PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
FACULDADE DE INFORMÁTICA
BACHARELADO EM SISTEMAS DE INFORMAÇÃO

INTERPRETADOR PARA UM SUBCONJUNTO DA SINTAXE DO AGENTSPEAK(L) PARA PYTHON

ANDRÉ LUIZ LEONHARDT DOS SANTOS

Proposta de Trabalho de Conclusão apresentada como requisito parcial à obtenção do grau de Bacharel em Sistemas de Informação na Pontifícia Universidade Católica do Rio Grande do Sul.

Orientador: Prof. Felipe Meneguzzi

Porto Alegre
2016

RESUMO

O desenvolvimento de sistemas multiagentes tem possibilitado a descrição, a solução e o entendimento de grandes sistemas distribuídos. Ao longo dos anos, linguagens de programação de agentes surgiram com o propósito de facilitar o desenvolvimento destes sistemas. Estas linguagens permitem criar agentes com comportamento baseado em crenças, desejos e intenções, e que interajam com um ambiente complexo. Entre as linguagens amplamente utilizadas está a linguagem AgentSpeak(L), cuja implementação mais comum é o JASON. Entretanto, apesar de suportar diversas extensões da linguagem AgentSpeak(L) o desempenho deste interpretador é limitado. Propomos, então, o desenvolvimento de um interpretador para um subconjunto da sintaxe do AgentSpeak(L) para Python. Neste sentido, realizaremos um levantamento teórico para determinar o ciclo de raciocínio mais adequado para os agentes, além de todas as fases e ferramentas necessárias para a criação de um interpretador.

Palavras-Chave: Agentes, Sistemas Multiagentes, AgentSpeak(L), Interpretadores, Python.

ABSTRACT

The development of multi-agent systems has made possible the description, solution, and understanding of large-scale distributed systems. Over the years, agent programming language has emerged with the aim of making systems development easier. Such languages allow the creation of agents of which behavior is based on beliefs, desires, and intentions, and which interact with a complex environment. Among the most widely used languages, it is the AgentSpeak(L), of which JASON is the most common implementation. However, though this interpreter can abide diferent language extensions, it delivers a poor performance. Thus, we propose the development of an interpreter for a subset of the AgentSpeak(L) syntax for Python. In this sense, we will conduct a theoretical survey in order to determine the most suitable reasoning cycle for the agents, besides all phases and tools necessary for creating an interpreter.

Keywords: Agents, Multi-Agent Systems, AgentSpeak (L), Interpreters, Python.

LISTA DE FIGURAS

LISTA DE SIGLAS

BDI – Belief Desire Intention

DMARS – Distributed Multi-Agent Reasoning System

IA – Inteligência Artificial

JASON – Java-based AgentSpeak interpreter used with SACI for multi-agent distribution Over the Net

PRS – Procedural Reasoning System

SMA – Sistema Multiagente

SUMÁRIO

1. INTRODUÇÃO

Inteligência Artificial (IA) é o campo de estudo da ciência da computação que tem como objetivo construir máquinas e programas de computadores com comportamento inteligente. Desde o início de suas pesquisas, em meados da década de 50, muitos avanços foram feitos neste sentido, como a criação de solucionadores de problemas gerais e a concepção de algoritmos genéticos, nos anos 60 e 70, respectivamente. Apesar de ter ocorrido um hiato com poucos avanços, a IA tornou-se, definitivamente, uma indústria nos anos 80 com o desenvolvimento de sistemas especialistas. Nesta época, intensificaram-se as pesquisas em áreas como redes neurais artificiais, aprendizado de máquina e mineração de dados [Russel and Norvig 2010, p. 19].

Uma outra área, a de agentes inteligentes e sistemas multiagentes (SMA), tem recebido atenção especial desde os anos 90 época pela sua capacidade de explorar as características presentes em sistemas distribuídos massivos e abertos, como a internet e a nossa sociedade [Wooldridge 2002, p. xi]. Nesta representação, diversos agentes, cada um com suas próprias percepções, crenças, desejos, intenções e ações, interagem entre si e com um ambiente. Este contexto proporcionou o desenvolvimento de linguagens de programação e ferramentas para auxiliarem na descrição, na solução e no entendimento de grandes sistemas distribuídos, como telecomunicações, tráfego aéreo e terrestre, previsão de demanda em energia elétrica, sistemas tutores e até viagens espaciais [d'Inverno et al. 1997].

Entre as linguagens amplamente utilizadas está a linguagem AgentSpeak(L), cuja implementação mais comum é o interpretador JASON. Desta forma, é possível criar agentes com comportamento baseado em crenças, desejos e intenções, e que interajam com um ambiente complexo. Neste caso, os agentes são descritos por meio do AgentSpeak(L), ao passo que o ambiente é desenvolvido utilizando a linguagem Java [Bordini and Hübner 2005]. Por sua vez, o Java, linguagem com a qual o JASON foi desenvolvido, é amplamente utilizado na industria de desenvolvimento de *software*[1] e no ensino de algoritmos utilizados em IA [Connelly and Goel 2013]. Outra linguagem presente neste mesmo cenário é a Python. No entanto, apesar de ser utilizado em aplicações científica com alto poder computacional [McKinney 2013, p. 3], o Python não possui nenhuma implementação para a linguagem AgentSpeak(L) que permita o desenvolvimento de SMA.

Propomos, então, o desenvolvimento de um interpretador para um subconjunto da sintaxe do AgentSpeak(L) para Python. Tendo em vista que muitos dos cenários de SMA possuem uma complexidade elevada, é necessário que o interpretador em questão seja enxuto e eficiente. Desta forma, o fato do Python ser amplamente utilizado em computação científica corrobora para que ele seja empregado como plataforma base no desenvolvimento deste trabalho.

Este documento está organizado da seguinte forma. O Capítulo 2 contém uma fundamentação teórica com os conceitos necessários para o entendimento deste trabalho, como: agentes, sistemas multiagentes, arquitetura BDI, AgenteSpeak(L) e interpretadores. Por sua vez, o Capítulo 3

[1] http://www.tiobe.com/tiobe_index

contém os objetivos pretendidos na realização deste trabalho, enquanto que o Capítulo 4 detalha as atividades necessárias para alcançar tais objetivos.

2. FUNDAMENTAÇÃO TEÓRICA

2.1 Agentes

Embora não exista um consenso geral sobre a definição do termo [Wooldridge 2002, p. 15], um agente é um sistema computacional que *percebe* o *ambiente* em que se encontra e é capaz de realizar *ações* de forma *autônoma* para alcançar seus objetivos propostos [Wooldridge and Jennings 1995]. A percepção do ambiente pelo agente pode ser feita através de sensores e as ações podem realizadas por atuadores [Russel and Norvig 2010, p. 34], conforme ilustrado na Figura 2.1.

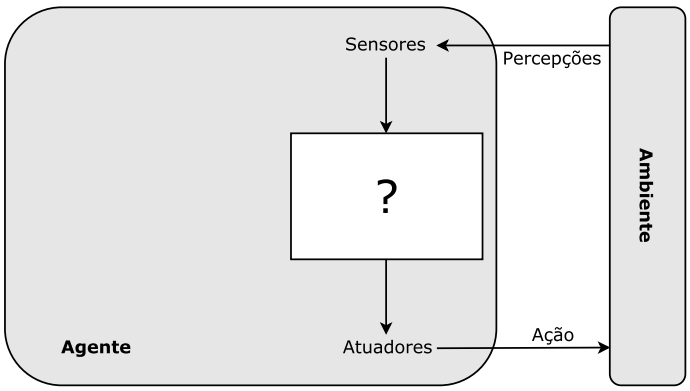

Figura 2.1 – Interação de um agente e seu ambiente.

Wooldridge e Jennings [Wooldridge and Jennings 1995] também caracterizam agentes com as seguintes propriedades:

- Autonomia: Operam sem a intervenção direta de humanos ou outros agentes, possuindo controle total sobre suas ações e estado interno.
- Habilidade Social: Interagem com outros agentes (humanos ou computacionais).
- Reatividade: Percebem e reagem às alterações do ambiente em que estão inseridos.
- Pró-atividade: Além de atuarem em resposta às alterações ocorridas em seu ambiente, apresentam um comportamento orientado a objetivos, tomando a iniciativa quando julgarem apropriado.

Para os autores, estas propriedades são apresentadas como sendo parte de uma noção *fraca* sobre agentes. Eles também citam características aplicadas a seres humanos, definida como uma noção *forte* sobre agentes, como conhecimento crenças, intenções e deveres.

2.1.1 Ambiente

O ambiente em que o agente obtém as informações e realiza suas ações pode, segundo Russel e Norvig [Russel and Norvig 2010, p. 41], ser classificados de acordo com suas propriedades:

- Totalmente observável x parcialmente observável: Em um ambiente totalmente observável, o agente pode obter informações completas e atualizadas através das percepções. No entanto, ruído ou dados imprecisos dos sensores tornam o ambiente parcialmente observável.
- Determinístico x não-determinístico: Em um ambiente determinístico, não há incerteza sobre qual será o resultado da execução de uma ação. Por outro lado, em um ambiente não-determinístico, a incerteza do resultado da ação é consequência do fato de um agente possuir uma influência limitada sobre o ambiente e que suas ações podem, por exemplo, falhar.
- Estático x dinâmico: Em um ambiente estático, não há mudanças enquanto um agente está deliberando, apenas quando uma ação é executada por um agente. Por sua vez, em um ambiente dinâmico, o agente precisa continuar observando o mundo enquanto está decidindo sobre a realização de uma ação e, caso ele ainda não tenha se decidido, será considerado a decisão de não fazer nada.
- Discreto x contínuo: Em um ambiente discreto, existe um número contável de ações, percepções e estados do ambiente, por sua vez, em um ambiente contínuo, este número torna-se incontável.

2.1.2 Arquitetura

O comportamento de um agente é descrito por um programa que mapeia, através de processos internos, uma sequência de percepções em uma ação [Russel and Norvig 2010, p. 46]. A especificação de como são estes processos internos é denominado de arquitetura do agente. Maes [Maes 1991, p. 115] define arquitetura de um agente como sendo:

> Uma arquitetura propõem uma metodologia específica para construir um agente inteligente. Ela especifica como o problema pode ser dividido em subproblemas, isto é, como a construção de um agente pode ser dividido em um conjunto de componentes modulares e como estes módulos devem interagir. O conjunto total de módulos e suas interações precisam prover uma resposta de como as percepções e o estado interno do agente irá determinar sua ação e o estado interno futuro do agente. Uma arquitetura engloba técnicas e algoritmos que auxiliem esta metodologia.

Russel e Norvig [Russel and Norvig 2010, p. 47] destacam quatro tipos básicos de arquiteturas de agentes:

1. Agente Reativo Simples
2. Agente Reativo Baseado em Modelos

3. Agente Baseado em Objetivos

4. Agente Baseado em Utilidade

O agente reativo simples executa suas ações baseadas apenas na percepção atual, ignorando todo o histórico de percepções anteriores. Isso torna a implementação desse tipo de agente mais fácil, pois o seu programa é composto por uma série de estruturas condicionais do tipo *if-then-else* que contém as regras para traduzir as percepções em uma ação, conforme mostra a Figura 2.2. No entanto, um comportamento mais complexo pode gerar uma implementação mais extensa.

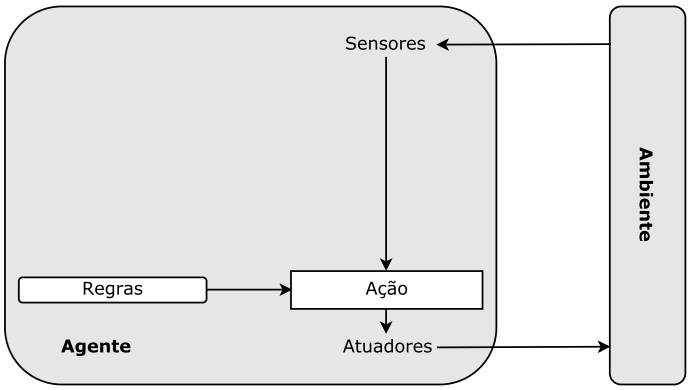

Figura 2.2 – Agente reativo simples.

Diferentemente do agente reativo simples, o agente baseado em modelos possui um estado interno que depende do histórico de percepções e que reflita os aspectos não observados no estado atual. As informações armazenadas neste estado interno são utilizadas no processo de escolha da ação de forma complementar às percepções, permitindo assim, lidar com ambientes parcialmente observáveis. A Figura 2.3 ilustra como a percepção atual é combinada com o estado interno antigo para gerar a descrição atualizada do estado atual.

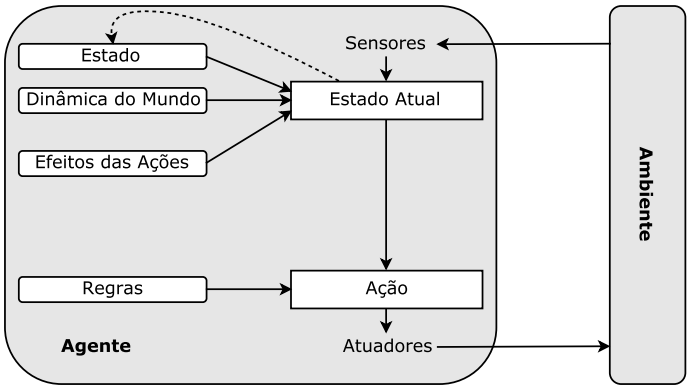

Figura 2.3 – Agente reativo baseado em modelos.

O agente baseado em objetivos utiliza informações de estados em que o agente deseja chegar e, combinado com informações do seu estado interno, escolhe as ações necessárias para alcançar o seu objetivo, conforme mostra a Figura 2.4. Busca e planejamento são subáreas da IA que auxiliam na identificação da sequência de ações utilizadas por este tipo de agente.

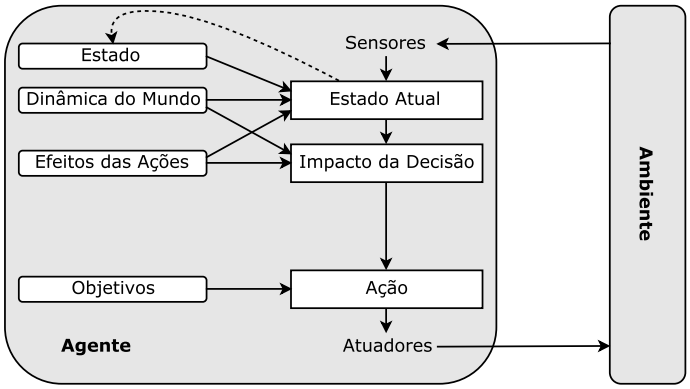

Figura 2.4 – Agente baseado em objetivos.

Objetivos isolados não são suficientes para gerar comportamentos adequados na maioria dos ambientes pois criam uma distinção binária dos estados. Diante disso, o agente baseado em utilidade mensura o quão bom um estado é em um número real para que, posteriormente, realize decisões racionais nos casos em que houverem objetivos conflitantes ou quando existirem muitos objetivos pretendidos pelo agente, conforme mostra a Figura 2.5.

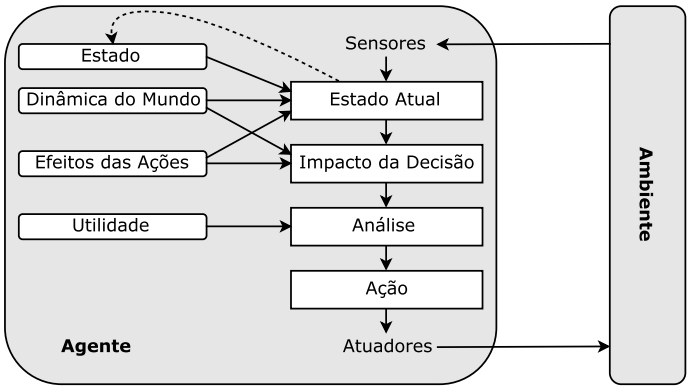

Figura 2.5 – Agente baseado em utilidade.

2.2 BDI

Uma das abordagens mais comuns para a construção de agentes baseados em objetivos é através de estados mentais semelhantes ao raciocínio prático humano como crenças, desejos e intenções [Bratman 1987, p. 1]. Uma das arquiteturas que implementam o conceito de estados mentais é a BDI (*Belief-Desire-Intention*) e, de acordo com Wooldridge [Wooldridge 2000, p. 7], possui as seguintes características:

- Crenças: Correspondem ao conhecimento que o agente possui em relação ao ambiente em que se encontra. Um agente pode ter crenças sobre o mundo, sobre outros agentes, sobre interações com outros agentes e crenças sobre suas próprias crenças, podendo ser, inclusive, incompletas ou incorretas.

- Desejos: Representam os estados objetivos que o agente deseja alcançar, motivando o agente a agir de forma a realizar suas metas.
- Intenções: São os desejos que o agente está comprometido em alcançar e determinam o processo de raciocínio prático, o qual leva o agente a ação.

O processo de raciocínio em um agente BDI genérico é ilustrado na Figura 2.6 e possui 7 componentes importantes [Wooldridge 1999, p. 31]:

- Um conjunto de crenças atual, que representa as informações que o agente tem sobre seu ambiente atual.
- Uma função de revisão de crenças, que, a partir da entrada percebida e com as crenças atuais do agente, determina um novo conjunto de crenças.
- Uma função de geração de opções, que determina as opções disponíveis para o agente, tendo como base suas crenças sobre seu ambiente e suas intenções atuais.
- Um conjunto de opções atual, que representa as ações possíveis disponíveis para o agente.
- Uma função de filtro, que representa o processo de raciocínio prático do agente, e que determina as intenções dele, tendo com base nas suas crenças, desejos e intenções atuais.
- Um conjunto de intenções atual, que representa o subconjunto de desejos que o agente está determinado a alcançar.
- Uma função de seleção de ação, que determina uma ação a ser executada, tendo como base suas intenções atuais.

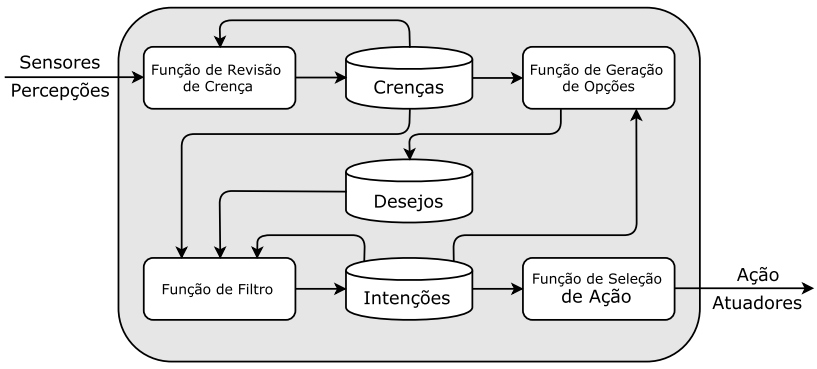

Figura 2.6 – Diagrama de uma arquitetura BDI genérica.

2.3 Sistemas Multiagentes

A utilização, na prática, de sistemas com um único agente são raros. Os casos mais comuns são os de agentes convivendo em um ambiente com outros agentes, criando um sistema multiagente [Bordini et al. 2007, p. 5].

Um SMA contém agentes conectados por algum tipo de relacionamento organizacional e interagem entre si através de uma forma de comunicação. Cada agente possui uma esfera de influência sobre o ambiente em que compartilha com os demais agentes. Estas esferas referem-se à parte do ambiente no qual um agente pode ter o controle parcial ou total e podem coincidir com as de outros agentes [Wooldridge 2002, p. 105].

O fato das esferas de influência de dois ou mais agentes coincidirem torna a escolha de ações destes agentes mais complicada, pois para que um deles alcance seu objetivo neste ambiente compartilhado é necessário levar em consideração as ações dos demais agentes [Bordini et al. 2007, p. 5]. A Figura 2.7 demonstra as interações entre os agentes e suas esferas de influência no ambiente.

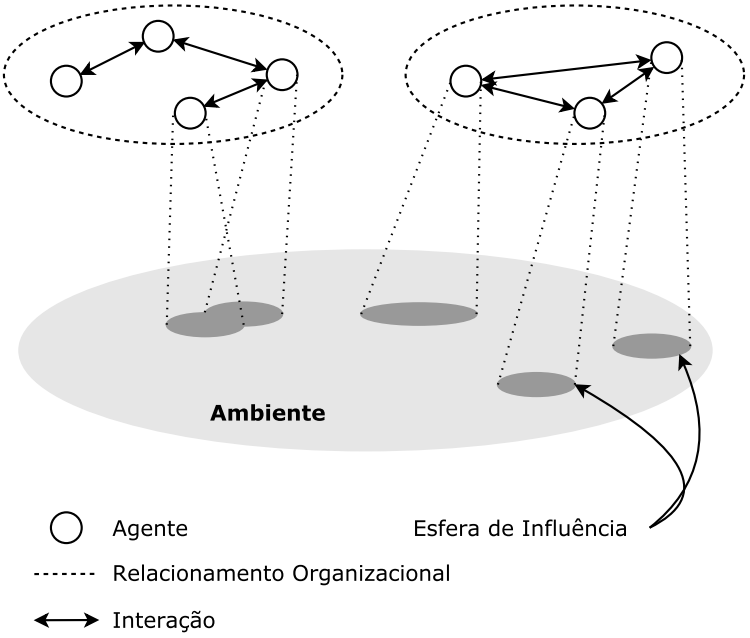

Figura 2.7 – Estrutura típica de um sistema multiagente.

Bond e Grasser [Bond and Gasser 1988, p. 34] enumeram algumas dificuldades inerentes ao projeto e implementação de SMA:

- Formulação, descrição, decomposição e alocação de problemas entre os agentes.
- Tipos de linguagens e protocolos para comunicação e interação entre os agentes.
- Desenvolver agentes individuais capazes de representar e raciocinar sobre as ações, planos e conhecimento de outros agentes para que estejam aptos a coordenarem suas ações.

- Reconhecimento e reconciliação de intenções conflitantes entre agentes que estejam tentando coordenar suas ações.

2.4 Implementações da arquitetura BDI

Existem inúmeras ferramentas e linguagens que permitem o desenvolvimento de agentes utilizando a arquitetura BDI [Mascardi et al. 2005]. Nesta seção são abordados o AgentSpeak(L), o dMARS e o PRS.

2.4.1 PRS

O ***P****rocedural* ***R****easoning* ***S****ystem* (PRS) é uma arquitetura BDI genérica para representação e raciocínio de ações e procedimentos em ambientes dinâmicos. O PRS foi desenvolvido em LISP pelo *Stanford Research Institute International* e utilizado no sistema de controle reativo de viagens espaciais da NASA (*National Aeronautics & Space Administration*). Segundo Georgeff e Lansky [Georgeff and Lansky 1987], a arquitetura do PRS pode ser subdividida em componentes periféricos e de raciocínio. Os componentes periféricos são:

- Sensores.
- Um monitor, que traduz informações dos sensores em crenças para o agente.
- Um gerador de comando, que traduz ações primitivas em comandos para os atuadores.
- Atuadores.

Os componentes de raciocínio são:

- Um banco de dados, que contém as crenças atuais e fatos, expressos em lógica de primeira ordem, sobre o mundo.
- Uma biblioteca de planos, que contém as representações simbólicas explicitas de crenças, desejos e intenções.
- Um conjunto de objetivos atual, que representa os desejos que são consistentes e que podem ser atingidos.
- Uma estrutura de intenção, que contem uma pilha com os planos selecionados em tempo de execução.
- Um interpretador, que gerencia o sistema, selecionando os planos aptos em resposta aos objetivos e crenças do sistema.

A Figura 2.8 ilustra a relação entre os componentes da arquitetura do PRS.

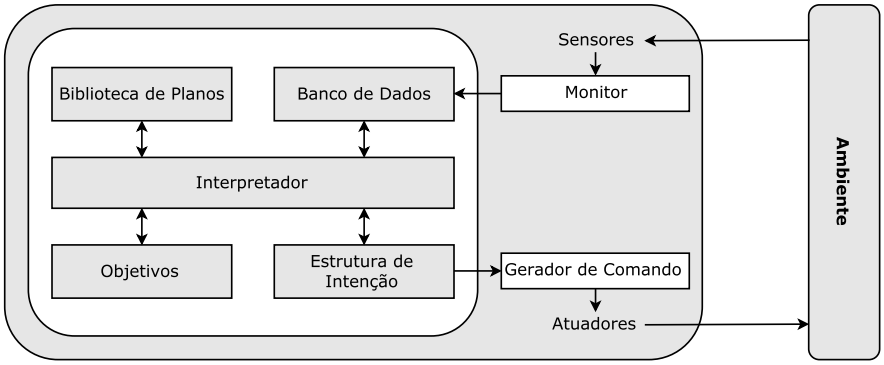

Figura 2.8 – Arquitetura do PRS.

2.4.2 dMARS

O ***D**istributed **M**ulti-**A**gent **R**easoning **S**ystem* (dMARS) é uma evolução da arquitetura PRS para construção de sistemas de domínios dinâmicos onde há conhecimento incerto e complexo. O dMARS foi desenvolvido em C++ pelo *Australian Artificial Intelligence Institute* como uma ferramenta comercial voltada para aplicações nas áreas de telecomunicações, viagens espaciais e tráfego aéreo. Segundo d'Inverno *et al.* [d'Inverno et al. 1997], os agentes utilizam a arquitetura BDI através de uma biblioteca de planos, que especifica o curso de ação que pode ser tomado pelo agente para realizar suas intenções. Cada plano pode conter um conjunto de componentes:

- Um gatilho, que descreve as circunstâncias em que o plano pode ser considerado, geralmente em termos de eventos.
- Um contexto, ou pré-condição, que especifica as circunstâncias iniciais para a execução de um plano.
- Uma condição de manutenção, que caracteriza as circunstâncias que devem permanecer verdadeiras enquanto o plano é executado.
- Um corpo com objetivos e ações primitivas, que define o curso de ação.

O interpretador, conforme mostra a Figura 2.9, é o responsável por manter a operação do agente e executa, continuamente, o seguinte ciclo:

- Observa o mundo e o estado interno do agente. Em seguida, atualiza a fila de eventos para refletir os novos eventos observados.
- Gera novos eventos possíveis (tarefas), a partir de planos encontrados com eventos compatíveis na lista de eventos.
- Seleciona, a partir do conjunto de planos compatíveis, um evento para execução.

- Adiciona o evento escolhido em uma pilha nova ou já existente de intenção, dependendo se o evento é ou não um subobjetivo.
- Seleciona uma pilha de intenção, pega o primeiro plano da pilha e executa o próximo passo do plano atual: se o passo for uma ação, executa esta ação; se for um subobjetivo, envia este para a fila de eventos.

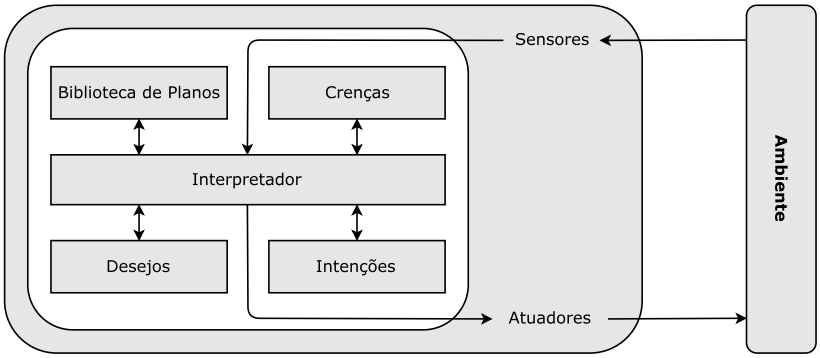

Figura 2.9 – Arquitetura do dMARS.

2.5 AgentSpeak(L)

AgentSpeak(L) é uma linguagem de programação orientada a agentes baseada em lógica de primeira ordem, com eventos e ações. Rao [Rao 1996] utilizou como base implementações da arquitetura BDI como o PRS e o dMARS para formalizar sua semântica operacional em uma linguagem abstrata de programação. Segundo o autor, o desenvolvedor utiliza as notações do AgentSpeak(L) para descrever o estado atual do agente, de seu ambiente e de outros agentes (crenças), os estados que o agente deseja atingir, baseado em estímulos externos e internos (desejos) e os planos que satisfazem um dado estímulo (intenções).

Por ser uma linguagem abstrata de programação, o AgentSpeak(L) precisa de uma implementação de um compilador ou interpretador para ser utilizado. Uma das ferramentas que surgiram para a construção de agentes utilizando o AgentSpeak(L) foi o Jason.

2.5.1 JASON

O ***J**ava-based **A**gentSpeak interpreter used with **S**ACI for multi-agent distribution **O**ver the **N**et* (JASON) é um interpretador de uma versão estendida do AgentSpeak(L). O JASON foi desenvolvido em Java e está disponível em código aberto sob a licença GNU LGPL. Segundo Bordini e Hübner [Bordini and Hübner 2005], as funcionalidades disponíveis no JASON, além daquelas definidas no AgentSpeak(L), são:

- Comunicação entre agentes baseada em atos de fala.
- Anotações nas crenças sobre as fontes de informação.
- Anotações nos rótulos dos planos para serem utilizados em funções de seleção mais elaboradas.
- Funções de seleção, funções de confiança e arquitetura geral do agente (percepção, revisão de crenças, comunicação entre agentes e ações) customizáveis em Java.
- Ambiente multiagente implementável em Java.

2.6 Interpretadores

Aho *et al.* [Aho et al. 2007, p. 2] definem interpretadores como programas que possuem como entrada um outro programa escrito em uma linguagem-fonte, e que executam suas operações de forma direta, instrução por instrução. Segundo os autores, os interpretadores possuem uma estrutura, conforme demonstra a Figura 2.10, com as seguintes fases:

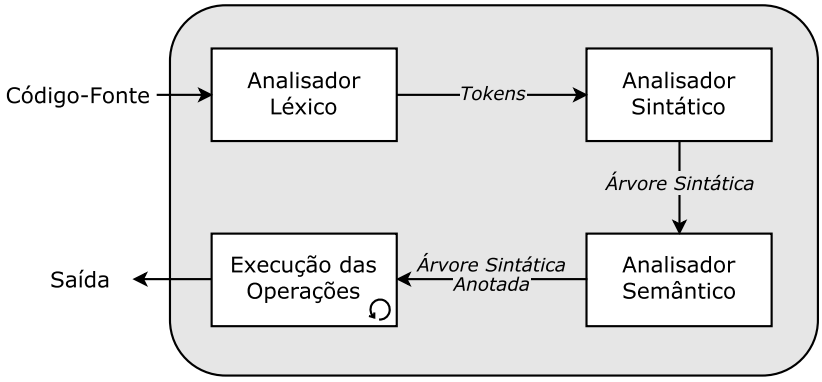

Figura 2.10 – Fases de um interpretador.

- Analisador Léxico: Realiza a leitura das linhas de caracteres que compõem o código-fonte e agrupa em uma sequência de símbolos, ou *tokens*. Os *tokens* representam palavras reservadas, identificadores, literais numéricos e de texto, operadores e pontuações.
- Analisador Sintático: Utiliza os *tokens* produzidos pelo analisador léxico e cria uma representação que determina os elementos estruturais do programa e seus relacionamentos. Esta representação é feita a partir de uma árvore sintática em que cada nó representa uma operação e os filhos representam os argumentos da operação.
- Analisador Semântico: Verifica se cada um dos operadores da árvore sintática possui um operando compatível e, se verdadeiro, tipifica o operador Se algum dos operadores não for compatível, rejeita o código-fonte.
- Execução das Operações: Se aprovado, o interpretador inicia a execução da árvore sintática anotada na etapa anterior.

2.7 Python

O Python é uma linguagem de programação criada por Guido van Rossum em 1991. De acordo com McKinney [McKinney 2013, p. 2], suas principais características são:

- Código-fonte interpretado.
- Permite o desenvolvimento de forma procedural, orientada a objetos ou funcional.
- Variáveis dinamicamente tipadas, ou seja, seu tipo pode ser alterado durante a execução do programa.
- Simplicidade no uso de estruturas de dados.
- Foco em produtividade e legibilidade, assemelhando-se com a descrição de algoritmos em pseudocódigo.
- Possui um extenso conjunto de bibliotecas que pode ser utilizado em diferentes domínios da IA, como o *SciPy* e *NumPy*, *ProbCog* e o *NLTK*. Estas bibliotecas são utilizadas, respectivamente, no desenvolvimento de aplicações voltadas para a computação científica, raciocínio probabilístico e no processamento da linguagem natural.
- Facilidade na integração com bibliotecas desenvolvidas em C, C++ e Fortran.
- Linguagem livre e multiplataforma.

Estas características permitiram que o Python se popularizasse tanto na industria de desenvolvimento de software como na comunidade científica.

3. OBJETIVOS

3.1 Objetivo Geral

O objetivo geral deste trabalho é a implementação de um interpretador para um subconjunto da sintaxe do AgentSpeak(L) para Python. Após concluído, também queremos disponibilizar o projeto livremente à comunidade para que o mesmo possa ter novas funcionalidades desenvolvidas, ou ser utilizado em outros projetos de IA.

3.2 Objetivos Específicos

- Estudar a sintaxe do AgentSpeak(L) e definir o subconjunto que será implementado.
- Estudar o ciclo de raciocínio e implementações da arquitetura BDI.
- Modelar o interpretador para Python.

4. DEFINIÇÃO DAS ATIVIDADES

4.1 Atividades Previstas

1. Estudar as definições da sintaxe do AgentSpeak(L) e definir aquelas que serão abordadas neste trabalho.
2. Estudar algumas implementações da arquitetura BDI e a forma como o ciclo de raciocínio do agente é realizado em cada uma delas. Em seguida, definir a estrutura mais adequada para abordar no interpretador.
3. Estudar as estruturas necessárias para a construção de interpretadores. Em seguida, modelar o interpretador baseado no ciclo de raciocínio definido no item anterior.
4. Escrever o volume final do TC I.

4.2 Cronograma das Atividades

A Figura 4.1 ilustra o cronograma para a realização deste trabalho. Nele, estão definidas a semana em que cada uma das atividades será executada.

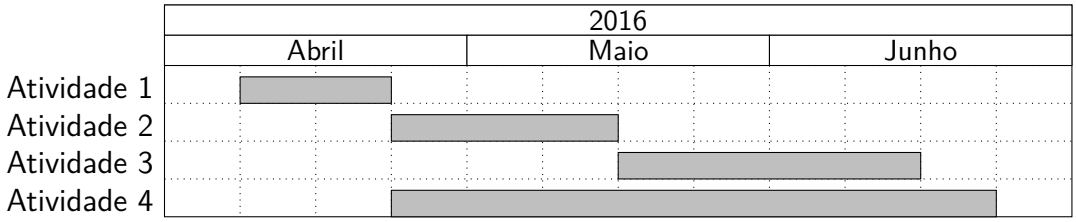

Figura 4.1 – Cronograma das atividades para o TC I.

REFERÊNCIAS BIBLIOGRÁFICAS

[Aho et al. 2007] Aho, A. V., Lam, M. S., Sethi, R., and Ullman, J. D. (2007). *Compilers : Principles, Techniques, & Tools*. Addison-Wesley, second edition.

[Bond and Gasser 1988] Bond, A. H. and Gasser, L. (1988). *Readings in Distributed Artificial Intelligence*. Morgan Kaufmann, San Francisco, CA.

[Bordini et al. 2007] Bordini, R. H., Hübner, J. F., and Wooldridge, M. (2007). *Programming Multi-Agent Systems in AgentSpeak Using Jason*. Wiley Series in Agent Technology. Wiley.

[Bordini and Hübner 2005] Bordini, R. H. and Hübner, J. F. (2005). Bdi agent programming in agentspeak using jason. In Toni, F. and Torroni, P., editors, *CLIMA*, volume 3900 of *Lecture Notes in Computer Science*, pages 143–164. Springer.

[Bratman 1987] Bratman, M. (1987). *Intention, plans, and practical reason*. Harvard University Press.

[Connelly and Goel 2013] Connelly, D. and Goel, A. K. (2013). Paradigms of artificial intelligence programming in python. *Fourth AAAI Symposium on Educational Advances in Artificial Intelligence*.

[d'Inverno et al. 1997] d'Inverno, M., Kinny, D., Luck, M., and Wooldridge, M. (1997). A formal specification of dmars. In Singh, M. P., Rao, A. S., and Wooldridge, M., editors, *ATAL*, volume 1365 of *Lecture Notes in Computer Science*, pages 155–176. Springer.

[Georgeff and Lansky 1987] Georgeff, M. and Lansky, A. L. (1987). Reactive reasoning and planning. In *Proceedings of AAAI-87*, pages 677–682.

[Maes 1991] Maes, P. (1991). The agent network architecture. *SIGART Bulletin*, 2(4):115–120.

[Mascardi et al. 2005] Mascardi, V., Demergasso, D., and Ancona, D. (2005). Languages for programming bdi-style agents: an overview. In Corradini, F., Paoli, F. D., Merelli, E., and Omicini, A., editors, *WOA*, pages 9–15. Pitagora Editrice Bologna.

[McKinney 2013] McKinney, W. (2013). *Python for data analysis.* O'Reilly.

[Rao 1996] Rao, A. S. (1996). Agentspeak(l): Bdi agents speak out in a logical computable language. In van Hoe, R., editor, *Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World*.

[Russel and Norvig 2010] Russel, S. and Norvig, P. (2010). *Artificial Intelligence: A Modern Approach*. Pearson Education Inc.

[Wooldridge 1999] Wooldridge, M. (1999). Intelligent agents. In Weiss, G., editor, *Multiagent Systems*, chapter 1, pages 27–77. MIT Press.

[Wooldridge 2000] Wooldridge, M. (2000). *Reasoning About Rational Agents*. MIT Press.

[Wooldridge 2002] Wooldridge, M. (2002). *An Introduction to MultiAgent Systems*. John wiley & Sons Ltd.

[Wooldridge and Jennings 1995] Wooldridge, M. and Jennings, N. (1995). Intelligent agents: Theory and practice. *Knowledge Engineering Review*, 2(10).