

timer executa a detecção de objetos em um determinado intervalo de tempo, ao passo em que o ambiente Jason simula as mudanças nas imagens.

Da mesma forma que nos projetos anteriores, ao perceber que possui todos os objetos necessários, o agente então realiza sua tarefa e serve o cliente. Em seguida verifica se conseguiu realizar seu objetivo com sucesso, analisando se recebeu uma percepção de que há café no ambiente. Neste projeto não ocorre a situação de uma percepção de detecção de objeto não estar embasada, visto que quem está gerando a percepção é o próprio provedor de percepções já embasadas.

4.3.3.6 Execução do projeto sgmodel_v3

Por fim, o projeto *sgmodel_v3* implementa o uso da ação interna de embasamento de símbolos. Esta abordagem permite realizar o embasamento das crenças com maior flexibilidade, dando maior controle para o desenvolvedor do agente sobre o que e quando exatamente embasar. Ao analisar o início da execução do projeto no MAS Console é possível notar que inicialmente não há qualquer comunicação com o módulo de detecção de objetos, devido à forma como o agente foi implementado.

O agente deste projeto foi desenvolvido de tal forma que a comunicação com o módulo de detecção de objetos só é feita quando as crenças do agente já indicam que ele possui os objetos necessários para realizar sua tarefa de servir café. Dessa forma, o agente executa o embasamento simbólico para estas crenças e, ao obter sucesso neste embasamento, executa seu objetivo principal.

4.3.4 Resultados

A avaliação dos experimentos deu-se, em sua maioria, mediante o uso de métricas qualitativas, conforme descrito na seção 4.5.2. As seções seguintes irão descrever os resultados de cada métrica analisadas nos experimentos.

4.3.4.1 O modelo é capaz de embasar símbolos?

Em relação ao embasamento de símbolos, considera-se que todos os modelos propostos são capazes de conectar os símbolos que são utilizados pelo agente com seus correspondentes no mundo real, na medida em que o objeto detectado é mapeado para o símbolo. Porém, apesar de conectar o símbolo a um objeto do mundo real, nenhum modelo é capaz de resolver o

Problema do Embasamento Simbólico, visto que, para mapear o objeto para um símbolo, a rede neural precisou ser previamente treinada e recebeu tais símbolos de forma extrínseca. Os modelos seguem as abordagens representacionalistas e, apesar de não resolverem o Problema do Embasamento Simbólico, são capazes de embasar símbolos conectando-os aos seus referentes.

4.3.4.2 O quanto o modelo facilita a tarefa de fundamentação de símbolos?

Por se tratar de uma biblioteca que fornece toda a estrutura para o embasamento de símbolos e permite ainda a sua extensão, os modelos facilitam consideravelmente a tarefa de fundamentação de símbolos. Os três modelos oferecem facilidades para a realização da tarefa, diferindo apenas no fluxo e nos papéis que cada parte do framework Jason assume. Para o contexto do experimento, o modelo que mais facilita a tarefa é o implementado pelo projeto *sgmodel_v2*, o qual utiliza o modelo de percepções embasadas. Após o desenvolvimento da configuração inicial, a tarefa de embasar qualquer símbolo se torna muito simples em todos os modelos apresentados. Além disso, os modelos permitem a reutilização de código feito para embasar símbolos, através da reutilização de provedores de embasamento e/ou de percepções embasadas. Isto permite que, na medida em que provedores são implementados, novos projetos ou evoluções de projetos existentes tenham seus símbolos embasados de maneira ainda mais fácil e rápida.

4.3.4.3 O quão adequada foi esta implementação para o contexto do experimento?

O contexto do experimento demanda a identificação de objetos e de pessoas no ambiente para que um café seja servido. Por este motivo, a funcionalidade do agente está diretamente ligada e é totalmente dependente da detecção de objetos que, em uma situação real, provavelmente seria executada em imagens capturadas de uma câmera.

O projeto *sgmodel_v1*, que implementa a utilização dos provedores de embasamento, não se mostrou adequado ao contexto, pois para realizar o embasamento das crenças na função de atualização de crenças do agente, é necessário que este receba novas percepções para que a função em questão seja executada. Por este motivo, para que o experimento funcionasse, foi necessário implementar a simulação de percepções no ambiente.

Também se mostrou inadequado ao contexto, pelo mesmo motivo, o projeto *sgmodel_v3*, que utiliza a ação interna de embasamento de símbolos. Também foi necessário

simular as percepções neste projeto para que ao final fosse executada a ação interna que verificaria o embasamento destas percepções para então realizar a tarefa de servir o café.

Já o projeto *sgmodel_v2*, que implementa a utilização de provedores de percepções embasadas se mostrou adequado ao experimento, pois ele se conecta diretamente com o módulo de detecção de objetos e gera diretamente para o agente as percepções embasadas. Apesar de haver simulações de trocas de imagens, estas simulações seriam removidas em um sistema real, que captaria diretamente da câmera as imagens a serem analisadas. Com a utilização deste modelo é possível fazer com que o agente funcione corretamente sem a necessidade de uma duplicidade das percepções, como seria necessário nos demais modelos.

4.3.4.4 Qual é o nível de complexidade de utilização do modelo?

O experimento implementado no projeto *sgmodel_v1* demonstra a facilidade em utilizar o modelo que utiliza provedores de embasamento em conjunto com uma classe de agente personalizada. Ao estender a classe *BaseGroundedAgent* disponível na biblioteca, o desenvolvedor precisa apenas implementar dois métodos básicos: o método que cria as configurações para o modelo e o método que instancia a registra os provedores de embasamento. Toda a lógica restante que é necessária para o funcionamento do modelo já está pronta. Isto facilita muito o trabalho do desenvolvedor que deseja utilizar a biblioteca. Este modelo também não exige grandes alterações nos planos do agente, exceto aqueles onde se deseja considerar o embasamento simbólico. Porém, ao utilizar este modelo, o desenvolvedor precisa utilizar uma classe de agente personalizada, seja estendendo a classe *BaseGroundedAgent* ou implementando por conta própria as funcionalidades necessárias para o funcionamento dos provedores.

O projeto *sgmodel_v2*, que implementa o modelo de provedores de percepções embasadas, também apresenta uma grande facilidade em sua utilização. Isto porque basta que o desenvolvedor estenda a classe *BaseGroundedPerceptTargetEnvironment* e defina a sua classe como sendo a que deve ser utilizada pelo ambiente. Da mesma forma que no projeto anterior, o desenvolvedor precisa apenas implementar dos métodos básicos: o método que cria as configurações para o modelo e o método que instancia a registra os provedores de de percepções embasadas. Aqui o desenvolvedor também possui a liberdade de implementar as interfaces e funcionalidades em seu próprio ambiente. Porém, para utilizar toda a estrutura pronta, disponível na biblioteca, o desenvolvedor precisa utilizar este ambiente personalizado.

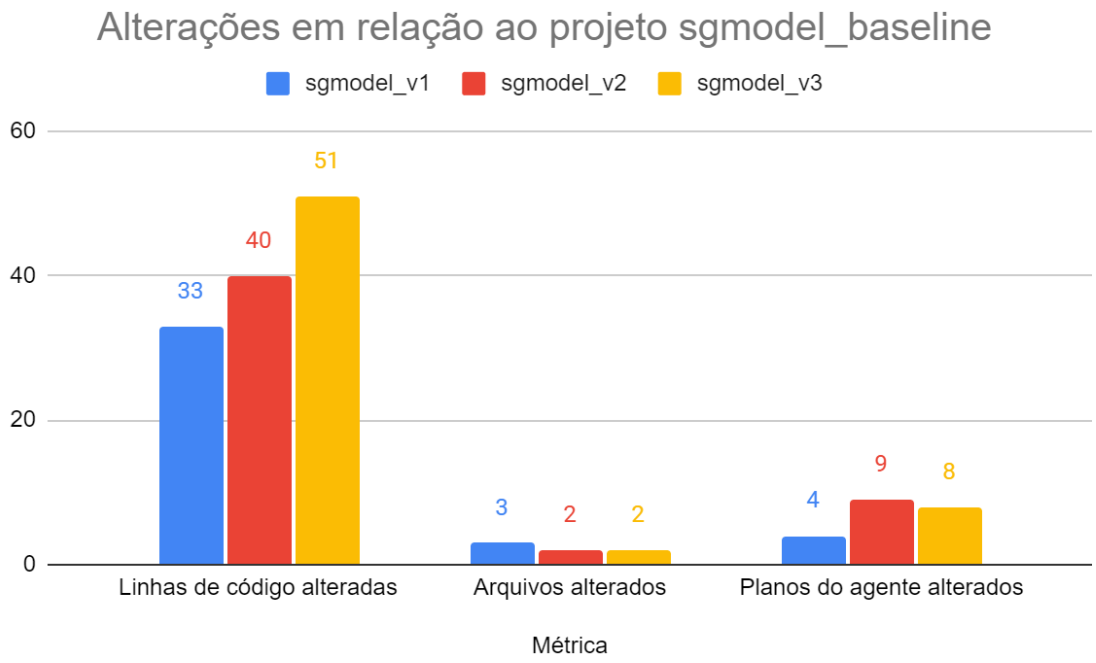
Neste modelo o desenvolvedor também possui a liberdade de implementar as interfaces em algum outro ponto do framework Jason. Para isso, ele precisa apenas implementar as interfaces *IGroundedPerceptTarget*, *IConfigurationProvider* no local onde acredita ser mais adequado ao seu projeto e realizar as configurações dos módulos, bem como a criação dos provedores de percepções embasadas. Assim como no modelo anterior, não é necessário realizar grandes alterações nos planos do agente, exceto aqueles onde se deseja considerar o embasamento simbólico.

Por fim, o projeto *sgmodel_v3*, que implementa o modelo da ação interna de embasamento simbólico mostra a flexibilidade fornecida. A utilização deste modelo não exige que o desenvolvedor utilize uma classe de ambiente nem de agente específicas ou modificadas. A utilização deste modelo demanda que o agente receba a configuração dos módulos de alguma forma. O experimento implementa através da adição de uma crença com a configuração gerada pelo ambiente. Desta forma, este experimento demandou mais alterações no código do agente e também a implementação da lógica que gera a configuração e envia para o agente por meio de uma percepção. A utilização deste modelo também se mostrou simples. A principal vantagem, como mencionado anteriormente, é a flexibilidade.

4.3.4.5 Métricas quantitativas

Por não se tratar de uma análise de performance ou de parâmetros puramente numéricos, a análise quantitativa dos modelos não é uma tarefa simples. Entretanto, visando colocar em números algumas perspectivas em relação aos modelos, foram definidas as três métricas apresentadas na seção 4.5.2. A Figura 18 contém um gráfico que mostra o resultado destas métricas. É importante destacar que todas as métricas foram calculadas comparando cada projeto do gráfico com o projeto *sgmodel_baseline* e que alterações relacionadas com simulações de mudança de percepções e também de troca das imagens a serem analisadas pelo módulo de detecção de objetos não foram consideradas. O principal objetivo em desconsiderar estas questões é quantificar as alterações que seriam realmente necessárias em um projeto real.

Figura 18 - Resultado das métricas quantitativas



Fonte: Elaborado pelo autor.

Em relação ao número de linhas alteradas, é possível verificar que o projeto *sgmodel_v3* foi o que obteve o maior valor. Grande parte das alterações deste experimento está concentrada no código do agente, visto que o mesmo executa duas vezes a ação interna (uma para cada percepção necessária). Porém, de modo geral, o número de alterações nas linhas é muito próximo entre os três experimentos.

Como os modelos se adaptam ao framework do Jason e a biblioteca já fornece grande parte da implementação necessária, os arquivos alterados são praticamente os mesmos, com exceção do primeiro experimento (*sgmodel_v1*) que exige a criação de um novo arquivo, com a classe do agente.

Já os planos do agente sofrem alterações distintas nos experimentos, porém em uma grandeza muito parecida. O experimento que mais exigiu alteração na lógica do agente foi o *sgmodel_v3* (apesar de não ser o de maior número de planos alterados). Isto porque este experimento demandou a criação de um plano para o recebimento das configurações e a alteração do plano de servir o café para a utilização da ação interna.

4.3.5 Discussão

Através da execução dos experimentos e da análise dos seus resultados, foi possível validar alguns pontos em relação aos modelos propostos neste trabalho. A Tabela 6 expõe um resumo dos resultados obtidos com a realização dos experimentos.

Tabela 6 - Resumo dos resultados das métricas verificadas para os experimentos

	sgmodel_v1	**sgmodel_v2**	**sgmodel_v3**
O modelo é capaz de embasar símbolos?	Sim	Sim	Sim
O quanto o modelo facilita a tarefa de fundamentação de símbolos?	Muito	Muito	Muito
O quão adequada foi esta implementação para o contexto do experimento?	Inadequado	Adequado	Inadequado
Qual é o nível de complexidade de utilização do modelo?	Baixo	Baixo	Baixo
Número de linhas de código alteradas	33	40	51
Número de arquivos alterados	3	2	2
Número de planos alterados ou incluídos no agente	4	9	8

Fonte: Elaborado pelo autor.

O modelo que se mostrou mais adequado ao contexto do experimento foi o do projeto *sgmodel_v2*, que implementa o uso de provedores de percepções embasadas. Isto porque, devido ao fato de que a tarefa do agente está diretamente ligada à detecção de objetos, e que o mesmo considera que seu objetivo só pode ser concluído quando suas percepções estão embasadas, os demais modelos se mostraram inadequados ao contexto, pois demandam neste caso uma fonte duplicada de percepções. Apesar disso, os demais modelos também realizam a tarefa de embasar os símbolos e facilitam este trabalho, com um baixo nível de dificuldade em sua utilização.

5 CONCLUSÃO

Este trabalho analisou o estado da arte em modelos para embasamento simbólico, com o objetivo de propor um modelo baseado em teoria de agentes BDI e modelos conexionistas para reconhecimento de objetos. Esta análise permitiu o contato com diversos problemas em aberto para resolver o problema do embasamento simbólico, que motivam as mais variadas abordagens para a tentativa de descoberta de uma solução para o mesmo.

Inspirado nas abordagens representacionalistas, este trabalho desenvolveu uma biblioteca que oferece uma estrutura de base para conexão dos símbolos manipulados por agentes BDI aos seus referentes no mundo real. A utilização de uma rede neural de detecção de objetos permitiu que objetos que podem ser detectados através de uma câmera sejam relacionados às crenças do agente BDI, realizando assim o seu embasamento. A estrutura desenvolvida permite a sua extensão de maneira simples e prática, necessitando apenas da implementação de novos provedores (sejam de embasamento ou de percepções embasadas) e suas devidas configurações. Para a utilização da rede neural de detecção de objetos, este trabalho desenvolveu dois provedores, que realizam a tarefa de conectar-se ao módulo de detecção de objetos e então realizar a tarefa do embasamento.

É importante ressaltar que existem algumas limitações e adaptações na conexão entre o módulo de detecção de objetos e o agente BDI. Devido ao fato de que o framework Jason possui algumas limitações sintáticas, foi necessário realizar transformações nas classes geradas pela detecção de objetos a fim de permitir seu uso nas crenças no agente. Por exemplo, tokens que iniciam com letra maiúscula no código do arquivo asl são considerados variáveis, portanto foi necessário garantir que a primeira letra da classe não seja maiúscula. Estes tratamentos foram realizados nos provedores, uma vez que a restrição se dá devido ao framework Jason. Além disso, a conexão e a troca de informações com o módulo também foi simplificada, visto que não se tratava do foco deste trabalho. Não foram realizados tratamentos de erros por falha de conexão e o recebimento das detecções se deu através da leitura de um arquivo em disco.

O desenvolvimento e a execução dos experimentos permitiu validar os modelos propostos e analisá-los em relação ao cenário definido. Todos os modelos se mostraram capazes de realizar o embasamento dos símbolos e facilitaram muito esta tarefa. O número de alterações necessárias no projeto de base foram muito parecidos entre os modelos, não sendo possível considerar tais critérios para escolha do modelo a ser utilizado. Entretanto, apenas o modelo de percepções embasadas se mostrou adequado ao contexto do experimento. Isto porque a tarefa do agente está diretamente ligada às percepções do ambiente, mais especificamente à detecção

dos objetos necessários e da pessoa a ser servida. Com isso, o mais adequado para este contexto é receber a percepção já embasada, ao invés de possuir uma fonte de percepções que não seja a própria detecção do ambiente.

Apesar de propor modelos para realizar o embasamento de símbolos de um agente BDI de maneira satisfatória, fornecendo uma biblioteca facilmente extensível e de fácil utilização, este trabalho não resolve o problema do embasamento simbólico. Isto porque a detecção de objetos é realizada por uma rede neural que foi previamente treinada com exemplos de imagens anotadas por seres humanos, ou seja, a fonte do conhecimento contido nesta rede é extrínseca.

Porém, mesmo sem oferecer uma solução absoluta para o problema, este trabalho contribui com a definição de um modelo que realiza uma parte importante do embasamento de símbolos e que pode ser facilmente estendido e evoluído. Além disso, este trabalho conseguiu atingir seus objetivos, na medida em que analisou o estado da arte em modelos para embasamento simbólico, detecção de objetos e agente BDI, propôs um modelo que integra partes do conhecimento do agente à uma respectiva representação conexionista, desenvolveu um protótipo do modelo e definiu e analisou os resultados de um experimento.

5.1 TRABALHOS FUTUROS

A biblioteca desenvolvida neste trabalho oferece diversas possibilidades de evolução. Trabalhos futuros podem desenvolver novos provedores de embasamento e de percepções embasadas, fornecendo outras fontes de embasamento para os símbolos. Um exemplo possível para esta evolução seria o desenvolvimento de um provedor de percepções embasadas através do reconhecimento de fala, o qual conectaria as palavras reconhecidas aos símbolos manipulados pelo agente.

Além de permitir a implementação de novos provedores, a biblioteca também permite o desenvolvimento de novas ações internas que se adequem a novos casos de uso. Seria possível, por exemplo, desenvolver uma ação interna que dispare um determinado evento somente no caso de uma determinada crença, passada por parâmetro, estar embasada. Esta ação facilitaria o controle do fluxo de execução dos planos do agente, ao mesmo tempo que manteria a liberdade do desenvolvedor.

Por utilizarem-se de uma conexão simplificada e limitada com o módulo de detecção de objetos, sugere-se que os provedores de embasamento e de percepções embasadas a partir da detecção de objetos sejam evoluídos, permitindo a troca direta de informações entre os módulos utilizando a conexão via socket, ao invés de realizar a leitura do arquivo JSON. Tal evolução

permitirá que o módulo de detecção de objetos esteja em um sistema remoto e também removerá a necessidade de espaço de armazenamento para o arquivo.

Além das possibilidades de extensão deste trabalho apresentadas anteriormente, sugere-se a evolução dos modelos através do desenvolvimento de uma nova estratégia de troca de informações entre os provedores e os agentes BDI. Além de embasar os símbolos, conectando-os aos seus referentes no mundo real, os provedores podem ser capazes de fornecer mais informações sobre tais referentes. Por exemplo, um robô que execute uma detecção de objetos e que, ao mesmo tempo, tenha informações de distância, tamanho, peso, cor, e tantas outras características e dados sobre os referentes no mundo real, poderia utilizar os provedores para fornecer tais informações ao agente, de maneira unificada e contextualizada. O agente, por sua vez, passa a ter um conhecimento maior sobre um determinado referente do mundo real, podendo tomar decisões mais precisas em direção ao seu objetivo.

Por fim, sugere-se o desenvolvimento de novos experimentos, com situações mais complexas e com diferentes objetivos para que o modelo possa ser avaliado de uma maneira mais ampla. A realização de novos testes, com agentes mais complexos, permitirão a expandir a validação do modelo em relação ao embasamento dos símbolos e também da sua flexibilidade.

REFERÊNCIAS

ALBAWI, Saad; MOHAMMED, Tareq Abed; AL-ZAWI, Saad. Understanding of a convolutional neural network. **2017 International Conference On Engineering And Technology (ICET)**, [S.L.], p. 1-6, ago. 2017. IEEE.

BORDINI, Rafael H.; HÜBNER, Jomi F.. BDI Agent Programming in AgentSpeak Using Jason. **Lecture Notes In Computer Science**, [S.L.], p. 143-164, 2006. Springer Berlin Heidelberg.

EICHSTAEDT, Leonardo Vailatti. **Problema de embasamento de símbolos em um sistema multi-contexto**. 2019. 138 f. TCC (Graduação) - Curso de Sistemas de Informação, Universidade Federal de Santa Catarina, Florianópolis, 2019.

HARNAD, S. The symbol grounding problem. Physica D: Nonlinear Phenomena, Elsevier Science, v. 42, 1990.

JIAO, Licheng; ZHANG, Fan; LIU, Fang; YANG, Shuyuan; LI, Lingling; FENG, Zhixi; QU, Rong. A Survey of Deep Learning-Based Object Detection. **IEEE Access**, [S.L.], v. 7, p. 128837-128868, 2019. Institute of Electrical and Electronics Engineers (IEEE).

LUGER, George F. **Artificial Intelligence**. New Mexico: Pearson Education, 2009. 779 p. ISBN 0321545893.

MA, Chih-Yao. **Toward grounded spatio-temporal reasoning**. 2020. 148 f. Tese (Doutorado) - Curso de Filosofia, Engenharia Elétrica e de Computação, Georgia Institute Of Technology, Atlanta, 2020.

MORENO, Marcio; CIVITARESE, Daniel; BRANDAO, Rafael; CERQUEIRA, Renato. Effective Integration of Symbolic and Connectionist Approaches through a Hybrid Representation. In: INTERNATIONAL WORKSHOP ON NEURAL-SYMBOLIC LEARNING AND REASONING, 14., 2019, Macao. **Proceedings [...]** . Macao: Ijcai, 2019. p. 76-78.

RAO, Anand s; GEORGEFF, Michael P.. BDI Agents: from theory to practice. In: INTERNATIONAL CONFERENCE ON MULTIAGENT SYSTEMS, 1., 1995, Melbourne. **Proceedings of the First International Conference on Multiagent Systems**. San Francisco: AAAI, 1995. p. 312-319.

RAUE, Federico; BYEON, Wonmin; BREUEL, Thomas M.; LIWICKI, Marcus. Symbol Grounding in Multimodal Sequences using Recurrent Neural Networks. In: COCO @ NIPS, 29., 2015, Montreal. **Proceedings of the NIPS Workshop on Cognitive Computation: Integrating Neural and Symbolic Approaches**. Montreal: Neural Information Processing Systems Foundation, 2015.

REN, Shaoqing; HE, Kaiming; GIRSHICK, Ross; SUN, Jian. Faster R-CNN: Towards real-time object detection with region proposal networks. **IEEE Transactions On Pattern Analysis And Machine Intelligence**, [S.L.], v. 39, n. 6, p. 1137-1149, 1 jun. 2017. Institute of Electrical and Electronics Engineers (IEEE).

RUSSELL, Stuart J.; NORVIG, Peter. **Inteligência artificial**. Rio de Janeiro: Elsevier, Campus, 2004. 1021 p. ISBN 8535211772.

SEARLE, J. R. Minds, brains, and programs. Behavioral and Brain Sciences, Cambridge University Press, v. 3, 9 1980.

SILVA, Lavindra de; MENEGUZZI, Felipe; LOGAN, Brian. BDI Agent Architectures: a survey. **Proceedings Of The Twenty-Ninth International Joint Conference On Artificial Intelligence**, [S.L.], p. 4914-4921, jul. 2020. International Joint Conferences on Artificial Intelligence Organization.

SPANGENBERG, Michael; HENRICH, Dominik. Symbol grounding for symbolic robot commands based on physical properties. **2016 IEEE International Conference On Information And Automation (ICIA)**, [S.L.], p. 62-68, ago. 2016.

TADDEO, Mariarosaria; FLORIDI, Luciano. Solving the symbol grounding problem: a critical review of fifteen years of research. **Journal Of Experimental & Theoretical Artificial Intelligence**, [S.L.], v. 17, n. 4, p. 419-445, dez. 2005. Informa UK Limited.

WOOLDRIDGE, Michael. **An Introduction to MultiAgent Systems**. Liverpool: John Wiley And Sons Ltd, 2002.

WOOLDRIDGE, Michael; JENNINGS, Nicholas R.. Intelligent Agents: theory and practice. **The Knowledge Engineering Review**. Cambridge, p. 115-152. nov. 1995.

ZHANG, Xin; YANG, Yee-Hong; HAN, Zhiguang; WANG, Hui; GAO, Chao. Object class detection. **ACM Computing Surveys**, [S.L.], v. 46, n. 1, p. 1-53, out. 2013. Association for Computing Machinery (ACM).

APÊNDICE A - Código fonte dos testes iniciais com Jason

A.1 Código fonte do agente

```
/* Initial beliefs and rules */
sabeSeEhCortavelOuNao(Objeto) :- ehCortavel(Objeto) | naoEhCortavel(Objeto).
sabeSeEhCortanteOuNao(Objeto) :- ehCortante(Objeto) | naoEhCortante(Objeto).
sabeSeEhCortavelPorOuNao(ObjCortavel, ObjCortante) :- ehCortavelPor(ObjCortavel, ObjCortante) |
                                                      naoEhCortavelPor(objCortavel, ObjCortante).

ehDesconhecido(Objeto) :- not sabeSeEhCortavelOuNao(Objeto) &
                          not sabeSeEhCortanteOuNao(Objeto).
possuiObjetoCortante(Objeto) :- possui(Objeto) & ehCortante(Objeto).
possuiObjetoCortavel(Objeto) :- possui(Objeto) & ehCortavel(Objeto).

podeCortar(ObjetoCortavel, ObjetoCortante) :- possuiObjetoCortante(ObjetoCortante) &
                                              possuiObjetoCortavel(ObjetoCortavel) &
                                              ehCortavelPor(ObjetoCortavel, ObjetoCortante).

/* Initial goals */
!cortarObjetos.

/* Plans */
+possui(Objeto) : ehDesconhecido(Objeto) <- perguntarSeEhCortavel(Objeto)
                                            perguntarSeEhCortante(Objeto).

+possui(Objeto) : not sabeSeEhCortavelOuNao(Objeto) <- perguntarSeEhCortavel(Objeto).

+possui(Objeto) : not sabeSeEhCortanteOuNao(Objeto) <- perguntarSeEhCortante(Objeto).

+ehCortavel(Objeto) <- !cortarObjetos.

+naoEhCortavel(Objeto).

+ehCortante(Objeto) <- !cortarObjetos.

+naoEhCortante(Objeto).

+ehCortavelPor(ObjetoCortavel, ObjetoCortante).

+naoEhCortavelPor(ObjCortavel, ObjetoCortante).

+!cortar(ObjetoCortavel, ObjetoCortante) : podeCortar(ObjetoCortavel, ObjetoCortante)
                                          <- cortar(ObjetoCortavel, ObjetoCortante)
                                             .abolish(possui(ObjetoCortavel)).

-!cortar(ObjetoCortavel, ObjetoCortante) : not possui(ObjetoCortavel) <- solicitarObjeto(ObjetoCortavel).

-!cortar(ObjetoCortavel, ObjetoCortante) : not possui(ObjetoCortante) <- solicitarObjeto(ObjetoCortante).

-!cortar(ObjetoCortavel, ObjetoCortante) : naoEhCortavelPor(ObjetoCortavel, ObjetoCortante)
                                          <- solicitarObjetoCortante(ObjetoCortavel, ObjetoCortante).

-!cortar(ObjetoCortavel, ObjetoCortante) : not sabeSeEhCortavelPorOuNao(ObjetoCortavel, ObjetoCortante)
                                          <- perguntarSeEhCortavelPor(ObjetoCortavel, ObjetoCortante).

+!cortarObjetos <- .findall(O, possuiObjetoCortante(O), L1)
                   for (.member(ObjCortante, L1)) {
```

```
                .findall(O, possuiObjetoCortavel(O), L2)
                for (.member(ObjCortavel, L2)) {
                    !cortar(ObjCortavel, ObjCortante)
                }
        }.
```

A.2 Código fonte do ambiente

```
// Environment code for project grounded_agent
package grounded_agent;

import org.json.*;

import jason.asSyntax.*;
import jason.environment.*;

import java.io.File;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.util.ArrayList;
import java.util.List;
import java.util.Timer;
import java.util.TimerTask;
import java.util.logging.*;

public class Ambiente extends Environment {

        private Timer timer;
        private int seconds = 5;
        private Logger logger = Logger.getLogger("grounded_agent." + Ambiente.class.getName());

    /** Called before the MAS execution with the args informed in .mas2j */
    @Override
    public void init(String[] args) {
        super.init(args);
        this.timer = new Timer();
        this.timer.schedule(new UpdateTask(), 0, seconds * 1000);
    }

    @Override
    public boolean executeAction(String nomeDoAgente, Structure acao) {
      String nomeDaAcao = acao.getFunctor();
      if (nomeDaAcao.equals("solicitarObjeto")) {
            String objetoSolicitado = acao.getTerm(0).toString();
            logger.info("O agente solicitou um objeto: " + objetoSolicitado);
                return true;
            } else if (nomeDaAcao.equals("cortar")) {
                String objetoCortavel = acao.getTerm(0).toString();
                String objetoCortante = acao.getTerm(1).toString();
                logger.info("O agente cortou " + objetoCortavel + " com " + objetoCortante);
                return true;
            } else if (nomeDaAcao.equals("perguntarSeEhCortante")) {
                String objetoQuestionado = acao.getTerm(0).toString();
                logger.info("O agente perguntou: o objeto \"" + objetoQuestionado + "\" é cortante?");
                responderSeEhCortante(objetoQuestionado);
                return true;
            } else if (nomeDaAcao.equals("perguntarSeEhCortavel")) {
```

```
                String objetoQuestionado = acao.getTerm(0).toString();
                logger.info("O agente perguntou: o objeto \"" + objetoQuestionado + "\" é cortável?");
                responderSeEhCortavel(objetoQuestionado);
                return true;
        } else if (nomeDaAcao.equals("solicitarObjetoCortante")) {
                String objCortavel = acao.getTerm(0).toString();
                String objCortante = acao.getTerm(1).toString();
                logger.info("O agente disse: O objeto " + objCortavel + " não é cortável por " +
                        objCortante + ".\nPreciso de um objeto para cortar " + objCortavel);
                return true;
        } else if (nomeDaAcao.equals("perguntarSeEhCortavelPor")) {
                String objCortavel = acao.getTerm(0).toString();
                String objCortante = acao.getTerm(1).toString();
                logger.info("O agente perguntou: o objeto \"" + objCortavel +
                        "\" é cortável por \"" + objCortante + "\"?");
                responderSeEhCortavelPor(objCortavel, objCortante);
                return true;
        }

        logger.info("executando: " + acao + ", mas não foi implementado!");
        return true;
}

private void responderSeEhCortante(String objeto) {
  String crenca = "naoEhCortante";
  if (objeto.equals("tesoura") || objeto.equals("estilete")) {
        crenca = "ehCortante";
  }
  String txtNovaCrenca = crenca + "(" + objeto + ")";
  Literal novaCrenca = Literal.parseLiteral(txtNovaCrenca);
  logger.info("Respondendo: " + txtNovaCrenca);
  addPercept(novaCrenca);
}

private void responderSeEhCortavel(String objeto) {
  String crenca = "naoEhCortavel";
  if (objeto.equals("papel") || objeto.equals("cartolina")) {
        crenca = "ehCortavel";
  }
  String txtNovaCrenca = crenca + "(" + objeto + ")";
  Literal novaCrenca = Literal.parseLiteral(txtNovaCrenca);
  logger.info("Respondendo: " + txtNovaCrenca);
  addPercept(novaCrenca);
}

private void responderSeEhCortavelPor(String objCortavel, String objCortante) {
  String crenca = "naoEhCortavelPor";
  if (objCortavel.equals("papel")) {
        if (objCortante.equals("tesoura") || objCortante.equals("estilete")) {
                crenca = "ehCortavelPor";
        }
  }
  String txtNovaCrenca = crenca + "(" + objCortavel + "," + objCortante + ")";
  Literal novaCrenca = Literal.parseLiteral(txtNovaCrenca);
  logger.info("Respondendo: " + txtNovaCrenca);
  addPercept(novaCrenca);
}

/** Called before the end of MAS execution */
@Override
```

```
  public void stop() {
        super.stop();
        this.timer.cancel();
  }

  private void removerPercepcoesDeObjetosNaoDetectados(ArrayList<Deteccao> novasDeteccoes,
List<Literal> percepcoesAtuais) {
    for (Literal percepcao : percepcoesAtuais) {
                if (percepcao.getFunctor().equals("possui")) {
                if (!verificarSeClasseExiste(novasDeteccoes, percepcao.getTerm(0).toString())) {
                        removePercept(percepcao);
                }
                }
        }
  }

  private void adicionarPercepcoesDeNovosObjetosDetectados(ArrayList<Deteccao> novasDeteccoes,
List<Literal> percepcoesAtuais) {
    for (Deteccao deteccao : novasDeteccoes) {
                if (!verificarSeClasseExiste(percepcoesAtuais, deteccao.getClasse().getNome())) {
                Literal possui = Literal.parseLiteral("possui(" + deteccao.getClasse().getNome() + ")");
                addPercept(possui);
                }
        }
  }

  private boolean verificarSeClasseExiste(List<Literal> percepcoes, String nomeDaClasse) {
    for (Literal percepcao : percepcoes) {
        if (percepcao.getFunctor().equals("possui")) {
                if (percepcao.getTerm(0).toString().equals(nomeDaClasse)) {
                return true;
                }
        }
        }
    return false;
  }

  private boolean verificarSeClasseExiste(ArrayList<Deteccao> deteccoes, String nomeDaClasse) {
    for (Deteccao deteccao : deteccoes) {
                if (deteccao.getClasse().getNome().equals(nomeDaClasse)) {
                return true;
                }
        }
    return false;
  }

  private void atualizarPercepcoes() {
    ArrayList<Deteccao> novasDeteccoes = lerDeteccoes();
    List<Literal> percepcoesAtuais = consultPercepts("agent1");
    removerPercepcoesDeObjetosNaoDetectados(novasDeteccoes, percepcoesAtuais);
    adicionarPercepcoesDeNovosObjetosDetectados(novasDeteccoes, percepcoesAtuais);
  }

  private ArrayList<Deteccao> lerDeteccoes() {
    try {
        ArrayList<Deteccao> deteccoes = new ArrayList<Deteccao>();
        Path filePath = new File("C:\\UFSC\\TCC\\detection_results\\agent.perception.json").toPath();
        String json = new String(Files.readAllBytes(filePath));
                JSONArray jsonDeteccoes = new JSONArray(json);
                if (jsonDeteccoes.length() > 0) {
```

```
                    for (int i = 0; i < jsonDeteccoes.length(); i++) {
                            JSONObject jsonObjeto = jsonDeteccoes.getJSONObject(i);
                            JSONObject jsonClasseDoObjeto = jsonObjeto.getJSONObject("class");
                            int id = jsonClasseDoObjeto.getInt("id");
                            String name = jsonClasseDoObjeto.getString("name");
                            ClasseDeObjeto classe = new ClasseDeObjeto(id, name);
                            Deteccao deteccao = new Deteccao(jsonObjeto.getFloat("score"), classe);
                            if (deteccao.getScore() >= 0.5) {
                            deteccoes.add(deteccao);
                            }
                    }
                    }
                    return deteccoes;
            } catch (IOException e) {
                    e.printStackTrace();
                    return null;
            }
    }

    private class UpdateTask extends TimerTask {
            @Override
            public void run() {
                    atualizarPercepcoes();
            }
    }

}
```

A.3 Classe de objeto

```
package grounded_agent;

public class ClasseDeObjeto {

        private int id;
        private String nome;

        public ClasseDeObjeto(int id, String nome) {
        this.id = id;
        this.nome = nome;
        }

        public int getId() {
        return this.id;
        }

        public String getNome() {
        return this.nome;
        }

}
```

A.4 Classe detecção

```
package grounded_agent;

public class Deteccao {
```

```
        private float score;
        private ClasseDeObjeto classe;

        public Deteccao(float score, ClasseDeObjeto classe) {
        this.score = score;
        this.classe = classe;
        }

        public float getScore() {
        return this.score;
        }

        public ClasseDeObjeto getClasse() {
        return this.classe;
        }

}
```

A.5 Arquivo .mas2j

```
MAS grounded_agent {

        infrastructure: Centralised

        environment: grounded_agent.Ambiente

        agents:
        agent1 card_maker_agent;

        aslSourcePath:
        "src/asl";
}
```

APÊNDICE B - Código fonte do módulo de detecção de objetos

B.1 Arquivo detection.py

```
import _thread
import socket
import hashlib
import os
import pathlib
import numpy as np
import six.moves.urllib as urllib
import sys
import tarfile
import tensorflow as tf
import zipfile
import json

from collections import defaultdict
from io import StringIO
from matplotlib import pyplot as plt
from PIL import Image
from IPython.display import display
from random import randint

from object_detection.utils import ops as utils_ops
from object_detection.utils import label_map_util
from object_detection.utils import visualization_utils as vis_util

def sendAnswer(msg):
    tcp = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    tcp.connect(('localhost', 49999))
    tcp.send(msg)
    tcp.close()

class ObjectDetection:

    def __init__(self):
        print('ObjectDetection init started')
        utils_ops.tf = tf.compat.v1
        tf.gfile = tf.io.gfile

        self.detectionsByImageHash = {}
        self.lastImagesHashes = {}

        dir_path = os.path.dirname(os.path.realpath(__file__))
        configFilePath = dir_path + "/config.json"
        print("Loading configuration from " + configFilePath)
        with open(configFilePath) as json_file:
            self.config = json.load(json_file)

        self.PATH_TO_LABELS = self.config["PATH_TO_LABELS"]
        print("PATH_TO_LABELS: " + self.PATH_TO_LABELS)

        self.PATH_TO_TEST_IMAGES_DIR =
pathlib.Path(self.config["PATH_TO_TEST_IMAGES_DIR"])
```

```
        print("PATH_TO_TEST_IMAGES_DIR: " + self.config["PATH_TO_TEST_IMAGES_DIR"])

        self.model_name = self.config["MODEL_NAME"]
        print("MODEL  NAME: " + self.model  name)

        print('loading category index...')
        try:
            self.category_index =
label  map  util.create  category  index  from  labelmap(self.PATH  TO  LABELS,
use  display  name=True)
            print('category index loaded')
        except Exception as e:
            print('Failed loading category index', e)

        self.detection_model = None

    def load_model(self):
        print("loading model...")
        if self.detection  model != None:
            return

        base_url = self.config["MODEL_BASE_URL"]
        print("BASE  URL: " + base  url)

        model_file = self.model_name + '.tar.gz'
        model_origin = base_url + model_file
        print("Loading model from: " + model_origin)

        model  dir = tf.keras.utils.get  file(fname=self.model  name, origin=model  origin, untar=True)

        model_dir = pathlib.Path(model_dir)/"saved_model"

        model = tf.saved  model.load(str(model  dir))
        model = model.signatures['serving  default']

        self.detection_model = model
        print("model loaded.")


    def run_inference_for_single_image(self, model, image):
        image = np.asarray(image)
        # The input needs to be a tensor, convert it using `tf.convert_to_tensor`.
        input  tensor = tf.convert  to  tensor(image)
        # The model expects a batch of images, so add an axis with `tf.newaxis`.
        input_tensor = input_tensor[tf.newaxis,...]

        # Run inference
        output_dict = model(input_tensor)

        #print("Printing output_dict...")
        #print(output_dict)

        # All outputs are batches tensors.
        # Convert to numpy arrays, and take index [0] to remove the batch dimension.
        # We're only interested in the first num_detections.
        num_detections = int(output_dict.pop('num_detections'))
        output_dict = {key:value[0, :num_detections].numpy()
```

```
            for key,value in output_dict.items()}
    output_dict['num_detections'] = num_detections

    # detection_classes should be ints.
    output_dict['detection_classes'] = output_dict['detection_classes'].astype(np.int64)

    # Handle models with masks:
    if 'detection_masks' in output_dict:
      # Reframe the the bbox mask to the image size.
      detection_masks_reframed = utils_ops.reframe_box_masks_to_image_masks(
          output_dict['detection_masks'], output_dict['detection_boxes'],
          image.shape[0], image.shape[1])
      detection_masks_reframed = tf.cast(detection_masks_reframed > 0.5,
                                         tf.uint8)
      output_dict['detection_masks_reframed'] = detection_masks_reframed.numpy()

    return output_dict

def deleteFile(self, file):
    if os.path.exists(file):
      os.remove(file)

def show_inference(self, model, image_path, basePath, fileNumber):
    print("running detection...")
    # the array based representation of the image will be used later in order to prepare the
    # result image with boxes and labels on it.
    image_np = np.array(Image.open(image_path))
    # Actual detection.
    output_dict = self.run_inference_for_single_image(model, image_np)
    # Visualization of the results of a detection.

    vis_util.visualize_boxes_and_labels_on_image_array(
      image_np,
      output_dict['detection_boxes'],
      output_dict['detection_classes'],
      output_dict['detection_scores'],
      self.category_index,
      instance_masks=output_dict.get('detection_masks_reframed', None),
      use_normalized_coordinates=True,
      line_thickness=8)

    currentDetectionJson = []
    for i in range(output_dict['num_detections']):
      currentDetectionJson.append({
        'percept': 'detected',
        'score': output_dict['detection_scores'][i].astype(float),
        'class': self.category_index[output_dict['detection_classes'][i]],
        'box': output_dict['detection_boxes'][i].tolist()
      })

    imageResult = Image.fromarray(image_np)
    jsonName = basePath + f"agent.perception{fileNumber}.json"
    imageName = basePath + f"agent.perception{fileNumber}.jpg"

    self.deleteFile(jsonName)
    self.deleteFile(imageName)
```

```
        f = open(jsonName, "w")
        json.dump(currentDetectionJson, f, indent=4)
        f.close()

        imageResult.save(imageName, "JPEG")
        print("detection result saved.")
        return currentDetectionJson

    def calculateHash(self, inputFile):
        BUF  SIZE = 65536

        md5 = hashlib.md5()
        sha1 = hashlib.sha1()

        with open(inputFile, 'rb') as f:
            while True:
                data = f.read(BUF_SIZE)
                if not data:
                    break
                md5.update(data)
                sha1.update(data)

        return "{0}".format(md5.hexdigest())

    def listChangedImages(self):
        imagesPaths = list(map(lambda i: {"path": i, "hash": None},
sorted(list(self.PATH_TO_TEST_IMAGES_DIR.glob("*.jpg")))))
        newHashes = {}
        shouldDetect = False

        for img in imagesPaths:
            imgPath = img["path"]
            hash = self.calculateHash(imgPath)
            img["hash"] = hash
            newHashes[imgPath] = hash
            if (imgPath not in self.lastImagesHashes) or (self.lastImagesHashes[imgPath] != hash):
                # if there is at least one file to detect, run detection to all images for the sake of simplicity
                shouldDetect = True

        self.lastImagesHashes = newHashes

        if shouldDetect:
            return imagesPaths
        else:
            return []

    def detect(self, shouldAnswer):
        self.load_model()

        print('finding changed test images...')
        testImagePaths = self.listChangedImages()
        qtd = len(testImagePaths)
        print(f'{qtd} test images found')
        answerMsg = b'unchanged'
        if qtd > 0:

            basePath = self.config["RESULT_BASE_PATH"] + "/"
```

```
        if not os.path.exists(basePath):
            os.mkdir(basePath)

        detections = []

        for image in testImagePaths:
            image_path = image["path"]
            hash = image["hash"]
            if hash in self.detectionsByImageHash:
                detections.append(self.detectionsByImageHash[hash]["detection_result"])
            else:
                detection_result = self.show_inference(self.detection_model, image_path, basePath,
len(detections))
                self.detectionsByImageHash[hash] = { "hash": hash, "path": image["path"], "detection_result":
detection_result }
                detections.append(detection_result)

        finalDetectionResult = sum(detections, [])

        jsonName = basePath + f"agent.perception.json"

        f = open(jsonName, "w")
        json.dump(finalDetectionResult, f, indent=4)
        f.close()

        answerMsg = b'ok'

    if shouldAnswer:
        _thread.start_new_thread(sendAnswer, tuple([answerMsg]))
```

B.2 Arquivo server.py

```
import _thread
from detection import ObjectDetection
import socket

detection = ObjectDetection()
detection.detect(False)

HOST = 'localhost'    # Endereco IP do Servidor
PORT = 50000          # Porta que o Servidor esta

def client_connected(con, client):
    print('Connected by', client)

    command = ""
    while True:
        msg = con.recv(1024)
        if not msg: break
        command += msg.decode("UTF-8")


    print('Command received: ', command)
    if command == 'detect':
```

```
            print('detection started')
            detection.detect(True)

        print('Closing client connection...', client)
        con.close()
        print('Client connection closed.')
        _thread.exit()

print('Creating the socket...')
tcp = socket.socket(socket.AF  INET, socket.SOCK  STREAM)

orig = (HOST, PORT)

print('Binding socket')
tcp.bind(orig)
tcp.listen(1)

while True:
    print('Waiting for new connection...')
    con, client = tcp.accept()
    _thread.start_new_thread(client_connected, tuple([con, client]))

tcp.close()
```

APÊNDICE C - Código fonte da biblioteca sglib

C.1 Classe sglib.agents.BaseGroundedAgents

```
package sglib.agents;

import java.util.ArrayList;
import java.util.Collection;
import java.util.List;
import java.util.logging.Logger;

import jason.asSemantics.Agent;
import jason.asSyntax.Literal;
import sglib.configuration.SGLibConfiguration;
import sglib.configuration.SGLibModule;
import sglib.providers.IConfigurationProvider;
import sglib.providers.IGroundProvider;

public abstract class BaseGroundedAgent extends Agent implements IConfigurationProvider {

	private static final long serialVersionUID = 1L;

	private Logger logger = Logger.getLogger("sglib.agents." + getClass().getName());
	protected ArrayList<IGroundProvider> groundProviders;
	private SGLibConfiguration configuration;

	public BaseGroundedAgent() {
	super();
	this.configuration = this.createSGLibConfiguration();
	this.loadGroundProviders();
	this.setProvidersConfiguration();
	}

	private void setProvidersConfiguration() {
	for (IGroundProvider provider : groundProviders) {
		provider.setConfiguration(this);
	}
	}

	@Override
	public SGLibModule getModule(String name) {
	return this.configuration.getModule(name);
	}

	protected abstract SGLibConfiguration createSGLibConfiguration();

	protected abstract ArrayList<IGroundProvider> createGroundProviders();

	private void loadGroundProviders() {
	this.groundProviders = this.createGroundProviders();
	if (this.groundProviders == null) {
		this.groundProviders = new ArrayList<IGroundProvider>();
	}
	}

	protected List<Literal> ground(List<Literal> percepts) {
	logger.info("ground() called. " + percepts.size() + " percepts received. Calling providers...");
	for (IGroundProvider provider : this.groundProviders) {
```

```
                percepts = provider.groundBeliefs(percepts);
        }
        return percepts;
        }

        @Override
        public int buf(Collection<Literal> percepts) {
        if (percepts != null && !percepts.isEmpty()) {
                percepts = this.ground(new ArrayList<Literal>(percepts));
        }

        return super.buf(percepts);
        }

        @Override
        public void initAg() {
        // TODO Auto-generated method stub
        super.initAg();
        }

        @Override
        public void stopAg() {
        // TODO Auto-generated method stub
        super.stopAg();
        }

}
```

C.2 Classe sglib.configuration.SGLibConfiguration

```
package sglib.configuration;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;
import java.util.logging.Logger;

import org.json.JSONArray;
import org.json.JSONObject;

import sglib.providers.objectDetection.ObjectDetectionGroundProvider;

public class SGLibConfiguration {

        public final static String PROVIDERS_CONFIGURATION_KEY = "providers";
        public final static String MODULES_CONFIGURATION_KEY = "modules";

        private ArrayList<String> providers = new ArrayList<>();
        private Map<String, SGLibModule> modules = new HashMap<>();
        private Logger logger = Logger.getLogger("sglib." +
ObjectDetectionGroundProvider.class.getName());

        public SGLibConfiguration(String json) {
        JSONObject jsonObj = new JSONObject(json);

        try {
                if (jsonObj.has(PROVIDERS_CONFIGURATION_KEY)) {
                JSONArray jsonProviders =
jsonObj.getJSONArray(PROVIDERS_CONFIGURATION_KEY);
                for (int i = 0; i < jsonProviders.length(); i++) {
```

```
                providers.add(jsonProviders.getString(i));
            }
            }
        } catch (Exception e) {
            logger.severe("Error reading providers: " + e.getMessage());
            throw e;
        }

        try {
            if (jsonObj.has(MODULES_CONFIGURATION_KEY)) {
            JSONArray jsonModules = jsonObj.getJSONArray(MODULES_CONFIGURATION_KEY);
            for (int i = 0; i < jsonModules.length(); i++) {
                JSONObject jsonModule = jsonModules.getJSONObject(i);
                SGLibModule module = new SGLibModule();
                module.setName(jsonModule.getString(SGLibModule.MODULE_NAME_KEY));

                for (String key : jsonModule.keySet()) {
                if (!key.equals(SGLibModule.MODULE_NAME_KEY)) {
                module.setConfiguration(key, jsonModule.get(key));
                }
                }

                modules.put(module.getName(), module);
            }
            }
        } catch (Exception e) {
            logger.severe("Error reading modules: " + e.getMessage());
            throw e;
        }
        }

        public ArrayList<String> getProviders() {
        return this.providers;
        }

        public SGLibModule getModule(String name) {
        return modules.get(name);
        }
}
```

C.3 Classe sglib.configuration.SGLibConfigurationBuilder

```
package sglib.configuration;

import java.util.ArrayList;
import java.util.Map;

import org.json.JSONArray;
import org.json.JSONObject;

public class SGLibConfigurationBuilder {

        public ArrayList<SGLibModule> modules = new ArrayList<>();
        public ArrayList<String> providers = new ArrayList<>();

        public SGLibConfigurationBuilder() {

        }
```

```
        public SGLibConfigurationBuilder withProvider(String providerName) {
        providers.add(providerName);
        return this;
        }

        public SGLibConfigurationBuilder withModule(SGLibModule module) {
        modules.add(module);
        return this;
        }

        public String toJson() {
        JSONObject obj = new JSONObject();

        JSONArray arrModules = new JSONArray();
        for (SGLibModule module : this.modules) {
                JSONObject objModule = new JSONObject();
                objModule.put(SGLibModule.MODULE_NAME_KEY, module.getName());
                Map<String, Object> configurations = module.getConfigurations();
                for (String configKey : configurations.keySet()) {
                objModule.put(configKey, configurations.get(configKey));
                }
                arrModules.put(objModule);
        }

        obj.put(SGLibConfiguration.PROVIDERS_CONFIGURATION_KEY, new
JSONArray(this.providers));
        obj.put(SGLibConfiguration.MODULES_CONFIGURATION_KEY, arrModules);

        return obj.toString();
        }

        public SGLibConfiguration build() {
        return new SGLibConfiguration(this.toJson());
        }

}
```

C.4 Classe sglib.configuration.SGLibModule

```
package sglib.configuration;

import java.util.HashMap;
import java.util.Map;

public class SGLibModule {

        public static final String MODULE_NAME_KEY = "name";

        private String name;
        private HashMap<String, Object> configurations;

        public SGLibModule() {
        this.configurations = new HashMap<String, Object>();
        }

        public String getName() {
        return name;
        }

        public void setName(String name) {
```

```
        this.name = name;
        }

        public Map<String, Object> getConfigurations() {
        return this.configurations;
        }

        public Object getConfiguration(String key, Object defaultValue) {
        return configurations.getOrDefault(key, defaultValue);
        }

        public void setConfiguration(String key, Object value) {
        this.configurations.put(key, value);
        }

}
```

C.5 Classe sglib.environments.BaseGroundedPerceptTargetEnvironment

```
package sglib.environments;

import java.util.ArrayList;
import java.util.List;
import java.util.logging.Logger;

import jason.environment.Environment;
import sglib.configuration.SGLibConfiguration;
import sglib.configuration.SGLibModule;
import sglib.percepts.GroundedPercept;
import sglib.providers.IConfigurationProvider;
import sglib.providers.IGroundedPerceptProvider;
import sglib.providers.IGroundedPerceptTarget;

public abstract class BaseGroundedPerceptTargetEnvironment extends Environment
        implements IGroundedPerceptTarget, IConfigurationProvider {

        protected SGLibConfiguration configuration;
        protected Logger logger = Logger.getLogger("sgmodel_v2." + this.getClass().getName());
        protected ArrayList<IGroundedPerceptProvider> providers = new ArrayList<>();

        /** Called before the MAS execution with the args informed in .mas2j */
        @Override
        public void init(String[] args) {
        super.init(args);
        this.configuration = this.createSGLibConfiguration();
        this.loadProvidersAndRegister();
        this.setProvidersConfiguration();
        this.startProviders();
        }

        private void setProvidersConfiguration() {
        for (IGroundedPerceptProvider provider : providers) {
                provider.setConfiguration(this);
        }
        }

        protected abstract SGLibConfiguration createSGLibConfiguration();

        protected abstract void loadProvidersAndRegister();
```

```
        protected void startProviders() {
        for (IGroundedPerceptProvider provider : this.providers) {
                provider.start();
        }
        }

        protected void stopProviders() {
        for (IGroundedPerceptProvider provider : this.providers) {
                provider.stop();
        }
        }

        @Override
        public SGLibModule getModule(String name) {
        return this.configuration.getModule(name);
        }

        /** Called before the end of MAS execution */
        @Override
        public void stop() {
        this.stopProviders();
        super.stop();
        }

        @Override
        public void addGroundedPercepts(List<GroundedPercept> percepts) {
        logger.info("Environment received " + percepts.size() + " percepts from provider to be added");
        for (GroundedPercept percept : percepts) {
                if (percept.forSpecificAgent())
                addPercept(percept.getAgentName(), percept.getPercept());
                else
                addPercept(percept.getPercept());
        }
        }

        @Override
        public void removeGroundedPercepts(List<GroundedPercept> percepts) {
        logger.info("Environment received " + percepts.size() + " percepts from provider to be removed");
        for (GroundedPercept percept : percepts) {
                logger.info("Environment removing percept: " + percept.getPercept().toString());
                if (percept.forSpecificAgent())
                removePercept(percept.getAgentName(), percept.getPercept());
                else
                removePercept(percept.getPercept());
        }
        }

}
```

C.6 Classe sglib.percepts.GroundedPercept

```
package sglib.percepts;

import jason.asSyntax.Literal;

public class GroundedPercept {

	private Literal percept;
	private String agentName;

	public GroundedPercept(String agentName, Literal percept) {
	this.agentName = agentName;
	this.percept = percept;
	}

	public Literal getPercept() {
	return percept;
	}

	public String getAgentName() {
	return agentName;
	}

	public boolean forSpecificAgent() {
	return this.agentName != null && !this.agentName.isEmpty();
	}

}
```

C.7 Classe sglib.providers.objectDetection.DetectionExecutionResult

```
package sglib.providers.objectDetection;

public class DetectionExecutionResult {

	private DetectionExecutionResultCode resultCode;
	private String message;

	public DetectionExecutionResult() {
	}

	public DetectionExecutionResult(DetectionExecutionResultCode resultCode, String message) {
		this.resultCode = resultCode;
		this.message = message;
	}

	public DetectionExecutionResultCode getResultCode() {
		return resultCode;
	}

	public void setResultCode(DetectionExecutionResultCode resultCode) {
		this.resultCode = resultCode;
	}

	public String getMessage() {
		return message;
	}
```

```
	public void setMessage(String message) {
		this.message = message;
	}
}
```

C.8 Enumeração DetectionExecutionResultCode

```
package sglib.providers.objectDetection;

public enum DetectionExecutionResultCode {
	OK, Unchanged, Error, Other
}
```

C.9 sglib.providers.objectDetection.ObjectDetectionExecutor

```
package sglib.providers.objectDetection;

import java.io.IOException;
import java.io.OutputStream;
import java.net.ServerSocket;
import java.net.Socket;
import java.nio.charset.Charset;
import java.util.Scanner;
import java.util.logging.Logger;

import sglib.configuration.SGLibModule;

public class ObjectDetectionExecutor {

	private SGLibModule module;

	public ObjectDetectionExecutor() {
	}

	public void configure(SGLibModule module) {
	this.module = module;
	}

	public DetectionExecutionResult execute(Logger logger) {
	logger.info("ObjectDetectionExecutor.execute() called");

	DetectionExecutionResult result = new
DetectionExecutionResult(DetectionExecutionResultCode.Other, "");
	try {
		sendDetectCommand(logger);

		String response = waitAndReadResponse(logger);

		if (response.equals("ok")) {
		result.setResultCode(DetectionExecutionResultCode.OK);
		result.setMessage("detection returned: ok.");
		logger.info(result.getMessage());
		} else if (response.equals("unchanged")) {
		result.setResultCode(DetectionExecutionResultCode.Unchanged);
		result.setMessage("detection returned: unchanged.");
		logger.info(result.getMessage());
		} else {
```

```
                result.setResultCode(DetectionExecutionResultCode.Other);
                result.setMessage("detection do not returned ok. Answer: " + response);
                logger.info(result.getMessage());
                }

        } catch (IOException e) {
                e.printStackTrace();
                result.setResultCode(DetectionExecutionResultCode.Error);
                result.setMessage(e.getMessage());
                logger.severe(result.getMessage());
        }

        logger.info("end of ObjectDetectionExecutor.execute()");

        return result;
        }

        private String waitAndReadResponse(Logger logger) throws IOException {
        int port = (int) module.getConfiguration(

        SGLibObjectDetectionModule.OBJECT_DETECTION_MODULE_LISTENER_PORT_CONFIGU
RATION_KEY, 49999);

        logger.info("waiting for answer...");
        ServerSocket server = new ServerSocket(port);
        Socket answerConn = server.accept();

        String response = "";

        logger.info("object detection module connected.");
        Scanner sc = new Scanner(answerConn.getInputStream());
        if (sc.hasNextLine()) {
                response = sc.nextLine();
        }

        sc.close();
        server.close();

        return response;
        }

        private void sendDetectCommand(Logger logger) throws IOException {
        String host = (String) module.getConfiguration(

        SGLibObjectDetectionModule.OBJECT_DETECTION_MODULE_HOST_CONFIGURATION_KE
Y, "localhost");
        int port = (int) module

        .getConfiguration(SGLibObjectDetectionModule.OBJECT_DETECTION_MODULE_PORT_CONFI
GURATION_KEY, 50000);

        int maxAttempts = 3;
        for (int i = 1; i <= maxAttempts; i++) {
                logger.info("trying to connecting with object detection module (" + module.getName() + ").
Host = " + host
                                + ":" + port + ". Attempts: " + i);
                Socket detectCommandConnection;
                try {
                detectCommandConnection = new Socket(host, port);
                OutputStream saida = detectCommandConnection.getOutputStream();
```

```
                    saida.write("detect".getBytes(Charset.forName("UTF-8")));
                    saida.flush();
                    saida.close();
                    detectCommandConnection.close();
                    logger.info("detect command sent to object detection module.");
                    break;
                    } catch (IOException e) {
                    logger.info(e.getMessage());
                    if (i == maxAttempts)
                            throw e;
                    }
            }

            }

}
```

C.10 Classe sglib.providers.objectDetection.ObjectDetectionGroundedPerceptProvider

```
package sglib.providers.objectDetection;

import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;
import java.util.Timer;
import java.util.TimerTask;
import java.util.logging.Logger;

import jason.asSyntax.Literal;
import sglib.configuration.SGLibModule;
import sglib.percepts.GroundedPercept;
import sglib.providers.IConfigurationProvider;
import sglib.providers.IGroundedPerceptProvider;
import sglib.providers.IGroundedPerceptTarget;
import sglib.utils.JasonUtils;

public class ObjectDetectionGroundedPerceptProvider implements IGroundedPerceptProvider {

            private ObjectDetectionResultReader reader;
            private static int runDetectionIntervalInSeconds = 10;
            private Timer timer = new Timer();
            private Map<String, Literal> detections = new HashMap<String, Literal>();
            private ArrayList<RegisteredTarget> targets = new ArrayList<>();
            private Logger logger = Logger.getLogger("sglib." +
ObjectDetectionGroundedPerceptProvider.class.getName());
            private ObjectDetectionExecutor executor;

            public ObjectDetectionGroundedPerceptProvider() {
            this.reader = new ObjectDetectionResultReader(logger);
            this.executor = new ObjectDetectionExecutor();
            }

            @Override
            public void register(IGroundedPerceptTarget target, String agentName) {
            targets.add(new RegisteredTarget(target, agentName));
            }

            @Override
            public void setConfiguration(IConfigurationProvider configurationProvider) {
```

```
        SGLibModule module =
configurationProvider.getModule(SGLibObjectDetectionModule.OBJECT_DETECTION_MODULE_NAME);
        if (module != null) {
                this.executor.configure(module);
                try {
                this.reader.readConfiguration(module);
                } catch (IOException e) {
                logger.severe("The configuration file could not be loaded");
                e.printStackTrace();
                }
        }
        }

        private void updateDetections(Map<String, Literal> newDetections) {
        if (newDetections.isEmpty()) {
                logger.info("New detections is empty. Exiting from update detections...");
                return;
        }

        ArrayList<Literal> toRemove = new ArrayList<>();
        for (Literal detection : this.detections.values()) {
                String detectionId = JasonUtils.generateLiteralId(detection);
                if (!newDetections.containsKey(detectionId)) {
                logger.info("Removing detection: " + detection.toString());
                toRemove.add(detection);
                }
        }

        removePerceptsFromTargets(toRemove);

        for (Literal removedDetection : toRemove) {
                this.detections.remove(JasonUtils.generateLiteralId(removedDetection));
        }

        ArrayList<Literal> toAdd = new ArrayList<>();
        for (Literal newDetection : newDetections.values()) {
                newDetection.addAnnot(Literal.parseLiteral("grounded"));
                newDetection.addAnnot(Literal.parseLiteral("visual"));
                String detectionId = JasonUtils.generateLiteralId(newDetection);
                if (!this.detections.containsKey(detectionId)) {
                this.detections.put(detectionId, newDetection);
                logger.info("Adding detection: " + newDetection.toString());
                toAdd.add(newDetection);
                }
        }

        addPerceptsToTargets(toAdd);
        }

        private void addPerceptsToTargets(ArrayList<Literal> toAdd) {
        if (toAdd.isEmpty())
                return;

        logger.info("Adding " + toAdd.size() + " percepts to targets");
        for (RegisteredTarget target : this.targets) {

        target.getTarget().addGroundedPercepts(convertToGroundedPerceptList(target.getAgentName(),
toAdd));
        }
        }
```

```
        private void removePerceptsFromTargets(ArrayList<Literal> toRemove) {
        if (toRemove.isEmpty())
                return;

        logger.info("Removing " + toRemove.size() + " percepts from targets");
        for (RegisteredTarget target : this.targets) {

        target.getTarget().removeGroundedPercepts(convertToGroundedPerceptList(target.getAgentName(),
toRemove));
        }
        }

        private static ArrayList<GroundedPercept> convertToGroundedPerceptList(String agentName,
ArrayList<Literal> list) {
        ArrayList<GroundedPercept> groundedList = new ArrayList<>();
        for (Literal percept : list) {
                groundedList.add(new GroundedPercept(agentName, percept));
        }
        return groundedList;
        }

        private void executeDetectionUpdate() {
        DetectionExecutionResult result = executor.execute(logger);
        if (result.getResultCode() == DetectionExecutionResultCode.OK
                || result.getResultCode() == DetectionExecutionResultCode.Unchanged) {
                logger.info("detection returned ok. Reading result file...");
                Map<String, Literal> detections = reader.readDetectionResult();
                if (detections != null) {
                updateDetections(detections);
                } else {
                logger.info("Read detection result returned null");
                }
        }
        scheduleNextDetectionExecution();
        }

        private void scheduleNextDetectionExecution() {
        this.timer.schedule(new CheckObjectsTimerTask(), runDetectionIntervalInSeconds * 1000);
        }

        @Override
        public void start() {
        logger.info("Provider started.");
        executeDetectionUpdate();
        }

        @Override
        public void stop() {
        if (this.timer != null) {
                this.timer.cancel();
        }
        }

        private class CheckObjectsTimerTask extends TimerTask {

        @Override
        public void run() {
                logger.info("Running detection from timer task...");
                executeDetectionUpdate();
```

```
		}

		}

		private class RegisteredTarget {

		private IGroundedPerceptTarget target;
		private String agentName;

		public RegisteredTarget(IGroundedPerceptTarget target, String agentName) {
			this.target = target;
			this.agentName = agentName;
		}

		public IGroundedPerceptTarget getTarget() {
			return target;
		}

		public String getAgentName() {
			return agentName;
		}
		}

}
```

C.11 Classe sglib.providers.objectDetection.ObjectDetectionGroundProvider

```
package sglib.providers.objectDetection;

import java.io.IOException;
import java.net.UnknownHostException;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import java.util.logging.Logger;
import jason.asSyntax.ListTerm;
import jason.asSyntax.ListTermImpl;
import jason.asSyntax.Literal;
import sglib.configuration.SGLibModule;
import sglib.providers.IConfigurationProvider;
import sglib.providers.IGroundProvider;
import sglib.utils.JasonUtils;

public class ObjectDetectionGroundProvider implements IGroundProvider {

		private ObjectDetectionResultReader reader;
		private ObjectDetectionExecutor executor;
		private Logger logger = Logger.getLogger("sglib." +
ObjectDetectionGroundProvider.class.getName());

		public ObjectDetectionGroundProvider() {
		this.reader = new ObjectDetectionResultReader(logger);
		this.executor = new ObjectDetectionExecutor();
		}

		private Map<String, Literal> runDetection() throws UnknownHostException, IOException {
		Map<String, Literal> detections = null;

		logger.info("runDetection called");
```

```
DetectionExecutionResult result = this.executor.execute(logger);
if (result.getResultCode() == DetectionExecutionResultCode.OK
        || result.getResultCode() == DetectionExecutionResultCode.Unchanged) {
        logger.info("Reading result file...");
        detections = reader.readDetectionResult();
} else {
        logger.info("Result: " + result.getMessage());
}

logger.info("end of runDetection");

return detections;
}

public ListTerm groundSymbol(Literal belief) {
ListTerm result = new ListTermImpl();

try {
        Map<String, Literal> detections = runDetection();
        if (detections != null && !detections.isEmpty()) {
        logger.info("detections is not empty");
        String beliefId = JasonUtils.generateLiteralId(belief);
        Literal detection = detections.get(beliefId);
        if (detection != null) {
                result.add(Literal.parseLiteral("visual"));
        }
        } else {
        logger.info("detections is null or empty");
        }
} catch (UnknownHostException e) {
        logger.severe(e.getMessage());
} catch (IOException e) {
        logger.severe(e.getMessage());
}

return result;
}

@Override
public void setConfiguration(IConfigurationProvider configurationProvider) {
SGLibModule module =
configurationProvider.getModule(SGLibObjectDetectionModule.OBJECT_DETECTION_MODULE_NAME);
if (module != null) {
        this.executor.configure(module);
        try {
        this.reader.readConfiguration(module);
        } catch (IOException e) {
        logger.severe("The configuration file could not be loaded");
        e.printStackTrace();
        }
}
}

@Override
public List<Literal> groundBeliefs(List<Literal> beliefs) {
List<Literal> result = new ArrayList<Literal>();

try {
        Map<String, Literal> detections = runDetection();
        if (detections != null && !detections.isEmpty()) {
```

```
                    logger.info("detections is not empty");

                    for (Literal belief : beliefs) {
                            String beliefId = JasonUtils.generateLiteralId(belief);
                            Literal detection = detections.get(beliefId);
                            if (detection != null) {
                            belief.addAnnot(Literal.parseLiteral("grounded"));
                            belief.addAnnot(Literal.parseLiteral("visual"));
                            }
                            result.add(belief);
                    }

                    } else {
                    logger.info("detections is null or empty");
                    }
            } catch (UnknownHostException e) {
                    logger.severe(e.getMessage());
            } catch (IOException e) {
                    logger.severe(e.getMessage());
            }

            return result;
            }

}
```

C.12 Classe sglib.providers.objectDetection.ObjectDetectionResultReader

```
package sglib.providers.objectDetection;

import java.io.IOException;
import java.util.HashMap;
import java.util.Map;
import java.util.logging.Logger;

import org.json.JSONArray;
import org.json.JSONObject;

import jason.asSyntax.Literal;
import sglib.configuration.SGLibModule;
import sglib.utils.FileUtils;
import sglib.utils.JasonUtils;

public class ObjectDetectionResultReader {

        private Logger logger;
        private DetectionConfiguration configuration;

        public ObjectDetectionResultReader(Logger logger) {
        this.logger = logger;
        }

        public void readConfiguration(SGLibModule module) throws IOException {
        String configFilePath = (String) module

.getConfiguration(SGLibObjectDetectionModule.OBJECT_DETECTION_MODULE_CONFIG_FILE_LOCATION_KEY, "");
        this.configuration = new DetectionConfiguration(configFilePath);
```

```
}

public Map<String, Literal> readDetectionResult() {
Map<String, Literal> detections = new HashMap<String, Literal>();

try {
        String detectionResultFile = configuration.getDetectionResultFilePath();
        logger.info("Reading from: " + detectionResultFile);
        String json = FileUtils.readFileContents(detectionResultFile);
        JSONArray jsonDetections = new JSONArray(json);

        for (int i = 0; i < jsonDetections.length(); i++) {
        JSONObject jsonObject = jsonDetections.getJSONObject(i);
        String percept = jsonObject.getString("percept");
        float score = jsonObject.getFloat("score");
        if (score >= 0.5) {
                JSONObject jsonObjectClass = jsonObject.getJSONObject("class");

                String className = jsonObjectClass.getString("name");
                Literal litPercept = Literal
                .parseLiteral(percept + "(" + convertClassToValidJasonValue(className) + ")");

                if (jsonObject.has("annots")) {
                JSONArray annots = jsonObject.getJSONArray("annots");
                for (int j = 0; j < annots.length(); j++) {
                Literal annot = Literal.parseLiteral(annots.getString(j));
                litPercept.addAnnot(annot);
                }
                }

                logger.info(litPercept.toString() + " was read from json file.");
                detections.put(JasonUtils.generateLiteralId(litPercept), litPercept);
        }
        }

        return detections;
} catch (IOException e) {
        logger.severe(e.getMessage());
        e.printStackTrace();
        return null;
}
}

private static String convertClassToValidJasonValue(String originalClassName) {
String result = originalClassName.replaceAll("[\\s,]", "_").replaceAll("[\"'()]", "");
result = result.substring(0, 1).toLowerCase() + result.substring(1);
return result;
}

private class DetectionConfiguration {

private String resultBasePath;

public DetectionConfiguration(String configFilePath) throws IOException {
        String json = FileUtils.readFileContents(configFilePath);
        JSONObject jsonObj = new JSONObject(json);

        setResultBasePath(jsonObj.getString("RESULT_BASE_PATH"));
}
```

```
	public String getResultBasePath() {
		return resultBasePath;
	}

	public void setResultBasePath(String resultBasePath) {
		this.resultBasePath = resultBasePath;
	}

	public String getDetectionResultFilePath() {
		return this.getResultBasePath() + "\\agent.perception.json";
	}

	}

}
```

C.13 Classe sglib.providers.objectDetection.SGLibObjectDetectionModule

```
package sglib.providers.objectDetection;

import sglib.configuration.SGLibModule;

public class SGLibObjectDetectionModule extends SGLibModule {

	public final static String OBJECT_DETECTION_MODULE_NAME = "objectDetection";
	public final static String OBJECT_DETECTION_MODULE_HOST_CONFIGURATION_KEY =
"host";
	public final static String OBJECT_DETECTION_MODULE_PORT_CONFIGURATION_KEY =
"serverPort";
	public final static String
OBJECT_DETECTION_MODULE_LISTENER_PORT_CONFIGURATION_KEY = "listenerPort";
	public final static String OBJECT_DETECTION_MODULE_CONFIG_FILE_LOCATION_KEY =
"configFilePath";

	public SGLibObjectDetectionModule() {
	setName(OBJECT_DETECTION_MODULE_NAME);
	}

	public void setHost(String host) {
	setConfiguration(OBJECT_DETECTION_MODULE_HOST_CONFIGURATION_KEY, host);
	}

	public void setServerPort(int port) {
	setConfiguration(OBJECT_DETECTION_MODULE_PORT_CONFIGURATION_KEY, port);
	}

	public void setListenerPort(int port) {

	setConfiguration(OBJECT_DETECTION_MODULE_LISTENER_PORT_CONFIGURATION_KE
Y, port);
	}

	public void setConfigFilePath(String configFilePath) {
	setConfiguration(OBJECT_DETECTION_MODULE_CONFIG_FILE_LOCATION_KEY,
configFilePath);
	}
}
```

C.14 Interface sglib.providers.IConfigurationProvider

```
package sglib.providers;
import sglib.configuration.SGLibModule;
public interface IConfigurationProvider {
        public SGLibModule getModule(String name);
}
```

C.15 Interface sglib.providers.IGroundedPerceptProvider

```
package sglib.providers;
public interface IGroundedPerceptProvider {
        public void start();
        public void stop();
        public void register(IGroundedPerceptTarget target, String agentName);
        void setConfiguration(IConfigurationProvider configurationProvider);
}
```

C.16 Interface sglib.providers.IGroundedPerceptTarget

```
package sglib.providers;
import java.util.List;
import sglib.percepts.GroundedPercept;
public interface IGroundedPerceptTarget {
        public void addGroundedPercepts(List<GroundedPercept> percepts);
        public void removeGroundedPercepts(List<GroundedPercept> percepts);
}
```

C.17 Interface sglib.providers.IGroundProvider

```
package sglib.providers;

import java.util.List;

import jason.asSyntax.ListTerm;
import jason.asSyntax.Literal;

public interface IGroundProvider {

	public void setConfiguration(IConfigurationProvider configurationProvider);

	public ListTerm groundSymbol(Literal belief);

	public List<Literal> groundBeliefs(List<Literal> beliefs);

}
```

C.18 Classe sglib.utils.FileUtils

```
package sglib.utils;

import java.io.File;
import java.io.IOException;
import java.nio.file.Files;

public class FileUtils {

	private FileUtils() {
	}

	public static String readFileContents(String filePath) throws IOException {
	return new String(Files.readAllBytes(new File(filePath).toPath()));
	}

}
```

C.19 Classe sglib.utils.JasonUtils

```
package sglib.utils;

import jason.asSyntax.Literal;
import jason.asSyntax.Term;

public class JasonUtils {

	private JasonUtils() {
	}

	public static String generateLiteralId(Literal percept) {
	String id = percept.getFunctor();

	String separator = "";
	for (Term term : percept.getTerms()) {
		id += separator + term.toString();
		separator = ",";
	}

	return id;
	}

}
```

C.20 Classe sglib.configureGrounders

```
package sglib;

import java.util.logging.Logger;

import jason.JasonException;
import jason.asSemantics.Agent;
import jason.asSemantics.DefaultInternalAction;
import jason.asSemantics.TransitionSystem;
import jason.asSemantics.Unifier;
import jason.asSyntax.StringTerm;
import jason.asSyntax.Term;

public class configureGrounders extends DefaultInternalAction {

	private static final long serialVersionUID = 1L;

	private Logger logger = Logger.getLogger("sglib." + configureGrounders.class.getName());

	@Override
	public int getMinArgs() {
	return 1;
	}

	@Override
	public int getMaxArgs() {
	return 1;
	}

	@Override
	protected void checkArguments(Term[] args) throws JasonException {
	super.checkArguments(args);
```

```
        if (!(args[0] instanceof StringTerm))
                throw JasonException.createWrongArgument(this,
                        "first argument must be a string with the configuration json");
        }

        @Override
        public Object execute(TransitionSystem ts, Unifier un, Term[] args) throws Exception {
        checkArguments(args);
        logger.info(configureGrounders.class.getName() + " internal action was called.");

        Agent agent = ts.getAg();
        groundSymbol ia = (groundSymbol) agent.getIA(groundSymbol.class.getTypeName());

        if (ia != null) {
                StringTerm json = (StringTerm) args[0];
                ia.configureProviders(json.getString());
                return true;
        }
        return false;
        }

}
```

C.21 Classe sglib.groundSymbol

```
package sglib;

import java.lang.reflect.Constructor;
import java.lang.reflect.InvocationTargetException;
import java.util.HashMap;
import java.util.logging.Logger;

import jason.JasonException;
import jason.asSemantics.DefaultInternalAction;
import jason.asSemantics.TransitionSystem;
import jason.asSemantics.Unifier;
import jason.asSyntax.ListTerm;
import jason.asSyntax.ListTermImpl;
import jason.asSyntax.Literal;
import jason.asSyntax.Term;
import sglib.configuration.SGLibConfiguration;
import sglib.configuration.SGLibModule;
import sglib.providers.IConfigurationProvider;
import sglib.providers.IGroundProvider;

public class groundSymbol extends DefaultInternalAction implements IConfigurationProvider {

	private static final long serialVersionUID = 1L;

	private Logger logger = Logger.getLogger("sglib." + groundSymbol.class.getName());
	private HashMap<String, IGroundProvider> groundProviders = new HashMap<String,
IGroundProvider>();
	private SGLibConfiguration configuration;

	public groundSymbol() {
	}

	@Override
	public int getMinArgs() {
	return 2;
	}

	@Override
	public int getMaxArgs() {
	return 2;
	}

	@Override
	protected void checkArguments(Term[] args) throws JasonException {
	super.checkArguments(args);
	if (!(args[0] instanceof Literal))
		throw JasonException.createWrongArgument(this, "first argument must be a literal");
	}

	public void configureProviders(String configurationJson) {
	this.configuration = new SGLibConfiguration(configurationJson);
	}

	@Override
	public Object execute(TransitionSystem ts, Unifier un, Term[] args) throws Exception {
	checkArguments(args);

	logger.info(groundSymbol.class.getName() + " internal action was called.");
```

```
		if (this.configuration == null)
			throw new JasonException(
				"You must configure ground providers before using the groundSymbol internal
action.");

		ListTerm result = new ListTermImpl();
		Literal belief = (Literal) args[0];

		for (String providerName : configuration.getProviders()) {
			if (!groundProviders.containsKey(providerName)) {
			try {
				Class<?> clazz = Class.forName(providerName);
				Constructor<?> constructor = clazz.getConstructor();
				IGroundProvider providerInstance = (IGroundProvider) constructor.newInstance();
				providerInstance.setConfiguration(this);
				groundProviders.put(providerName, providerInstance);
			} catch (ClassNotFoundException e) {
				e.printStackTrace();
			} catch (NoSuchMethodException e) {
				e.printStackTrace();
			} catch (SecurityException e) {
				e.printStackTrace();
			} catch (InstantiationException e) {
				e.printStackTrace();
			} catch (IllegalAccessException e) {
				e.printStackTrace();
			} catch (IllegalArgumentException e) {
				e.printStackTrace();
			} catch (InvocationTargetException e) {
				e.printStackTrace();
			}
			}

			IGroundProvider providerInstance = groundProviders.get(providerName);
			if (providerInstance == null)
			continue;

			ListTerm providerResult = providerInstance.groundSymbol(belief);

			if (providerResult != null)
			result.addAll(providerResult);
		}

		logger.info(result.size() + " grounds found to belief " + belief.toString());
		logger.info(result.toString());

		return un.unifies(args[1], result);
		}

		@Override
		public SGLibModule getModule(String name) {
		return configuration.getModule(name);
		}
}
```

APÊNDICE D - Código fonte dos experimentos

D.1 Projeto *sgmodel_baseline*

D.1.1 Código fonte do agente smartcoffeemachine_agent.asl

```
/* Initial beliefs and rules */

detectedAndCanServeWith(Object) :- detected(Object) & canServeWith(Object).
detectedAndCanServeIn(Object) :- detected(Object) & canServeIn(Object).

isAbleToServe :- detectedAndCanServeWith(ObjectToServeWith) &
detectedAndCanServeIn(ObjectToServeIn).

canServeWith(kettle).
canServeIn(coffee_cup).

/* Initial goals */

!start.

/* Plans */

+!start <- .print("Hello, just give me a kettle and a coffee cup and I'll pour you some coffee.")
           !refreshPerceptions.

+canServeWith(Object).
-canServeWith(Object).

+canServeIn(Object).
-canServeIn(Object).

+detected(woman) <- +detected(person).
+detected(man) <- +detected(person).

-detected(woman) <- -detected(person).
-detected(man) <- -detected(person).

+detected(person) <- !serve.
-detected(person) <- .remove_plan(serve).

+detected(Object) : not canServeWith(Object) & not canServeIn(Object) & not (Object == coffee) <-
askTheUtility(Object).
+detected(Object) : detected(person) <- !serve.

+!updatePerceptions : not serving <- .print("generating event to refreshPercetions (now +10 s)");
                                    .at("now +10 s", {+!refreshPerceptions}).

+!refreshPerceptions : not serving <- .print("Agent will refreshPerceptions (external action)...");
                                      refreshPerceptions;
                                      !updatePerceptions.

+!refreshPerceptions : serving.

+!serve : isAbleToServe & detected(person) <- .findall(O, detectedAndCanServeWith(O), L1)
                                              .nth(0, L1, ObjToServeWith)
                                              .findall(O, detectedAndCanServeIn(O), L2)
                                              .nth(0, L2, ObjToServeIn)
```

```
                                        .print("It will serve with ", ObjToServeWith, " in ", ObjToServeIn)
                                        !serveWith(ObjToServeWith, ObjToServeIn).

+!serve : not detected(person).

+!serve : detectedAndCanServeIn(Object) <- requestObjectToServeWith.

+!serve : detectedAndCanServeWith(Object) <- requestObjectToServeIn.

+!serve <- .print("Waiting for the objects to serve the client with.").

+!serveWith(ObjToServeWith, ObjToServeIn) : not serving
                                        <- +serving;
                                .print("Serving the client...");
                                        .print("Using: ", ObjToServeWith);
                                                .print("To fill: ", ObjToServeIn);
                                                serveClient(ObjToServeWith, ObjToServeIn);
                                                .print("generating event o check if was successful... (now
+30 s)");
                                                .at("now +30 s",
{+!refreshPerceptionsAndCheckSuccessful}).

+!serveWith(ObjToServeWith, ObjToServeIn) : serving.

+!refreshPerceptionsAndCheckSuccessful <- .print("The agent will verify that it was successful...");
                                        verifyThatWasSuccessful;
                                        !checkSuccessful;
                                        -serving;
                                        !updatePerceptions.

+!checkSuccessful : detected(coffee) <- .print("The client was served successfully.").

+!checkSuccessful : not detected(coffee) <- .print("Oops! There was some problem when serving the client.").
```

D.1.2 Código fonte do ambiente

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;
import java.util.logging.Logger;

import jason.asSyntax.Literal;
import jason.asSyntax.Structure;
import jason.environment.Environment;
import sglib.utils.JasonUtils;

public class SGEnvironmentBaseline extends Environment {

    private int currentSituation = -1;
    private Logger logger = Logger.getLogger("sgmodel_baseline." + SGEnvironmentBaseline.class.getName());
    private Map<String, Literal> currentDetections = new HashMap<>();
    private Map<String, Literal>[] perceptsSituations;

    @Override
    public void init(String[] args) {
        super.init(args);
        generatePerceptsSituations();
    }
```

```
    @SuppressWarnings("unchecked")
    private void generatePerceptsSituations() {
            perceptsSituations = (Map<String, Literal>[]) new Map[5];
            perceptsSituations[0] = createPerceptsMap("detected(kettle)");
            perceptsSituations[1] = createPerceptsMap("detected(person)", "detected(coffee_cup)");
            perceptsSituations[2] = createPerceptsMap("detected(person)", "detected(kettle)", "detected(cup)");
            perceptsSituations[3] = createPerceptsMap("detected(person)", "detected(kettle)",
"detected(coffee_cup)");
            perceptsSituations[4] = createPerceptsMap("detected(person)", "detected(kettle)",
"detected(coffee_cup)", "detected(coffee)");
    }

    private HashMap<String, Literal> createPerceptsMap(String... percepts) {
            HashMap<String, Literal> map = new HashMap<String, Literal>();
            for (String percept : percepts) {
            Literal litPercept = Literal.parseLiteral(percept);
            map.put(JasonUtils.generateLiteralId(litPercept), litPercept);
            }
            return map;
    }

    @Override
    public boolean executeAction(String agName, Structure action) {

            String actionName = action.getFunctor();
            if (actionName.equals("refreshPerceptions")) {
            logger.info("The agent is refreshing the percepts...");
            updatePerceptions();
            informAgsEnvironmentChanged();
            return true;
            } else if (actionName.equals("verifyThatWasSuccessful")) {
            logger.info("The agent is vertifying if it was successful...");
            updatePerceptions();
            informAgsEnvironmentChanged();
            return true;
            } else if (actionName.equals("askTheUtility")) {
            String askedObject = action.getTerm(0).toString();
            logger.info("The agent asked for the utility of the object: " + askedObject);
            answerTheUtilityOf(askedObject);
            return true;
            } else if (actionName.equals("requestObjectToServeWith")) {
            logger.info("The agent requested an object to serve coffee with");
            return true;
            } else if (actionName.equals("requestObjectToServeIn")) {
            logger.info("The agent requested an object to serve coffee in");
            return true;
            } else if (actionName.equals("serveClient")) {
            String objectToServeWith = action.getTerm(0).toString();
            String objectToServeIn = action.getTerm(1).toString();
            logger.info("The agent served the client using " + objectToServeWith + " to fill " + objectToServeIn);
            return true;
            }

            logger.info("executing: " + action + ", but not implemented!");
            return false;
    }

    private void answerTheUtilityOf(String askedObject) {
            String percept = "unkownUtility";
            if (askedObject.equals("teapot") || askedObject.equals("kettle")) {
```

```
            percept = "canServeWith";
            } else if (askedObject.equals("mug") || askedObject.equals("coffee_cup") || askedObject.equals("cup"))
{
            percept = "canServeIn";
            }
            String newPerceptText = percept + "(" + askedObject + ")";
            logger.info("Answering: " + newPerceptText);
            addPercept(Literal.parseLiteral(newPerceptText));
    }

    private void updatePerceptions() {
            updatePerceptions(-1);
    }

    private void updatePerceptions(int situationIndex) {
            if (situationIndex == -1) {
            currentSituation++;

            if (currentSituation == perceptsSituations.length)
            currentSituation = 0;

            situationIndex = currentSituation;
            } else {
            currentSituation = situationIndex;
            }

            logger.info("Updating percepts...");
            Map<String, Literal> newDetections = perceptsSituations[currentSituation];

            logger.info(newDetections.size() + " new detections read.");
            ArrayList<Literal> toRemove = new ArrayList<>();
            ArrayList<Literal> toAdd = new ArrayList<>();

            for (String key : newDetections.keySet()) {
            if (!this.currentDetections.containsKey(key)) {
             toAdd.add(newDetections.get(key));
            }
            }

            for (String key : this.currentDetections.keySet()) {
            if (!newDetections.containsKey(key)) {
            toRemove.add(this.currentDetections.get(key));
            }
            }

            logger.info(toRemove.size() + " percepts to remove.");
            for (Literal percept : toRemove) {
            this.currentDetections.remove(JasonUtils.generateLiteralId(percept));
            removePercept(percept);
            }

            logger.info(toAdd.size() + " percepts to add.");
            for (Literal percept : toAdd) {
            this.currentDetections.put(JasonUtils.generateLiteralId(percept), percept);
            }
            addPercept(toAdd.toArray(new Literal[toAdd.size()]));
    }

}
```

D.1.3 Arquivo .mas2j

```
MAS sgmodel_baseline {

        infrastructure: Centralised

        environment: SGEnvironmentBaseline

        agents:
        agent1 smartcoffeemachine_agent;

        aslSourcePath:
        "src/asl";
}
```

D.2 Projeto *sgmodel_v1*

D.2.1 Código fonte do agente smartcoffeemachine_agent.asl

```
/* Initial beliefs and rules */

detectedAndCanServeWith(Object) :- detected(Object)[grounded] & canServeWith(Object).
detectedAndCanServeIn(Object) :- detected(Object)[grounded] & canServeIn(Object).

isAbleToServe :- detectedAndCanServeWith(ObjectToServeWith) &
detectedAndCanServeIn(ObjectToServeIn).

canServeWith(kettle).
canServeIn(coffee_cup).

/* Initial goals */

!start.

/* Plans */

+!start <- .print("Hello, just give me a kettle and a coffee cup and I'll pour you some coffee.")
           !refreshPerceptions.

+canServeWith(Object).
-canServeWith(Object).

+canServeIn(Object).
-canServeIn(Object).

+detected(woman) <- +detected(person).
+detected(man) <- +detected(person).

-detected(woman) <- -detected(person).
-detected(man) <- -detected(person).

+detected(person) <- !serve.
-detected(person) <- .remove_plan(serve).

+detected(Object) : not canServeWith(Object) & not canServeIn(Object) & not (Object == coffee) <-
askTheUtility(Object).
+detected(Object) : detected(person) <- !serve.
```

```
+!updatePerceptions : not serving <- .print("generating event to refreshPercetions (now +10 s)");
                                     .at("now +10 s", {+!refreshPerceptions}).

+!refreshPerceptions : not serving <- .print("Agent will refreshPerceptions (external action)...");
                                      refreshPerceptions;
                                                !updatePerceptions.

+!refreshPerceptions : serving.

+!serve : isAbleToServe & detected(person) <- .findall(O, detectedAndCanServeWith(O), L1)
                                              .nth(0, L1, ObjToServeWith)
                                                        .findall(O, detectedAndCanServeIn(O), L2)
                                                        .nth(0, L2, ObjToServeIn)
                                              .print("It will serve with ", ObjToServeWith, " in ", ObjToServeIn)
                                              !serveWith(ObjToServeWith, ObjToServeIn).

+!serve : not detected(person).

+!serve : detectedAndCanServeIn(Object) <- requestObjectToServeWith.

+!serve : detected(Object) & canServeIn(Object) <- .print("Agent has detected(", Object,") to serve in, but it's
not grounded. Therefore, it will not be considered valid to be used.").

+!serve : detectedAndCanServeWith(Object) <- requestObjectToServeIn.

+!serve : detected(Object) & canServeWith(Object) <- .print("Agent has detected(", Object,") to serve with, but
it's not grounded. Therefore, it will not be considered valid to be used.").

+!serve <- .print("Waiting for the objects to serve the client with.").

+!serveWith(ObjToServeWith, ObjToServeIn) : not serving
                                              <- +serving;
                                    .print("Serving the client...");
                                              .print("Using: ", ObjToServeWith);
                                                        .print("To fill: ", ObjToServeIn);
                                                        serveClient(ObjToServeWith, ObjToServeIn);
                                                        .print("generating event o check if was successful... (now
+30 s)");
                                                        .at("now +30 s",
{+!refreshPerceptionsAndCheckSuccessful}).

+!serveWith(ObjToServeWith, ObjToServeIn) : serving.

+!refreshPerceptionsAndCheckSuccessful <- .print("The agent will verify that it was successful...");
                                          verifyThatWasSuccessful;
                                          !checkSuccessful;
                                          -serving;
                                          !updatePerceptions.

+!checkSuccessful : detected(coffee) <- .print("The client was served successfully.").

+!checkSuccessful : not detected(coffee) <- .print("Oops! There was some problem when serving the client.").
```

D.2.2 Classe SGEnvironmentV1

```
import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;
import java.util.logging.Logger;

import jason.asSyntax.Literal;
import jason.asSyntax.Structure;
import jason.environment.Environment;
import sglib.utils.JasonUtils;

public class SGEnvironmentV1 extends Environment {

    private Map<String, Literal> currentDetections = new HashMap<>();
    private Logger logger = Logger.getLogger("sgmodel_v1." + SGEnvironmentV1.class.getName());
    private int currentSituation = -1;
    private String[] objectDetectionSituations = {
            "C:\\UFSC\\TCC\\test-images\\1.person.bat",
            "C:\\UFSC\\TCC\\test-images\\2.person_kettle.bat",
            "C:\\UFSC\\TCC\\test-images\\3.person_kettle_cup.bat",
            "C:\\UFSC\\TCC\\test-images\\3.person_kettle_cup.bat",
            "C:\\UFSC\\TCC\\test-images\\4.person_kettle_coffee_cup.bat"
    };

    private Map<String, Literal>[] perceptsSituations;

    @Override
    public void init(String[] args) {
            super.init(args);
            generatePerceptsSituations();
    }

    @SuppressWarnings("unchecked")
    private void generatePerceptsSituations() {
            perceptsSituations = (Map<String, Literal>[]) new Map[5];
            perceptsSituations[0] = createPerceptsMap("detected(kettle)");
            perceptsSituations[1] = createPerceptsMap("detected(person)", "detected(coffee_cup)");
            perceptsSituations[2] = createPerceptsMap("detected(person)", "detected(kettle)", "detected(cup)");
            perceptsSituations[3] = createPerceptsMap("detected(person)", "detected(kettle)",
"detected(coffee_cup)");
            perceptsSituations[4] = createPerceptsMap("detected(person)", "detected(kettle)",
"detected(coffee_cup)", "detected(coffee)");
    }

    private HashMap<String, Literal> createPerceptsMap(String... percepts) {
            HashMap<String, Literal> map = new HashMap<String, Literal>();
            for (String percept : percepts) {
            Literal litPercept = Literal.parseLiteral(percept);
            map.put(JasonUtils.generateLiteralId(litPercept), litPercept);
            }
            return map;
    }

    @Override
    public boolean executeAction(String agName, Structure action) {

            String actionName = action.getFunctor();
```

```
            if (actionName.equals("refreshPerceptions")) {
            logger.info("The agent is refreshing the percepts...");
            simulateObjectDetectionChanges();
            updatePerceptions();
            informAgsEnvironmentChanged();
            return true;
            } else if (actionName.equals("verifyThatWasSuccessful")) {
            logger.info("The agent is vertifying if it was successful...");
            simulateObjectDetectionChanges(objectDetectionSituations.length - 1);
            updatePerceptions();
            informAgsEnvironmentChanged();
            return true;
            } else if (actionName.equals("askTheUtility")) {
            String askedObject = action.getTerm(0).toString();
            logger.info("The agent asked for the utility of the object: " + askedObject);
            answerTheUtilityOf(askedObject);
            return true;
            } else if (actionName.equals("requestObjectToServeWith")) {
            logger.info("The agent requested an object to serve coffee with");
            return true;
            } else if (actionName.equals("requestObjectToServeIn")) {
            logger.info("The agent requested an object to serve coffee in");
            return true;
            } else if (actionName.equals("serveClient")) {
            String objectToServeWith = action.getTerm(0).toString();
            String objectToServeIn = action.getTerm(1).toString();
            logger.info("The agent served the client using " + objectToServeWith + " to fill " + objectToServeIn);
            return true;
            }

            logger.info("executing: " + action + ", but not implemented!");
            return false;
    }

    private void answerTheUtilityOf(String askedObject) {
            String percept = "unkownUtility";
            if (askedObject.equals("teapot") || askedObject.equals("kettle")) {
            percept = "canServeWith";
            } else if (askedObject.equals("mug") || askedObject.equals("coffee_cup") || askedObject.equals("cup"))
{
            percept = "canServeIn";
            }
            String newPerceptText = percept + "(" + askedObject + ")";
            logger.info("Answering: " + newPerceptText);
            addPercept(Literal.parseLiteral(newPerceptText));
    }

    private void simulateObjectDetectionChanges() {
            simulateObjectDetectionChanges(-1);
    }

    private void simulateObjectDetectionChanges(int situationIndex) {
            // Simulates object detection changes for the next execution
            logger.info("Simulating object detection changes...");

            if (situationIndex == -1) {
            currentSituation++;

            if (currentSituation == objectDetectionSituations.length)
            currentSituation = 0;
```

```
            situationIndex = currentSituation;
            } else {
            currentSituation = situationIndex;
            }

            String situation = objectDetectionSituations[situationIndex];
            logger.info("Running the situation: " + situation);

            try {
            Runtime.getRuntime().exec("cmd /c start /B " + situation);
            Thread.sleep(1000);
            } catch (IOException e) {
            e.printStackTrace();
            } catch (InterruptedException e) {
            e.printStackTrace();
            }
    }

    private void updatePerceptions() {
            // Changes perceptions using predefined situations to simulate ungrounded
            // percepts
            logger.info("Updating percepts...");
            Map<String, Literal> newDetections = perceptsSituations[currentSituation];

            logger.info(newDetections.size() + " new detections read.");
            ArrayList<Literal> toRemove = new ArrayList<>();
            ArrayList<Literal> toAdd = new ArrayList<>();

            for (String key : newDetections.keySet()) {
            if (!this.currentDetections.containsKey(key)) {
            toAdd.add(newDetections.get(key));
            }
            }

            for (String key : this.currentDetections.keySet()) {
            if (!newDetections.containsKey(key)) {
            toRemove.add(this.currentDetections.get(key));
            }
            }

            logger.info(toRemove.size() + " percepts to remove.");
            for (Literal percept : toRemove) {
            this.currentDetections.remove(JasonUtils.generateLiteralId(percept));
            removePercept(percept);
            }

            logger.info(toAdd.size() + " percepts to add.");
            for (Literal percept : toAdd) {
               this.currentDetections.put(JasonUtils.generateLiteralId(percept), percept);
            }
            addPercept(toAdd.toArray(new Literal[toAdd.size()]));
    }

}
```

D.2.3 Classe SmartcoffeMachineAgent

```
import java.util.ArrayList;

import sglib.agents.BaseGroundedAgent;
import sglib.configuration.SGLibConfiguration;
import sglib.configuration.SGLibConfigurationBuilder;
import sglib.providers.IGroundProvider;
import sglib.providers.objectDetection.ObjectDetectionGroundProvider;
import sglib.providers.objectDetection.SGLibObjectDetectionModule;

public class SmartcoffeMachineAgent extends BaseGroundedAgent {

   private static final long serialVersionUID = 1L;

   public SmartcoffeMachineAgent() {
         super();
   }

   @Override
   protected SGLibConfiguration createSGLibConfiguration() {
         SGLibObjectDetectionModule module = new SGLibObjectDetectionModule();

         module.setHost("localhost");
         module.setServerPort(50000);
         module.setListenerPort(49999);
         module.setConfigFilePath("D:\\GitHub\\tcc\\object_detection\\config.json");

         return new SGLibConfigurationBuilder().withModule(module).build();
   }

   @Override
   protected ArrayList<IGroundProvider> createGroundProviders() {
         ArrayList<IGroundProvider> result = new ArrayList<IGroundProvider>();
         result.add(new ObjectDetectionGroundProvider());
         return result;
   }

}
```

D.2.4 Arquivo .mas2j

```
MAS sgmodel_v1 {

         infrastructure: Centralised

         environment: SGEnvironmentV1

         agents:
         agent1 smartcoffeemachine_agent.asl agentClass SmartcoffeMachineAgent;

         aslSourcePath:
         "src/asl";
}
```

D.3 Projeto sgmodel_v2

D.3.1 Código fonte do agente smartcoffeemachine_agent.asl

```
/* Initial beliefs and rules */

detectedAndCanServeWith(Object) :- detected(Object)[grounded] & canServeWith(Object).
detectedAndCanServeIn(Object) :- detected(Object)[grounded] & canServeIn(Object).

isAbleToServe :- detectedAndCanServeWith(ObjectToServeWith) &
detectedAndCanServeIn(ObjectToServeIn).

canServeWith(kettle).
canServeIn(coffee_cup).

/* Initial goals */

!start.

/* Plans */

+!start <- .print("Hello, just give me a kettle and a coffee cup and I'll pour you some coffee.").

+canServeWith(Object).
-canServeWith(Object).

+canServeIn(Object).
-canServeIn(Object).

+detected(woman) <- +detected(person).
+detected(man) <- +detected(person).

-detected(woman) <- -detected(person).
-detected(man) <- -detected(person).

+detected(person) <- !serve.
-detected(person) <- .remove_plan(serve).

+detected(Object) : not canServeWith(Object) & not canServeIn(Object) & not (Object == coffee) <-
askTheUtility(Object).
+detected(Object) : detected(person) <- !serve.

+!serve : isAbleToServe & detected(person) <- .findall(O, detectedAndCanServeWith(O), L1)
                                              .nth(0, L1, ObjToServeWith)
                                                    .findall(O, detectedAndCanServeIn(O), L2)
                                                    .nth(0, L2, ObjToServeIn)
                                              .print("It will serve with ", ObjToServeWith, " in ", ObjToServeIn)
                                              !serveWith(ObjToServeWith, ObjToServeIn).

+!serve : not detected(person).

+!serve : detectedAndCanServeIn(Object) <- requestObjectToServeWith.

+!serve : detected(Object) & canServeIn(Object) <- .print("Agent has detected(", Object,") to serve in, but it's
not grounded. Therefore, it will not be considered valid to be used.").

+!serve : detectedAndCanServeWith(Object) <- requestObjectToServeIn.
```

```
+!serve : detected(Object) & canServeWith(Object) <- .print("Agent has detected(", Object,") to serve with, but
it's not grounded. Therefore, it will not be considered valid to be used.").

+!serve <- .print("Waiting for the objects to serve the client with.").

+!serveWith(ObjToServeWith, ObjToServeIn) : not serving
                                            <- +serving;
                                            .print("Serving the client...");
                                            .print("Using: ", ObjToServeWith);
                                                .print("To fill: ", ObjToServeIn);
                                                serveClient(ObjToServeWith, ObjToServeIn);
                                                .print("generating event o check if was successful... (now
+30 s)");
                                                .at("now +30 s",
{+!refreshPerceptionsAndCheckSuccessful}).

+!serveWith(ObjToServeWith, ObjToServeIn) : serving.

+!refreshPerceptionsAndCheckSuccessful <- .print("The agent will verify that it was successful...");
                                          verifyThatWasSuccessful;
                                          !checkSuccessful;
                                          -serving.

+!checkSuccessful : detected(coffee) <- .print("The client was served successfully.").

+!checkSuccessful : not detected(coffee) <- .print("Oops! There was some problem when serving the client.").
```

D.3.2 Classe SGEnvironmentV2

```
import java.io.IOException;
import java.util.Timer;
import java.util.TimerTask;
import java.util.logging.Logger;

import jason.asSyntax.Literal;
import jason.asSyntax.Structure;
import sglib.configuration.SGLibConfiguration;
import sglib.configuration.SGLibConfigurationBuilder;
import sglib.environments.BaseGroundedPerceptTargetEnvironment;
import sglib.providers.IGroundedPerceptProvider;
import sglib.providers.IGroundedPerceptTarget;
import sglib.providers.objectDetection.ObjectDetectionGroundProvider;
import sglib.providers.objectDetection.ObjectDetectionGroundedPerceptProvider;
import sglib.providers.objectDetection.SGLibObjectDetectionModule;

public class SGEnvironmentV2 extends BaseGroundedPerceptTargetEnvironment implements
IGroundedPerceptTarget {

    private boolean serving = false;
    private Object lock = new Object();
    private Timer timer;
    private int simulateChangesIntervalInSeconds = 15;
    private Logger logger = Logger.getLogger("sgmodel_v2." + SGEnvironmentV2.class.getName());
    private int currentSituation = -1;
    private String[] objectDetectionSituations = {
            "C:\\UFSC\\TCC\\test-images\\1.person.bat",
            "C:\\UFSC\\TCC\\test-images\\2.person_kettle.bat",
```

```
            "C:\\UFSC\\TCC\\test-images\\3.person_kettle_cup.bat",
            "C:\\UFSC\\TCC\\test-images\\4.person_kettle_coffee_cup.bat"
    };

    /** Called before the MAS execution with the args informed in .mas2j */
    @Override
    public void init(String[] args) {
            super.init(args);
            this.timer = new Timer();
            this.timer.schedule(new ChangeSituationTask(), 0, simulateChangesIntervalInSeconds * 1000);
    }

    @Override
    protected SGLibConfiguration createSGLibConfiguration() {
            SGLibObjectDetectionModule module = new SGLibObjectDetectionModule();

            module.setHost("localhost");
            module.setServerPort(50000);
            module.setListenerPort(49999);
            module.setConfigFilePath("D:\\GitHub\\tcc\\object_detection\\config.json");

            return new
SGLibConfigurationBuilder().withProvider(ObjectDetectionGroundProvider.class.getTypeName())
            .withModule(module).build();
    }

    @Override
    protected void loadProvidersAndRegister() {
            IGroundedPerceptProvider provider = new ObjectDetectionGroundedPerceptProvider();
            provider.register(this, null);
            this.providers.add(provider);
    }

    @Override
    public boolean executeAction(String agName, Structure action) {
            String actionName = action.getFunctor();
            if (actionName.equals("verifyThatWasSuccessful")) {
            synchronized (lock) {
            logger.info("The agent is vertifying if it was successful...");
            simulateObjectDetectionChanges(objectDetectionSituations.length - 1);
            informAgsEnvironmentChanged();
            serving = false;
            return true;
            }
            } else if (actionName.equals("askTheUtility")) {
            String askedObject = action.getTerm(0).toString();
            logger.info("The agent asked for the utility of the object: " + askedObject);
            answerTheUtilityOf(askedObject);
            return true;
            } else if (actionName.equals("requestObjectToServeWith")) {
            logger.info("The agent requested an object to serve coffee with");
            return true;
            } else if (actionName.equals("requestObjectToServeIn")) {
            logger.info("The agent requested an object to serve coffee in");
            return true;
            } else if (actionName.equals("serveClient")) {
            synchronized (lock) {
            serving = true;
            String objectToServeWith = action.getTerm(0).toString();
            String objectToServeIn = action.getTerm(1).toString();
```

```
            logger.info("The agent served the client using " + objectToServeWith + " to fill " + objectToServeIn);
            return true;
            }
            }

            logger.info("executing: " + action + ", but not implemented!");
            return false;
    }

    private void answerTheUtilityOf(String askedObject) {
            String percept = "unkownUtility";
            if (askedObject.equals("teapot") || askedObject.equals("kettle")) {
            percept = "canServeWith";
            } else if (askedObject.equals("mug") || askedObject.equals("coffee_cup") || askedObject.equals("cup"))
{
            percept = "canServeIn";
            }
            String newPerceptText = percept + "(" + askedObject + ")";
            logger.info("Answering: " + newPerceptText);
            addPercept(Literal.parseLiteral(newPerceptText));
    }

    private void simulateObjectDetectionChanges() {
            synchronized (lock) {
            if (!serving) {
             simulateObjectDetectionChanges(-1);
            } else {
            logger.info("The agent1 is serving the client, waiting for simulate new changes...");
            }
            }
    }

    private void simulateObjectDetectionChanges(int situationIndex) {
            // Simulates object detection changes for the next execution
            logger.info("Simulating object detection changes...");

            if (situationIndex == -1) {
            currentSituation++;

            if (currentSituation == objectDetectionSituations.length)
            currentSituation = 0;

            situationIndex = currentSituation;
            }

            String situation = objectDetectionSituations[situationIndex];
            logger.info("Running the situation: " + situation);

            try {
            Runtime.getRuntime().exec("cmd /c start /B " + situation);
            Thread.sleep(1000);
            } catch (IOException e) {
            e.printStackTrace();
            } catch (InterruptedException e) {
            e.printStackTrace();
            }
    }

    private class ChangeSituationTask extends TimerTask {
```

```
        @Override
        public void run() {
        simulateObjectDetectionChanges();
        }

    }
}
```

D.3.3 Arquivo .mas2j

```
MAS sgmodel_v2 {

        infrastructure: Centralised

        environment: SGEnvironmentV2

        agents:
        agent1 smartcoffeemachine_agent;

        aslSourcePath:
        "src/asl";
}
```

D.4 Projeto sgmodel_v3

D.4.1 Código fonte do agente smartcoffeemachine_agent.asl

```
/* Initial beliefs and rules */

detectedAndCanServeWith(Object) :- detected(Object) & canServeWith(Object).
detectedAndCanServeIn(Object) :- detected(Object) & canServeIn(Object).

isAbleToServe :- detectedAndCanServeWith(ObjectToServeWith) &
detectedAndCanServeIn(ObjectToServeIn).

canServeWith(kettle).
canServeIn(coffee_cup).

/* Initial goals */

/* Plans */

+configure(Json) <- sglib.configureGrounders(Json);
                    configured.

+start <- .print("Hello, just give me a kettle and a coffee cup and I'll pour you some coffee.").

+canServeWith(Object).
-canServeWith(Object).

+canServeIn(Object).
-canServeIn(Object).

+detected(woman) <- +detected(person).
+detected(man) <- +detected(person).
```

```
-detected(woman) <- -detected(person).
-detected(man) <- -detected(person).

+detected(person) <- !serve.
-detected(person) <- .remove_plan(serve).

+detected(Object) : not canServeWith(Object) & not canServeIn(Object) & not (Object == coffee) <-
askTheUtility(Object).
+detected(Object) : detected(person) <- !serve.

+!serve : isAbleToServe & detected(person) <- .findall(O, detectedAndCanServeWith(O), L1)
                                              .nth(0, L1, ObjToServeWith)
                                                    .findall(O, detectedAndCanServeIn(O), L2)
                                                    .nth(0, L2, ObjToServeIn)

                                                    sglib.groundSymbol(detected(ObjToServeWith),
ObjToServeWithGrounds);
                                              if (.length(ObjToServeWithGrounds) > 0) {
                                                    sglib.groundSymbol(detected(ObjToServeIn),
ObjToServeInGrounds);
                                                    if (.length(ObjToServeInGrounds) > 0) {
                                                    .print("It will serve with ", ObjToServeWith, " in ",
ObjToServeIn)
                                                          !serveWith(ObjToServeWith, ObjToServeIn)

                                                    } else {
                                                          .print("Agent has detected(", ObjToServeIn,") to
serve with, but it's not grounded. Therefore, it will not be considered valid to be used.")
                                                    }
                                              } else {
                                                    .print("Agent has detected(", ObjToServeWith,") to serve
in, but it's not grounded. Therefore, it will not be considered valid to be used.")
                                              }.

+!serve : not detected(person).

+!serve : detectedAndCanServeIn(Object) <- requestObjectToServeWith.

+!serve : detectedAndCanServeWith(Object) <- requestObjectToServeIn.

+!serve <- .print("Waiting for the objects to serve the client with.").

+!serveWith(ObjToServeWith, ObjToServeIn) : not serving
                                              <- +serving;
                                              .print("Serving the client...");
                                             .print("Using: ", ObjToServeWith);
                                                    .print("To fill: ", ObjToServeIn);
                                                    serveClient(ObjToServeWith, ObjToServeIn);
                                                    .print("generating event o check if was successful... (now
+30 s)");
                                                    .at("now +30 s",
{+!refreshPerceptionsAndCheckSuccessful}).

+!serveWith(ObjToServeWith, ObjToServeIn) : serving.

+!refreshPerceptionsAndCheckSuccessful <- .print("The agent will verify that it was successful...");
                                              verifyThatWasSuccessful;
                                              .print("Verfied on environment. Checking successful based on
beliefs...");
                                              !checkSuccessful;
```

```
                                        -serving.

+!checkSuccessful : detected(coffee) <- .print("The client was served successfully.").

+!checkSuccessful : not detected(coffee) <- .print("Oops! There was some problem when serving the client.").
```

D.4.2 Classe SGEnvironmentV3

```
import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;
import java.util.Timer;
import java.util.TimerTask;
import java.util.logging.Logger;

import jason.asSyntax.ASSyntax;
import jason.asSyntax.Literal;
import jason.asSyntax.StringTermImpl;
import jason.asSyntax.Structure;
import jason.asSyntax.parser.ParseException;
import jason.asSyntax.parser.TokenMgrError;
import jason.environment.Environment;
import sglib.configuration.SGLibConfigurationBuilder;
import sglib.providers.objectDetection.ObjectDetectionGroundProvider;
import sglib.providers.objectDetection.SGLibObjectDetectionModule;
import sglib.utils.JasonUtils;

public class SGEnvironmentV3 extends Environment {

    private Logger logger = Logger.getLogger("sgmodel_v3." + SGEnvironmentV3.class.getName());
    private boolean serving = false;
    private Object lock = new Object();
    private Timer timer;
    private int simulateChangesIntervalInSeconds = 15;
    private int currentSituation = -1;
    private String[] objectDetectionSituations = {
            "C:\\UFSC\\TCC\\test-images\\1.person.bat",
            "C:\\UFSC\\TCC\\test-images\\2.person_kettle.bat",
               "C:\\UFSC\\TCC\\test-images\\3.person_kettle_cup.bat",
            "C:\\UFSC\\TCC\\test-images\\3.person_kettle_cup.bat",
            "C:\\UFSC\\TCC\\test-images\\4.person_kettle_coffee_cup.bat"
    };
    private Map<String, Literal>[] perceptsSituations;
    private Map<String, Literal> currentDetections = new HashMap<>();

    /** Called before the MAS execution with the args informed in .mas2j */
    @Override
    public void init(String[] args) {
            super.init(args);
            generatePerceptsSituations();

            try {
            configureAgents();
            } catch (ParseException e) {
            e.printStackTrace();
            } catch (TokenMgrError e) {
            e.printStackTrace();
            }
```

```
        this.timer = new Timer();
        this.timer.schedule(new ChangeSituationTask(), 10 * 1000, simulateChangesIntervalInSeconds *
1000);
    }

    @SuppressWarnings("unchecked")
    private void generatePerceptsSituations() {
        perceptsSituations = (Map<String, Literal>[]) new Map[5];
        perceptsSituations[0] = createPerceptsMap("detected(kettle)");
        perceptsSituations[1] = createPerceptsMap("detected(person)", "detected(coffee_cup)");
        perceptsSituations[2] = createPerceptsMap("detected(person)", "detected(kettle)", "detected(cup)");
        perceptsSituations[3] = createPerceptsMap("detected(person)", "detected(kettle)",
"detected(coffee_cup)");
        perceptsSituations[4] = createPerceptsMap("detected(person)", "detected(kettle)", "detected(coffee)");
    }

    private HashMap<String, Literal> createPerceptsMap(String... percepts) {
        HashMap<String, Literal> map = new HashMap<String, Literal>();
        for (String percept : percepts) {
        Literal litPercept = Literal.parseLiteral(percept);
        map.put(JasonUtils.generateLiteralId(litPercept), litPercept);
        }
        return map;
    }

    private void configureAgents() throws ParseException, TokenMgrError {
        Literal cfg = ASSyntax.parseLiteral("configure");
        cfg.addTerm(new StringTermImpl(getConfiguration()));
        addPercept(cfg);
    }

    private String getConfiguration() {
        SGLibObjectDetectionModule module = new SGLibObjectDetectionModule();

        module.setHost("localhost");
        module.setServerPort(50000);
        module.setListenerPort(49999);
        module.setConfigFilePath("D:\\GitHub\\tcc\\object_detection\\config.json");

        return new
SGLibConfigurationBuilder().withProvider(ObjectDetectionGroundProvider.class.getTypeName())
        .withModule(module).toJson();
    }

    @Override
    public boolean executeAction(String agName, Structure action) {
        String actionName = action.getFunctor();
        if (actionName.equals("configured")) {
        addPercept(agName, Literal.parseLiteral("start"));
        return true;
        } else if (actionName.equals("verifyThatWasSuccessful")) {
        logger.info("The agent is vertifying if it was successful...");
        simulateObjectDetectionChanges(objectDetectionSituations.length - 1);
        informAgsEnvironmentChanged();
        timer.schedule(new FinishServingTask(), 60 * 1000);
        return true;
        } else if (actionName.equals("askTheUtility")) {
        String askedObject = action.getTerm(0).toString();
        logger.info("The agent asked for the utility of the object: " + askedObject);
```

```
            answerTheUtilityOf(askedObject);
            return true;
            } else if (actionName.equals("requestObjectToServeWith")) {
            logger.info("The agent requested an object to serve coffee with");
            return true;
            } else if (actionName.equals("requestObjectToServeIn")) {
            logger.info("The agent requested an object to serve coffee in");
            return true;
            } else if (actionName.equals("serveClient")) {
            synchronized (lock) {
            serving = true;
            String objectToServeWith = action.getTerm(0).toString();
            String objectToServeIn = action.getTerm(1).toString();
            logger.info("The agent served the client using " + objectToServeWith + " to fill " + objectToServeIn);
            return true;
            }
            }

            logger.info("executing: " + action + ", but not implemented!");
            return false;
    }

    private void answerTheUtilityOf(String askedObject) {
            String percept = "unkownUtility";
            if (askedObject.equals("teapot") || askedObject.equals("kettle")) {
            percept = "canServeWith";
            } else if (askedObject.equals("mug") || askedObject.equals("coffee_cup") || askedObject.equals("cup"))
{
            percept = "canServeIn";
            }
            String newPerceptText = percept + "(" + askedObject + ")";
            logger.info("Answering: " + newPerceptText);
            addPercept(Literal.parseLiteral(newPerceptText));
    }

    private void simulateObjectDetectionChanges() {
            synchronized (lock) {
            if (!serving) {
             simulateObjectDetectionChanges(-1);
            } else {
            logger.info("The agent1 is serving the client, waiting for simulate new changes...");
            }
            }
    }

    private void simulateObjectDetectionChanges(int situationIndex) {
            // Simulates object detection changes for the next execution
            logger.info("Simulating object detection changes...");

            if (situationIndex == -1) {
            currentSituation++;

            if (currentSituation == objectDetectionSituations.length)
            currentSituation = 0;

            situationIndex = currentSituation;
            } else {
            currentSituation = situationIndex;
            }
```

```
        String situation = objectDetectionSituations[situationIndex];
        logger.info("Running the situation: " + situation);

        try {
        Runtime.getRuntime().exec("cmd /c start /B " + situation);
        Thread.sleep(1000);
           updatePerceptions();
        } catch (IOException e) {
        e.printStackTrace();
        } catch (InterruptedException e) {
        e.printStackTrace();
        }
}

private void updatePerceptions() {
        // Changes perceptions using predefined situations to simulate ungrounded
        // percepts
        logger.info("Updating percepts...");
        Map<String, Literal> newDetections = perceptsSituations[currentSituation];

        logger.info(newDetections.size() + " new detections read.");
        ArrayList<Literal> toRemove = new ArrayList<>();
        ArrayList<Literal> toAdd = new ArrayList<>();

        for (String key : newDetections.keySet()) {
        if (!this.currentDetections.containsKey(key)) {
        toAdd.add(newDetections.get(key));
        }
        }

        for (String key : this.currentDetections.keySet()) {
        if (!newDetections.containsKey(key)) {
        toRemove.add(this.currentDetections.get(key));
        }
        }

        logger.info(toRemove.size() + " percepts to remove.");
        for (Literal percept : toRemove) {
        this.currentDetections.remove(JasonUtils.generateLiteralId(percept));
        removePercept(percept);
        }

        logger.info(toAdd.size() + " percepts to add.");
        for (Literal percept : toAdd) {
        this.currentDetections.put(JasonUtils.generateLiteralId(percept), percept);
        }
        addPercept(toAdd.toArray(new Literal[toAdd.size()]));
}

private class ChangeSituationTask extends TimerTask {

        @Override
        public void run() {
        simulateObjectDetectionChanges();
        }

}

private class FinishServingTask extends TimerTask {
```

```
        @Override
        public void run() {
        synchronized (lock) {
        serving = false;
        }
        }

    }
}
```

D.4.3 Arquivo .mas2j

```
MAS sgmodel_v3 {

        infrastructure: Centralised

        environment: SGEnvironmentV3

        agents:
        agent1 smartcoffeemachine_agent;

        aslSourcePath:
        "src/asl";
}
```

APÊNDICE D - Repositório público com o código fonte completo

O código fonte completo, incluindo os experimentos, está disponível também em um repositório público do GitHub, podendo ser acessado através do seguinte endereço: https://github.com/fsilvino/SymbolGroundingLibrary.

APÊNCICE E - Artigo

Modelo para embasamento simbólico fundamentado em teoria de Agentes BDI e Redes Neurais

Flávio Silvino[1]

[1]Departamento de Informática e Estatística – Universidade Federal de Santa Catarina (UFSC)
Caixa Postal 476 – 88.040-900 – Florianópolis – SC – Brasil

flavio.silvino@grad.ufsc.br

Abstract. *The inability of computers to relate symbols to their referents in the real world characterizes the Symbol Grounding Problem, defined by Harnad, in 1990. Since its definition, several solutions attempts have been proposed for the problem, and they can be classified into representationalist, semi-representationalist and non-representationalist approaches. Inspired by representationalist approaches, this work proposes a model for the symbol grounding based on the theory of BDI agents and connectionist models for object detection. To this end, the state of the art in models for symbol grounding, object detection and BDI agents is analyzed. A model that integrates parts of the agent's knowledge with a respective connectionist representation is proposed. A prototype of the model is presented and an experiment is defined for its evaluation. Finally, the results of the experiment are analyzed and the conclusions are presented. Despite contributing to the task of grounding symbols of the BDI agent, connecting them to referents in the real world, through the provision of a library with three models for grounding beliefs of a BDI agent that has an architecture that is easy to use and extend, this work does not solve the Symbol Grounding Problem, since the neural network of object detection needs to be trained with data that are provided extrinsically. Therefore, the Symbol Grounding Problem remains open for the exploration of future works, with the possibility of evolution of the model proposed in this work.*

Resumo. *A incapacidade dos computadores de relacionar os símbolos com os seus referentes no mundo real caracteriza o Problema do Embasamento Simbólico, definido por Harnad, em 1990. Desde a sua definição, diversas tentativas de soluções foram propostas para o problema, sendo que as mesmas podem ser classificadas em abordagens representacionalistas, semi-representacionalistas e não-representacionalistas. Inspirado nas abordagens representacionalistas, este trabalho propõe um modelo para o embasamento simbólico fundamentado em teoria de agentes BDI e modelos conexionistas para reconhecimento de objetos. Para tal, é analisado o estado da arte em modelos para embasamento simbólico, detecção de objetos e agentes BDI. Um modelo que integra partes do conhecimento do agente a uma respectiva representação conexionista é proposto. Um protótipo deste modelo é apresentado e um experimento é definido para a avaliação do mesmo. Por fim, é realizada a análise dos resultados do experimento e são apresentadas as conclusões. Apesar de contribuir para a tarefa de embasamento de símbolos do agente BDI, conectando-os aos referentes no mundo real, através da disponibilização de uma biblioteca com três modelos para embasamento de crenças de um agente BDI que possui uma arquitetura de fácil utilização e extensão, este trabalho não resolve o problema do embasamento simbólico, visto que a rede neural de detecção de objetos precisa ser treinada com dados que são fornecidos de forma extrínseca. Portanto, o Problema do Embasamento Simbólico segue em aberto para a exploração de*

trabalhos futuros, sendo que há a possibilidade de evolução do modelo proposto neste trabalho.

1. Introdução

A Inteligência Artificial (IA) abrange as mais diversas áreas, desde aprendizado e percepção até jogos de xadrez ou diagnóstico de doenças. Por automatizar as tarefas intelectuais, a IA possui grande relevância para qualquer área da atividade intelectual humana (NORVIG; RUSSEL, 2004). Segundo Luger (2009), a Inteligência Artificial pode ser definida como um ramo da ciência da computação que se preocupa com a automação do comportamento inteligente. Porém, a IA possui várias definições, visto que o tema é abordado de forma diferente por diferentes autores.

As definições de IA, em linhas gerais, dividem-se entre as que estão relacionadas a processos de pensamento e raciocínio e as que se referem ao comportamento. Além disso, as definições medem o sucesso de maneiras diferentes, podendo fazê-lo em termos da fidelidade ao desempenho humano ou comparando-o a um conceito ideal de inteligência (NORVIG; RUSSEL, 2004).

Independentemente da definição, o estudo da Inteligência Artificial envolve a busca pelo entendimento do funcionamento do pensamento humano e, para tanto, conta com contribuições de diversas áreas como a Filosofia e a Psicologia. Segundo Searle (1980), a IA pode ser dividida em dois grandes grupos: a IA fraca e a IA forte. A primeira, caracteriza-se pela simulação do comportamento inteligente, ou seja, ela permite que sejam testadas hipóteses de como a mente funciona, mas não pretende alcançar a inteligência humana. Já a IA forte afirma que os programas, se desenvolvidos corretamente, são capazes de entender e de possuir estados cognitivos, assim como as mentes humanas. Um dos principais desafios é entender como a mente humana atribui significado aos símbolos e como funciona o processo de cognição.

Símbolos normalmente são uma convenção formal de notação utilizada pelos usuários do sistema simbólico ao qual fazem parte e não estão ligados aos seus respectivos significados (HARNAD, 1990). Em 1990, Harnad formulou o que chamou de “Symbol Grounding Problem” (Problema do Embasamento Simbólico). Ele coloca o problema em duas perguntas principais: “Como a interpretação semântica de um sistema formal de símbolos pode ser intrínseca ao sistema, em vez de apenas parasitária nos significados de nossas cabeças? Como os significados de símbolos sem sentido, manipulados exclusivamente com base em suas formas (arbitrárias), podem ser fundamentados em qualquer coisa, exceto em outros símbolos sem significado?”. Em resumo, o Problema do Embasamento Simbólico questiona como desenvolver um sistema inteligente que seja capaz de atribuir significado a um símbolo de maneira autônoma, sem qualquer influência externa.

As propostas de soluções para o Problema do Embasamento Simbólico são organizadas em três abordagens principais: as representacionalistas, as semi-representacionalistas e as não representacionalistas (TADDEO; FLORIDI, 2005). As abordagens representacionalistas, em especial, buscam resolver o problema através da conexão das percepções do agente com os símbolos. Tratando-se de processamento de percepções, as abordagens de inteligência artificial conexionista, especialmente as Redes Neurais, possuem grandes vantagens em relação às abordagens simbólicas.

Com base nesta abordagem, o presente trabalho tem o objetivo de propor um modelo que aplica a abordagem representacionalista, realizando o embasamento simbólico das crenças de um Agente BDI através do uso de Redes Neurais que farão parte das percepções do agente.

2. Fundamentação teórica

2.1. Problema do embasamento simbólico

Definido por Harnad, em 1990, o Symbol Grounding Problem (Problema do Embasamento Simbólico) questiona como os símbolos são conectados com os seus respectivos significados, de forma que não estejam embasados apenas em outros símbolos sem embasamento. Em outras palavras, o problema questiona como a interpretação semântica de um sistema formal de símbolos pode se tornar intrínseca ao sistema e não depender dos significados em nossas cabeças.

Ao definir o Problema do Embasamento Simbólico, Harnad (1990) também propõe um caminho para uma possível solução. Visando beneficiar-se do melhor dos dois mundos, Harnad propõe a utilização de um modelo híbrido de Inteligência Artificial (IA), combinando a IA Simbólica com a IA Conexionista. A primeira possui a capacidade de manipulação de símbolos e simulação de raciocínio, e a segunda está mais próxima daquilo que se conhece sobre o funcionamento do cérebro humano e possui a capacidade de aprendizado de padrões, podendo ser utilizada para a extração de características invariantes das suas projeções sensoriais. Através do relacionamento daquilo que é percebido através de sensores com seus referentes simbólicos, os símbolos estariam então embasados com seus referentes no mundo real.

Taddeo e Floridi (2005) classificaram as soluções propostas para o Problema do Embasamento Simbólico em três abordagens: representacionalista, semi-representacionalista e não-representacionalista. A abordagem representacionalista, segundo os autores, procura resolver o problema fundamentando os símbolos de um Agente Artificial com base nas representações extraídas dos dados perceptivos do agente. Já a abordagem semi-representacionalista, ainda possui como base a abordagem representacionalista, porém baseia-se nos princípios da robótica baseada em comportamento para realizar o aprendizado e o embasamento dos símbolos. Por fim, a abordagem não-representacionalista defende que não é necessária uma representação simbólica, pois o comportamento inteligente pode ser resultado de interações entre um Agente Artificial corporificado e situado em seu ambiente, bastando acoplamentos sensório-motores.

2.2. Redes neurais e detecção de objetos

Os neurônios artificiais, que compõem as Redes Neurais, são um modelo matemático inspirado nos neurônios biológicos presentes no cérebro humano. Basicamente eles são ativados quando uma combinação linear de suas entradas excede algum limiar, sendo que a esta soma é aplicada uma função de ativação. As Redes Neurais Artificiais consistem na combinação de vários neurônios artificiais (NORVIG; RUSSEL, 2004). Segundo Luger (2009), um dos principais pontos fortes das redes neurais é que, por serem geralmente treinadas ou condicionadas, ao invés de programadas explicitamente, as redes neurais são capazes de capturar invariâncias no mundo se possuírem uma arquitetura de rede e um algoritmo de aprendizagem apropriadamente projetados.

Com a evolução dos estudos sobre as redes neurais e com o aumento da disponibilidade de poder computacional, diversas arquiteturas e algoritmos de aprendizagem foram desenvolvidos e o campo de aplicação das mesmas foi extensamente ampliado. Uma importante aplicação das redes neurais, na área de visão computacional, é a detecção de objetos.

JIAO et. al. (2019), afirmam que:

> "A detecção de objetos é uma tecnologia computacional relacionada à visão computacional e ao processamento de imagens que trata da detecção de instâncias de objetos semânticos de uma determinada classe (como humanos, edifícios ou carros) em imagens e vídeos digitais. Domínios bem pesquisados de detecção de objetos incluem detecção de multicategorias, detecção de bordas, detecção de objetos salientes, detecção de pose, detecção de texto de cena, detecção de rosto e detecção de pedestres, etc. Como uma parte importante da compreensão de cena, a detecção de objeto tem sido amplamente utilizada em muitos campos da vida moderna, como campo de segurança, campo militar, campo de transporte, campo médico e campo de vida."

O uso de Redes Neurais Convolucionais (Convolutional Neural Networks CNN, em inglês) na tarefa de detecção de objetos proporcionou um grande avanço na área e diversos detectores foram propostos desde então. As redes neurais convolucionais iniciam a detecção a partir de pequenas regiões nas imagens, identificando features simples e realizando composições das mesmas para então chegar a uma classificação mais abstrata da classe que está sendo detectada. Por exemplo, para detectar um quadrado, uma CNN poderia identificá-lo como uma composição de quatro retas em uma determinada disposição. Por tratarem de pequenas regiões da imagem e manterem os pesos para os vizinhos da próxima camada, as CNNs reduzem drasticamente a complexidade da rede, diminuindo o número de conexões necessárias entre os neurônios e, além disso, permitem detectar os objetos independentemente da sua localização na imagem (ALBAWI; MOHAMMED; AL-ZAWI, 2017).

Durante anos de pesquisas, os detectores de objetos foram aprimorados muitas vezes a partir de detectores propostos anteriormente. O detector de objetos Faster R-CNN, proposto por Ren et. al. (2017), introduziu as chamadas Region Proposal Networks (RPN), que são formadas por uma rede totalmente convolucional e que compartilham as camadas convolucionais com um detector de objetos Fast R-CNN. Estas redes possuem o papel de gerar as regiões nas quais o detector de objetos realizará a busca pelos objetos, atribuindo um score a cada região. Através do compartilhamento das camadas convolucionais, os autores conseguiram aumentar a performance final do processo de detecção (REN et. al., 2017).

2.3. Agentes BDI

A definição de agentes inteligentes não é única, pois muitas delas baseiam-se no objetivo de sua aplicação. Segundo Wooldridge (2002), um agente "é um sistema de computador que está situado em algum ambiente, e que é capaz de ação autônoma nesse ambiente a fim de atender aos seus objetivos de projeto". Segundo Wooldridge e Jennings (1995), apesar das diversas definições para agentes, algumas propriedades devem estar presentes nos mesmos, sendo elas: 1) autonomia: o agente possuir algum tipo de controle sobre suas ações e seu estado interno e deve operar sem intervenção direta de humanos ou outros agentes; 2) habilidade social: ele deve comunicar-se com outros agentes e, possivelmente, humanos; 3) reatividade: o agente deve perceber o ambiente em que se encontra e reagir a tempo às mudanças; e 4) proatividade: o agente não pode apenas reagir às mudanças, mas também deve tomar iniciativas para alcançar seus objetivos.

A arquitetura de agentes BDI (Belief-Desire-Intention) foi inspirada no trabalho de Bratman (1987 apud Silva, Meneguzzi, Logan, 2020), que afirmou que o raciocínio prático pode ser visto como o ato de pesar múltiplas considerações conflitantes a favor ou contra escolhas conflitantes, à luz do que o agente acredita, valoriza e se preocupa. O raciocínio pode ser dividido em duas etapas, sendo a primeira a deliberação, onde o agente decide qual estado deseja alcançar com base em seus desejos, e a segunda etapa, conhecida como raciocínio meios-

fins, onde o agente decide como irá fazer para alcançar tal estado, ou seja, ele escolhe um plano de ação (SILVA; MENEGUZZI; LOGAN, 2020).

As crenças (beliefs) do agente são as informações sobre o ambiente que o agente possui e são atualizadas por meio de ações de detecção. É através delas que o agente se torna capaz de identificar a possibilidade ou não de realizar ações para alcançar seus objetivos e entender os efeitos causados pelas mesmas. Os desejos (desires) do agente são os objetivos a serem alcançados e estão associados a prioridades ou recompensas. Para alcançar seus objetivos, o agente precisa analisar o ambiente para identificar se possui condições de realizar determinadas ações em busca dos mesmos. Porém, por se tratar de questões não-determinísticas, é possível que imediatamente antes ou durante uma ação do agente ocorra algum evento que altere o ambiente, e que as condições impeçam a realização da ação. Por isso, é necessário que o agente possua uma forma de verificar tal situação sem que passe todo o tempo apenas verificando-as e nunca realizando suas ações. Por este motivo, as intenções (intentions) representam a escolha atual da ação a ser tomada pelo agente, que é baseada em seus desejos (RAO; GEORGEFF, 1995).

Bordini e Hübner (2006) desenvolveram uma plataforma de desenvolvimento de agentes BDI, chamada Jason, baseada na linguagem AgentSpeak em que forneceram diversas extensões que permitem o desenvolvimento prático de sistemas multi-agentes, visto que a linguagem AgentSpeak é apenas abstrata, realizando também algumas adaptações na linguagem.

Jason é open source, foi desenvolvido em Java e possui diversas features. Dentre elas: comunicação interagente baseada em atos de fala, anotações em rótulos de planos, possibilidade de executar um sistema multi-agente distribuído em rede e funções de seleção totalmente personalizadas.

3. Trabalhos relacionados

Em Moreno et. al. (2019), os autores defendem a utilização de um modelo híbrido de representação que promova uma integração efetiva entre os subsistemas simbólico e conexionista e seja capaz de manipulá-los de maneiras diferentes em um sentido mais amplo, fornecendo uma abstração que permite tal manipulação definindo a relação entre as estruturas simbólicas, as redes neurais e seus processadores correspondentes.

Para tanto, os autores propuseram uma representação híbrida capaz de combinar características específicas dos modelos, baseada nos fundamentos de nós e ligações. O foco do estudo foi em ligações SPO (sujeito-predicado-objeto) que são capazes de especificar fatos como em RDF (Resource Description Framework) e ligações causais onde há condições e ações envolvidas na relação. As ligações causais definem a execução da representação e as ligações SPO permitem o raciocínio sobre as ligações simbólicas e subsimbólicas. Basicamente, o trabalho realizado consiste em descrever os componentes de uma rede neural através de uma representação simbólica.

Em Raue et. al. (2015), os autores utilizaram duas redes neurais recorrentes do tipo LSTM para promover a associação entre entradas multimodais (neste caso entradas visuais e de áudio) que contém a mesma sequência semântica, porém sem qualquer segmentação prévia (diferente de outros trabalhos que utilizam entradas segmentadas). Ao combinar estas redes com técnicas de segmentação e de sincronização, os autores foram capazes de realizar um "coembasamento" das entradas visuais com as entradas auditivas.

O trabalho de Eichstaedt (2019) fez uma análise filosófica do Problema do Embasamento Simbólico e, através da utilização de agentes BDI em conjunto com uma rede

neural para reconhecimento de dígitos numéricos, o autor realizou o relacionamento das crenças do agente em relação à detecção do dígito com a representação conexionista do mesmo.

4. Desenvolvimento

Os primeiros passos do desenvolvimento do trabalho se deram através da pesquisa de redes neurais pré-treinadas para detecção de objetos, visando a sua utilização em testes dos modelos. Como o objetivo do trabalho não está relacionado com a tarefa de detecção, mas sim com a utilização do resultado desta para o embasamento de símbolos, optou-se por esta estratégia de implementação, focada na integração das IAs conexionista e simbólica.

Através da utilização de um código de exemplo, disponibilizado no repositório do GitHub do projeto Tensor Flow, foram testadas algumas redes neurais de detecção de objetos pré-treinadas, disponíveis também no repositório do projeto, sendo alguns exemplos: ssd_mobiledet_cpu_coco, ssd_resnet_101_fpn_oidv4 e faster_rcnn_inception_resnet_v2_atrous_oidv4. Por se tratarem de redes pré-treinadas, os objetos a serem detectados dependem fortemente do dataset utilizado no treino, e isso acarreta em uma restrição do contexto em que o agente BDI poderá ser desenvolvido. Devido à intenção de se utilizar uma rede Faster R-CNN, bem como de utilizar imagens de objetos comuns, optou-se pelo uso da rede faster_rcnn_inception_resnet_v2_atrous_oidv4, que foi treinada com o dataset Open Images, que possui uma grande variedade de objetos.

A partir da escolha da rede a ser utilizada, foi desenvolvido o módulo de detecção de objetos a ser utilizado no trabalho com base no código de exemplo mencionado anteriormente. Inicialmente, o código fonte foi alterado para realizar a leitura de imagens em um diretório e realizar a detecção dos objetos nestas imagens. Além disso, junto com a imagem de resultado da detecção de objetos (com as caixas de detecção desenhadas, junto com os nomes de objetos detectados) , foi programado para que fosse salvo também em disco um arquivo JSON com os detalhes do resultado da detecção dos objetos. O arquivo foi gravado para ser a fonte de informações para o agente BDI. Com esta parte desenvolvida, a rede foi testada com várias imagens.

Durante a implementação inicial do agente BDI, procurou-se realizar testes de possíveis abordagens para a elaboração do modelo para embasamento simbólico. Com isso, inicialmente foi desenvolvido um agente BDI com o objetivo de cortar objetos. O agente então possui planos para realizar esta tarefa e também para tentar descobrir se é possível realizar o corte de determinado objeto utilizando outro objeto disponível. Para isso, o agente executa ações para solicitar ao ambiente mais informações sobre os objetos detectados, como, por exemplo, saber se determinado objeto é cortável ou cortante, e também se determinado objeto cortável pode ser cortado por um objeto cortante disponível.

Para que o agente recebesse a informação dos objetos disponíveis, o ambiente Jason realizava periodicamente a leitura do arquivo JSON gerado pela detecção de objetos, assumindo que a mesma estivesse sempre atualizada (nesta fase inicial, a detecção de objetos era executada manualmente, através de uma linha de comando).

Após a finalização deste teste, foi desenvolvido um outro agente que seguiu a mesma proposta do anterior, porém possuía outro objetivo e também considerava o resultado da sua ação, tentando perceber no ambiente se sua ação foi efetiva. Este segundo agente BDI foi modelado para simular um robô que serve café e inicia com duas crenças básicas: a) é possível servir com uma chaleira; b) é possível servir em uma xícara. Assim que é iniciado, o agente realiza a ação externa de perceber o ambiente, que sinaliza para o ambiente Jason a necessidade de ler o arquivo JSON. O ambiente Jason por sua vez, realiza a leitura deste arquivo JSON

gerado pelo módulo de detecção de objetos e atualiza as crenças do agente, adicionando novos objetos detectados e removendo objetos que deixaram de ser detectados.

Figura 1. Imagem do resultado da detecção de objetos em uma imagem simulada.


```
[
    {
        "score": 0.9849456548690796,
        "class": {
            "id": 259,
            "name": "Kettle"
        },
        "box": [
            0.09529615193605423,
            0.5756107568740845,
            0.7043643593788147,
            0.9246169924736023
        ]
    },
```


Figura 2. Trecho do JSON gerado pelo módulo de detecção de objetos.

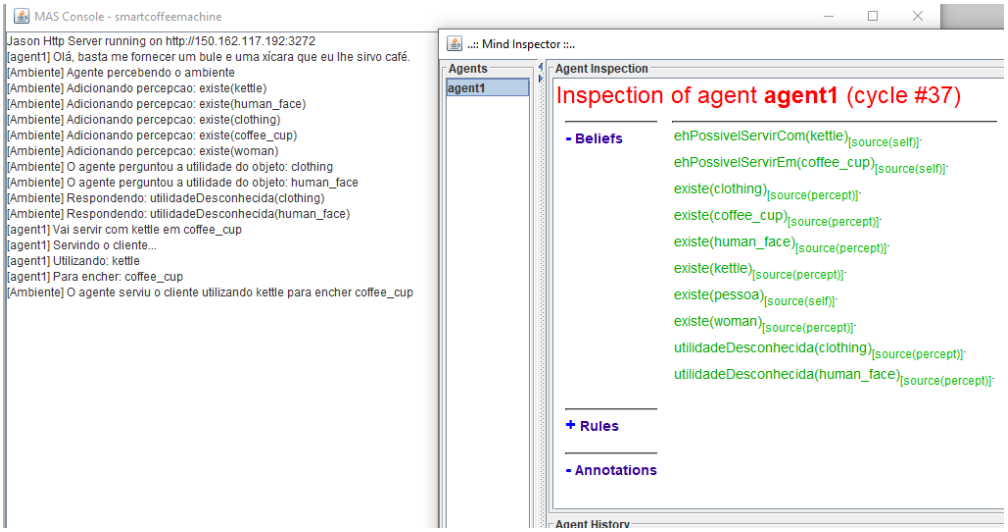


Figura 3. Teste de execução do agente BDI.

Após a análise inicial das possibilidades de implementação e abordagens para o desenvolvimento do modelo, estabeleceu-se que seria mantida a estratégia de geração de um arquivo JSON que seria gravado em um diretório pelo módulo de detecção de objetos (módulo conexionista) e lido pelo agente BDI (módulo simbólico). Para fins de simplificação, foi estabelecido que a comunicação entre os módulos seria feita de maneira simples, através de trocas de mensagens utilizando sockets. Com isso, ambos os módulos devem de alguma forma conhecer a priori a localização do arquivo JSON. Tais pressupostos foram estabelecidos exclusivamente com o intuito de simplificar a implementação daquilo que não faz parte dos objetivos do trabalho.

Além disso, algumas abordagens para o desenvolvimento foram esquematizadas e analisadas quanto às suas vantagens e desvantagens e as informações relativas a esta análise estão na Tabela 1.

Tabela 1. Análise de possíveis abordagens para o desenvolvimento do modelo

Estratégia	Vantagem	Desvantagem
Criar anotação automática de crenças [grounded]	Flexibilidade para o desenvolvedor	Precisa ter alguma forma de definir quando uma crença pode ser anotada como grounded e envolve a alteração da engine do Jason também
Ação interna que verifique se determinada crença está embasada ou não	Permite ao desenvolvedor utilizar o mecanismo na medida de sua necessidade	Vai demandar o desenvolvimento da conexão com a detecção de objetos
Classe de agente + Classe de Ambiente + Ação interna	Cria um modelo completo para realizar a comunicação com a detecção de objetos, trata a informação repassada pelo ambiente customizado (sobre o embasamento) e permite a verificação por parte do desenvolvedor através da ação interna	Obriga o desenvolvedor a utilizar uma classe de agente específica

4.1. Personalização da classe Agent e introdução de provedores de embasamento

Esta implementação visou analisar a possibilidade de personalização da classe Agent do Jason, com o objetivo de utilizar a função de atualização de crenças (*buf*) como ponto de partida para a conexão com a detecção de objetos através de Provedores de Embasamento. Tais provedores foram definidos de forma que podem ser implementados pelas mais diversas fontes de embasamento possíveis. Eles consistem em uma classe que fica responsável por receber uma lista de crenças e retorná-la com as devidas crenças anotadas. Neste trabalho, a implementação do provedor deu-se através da conexão, de forma limitada e simplificada, com o módulo de detecção de objetos. A Figura 4 ilustra, de maneira simplificada, o fluxo executado pelo agente, que e consiste em:

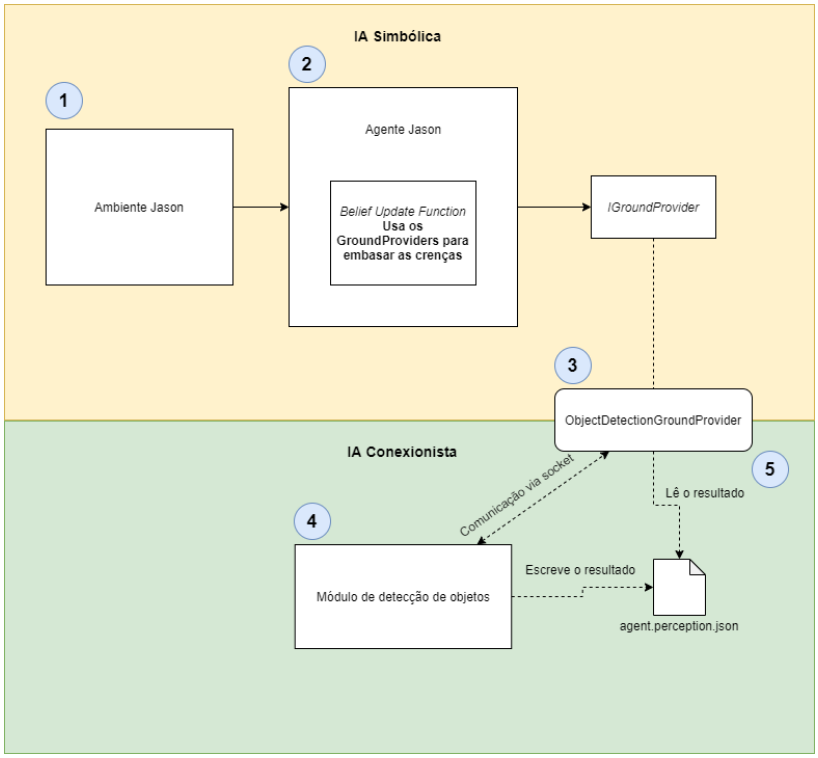


Figura 4. Modelo utilizando provedores de embasamento.

1. O ambiente Jason envia para o agente uma ou mais percepções.
2. Ao executar a função de atualização de crenças, o agente personalizado solicita a todos os provedores de embasamento que realizem o embasamento das crenças que foram recém recebidas pelo agente. Desta forma, o provedor de embasamento está livre para inserir novas crenças, além de embasar as crenças fornecidas pelo agente.
3. No caso do provedor de embasamento implementado neste trabalho, tal função é feita através da comunicação com o módulo de detecção de objetos, que foi desenvolvido em Python. O provedor de embasamento abre uma conexão via socket e envia um comando para o módulo de detecção de objetos para que este execute a detecção.
4. Após terminar a detecção, o módulo de detecção de objetos grava, em um arquivo JSON, o resultado da detecção e envia uma resposta para o provedor de embasamento, para que este possa ler o arquivo de resultado. É de se destacar aqui que o foco deste trabalho não está na integração entre as tecnologias, mas no modelo para o embasamento simbólico em si. Por este motivo, esta integração foi simplificada e não controla erros de

comunicação e também assume que tanto o provedor quanto o módulo de detecção de objetos conhecem o local do arquivo a ser gerado com o resultado.

5. Ao receber a resposta do módulo de detecção de objetos, o provedor de embasamento lê o arquivo de resultado da detecção e analisa quais percepções podem ser embasadas a partir das detecções realizadas pelo módulo correspondente. Em seguida, o provedor adiciona anotações nas crenças que podem ser embasadas. No caso deste provedor, optou-se por incluir duas anotações: *[grounded]*, que indica que aquela crença está embasada (e tal anotação foi definida como um padrão para o embasamento) e, além desta, a anotação *[visual]* para indicar que o embasamento daquela crença está sendo feito com base em uma percepção visual.

4.2. Personalização da classe Environment e introdução de provedores de percepções embasadas

Outra abordagem experimentada foi a da personalização da classe Environment do Jason, permitindo o uso de Provedores de Percepções Embasadas. Neste modelo, ao invés de ser o agente o responsável pela busca do embasamento de suas crenças, o ambiente é quem recebe as percepções embasadas e as repassa aos agentes. Desta forma, os Provedores de Percepções Embasadas possuem um papel ativo, diferente do modelo anterior em que eram requisitados pelo agente. A Figura 5 ilustra o modelo, que consiste em:

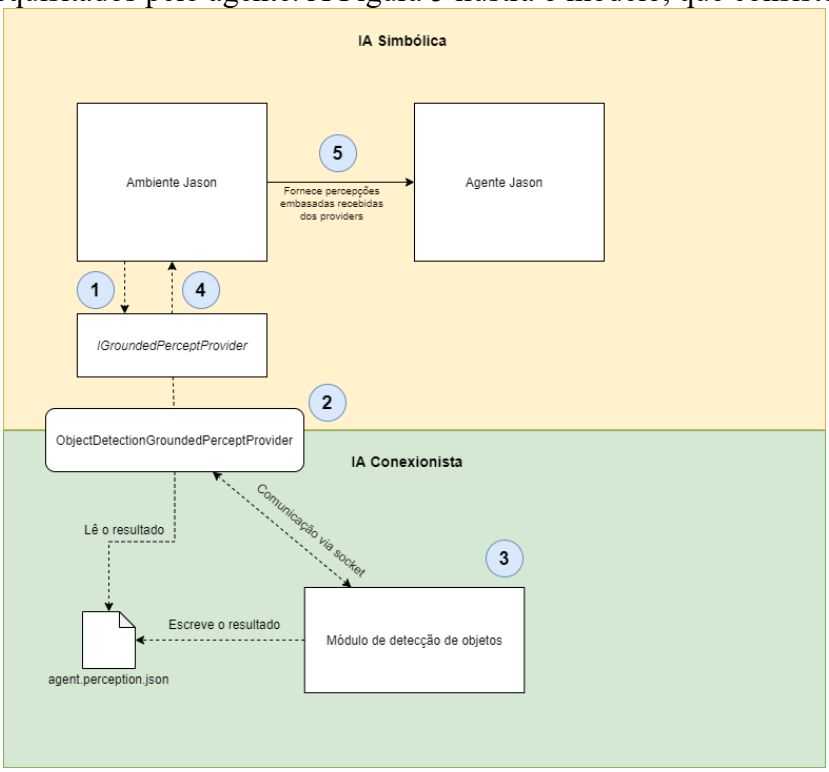


Figura 5. Modelo utilizando provedores de percepções embasadas.

1. O ambiente Jason cria seus Provedores de Percepções Embasadas e os inicia.
2. O Provedor de Percepção Embasada, ao ser iniciado pelo ambiente, começa a realizar seu processamento. No caso deste trabalho, foi realizada a implementação de um provedor de percepções através da detecção de objetos. O provedor realiza a comunicação com o módulo de detecção de objetos periodicamente, enviando um comando através de uma conexão via socket, para que o módulo de detecção execute a mesma.

3. Ao receber o comando para realizar a detecção, o módulo de detecção de objetos realiza sua tarefa, grava o resultado da detecção no arquivo JSON e responde ao provedor de percepções embasadas, para que este realize a leitura do resultado.
4. Ao receber a resposta do módulo de detecção de objetos, o provedor de percepções embasadas realiza a leitura do resultado e envia as percepções para o ambiente Jason, devidamente anotadas (com *[grounded]* para indicar o embasamento e *[visual]* para indicar que é uma percepção visual).
5. Ao receber as percepções embasadas dos provedores de percepções embasadas, o ambiente Jason as encaminha para o agente, que pode tratá-las da forma que for conveniente, utilizando as anotações para a tomada de decisões.

4.3. Ação interna para embasamento de símbolos

Para o desenvolvimento do modelo de embasamento através da utilização de ação interna foi necessário realizar alguns testes para definir de qual a seria a funcionalidade da ação, quais seriam seus parâmetros e como ela poderia ser utilizada.

Inicialmente considerou-se a criação de uma ação interna *sglib.isGrounded(s)*, onde *s* seria o parâmetro com o símbolo a ser embasado, e a ação retornaria *true* ou *false*. Porém, ao realizar os testes iniciais, constatou-se que o Jason considera que a ação interna falhou caso seu retorno seja *false*, o que inviabilizou esta implementação.

Uma segunda possibilidade analisada foi a de retornar o símbolo passado por parâmetro para a ação interna com as anotações sobre o embasamento, no mesmo padrão utilizado nas abordagens anteriores, ou seja, uma anotação *grounded* e outra *visual* para o caso da detecção de objetos. Porém, esta implementação tornaria complexa a utilização dentro do código do agente e por este motivo foi também descartada.

Por fim, a implementação escolhida foi a de uma ação interna chamada *sglib.groundSymbol(s,G)*, onde *s* é o símbolo a ser embasado e *G* é a variável que receberá a lista de embasamentos daquele símbolo. Caso o símbolo não esteja embasado, a lista estará vazia. Dessa forma, ao receber a lista em uma variável, o desenvolvedor possui a flexibilidade de utilizar os dados da lista conforme sua necessidade. Caso a intenção seja apenas verificar se determinado símbolo está embasado, basta verificar se a lista não está vazia. Além disso, se for necessário tomar decisões com base em qual tipo de embasamento está disponível, o desenvolvedor pode procurar na lista o tipo desejado, como por exemplo '*visual'*.

O desenvolvimento desta ação interna foi realizado em duas iterações. Isto porque para realizar o embasamento do símbolo recebido por parâmetro, a ação interna utiliza provedores de embasamento. Como o objetivo é deixar o modelo extensível, na primeira iteração o detentor dos provedores de embasamento era o agente, que deveria implementar uma interface que permitia à ação interna solicitar o embasamento dos símbolos. Com isso, o desenvolvedor poderia configurar, através da classe do agente, quais provedores de embasamento seriam utilizados pelo mesmo.

Esta iteração serviu como prova de conceito para o uso da ação interna e permitiu a sua validação com um fluxo completo de execução. Porém, ela apresenta alguns pontos negativos em relação à generalização do modelo e também delega mais responsabilidades para o desenvolvedor, o que se deseja reduzir. O principal ponto negativo é a necessidade de implementação da interface *IGroundedAgent* na classe de todos os agentes que visem utilizar a ação interna. Isto torna o modelo menos flexível e demanda um trabalho inicial maior para a sua utilização. Para resolver estes problemas, foram analisadas possíveis soluções para realizar a configuração dos provedores a serem utilizados pela ação interna, de tal modo que esta fosse capaz de realizar o embasamento sem depender diretamente do agente que a está executando.

As ações internas no Jason são implementadas através da criação de uma classe que herda de *DefaultInternalAction*. Cada agente, ao executar a ação pela primeira vez, cria uma instância da classe da ação interna e utiliza esta única instância durante todo o seu ciclo de vida. Isto permite que a ação possua um estado por agente, ou seja, é possível guardar dados relativos à sua execução naquele agente ou qualquer outra informação que seja conveniente. Considerando esta característica, identificou-se a possibilidade de guardar na instância da ação interna as configurações dos provedores de embasamento e demais configurações necessárias para seu funcionamento.

Visando enviar para o agente a configuração da ação interna *sglib.groundSymbol(s,G)*, foi desenvolvida uma segunda ação interna chamada *sglib.configureGrounders(j)*, onde *j* é uma string contendo um JSON com as configurações que informam quais os provedores de embasamento devem ser utilizados por aquele agente e quais são as configurações dos módulos dos quais os providers selecionados dependem.

A função da ação interna *sglib.configureGrounders(j)* é enviar para a instância da ação interna *sglib.groundSymbol(s,G)* a configuração informada. Para isto, a ação interna de configuração obtém a instância da ação de embasamento acessando o agente e enviando o JSON de configuração através de um método disponível na classe da ação interna de embasamento de símbolos. Após executar a ação de configuração, o agente está pronto para utilizar a ação interna de embasamento de símbolos, independentemente da sua classe ou do ambiente no qual está situado.

A Figura 6 ilustra a segunda versão do modelo de embasamento através da utilização de ação interna, que consiste em:

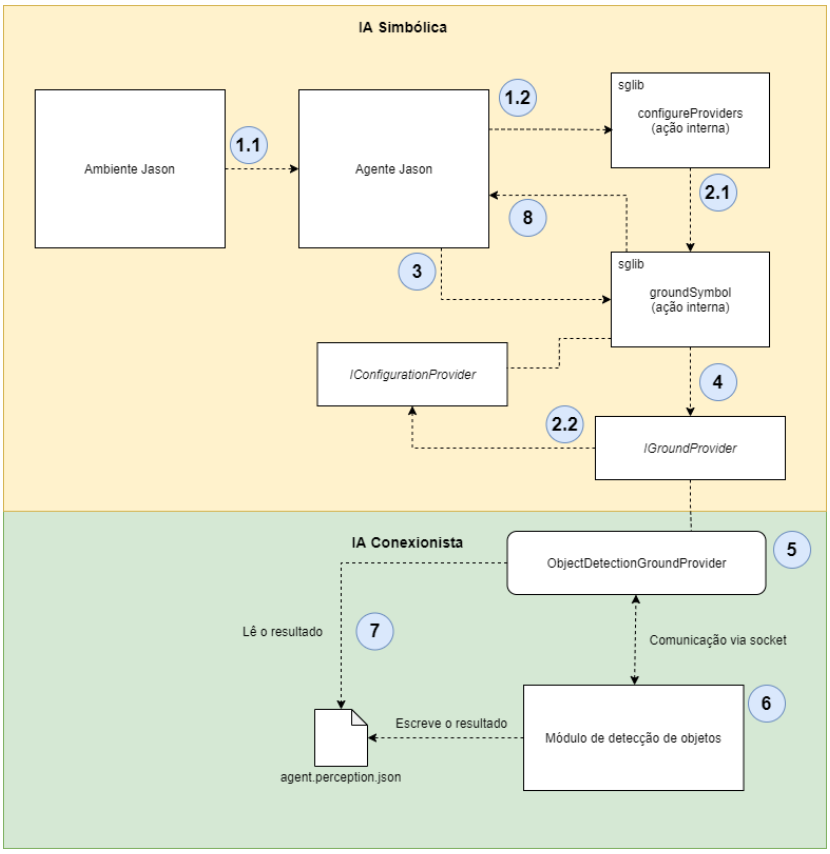


Figura 6. Segunda versão do modelo com ação interna de embasamento de símbolos.

1. A primeira etapa necessária para a utilização deste modelo pode ser executada de duas maneiras, sendo de livre escolha do desenvolvedor:

   1.1. Através do ambiente, utilizando um código fonte em Java, o desenvolvedor gera o JSON com as configurações e, através da adição de uma percepção, envia a configuração para o agente, que por sua vez deve possuir um plano para o evento de adição desta crença que irá executar a ação interna *sglib.configureGrounders(j)*. A geração do JSON pode ser feita manualmente, o arquivo pode ser lido do disco, ou ainda é possível utilizar um construtor de configurações que foi desenvolvido para facilitar o uso da biblioteca. A classe *SGLibConfigurationBuilder* pode ser instanciada e o desenvolvedor pode adicionar os nomes dos provedores a serem utilizados, bem como as configurações dos módulos, instanciando objetos do tipo *SGLibModule* ou tipos derivados deste, como por exemplo o *SGLibObjectDetectionModule* que já possui métodos para facilitar a configuração do módulo de detecção de objetos já desenvolvido neste trabalho.

   1.2. Este passo deve ser obrigatoriamente executado, ele pode ser programado para ser executado diretamente pelo agente, através do código *asl*, passando o JSON diretamente para a ação interna de configuração, ou através do evento gerado pelo passo 1.1. Aqui o agente executa a ação interna de configuração e passa o JSON.

2. A ação interna de configuração envia para a ação interna de embasamento de símbolos o JSON de configuração.

   2.1. A ação interna por sua vez lê as configurações e inicia os providers e módulos necessários para o seu funcionamento. A ação interna *sglib.groundSymbol(s,G)* implementa a interface *IConfigurationProvider* para que os provedores de embasamento possam buscar as configurações relevantes para o seu funcionamento.

   2.2. Os provedores de embasamento então buscam na ação interna, através da interface *IConfigurationProvider*, as configurações que são necessárias para o seu funcionamento.

3. Após realizar a configuração, o agente está livre para executar a ação interna de embasamento de símbolos. A ação por sua vez executa todos os provedores de embasamentos configurados para serem utilizados e realiza o embasamento propriamente dito.

4. Neste trabalho, a ação interna irá executar o provedor de embasamento de detecção de objetos.

5. O provedor de embasamento de detecção de objetos realiza a comunicação com o módulo de detecção de objetos através de uma conexão via socket, enviando um comando para realizar a detecção.

6. O módulo de detecção de objetos realiza a sua tarefa e grava o resultado da detecção no arquivo de resultados. Depois deste processo, notifica o provedor para que este leia o resultado no arquivo.

7. O provedor de embasamento lê o resultado da detecção de objetos e realiza o embasamento, caso seja possível.

8. A ação interna retorna a lista de embasamentos do símbolo para a variável passada por parâmetro. Caso não haja embasamento, é retornada uma lista vazia.

4.4. Experimentos

4.4.1. Objetivo

Visando desenvolver um protótipo do modelo proposto, utilizando-se de um modelo para reconhecimento de objetos e uma arquitetura para agentes BDI, os experimentos a serem apresentados implementaram os modelos propostos neste trabalho para que sua utilização pudesse ser analisada. A utilização experimental dos três modelos apresentados anteriormente permite a análise das suas vantagens e desvantagens em relação ao contexto desenvolvido para o experimento. Além disso, permite que seja analisada a capacidade dos modelos de fundamentar símbolos e de conectar os componentes de IA Simbólica e IA Conexionista.

4.4.2. Métricas de avaliação

Devido à natureza do problema que o modelo pretende tratar, a maior parte da sua avaliação se torna qualitativa, visto que este trabalho não visa melhoria de performance de quaisquer partes envolvidas, como por exemplo a acurácia das detecções realizadas pela rede neural de detecção de objetos, ou ainda do seu tempo de processamento. O foco deste trabalho está no problema do embasamento simbólico e como os modelos propostos auxiliam no embasamento de símbolos. Além disso, deseja-se avaliar o quanto os modelos facilitam a tarefa de embasamento de símbolos. A Tabela 2 especifica as métricas estabelecidas para a avaliação dos experimentos.

Uma versão deste sistema foi desenvolvida como *baseline* para a comparação dos experimentos de cada um dos modelos propostos neste trabalho. Esta versão não realiza nenhuma consideração a respeito no embasamento das crenças do agente. As mudanças no ambiente são simuladas através da classe ambiente do Jason e o agente reage às mudanças conforme elas são simuladas. O mesmo comportamento de simulação das mudanças no ambiente foi implementado nos demais experimentos, com os devidos ajustes para se encaixar nos requisitos de cada modelo.

A partir da versão de base do experimento, foram então criadas novas versões para a implementação da utilização de cada um dos modelos propostos neste trabalho. Com isto, foi possível identificar quais seriam as alterações e adaptações necessárias a serem realizadas para a utilização de cada modelo.

Tabela 2. Métricas de avaliação dos experimentos

Tipo	Métrica
Qualitativa	O modelo é capaz de embasar símbolos?
	O quanto o modelo facilita a tarefa de fundamentação de símbolos?
	O quão adequada foi esta implementação para o contexto do experimento?
	Qual é o nível de complexidade de utilização do modelo?
Quantitativa	Número de linhas de código alteradas
	Número de arquivos alterados
	Número de planos alterados ou incluídos no agente

Para o desenvolvimento dos experimentos foi definido um cenário hipotético de um agente que simula um robô de café inteligente, que utiliza objetos para servir o café, conforme foi descrito na seção 4.2. O agente espera possuir um objeto que será utilizado para servir o café ao cliente, como por exemplo uma xícara, e outro objeto que será utilizado para despejar o café no primeiro objeto, como por exemplo uma chaleira. Ao perceber que há uma pessoa no ambiente e que estão disponíveis os objetos necessários para a realização da tarefa, o agente serve o café e verifica se sua ação foi bem sucedida.

Para a execução do experimento, foram simuladas de forma simples algumas situações que o robô poderia vivenciar no mundo real para que o mesmo realizasse suas tarefas. Para simplificar o experimento e, ao mesmo tempo, determinar a eficiência dos modelos, foram preparadas cinco situações. A Tabela 3 resume as situações utilizadas durante os experimentos.

Devido ao fato de que a rede neural utilizada detecta a imagem simulada com a classe woman, os agentes foram programados para adicionarem uma crença detected(person) ao receber a crença detected(woman). Isto foi feito com o intuito de facilitar os testes.

O módulo de detecção de objetos possui suas classes já pré-definidas e a saída da rede neural respeita tais classes. Porém, visto que o Jason possui algumas restrições sintáticas, foi necessário realizar um tratamento no nome das classes recebidas da detecção de objetos para que não houvesse um problema de compatibilidade com o Jason. Visto que a limitação é imposta pelo Jason, optou-se por realizar tal tratamento nos provedores de embasamento, já que estes estão ligados à arquitetura do Jason. Na Tabela 3, as percepções que devem ser embasadas por estarem conectadas aos seus respectivos objetos detectados pela rede neural estão destacados.

Tabela 3. Situações de ambiente simuladas nos experimentos

Situação	Percepções simuladas	Imagem	Detecções	Percepção gerada pela detecção (transformada)
Nº 1	detected(kettle)		Human face Clothing Woman Girl	detected(human_face) detected(clothing) detected(woman) detected(girl)
Nº 2	detected(person) detected(coffee_cup)		Kettle Human face Clothing Woman Human hair	detected(kettle) detected(human_face) detected(clothing) detected(woman) detected(human_hair)
Nº 3	detected(person) detected(kettle) detected(cup)		Kettle Human face Clothing Coffee cup Woman Saucer	detected(kettle) detected(human_face) detected(clothing) detected(coffee_cup) detected(woman) detected(saucer)
Nº 4	detected(person) detected(kettle) detected(coffee_cup)		Kettle Human face Clothing Coffee cup Woman Saucer	detected(kettle) detected(human_face) detected(clothing) detected(coffee_cup) detected(woman) detected(saucer)

Nº 5	detected(person) detected(kettle) detected(coffee_cup) detected(coffee)		Kettle Human face Coffee Saucer Clothing Coffee cup Woman Girl	detected(kettle) detected(human_face) detected(coffee) detected(saucer) detected(clothing) detected(coffee_cup) detected(woman) detected(girl)

Cada situação visa demonstrar possíveis etapas de percepção do agente, bem como seus embasamentos. Na primeira situação há apenas a percepção de detecção de uma chaleira simulada pelo ambiente, porém esta percepção não está embasada, visto que na detecção de objetos a imagem não mostra este objeto. A segunda situação apresenta a percepção de detecção de uma pessoa e de uma xícara de café. A primeira percepção deve ser embasada, pois existe uma pessoa na imagem, já a xícara não. Nas situações 3 e 4 a imagem utilizada é a mesma, porém o conjunto de percepções simuladas pelo ambiente é diferente, visando demonstrar a diferença nas decisões do agente com base no embasamento de suas crenças. Por fim, a quinta situação é a que simula o término da tarefa de servir o cliente, utilizando uma imagem que possui o café na xícara, bem como as detecções correspondentes simuladas pelo ambiente.

4.4.3. Execução

O módulo de detecção de objetos, desenvolvido em Python a partir do código de exemplo disponibilizado pelo TensorFlow, consiste em dois arquivos:

detection.py: Neste arquivo está a lógica da detecção de objetos propriamente dita. A classe ObjectDetection realiza toda a tarefa de download da rede neural, descoberta da imagem a ser detectada, execução da detecção e persistência do resultado em arquivo JSON.

server.py: Aqui está definido o servidor socket que é utilizado para comunicação com o agente BDI. Ele executa a detecção a partir do comando recebido e responde após a finalização.

O módulo do agente BDI, por sua vez, foi estruturado em projetos, onde cada projeto possui um papel específico para o desenvolvimento deste trabalho, bem como para a execução e avaliação dos experimentos. Os projetos desenvolvidos são:

sglib: O nome deste projeto deriva de *Symbol Grounding Library* e é a biblioteca que implementa os modelos propostos. É neste projeto que se encontram todas as interfaces e classes desenvolvidas para os modelos. Aqui estão os provedores de embasamento, os provedores de percepções embasadas, o agente e o ambiente personalizados, a ação interna, etc.

sgmodel_baseline: Este projeto realiza a implementação do cenário dos experimentos sem qualquer utilização de embasamento simbólico. Ele serve como base para a comparação com os demais projetos.

sgmodel_v1: Implementa o experimento do modelo de personalização da classe Agent e uso de provedores de embasamento.

sgmodel_v2: Implementa o experimento do modelo de personalização da classe Environment e uso de provedores de percepções embasadas.

sgmodel_v3: Implementa o experimento do modelo de ação interna para embasamento de símbolos.

4.4.4. Resultados

A avaliação dos experimentos deu-se, em sua maioria, mediante o uso de métricas qualitativas, conforme descrito na seção 4.5.2. As seções seguintes irão descrever os resultados de cada métrica analisadas nos experimentos.

4.4.4.1. O modelo é capaz de embasar símbolos?

Em relação ao embasamento de símbolos, considera-se que todos os modelos propostos são capazes de conectar os símbolos que são utilizados pelo agente com seus correspondentes no mundo real, na medida em que o objeto detectado é mapeado para o símbolo. Porém, apesar de conectar o símbolo a um objeto do mundo real, nenhum modelo é capaz de resolver o Problema do Embasamento Simbólico, visto que, para mapear o objeto para um símbolo, a rede neural precisou ser previamente treinada e recebeu tais símbolos de forma extrínseca. Os modelos seguem as abordagens representacionalistas e, apesar de não resolverem o Problema do Embasamento Simbólico, são capazes de embasar símbolos conectando-os aos seus referentes.

4.4.4.2. O quanto o modelo facilita a tarefa de fundamentação de símbolos?

Por se tratar de uma biblioteca que fornece toda a estrutura para o embasamento de símbolos e permite ainda a sua extensão, os modelos facilitam consideravelmente a tarefa de fundamentação de símbolos. Os três modelos oferecem facilidades para a realização da tarefa, diferindo apenas no fluxo e nos papéis que cada parte do framework Jason assume. Para o contexto do experimento, o modelo que mais facilita a tarefa é o implementado pelo projeto *sgmodel_v2*, o qual utiliza o modelo de percepções embasadas. Após o desenvolvimento da configuração inicial, a tarefa de embasar qualquer símbolo se torna muito simples em todos os modelos apresentados. Além disso, os modelos permitem a reutilização de código feito para embasar símbolos, através da reutilização de provedores de embasamento e/ou de percepções embasadas. Isto permite que, na medida em que provedores são implementados, novos projetos ou evoluções de projetos existentes tenham seus símbolos embasados de maneira ainda mais fácil e rápida.

4.4.4.3. O quão adequada foi esta implementação para o contexto do experimento?

O contexto do experimento demanda a identificação de objetos e de pessoas no ambiente para que um café seja servido. Por este motivo, a funcionalidade do agente está diretamente ligada e é totalmente dependente da detecção de objetos que, em uma situação real, provavelmente seria executada em imagens capturadas de uma câmera.

O projeto *sgmodel_v1*, que implementa a utilização dos provedores de embasamento, não se mostrou adequado ao contexto, pois para realizar o embasamento das crenças na função de atualização de crenças do agente, é necessário que este receba novas percepções para que a função em questão seja executada. Por este motivo, para que o experimento funcionasse, foi necessário implementar a simulação de percepções no ambiente.

Também se mostrou inadequado ao contexto, pelo mesmo motivo, o projeto *sgmodel_v3*, que utiliza a ação interna de embasamento de símbolos. Também foi necessário simular as percepções neste projeto para que ao final fosse executada a ação interna que verificaria o embasamento destas percepções para então realizar a tarefa de servir o café.

Já o projeto *sgmodel_v2*, que implementa a utilização de provedores de percepções embasadas se mostrou adequado ao experimento, pois ele se conecta diretamente com o módulo de detecção de objetos e gera diretamente para o agente as percepções embasadas. Apesar de haver simulações de trocas de imagens, estas simulações seriam removidas em um sistema real,

que captaria diretamente da câmera as imagens a serem analisadas. Com a utilização deste modelo é possível fazer com que o agente funcione corretamente sem a necessidade de uma duplicidade das percepções, como seria necessário nos demais modelos.

4.4.4.4. Qual é o nível de complexidade de utilização do modelo?

O experimento implementado no projeto *sgmodel_v1* demonstra a facilidade em utilizar o modelo que utiliza provedores de embasamento em conjunto com uma classe de agente personalizada. Ao estender a classe *BaseGroundedAgent* disponível na biblioteca, o desenvolvedor precisa apenas implementar dois métodos básicos: o método que cria as configurações para o modelo e o método que instancia a registra os provedores de embasamento. Toda a lógica restante que é necessária para o funcionamento do modelo já está pronta. Isto facilita muito o trabalho do desenvolvedor que deseja utilizar a biblioteca. Este modelo também não exige grandes alterações nos planos do agente, exceto aqueles onde se deseja considerar o embasamento simbólico. Porém, ao utilizar este modelo, o desenvolvedor precisa utilizar uma classe de agente personalizada, seja estendendo a classe *BaseGroundedAgent* ou implementando por conta própria as funcionalidades necessárias para o funcionamento dos provedores.

O projeto *sgmodel_v2*, que implementa o modelo de provedores de percepções embasadas, também apresenta uma grande facilidade em sua utilização. Isto porque basta que o desenvolvedor estenda a classe *BaseGroundedPerceptTargetEnvironment* e defina a sua classe como sendo a que deve ser utilizada pelo ambiente. Da mesma forma que no projeto anterior, o desenvolvedor precisa apenas implementar dos métodos básicos: o método que cria as configurações para o modelo e o método que instancia a registra os provedores de de percepções embasadas. Aqui o desenvolvedor também possui a liberdade de implementar as interfaces e funcionalidades em seu próprio ambiente. Porém, para utilizar toda a estrutura pronta, disponível na biblioteca, o desenvolvedor precisa utilizar este ambiente personalizado.

Neste modelo o desenvolvedor também possui a liberdade de implementar as interfaces em algum outro ponto do framework Jason. Para isso, ele precisa apenas implementar as interfaces *IGroundedPerceptTarget*, *IConfigurationProvider* no local onde acredita ser mais adequado ao seu projeto e realizar as configurações dos módulos, bem como a criação dos provedores de percepções embasadas. Assim como no modelo anterior, não é necessário realizar grandes alterações nos planos do agente, exceto aqueles onde se deseja considerar o embasamento simbólico.

Por fim, o projeto *sgmodel_v3*, que implementa o modelo da ação interna de embasamento simbólico mostra a flexibilidade fornecida. A utilização deste modelo não exige que o desenvolvedor utilize uma classe de ambiente nem de agente específicas ou modificadas. A utilização deste modelo demanda que o agente receba a configuração dos módulos de alguma forma. O experimento implementa através da adição de uma crença com a configuração gerada pelo ambiente. Desta forma, este experimento demandou mais alterações no código do agente e também a implementação da lógica que gera a configuração e envia para o agente por meio de uma percepção. A utilização deste modelo também se mostrou simples. A principal vantagem, como mencionado anteriormente, é a flexibilidade.

4.4.4.5. Métricas quantitativas

Por não se tratar de uma análise de performance ou de parâmetros puramente numéricos, a análise quantitativa dos modelos não é uma tarefa simples. Entretanto, visando colocar em números algumas perspectivas em relação aos modelos, foram definidas as três métricas apresentadas na seção 4.5.2. A Figura 8 contém um gráfico que mostra o resultado destas

métricas. É importante destacar que todas as métricas foram calculadas comparando cada projeto do gráfico com o projeto *sgmodel_baseline* e que alterações relacionadas com simulações de mudança de percepções e também de troca das imagens a serem analisadas pelo módulo de detecção de objetos não foram consideradas. O principal objetivo em desconsiderar estas questões é quantificar as alterações que seriam realmente necessárias em um projeto real.

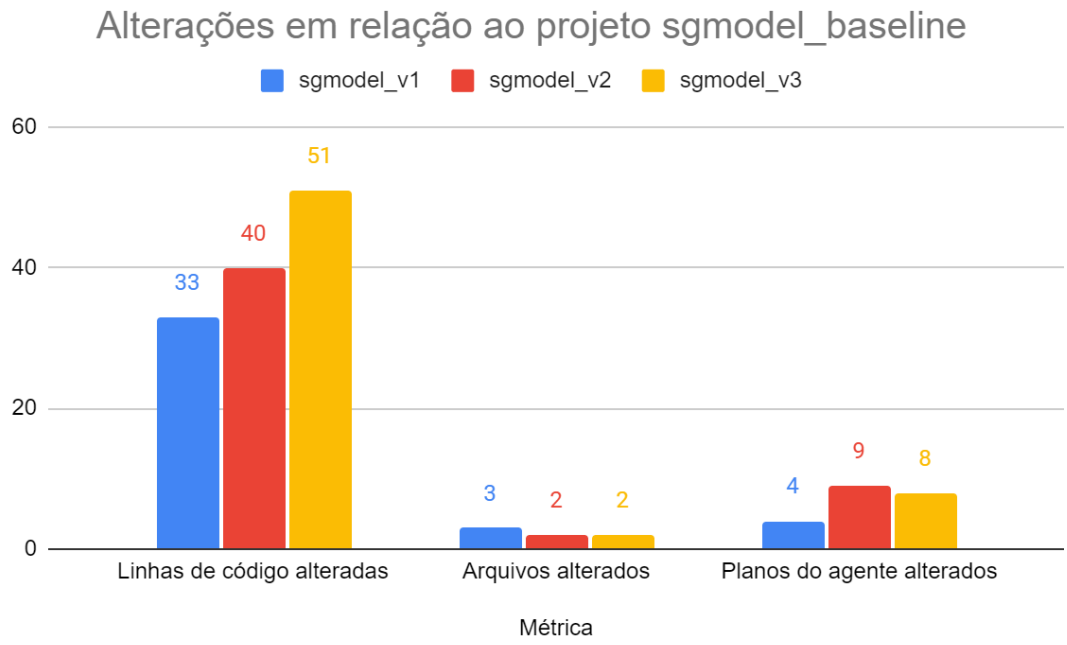


Figura 7. Resultado das métricas quantitativas.

Em relação ao número de linhas alteradas, é possível verificar que o projeto *sgmodel_v3* foi o que obteve o maior valor. Grande parte das alterações deste experimento está concentrada no código do agente, visto que o mesmo executa duas vezes a ação interna (uma para cada percepção necessária). Porém, de modo geral, o número de alterações nas linhas é muito próximo entre os três experimentos.

Como os modelos se adaptam ao framework do Jason e a biblioteca já fornece grande parte da implementação necessária, os arquivos alterados são praticamente os mesmos, com exceção do primeiro experimento (*sgmodel_v1*) que exige a criação de um novo arquivo, com a classe do agente.

Já os planos do agente sofrem alterações distintas nos experimentos, porém em uma grandeza muito parecida. O experimento que mais exigiu alteração na lógica do agente foi o *sgmodel_v3* (apesar de não ser o de maior número de planos alterados). Isto porque este experimento demandou a criação de um plano para o recebimento das configurações e a alteração do plano de servir o café para a utilização da ação interna.

4.4.5. Discussão

Através da execução dos experimentos e da análise dos seus resultados, foi possível validar alguns pontos em relação aos modelos propostos neste trabalho. A Tabela 4 expõe um resumo dos resultados obtidos com a realização dos experimentos.

Tabela 4. Resumo dos resultados das métricas verificadas para os experimentos

	sgmodel_v1	**sgmodel_v2**	**sgmodel_v3**
O modelo é capaz de embasar símbolos?	Sim	Sim	Sim
O quanto o modelo facilita a tarefa de fundamentação de símbolos?	Muito	Muito	Muito
O quão adequada foi esta implementação para o contexto do experimento?	Inadequado	Adequado	Inadequado
Qual é o nível de complexidade de utilização do modelo?	Baixo	Baixo	Baixo
Número de linhas de código alteradas	33	40	51
Número de arquivos alterados	3	2	2
Número de planos alterados ou incluídos no agente	4	9	8

O modelo que se mostrou mais adequado ao contexto do experimento foi o do projeto *sgmodel_v2*, que implementa o uso de provedores de percepções embasadas. Isto porque, devido ao fato de que a tarefa do agente está diretamente ligada à detecção de objetos, e que o mesmo considera que seu objetivo só pode ser concluído quando suas percepções estão embasadas, os demais modelos se mostraram inadequados ao contexto, pois demandam neste caso uma fonte duplicada de percepções. Apesar disso, os demais modelos também realizam a tarefa de embasar os símbolos e facilitam este trabalho, com um baixo nível de dificuldade em sua utilização.

5. Conclusão

Este trabalho analisou o estado da arte em modelos para embasamento simbólico, com o objetivo de propor um modelo baseado em teoria de agentes BDI e modelos conexionistas para reconhecimento de objetos. Esta análise permitiu o contato com diversos problemas em aberto para resolver o problema do embasamento simbólico, que motivam as mais variadas abordagens para a tentativa de descoberta de uma solução para o mesmo.

Inspirado nas abordagens representacionalistas, este trabalho desenvolveu uma biblioteca que oferece uma estrutura de base para conexão dos símbolos manipulados por agentes BDI aos seus referentes no mundo real. A utilização de uma rede neural de detecção de objetos permitiu que objetos que podem ser detectados através de uma câmera sejam relacionados às crenças do agente BDI, realizando assim o seu embasamento. A estrutura desenvolvida permite a sua extensão de maneira simples e prática, necessitando apenas da implementação de novos provedores (sejam de embasamento ou de percepções embasadas) e suas devidas configurações. Para a utilização da rede neural de detecção de objetos, este trabalho desenvolveu dois provedores, que realizam a tarefa de conectar-se ao módulo de detecção de objetos e então realizar a tarefa do embasamento.

É importante ressaltar que existem algumas limitações e adaptações na conexão entre o módulo de detecção de objetos e o agente BDI. Devido ao fato de que o framework Jason possui algumas limitações sintáticas, foi necessário realizar transformações nas classes geradas pela

detecção de objetos a fim de permitir seu uso nas crenças no agente. Por exemplo, tokens que iniciam com letra maiúscula no código do arquivo asl são considerados variáveis, portanto foi necessário garantir que a primeira letra da classe não seja maiúscula. Estes tratamentos foram realizados nos provedores, uma vez que a restrição se dá devido ao framework Jason. Além disso, a conexão e a troca de informações com o módulo também foi simplificada, visto que não se tratava do foco deste trabalho. Não foram realizados tratamentos de erros por falha de conexão e o recebimento das detecções se deu através da leitura de um arquivo em disco.

O desenvolvimento e a execução dos experimentos permitiu validar os modelos propostos e analisá-los em relação ao cenário definido. Todos os modelos se mostraram capazes de realizar o embasamento dos símbolos e facilitaram muito esta tarefa. O número de alterações necessárias no projeto de base foram muito parecidos entre os modelos, não sendo possível considerar tais critérios para escolha do modelo a ser utilizado. Entretanto, apenas o modelo de percepções embasadas se mostrou adequado ao contexto do experimento. Isto porque a tarefa do agente está diretamente ligada às percepções do ambiente, mais especificamente à detecção dos objetos necessários e da pessoa a ser servida. Com isso, o mais adequado para este contexto é receber a percepção já embasada, ao invés de possuir uma fonte de percepções que não seja a própria detecção do ambiente.

Apesar de propor modelos para realizar o embasamento de símbolos de um agente BDI de maneira satisfatória, fornecendo uma biblioteca facilmente extensível e de fácil utilização, este trabalho não resolve o problema do embasamento simbólico. Isto porque a detecção de objetos é realizada por uma rede neural que foi previamente treinada com exemplos de imagens anotadas por seres humanos, ou seja, a fonte do conhecimento contido nesta rede é extrínseca.

Porém, mesmo sem oferecer uma solução absoluta para o problema, este trabalho contribui com a definição de um modelo que realiza uma parte importante do embasamento de símbolos e que pode ser facilmente estendido e evoluído. Além disso, este trabalho conseguiu atingir seus objetivos, na medida em que analisou o estado da arte em modelos para embasamento simbólico, detecção de objetos e agente BDI, propôs um modelo que integra partes do conhecimento do agente à uma respectiva representação conexionista, desenvolveu um protótipo do modelo e definiu e analisou os resultados de um experimento.

5.1. Trabalhos futuros

A biblioteca desenvolvida neste trabalho oferece diversas possibilidades de evolução. Trabalhos futuros podem desenvolver novos provedores de embasamento e de percepções embasadas, fornecendo outras fontes de embasamento para os símbolos. Um exemplo possível para esta evolução seria o desenvolvimento de um provedor de percepções embasadas através do reconhecimento de fala, o qual conectaria as palavras reconhecidas aos símbolos manipulados pelo agente.

Além de permitir a implementação de novos provedores, a biblioteca também permite o desenvolvimento de novas ações internas que se adequem a novos casos de uso. Seria possível, por exemplo, desenvolver uma ação interna que dispare um determinado evento somente no caso de uma determinada crença, passada por parâmetro, estar embasada. Esta ação facilitaria o controle do fluxo de execução dos planos do agente, ao mesmo tempo que manteria a liberdade do desenvolvedor.

Por utilizarem-se de uma conexão simplificada e limitada com o módulo de detecção de objetos, sugere-se que os provedores de embasamento e de percepções embasadas a partir da detecção de objetos sejam evoluídos, permitindo a troca direta de informações entre os módulos utilizando a conexão via socket, ao invés de realizar a leitura do arquivo JSON. Tal evolução

permitirá que o módulo de detecção de objetos esteja em um sistema remoto e também removerá a necessidade de espaço de armazenamento para o arquivo.

Além das possibilidades de extensão deste trabalho apresentadas anteriormente, sugere-se a evolução dos modelos através do desenvolvimento de uma nova estratégia de troca de informações entre os provedores e os agentes BDI. Além de embasar os símbolos, conectando-os aos seus referentes no mundo real, os provedores podem ser capazes de fornecer mais informações sobre tais referentes. Por exemplo, um robô que execute uma detecção de objetos e que, ao mesmo tempo, tenha informações de distância, tamanho, peso, cor, e tantas outras características e dados sobre os referentes no mundo real, poderia utilizar os provedores para fornecer tais informações ao agente, de maneira unificada e contextualizada. O agente, por sua vez, passa a ter um conhecimento maior sobre um determinado referente do mundo real, podendo tomar decisões mais precisas em direção ao seu objetivo.

Por fim, sugere-se o desenvolvimento de novos experimentos, com situações mais complexas e com diferentes objetivos para que o modelo possa ser avaliado de uma maneira mais ampla. A realização de novos testes, com agentes mais complexos, permitirão a expandir a validação do modelo em relação ao embasamento dos símbolos e também da sua flexibilidade.

6. Referências

ALBAWI, Saad; MOHAMMED, Tareq Abed; AL-ZAWI, Saad. Understanding of a convolutional neural network. **2017 International Conference On Engineering And Technology (ICET)**, [S.L.], p. 1-6, ago. 2017. IEEE.

BORDINI, Rafael H.; HÜBNER, Jomi F.. BDI Agent Programming in AgentSpeak Using Jason. **Lecture Notes In Computer Science**, [S.L.], p. 143-164, 2006. Springer Berlin Heidelberg.

EICHSTAEDT, Leonardo Vailatti. **Problema de embasamento de símbolos em um sistema multi-contexto**. 2019. 138 f. TCC (Graduação) - Curso de Sistemas de Informação, Universidade Federal de Santa Catarina, Florianópolis, 2019.

HARNAD, S. The symbol grounding problem. Physica D: Nonlinear Phenomena, Elsevier Science, v. 42, 1990.

JIAO, Licheng; ZHANG, Fan; LIU, Fang; YANG, Shuyuan; LI, Lingling; FENG, Zhixi; QU, Rong. A Survey of Deep Learning-Based Object Detection. **IEEE Access**, [S.L.], v. 7, p. 128837-128868, 2019. Institute of Electrical and Electronics Engineers (IEEE).

LUGER, George F. **Artificial Intelligence**. New Mexico: Pearson Education, 2009. 779 p. ISBN 0321545893.

MORENO, Marcio; CIVITARESE, Daniel; BRANDAO, Rafael; CERQUEIRA, Renato. Effective Integration of Symbolic and Connectionist Approaches through a Hybrid Representation. In: INTERNATIONAL WORKSHOP ON NEURAL-SYMBOLIC LEARNING AND REASONING, 14., 2019, Macao. **Proceedings [...]** . Macao: Ijcai, 2019. p. 76-78.

RAO, Anand s; GEORGEFF, Michael P.. BDI Agents: from theory to practice. In: INTERNATIONAL CONFERENCE ON MULTIAGENT SYSTEMS, 1., 1995, Melbourne. **Proceedings of the First International Conference on Multiagent Systems**. San Francisco: AAAI, 1995. p. 312-319.

RAUE, Federico; BYEON, Wonmin; BREUEL, Thomas M.; LIWICKI, Marcus. Symbol Grounding in Multimodal Sequences using Recurrent Neural Networks. In: COCO @ NIPS,

29., 2015, Montreal. **Proceedings of the NIPS Workshop on Cognitive Computation: Integrating Neural and Symbolic Approaches**. Montreal: Neural Information Processing Systems Foundation, 2015.

REN, Shaoqing; HE, Kaiming; GIRSHICK, Ross; SUN, Jian. Faster R-CNN: Towards real-time object detection with region proposal networks. **IEEE Transactions On Pattern Analysis And Machine Intelligence**, [S.L.], v. 39, n. 6, p. 1137-1149, 1 jun. 2017. Institute of Electrical and Electronics Engineers (IEEE).

RUSSELL, Stuart J.; NORVIG, Peter. **Inteligência artificial**. Rio de Janeiro: Elsevier, Campus, 2004. 1021 p. ISBN 8535211772.

SEARLE, J. R. Minds, brains, and programs. Behavioral and Brain Sciences, Cambridge University Press, v. 3, 9 1980.

SILVA, Lavindra de; MENEGUZZI, Felipe; LOGAN, Brian. BDI Agent Architectures: a survey. **Proceedings Of The Twenty-Ninth International Joint Conference On Artificial Intelligence**, [S.L.], p. 4914-4921, jul. 2020. International Joint Conferences on Artificial Intelligence Organization.

TADDEO, Mariarosaria; FLORIDI, Luciano. Solving the symbol grounding problem: a critical review of fifteen years of research. **Journal Of Experimental & Theoretical Artificial Intelligence**, [S.L.], v. 17, n. 4, p. 419-445, dez. 2005. Informa UK Limited.

WOOLDRIDGE, Michael. **An Introduction to MultiAgent Systems**. Liverpool: John Wiley And Sons Ltd, 2002.

WOOLDRIDGE, Michael; JENNINGS, Nicholas R.. Intelligent Agents: theory and practice. **The Knowledge Engineering Review**. Cambridge, p. 115-152. nov. 1995.