



Rafael Cauê Cardoso

**PLANEJAMENTO DE PERCEPÇÕES E AÇÕES DE AGENTES JASON NO ROBÔ
LEGO® MINDSTORMS® NXT 1.0**

Santa Maria, RS

2011

Rafael Cauê Cardoso

**PLANEJAMENTO DE PERCEPÇÕES E AÇÕES DE AGENTES JASON NO ROBÔ
LEGO® MINDSTORMS® NXT 1.0**

Trabalho Final de Graduação apresentado ao Curso de Ciência da Computação – Área de Ciências Tecnológicas, do Centro Universitário Franciscano, como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Alexandre de Oliveira Zamberlan

Santa Maria, RS

2011

Rafael Cauê Cardoso

**PLANEJAMENTO DE PERCEPÇÕES E AÇÕES DE AGENTES JASON NO ROBÔ
LEGO® MINDSTORMS® NXT 1.0**

Trabalho final de graduação apresentado ao Curso de Ciência da Computação – Área de Ciências Tecnológicas, do Centro Universitário Franciscano, como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Alexandre de Oliveira Zamberlan – Orientador (UNIFRA)

Henrique Gabriel Gularte Pereira (UNIFRA)

Guilherme Chagas Kurtz (UNIFRA)

Aprovado em 30 de novembro de 2011

TERMO DE RESPONSABILIDADE

Eu, _____, brasileiro(a),
_____, portador de RG nº _____,
estudante do curso de _____ do
Centro Universitário Franciscano – UNIFRA, Santa Maria/RS, declaro
para os devidos fins que assumo integralmente a responsabilidade pelo
conteúdo, idéias e citações constantes em meu Trabalho Final de
Graduação – TFG, bem como o *software* desenvolvido no âmbito de
meu trabalho, isentando a Universidade, o professor orientador e os
professores componentes da banca de qualquer responsabilidade.

Declaro, ainda, que estou ciente de que na hipótese de
constatação de plágio poderei responder administrativa, civil e
criminalmente, sob as penas da lei.

Santa Maria, 15 de junho de 2011.

Testemunha

Testemunha

RESUMO

Este trabalho busca integrar duas áreas de estudo da Inteligência Artificial, a Robótica e Agentes Inteligentes. Apesar de já existir um grande número de estudos científicos sobre ambas as áreas separadamente, quando aplicadas em conjunto esse número diminui consideravelmente. Por esse motivo, objetiva-se realizar o planejamento de ações e percepções de agentes inteligentes BDI para robôs LEGO Mindstorms NXT. Para isso, é utilizado o *framework* Jason para programação de sistemas multiagentes, a linguagem de programação leJOS NXJ para a comunicação entre o computador e o robô, e a modelagem do sistema via a metodologia Prometheus. Assim, espera-se demonstrar a utilidade na implementação de agentes inteligentes em robôs reais (físicos).

Palavras-chaves: Robótica. LEGO Mindstorms NXT. Agentes BDI. Prometheus. Jason.

ABSTRACT

This project intends to integrate two study fields from Artificial Intelligence - Robotics and Intelligent Agents. Despite the current large number of scientific studies about both fields separately, when applied together this number drops considerably. For this reason, it is performed the planning of actions and perceptions of BDI intelligent agents for the brand of robots LEGO Mindstorms NXT. It is utilized the Jason framework for multi-agent programming, the leJOS NXJ programming language for the communication between computer and robot, and the Prometheus methodology for the system modeling. Thus, a demonstration about the usefulness in implementing intelligent agents on physical robots is expected.

Keywords: Robotics. LEGO Mindstorms NXT. BDI Agents. Prometheus. Jason.

LISTA DE ABREVIATURAS E SIGLAS

AOS	<i>Agent Oriented Software</i>
AOSE	<i>Agent-Oriented Software Engineering</i>
API	<i>Application Programming Interface</i>
BDI	<i>Beliefs, Desires and Intentions</i>
BricxCC	<i>Bricx Command Center</i>
GUI	<i>Graphical User Interface</i>
IDE	<i>Integrated Development Environment</i>
JVM	<i>Java Virtual Machine</i>
leJOS	<i>LEGO Java Operating System</i>
MAS	<i>Multi-Agent System</i>
MIT	<i>Massachusetts Institute of Technology</i>
NASA	<i>National Aeronautics and Space Administration</i>
NBC	<i>Next Byte Codes</i>
NXC	<i>Not Exactly C</i>
NXJ	<i>NXT Java</i>
NXT	<i>Next - Como em "o próximo passo"</i>
PDT	<i>Prometheus Design Tool</i>
PRS	<i>Procedural Reasoning System</i>
RCX	<i>Robotic Control Explorer</i>
RIS	<i>Robotics Invention System</i>
SMA	<i>Sistema Multiagente</i>
URBI	<i>Universal Robot Body Interface</i>

LISTA DE FIGURAS

Figura 1 - Kit original do <i>Robotics Invention System</i> (RIS).....	12
Figura 2 - Modelos de robôs das diferentes versões dos Kits LEGO.....	13
Figura 3 - Caixa do Kit LEGO Mindstorms NXT 1.0.....	14
Figura 4 - Interface do <i>brick</i> programável NXT.....	15
Figura 5 - Visão geral do <i>brick</i> programável.....	16
Figura 6 - O Sojourner da NASA, robô que explorou a superfície de Marte.....	17
Figura 7 - Robôs RCX utilizados na simulação da coleta de lixo hospitalar	18
Figura 8 - Modelo geral de um agente.....	23
Figura 9 - Esquema genérico do modelo BDI.....	24
Figura 10 - Exemplo de planos em AgentSpeak(L)	26
Figura 11 - Sintaxe da linguagem AgentSpeak(L)	27
Figura 12 - Esquema com as três fases de desenvolvimento da metodologia Prometheus	29
Figura 13 - Legenda dos componentes utilizados na modelagem deste trabalho.....	30
Figura 14 - Esquema do Agente Robótico.....	33
Figura 15 - Modelo da pista a ser percorrida pelo agente	34
Figura 16 - Estrutura do robô para a tarefa proposta.....	35
Figura 17 - Diagrama da visão geral de planos	36
Figura 18 - Diagrama de cenários.....	36
Figura 19 - Diagrama das funcionalidades do sistema.....	37
Figura 20 - Diagrama de acoplamento de dados	38
Figura 21 - Diagrama de acoplamento de funcionalidades dos agentes.....	38
Figura 22 - Diagrama de visão geral do sistema	39
Figura 23 - Diagrama de visão geral do Agente Robô	39
Figura 24 - Diagrama da Capacidade permanecer dentro no circuito	40
Figura 25 - Diagrama da Capacidade cruzar linha de chegada	40
Figura 26 - Diagrama da Capacidade desviar de um obstáculo	40
Figura 27 - Estrutura de diretórios.....	41
Figura 28 - Exemplo de um projeto em Jason-NXT	42
Figura 29 - Padrão de declaração de agentes.....	42
Figura 30 - Códificação do agente 'tfg.asl'	44
Figura 31 – Interface gráfica do usuário (traduzida)	45
Figura 32 - Erro encontrado durante a implementação	46
Figura 33 - Solução para o erro da Figura 32.....	46
Figura 34 – Pista de corrida alterada	46

LISTA DE QUADROS

Quadro 1 - Algumas características das linguagens de programação para o NXT	20
Quadro 2 - Algumas linguagens/ambientes de programação BDI	25
Quadro 3 - Valores que podem ser assumidos pelos argumentos	42
Quadro 4 - Valores assumidos nos argumentos do agente 'tfg.asl'	43

SUMÁRIO

1 INTRODUÇÃO	10
2 ROBÓTICA LEGO MINDSTORMS	12
2.1 LEGO MINDSTORMS NXT 1.0	14
2.1.1 Especificações Técnicas	14
2.1.2 Aplicabilidade	17
2.1.3 Comunicação	18
2.1.4 Linguagens de Programação	19
2.2 LINGUAGEM DE PROGRAMAÇÃO LEJOS NXJ	21
2.2.1 Características	21
3 TEORIA DE AGENTES	22
3.1 PROPRIEDADES	22
3.2 TIPOS DE AGENTES	23
3.2.1 Modelo BDI.....	24
3.3 LINGUAGENS E AMBIENTES DE PROGRAMAÇÃO	25
3.3.1 AgentSpeak(L)	26
3.3.2 Jason.....	27
3.4 MODELAGEM DE AGENTES	28
3.4.1 Metodologia Prometheus.....	29
4 TRABALHOS CORRELATOS	31
5 PROPOSTA DE TRABALHO.....	33
5.1 MODELAGEM.....	35
5.1.1 Especificação do Sistema	36
5.1.2 Projeto Arquitetural.....	38
5.1.3 Projeto Detalhado.....	39
5.2 IMPLEMENTAÇÃO	40
6 CONCLUSÃO	48
REFERÊNCIAS BIBLIOGRÁFICAS	50
ANEXO A - Extensões da sintaxe do Jason.....	57
ANEXO B - Estrutura dos códigos fontes.....	60
APÊNDICE A - Tutorial para a instalação dos componentes	63

1 INTRODUÇÃO

A Robótica é uma área que se encontra em constante expansão, pois ela é utilizada em conjunto com diversas outras áreas como por exemplo Medicina e Educação. Na Medicina, há microcirurgias com o sistema HipNav (DIGOIA; KANADE; WELLS, 1996), que cria um modelo tridimensional da anatomia de um paciente, e utiliza controle robótico para orientar a inserção de uma prótese. Já para a área da Educação (lúdica), existe uma linha de brinquedos robóticos conhecida como LEGO Mindstorms, que possibilita a montagem de robôs utilizando peças LEGO e a programação do comportamento desses robôs por meio de um *brick* programável (bloco programável).

A Robótica também está associada à teoria de agentes da área da Inteligência Artificial. De acordo com Jensen (2010), a maioria dos trabalhos em agentes inteligentes são realizados com agentes virtuais (agentes de *software*). Dessa forma, pretende-se explorar o uso de agentes inteligentes em robôs reais, e para isso, será projetado e implementado um agente através do *framework* para sistemas multiagentes Jason, que realiza a comunicação com o robô LEGO Mindstorms NXT implementado na linguagem leJOS NXJ. A modelagem do sistema proposto é realizada com a metodologia Prometheus da AOSE (Engenharia de *Software* Orientada a Agentes).

A linguagem leJOS substitui o *firmware* padrão da LEGO implementando no *brick* do robô uma máquina virtual, o que permite a programação no ambiente Java (LEJOS, 2011). Já o *framework* Jason é um interpretador para a linguagem de programação de agentes AgentSpeak(L), que implementa as semânticas operacionais dessa linguagem, e fornece uma plataforma para desenvolvimento de sistemas multiagentes (BORDINI; VIEIRA, 2003). Um agente, segundo Russel e Norvig (2004, p.33), "é tudo o que pode ser considerado capaz de perceber seu ambiente por meio de sensores e de agir sobre esse ambiente por intermédio de atuadores".

Com base no contexto apresentado, projeta-se que o agente deve ser composto por 'cérebro', 'atuadores' e 'olhos', onde a programação em Jason representa o 'cérebro' do agente, via crenças e planos, executando em um computador. Essa programação faz a comunicação com os 'atuadores' (motores) e 'olhos' (sensores) do robô LEGO Mindstorms NXT, por meio do seu *brick* que executa o código na linguagem leJOS. Então, os sensores do robô devem captar as mudanças ocorridas no ambiente e enviar essas percepções para o 'cérebro' do agente

no computador, onde se encontra todo o planejamento de seu comportamento, que por sua vez envia a intenção resultante do plano escolhido ao robô que realiza a sua execução.

Sendo assim, o presente trabalho tem como objetivo geral projetar e implementar um agente via o *framework* Jason que realize comunicação com o robô LEGO Mindstorms NXT. A tarefa à ser realizada por este agente consiste em achar a saída de um simples percurso com obstáculos, apenas para demonstrar algumas das vantagens da modelagem e implementação de agentes com a extensão do *framework* Jason para robôs LEGO Mindstorms NXT utilizando o *firmware* leJOS NXJ. E como objetivos específicos assumem-se:

- modelar o comportamento do agente segundo a metodologia Prometheus;
- pesquisar e testar a programação de agentes utilizando Jason;
- pesquisar e testar a linguagem de programação leJOS para o robô LEGO Mindstorms NXT;
- pesquisar e testar a comunicação entre Jason e leJOS.

Finalmente, para melhor compreensão do trabalho proposto, o texto está dividido em seis capítulos. No Capítulo 2 é abordada a robótica LEGO, características e linguagens de programação para o Kit LEGO. O Capítulo 3 apresenta a teoria de agentes, com alguns conceitos fundamentais e, principalmente, as linguagens de implementação de agentes. No Capítulo 4, disserta-se sobre alguns trabalhos científicos que envolvem a tecnologia LEGO Mindstorms NXT e a área de Inteligência Artificial. O Capítulo 5 apresenta a proposta, a modelagem, e a implementação deste trabalho final de graduação. No capítulo 6, encontram-se os resultados e algumas conclusões sobre este trabalho, e por fim, as referências bibliográficas.

2 ROBÓTICA LEGO MINDSTORMS

Neste capítulo, são apresentadas as diferentes versões dos robôs LEGO e suas linguagens de programação, mais especificamente o robô LEGO Mindstorms NXT 1.0 e a linguagem de programação leJOS NXJ, que foram utilizados na proposta deste trabalho.

Existem várias definições para o que é um robô, porém a que mais se encaixa com a proposta deste trabalho é a definição sugerida por Stuart Russel e Peter Norvig (RUSSEL; NORVIG, 2004, p.870):

Os **robôs** são agentes físicos que executam tarefas manipulando o mundo físico. Para isso, eles são equipados com **efetadores**¹ como pernas, rodas, articulações e garras. Os efetadores têm um único propósito: exercer forças físicas sobre o ambiente. Os robôs também estão equipados com **sensores**, que lhes permitem perceber seu ambiente.

O Kit² LEGO Mindstorms (MINDSTORMS, 2011), desenvolvido pela LEGO, é composto de peças de brinquedo (*bricks*) LEGO que podem ser facilmente conectadas e utilizadas com motores e sensores, tornando possível projetar e construir robôs reais (robôs físicos que atuam em um mundo físico) com facilidade.

Em 1980³, o cientista Seymour Papert publicou a primeira edição do livro (PAPERT, 1980). O livro envolvia o conceito de não apenas usar computadores para avaliar as crianças, mas também para que elas utilizassem os computadores para invenções e na solução de problemas. Foi através da ideia proposta por este livro que o nome Mindstorms foi adotado pela LEGO (FORD JR, 2011).



Figura 1 - Kit original do *Robotics Invention System* (RIS) (GASPERI; HURBAIN; HURBAIN, 2007, p.2)

¹ Apesar de Russel e Norvig utilizarem o termo "efetadores", outros autores preferem o uso do termo "atuadores". Por questão de simplificação, neste trabalho é utilizado "atuadores".

² Conjunto de peças ou materiais a serem montadas (MICHAELIS, 2011).

³ Na bibliografia consultada, existe uma divergência no ano de publicação do livro (1980 e 1993), pois em 1993 foi lançada a segunda edição do livro.

A primeira versão do produto (Figura 1), denominada RIS (*Robotics Invention System*), foi lançada em 1998 e possuía um *brick* programável⁴ chamado RCX (*Robotic Control Explorer*). Com o sucesso do RIS, a LEGO começou a trabalhar em seu sucessor, que apareceria no mercado oito anos após o lançamento do Kit RIS.

Seu sucessor ficou conhecido em 2006 como LEGO Mindstorms NXT 1.0, que demonstrou ser um aprimoramento do RIS em todos os aspectos. A LEGO não investiu apenas no nível do produto em si, mas também no seu lançamento e na documentação detalhada (GASPERI; HURBAIN; HURBAIN, 2007).

Com tantos aprimoramentos, o LEGO Mindstorms NXT 1.0 conseguiu fazer mais sucesso que o seu antecessor, o que motivou a LEGO a lançar uma nova versão do Kit Mindstorms. Em 2009, apenas três anos após lançamento do 1.0, já se encontrava no mercado o LEGO Mindstorms NXT 2.0.

Alguns exemplos do que pode ser construído com as diferentes versões do Kit LEGO Mindstorms podem ser observados na Figura 2. Na Figura 2.a é possível visualizar quatro modelos que o Kit Mindstorms 2.0 ensina a construir e programar (MINDSTORMS, 2011). Já na Figura 2.b, encontra-se um modelo desenvolvido para subir escadas (AGULLÓ et al, 2003, p.1) utilizando o Kit RIS. Finalmente, na Figura 2.c é apresentado o robô JohnNXT (BENEDETTELLI, 2008, p.516), uma réplica do robô Johnny 5 dos filmes de 1980 *Short Circuit*⁵ criado com o Kit Mindstorms 1.0.

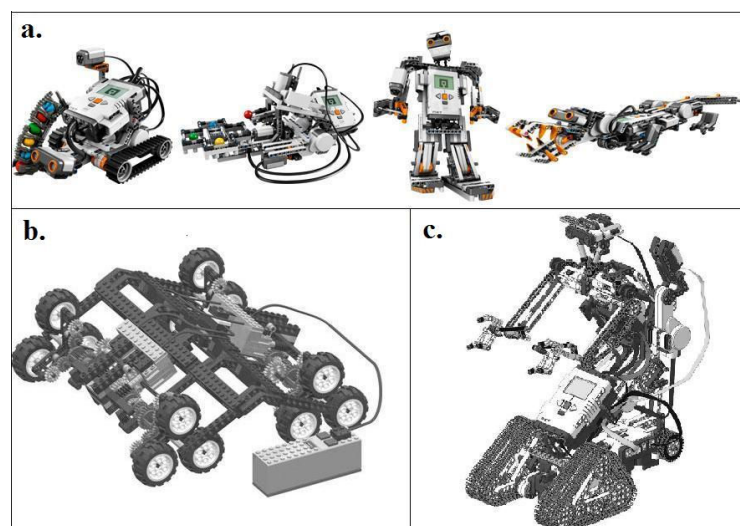


Figura 2 - Modelos de robôs das diferentes versões dos Kits LEGO

⁴ Microcomputador desenvolvido primeiramente no M.I.T. (Instituto de Tecnologia de Massachusetts) *Media Lab*, e mais tarde comercializado pela LEGO (BULL, 2005).

⁵ No filme, o robô chamado Johnny 5, um dos cinco protótipos construídos para finalidades militares, é atingido por um raio enquanto estava sendo recarregado. Obtendo autoconsciência, senso de humor, e uma compreensão do valor da vida.

2.1 LEGO MINDSTORMS NXT 1.0

A principal diferença do Kit LEGO Mindstorms NXT 2.0 para o Kit 1.0 (Figura 3), é a troca do sensor de luz por um sensor de cor que possui três funcionalidades: atua como o sensor de luz, detectando a intensidade da luz; atua como o sensor de cor, capaz de distinguir entre seis cores (branco, preto, amarelo, vermelho, verde e azul); e também funciona como uma lâmpada colorida (MINDSTORMS, 2011).



Figura 3 - Caixa do Kit LEGO Mindstorms NXT 1.0
(GASPERI; HURBAIN; HURBAIN, 2007, p.4)

O sensor de som não está mais incluído no conteúdo da caixa padrão, porém ele funciona caso seja adquirido separadamente, assim como os demais sensores do Kit LEGO Mindstorms NXT 1.0. No lugar do sensor de som foi incluído mais um sensor de contato⁶, e foram apresentados quatro novos modelos de robôs (Figura 2.a) e seus respectivos guias de montagem e programação.

2.1.1 Especificações Técnicas

Conteúdo do Kit LEGO Mindstorms NXT 1.0 (LEGO, 2006):

- 619 peças de brinquedo LEGO Technic⁷ - elementos de montagem, rodas, engrenagens, etc.;

⁶ Também chamado de sensor de toque, do inglês *touch sensor*.

⁷ Blocos e peças um pouco mais sofisticadas do que as das linhas básicas e temáticas (LEGO, 2006).

- Três servomotores⁸ interativos, cada um com um sensor de rotação capaz de medir a rotação em graus;
- Um sensor ultrassônico - detecta objetos e a sua distância aos objetos;
- Um sensor de som - permite medir o nível de ruído no ambiente;
- Um sensor de luz - determina a intensidade de luz (presença/ausência em porcentagem);
- Um sensor de contato - reage ao contato ou liberação a objetos;
- Manual do usuário;
- CD-ROM com a linguagem de programação NXT-G;
- Cartolina com percurso de teste para testar os modelos padrões de robôs;
- Um microcomputador (*brick* programável) NXT - atua como o cérebro⁹ do robô.

Na Figura 4, observa-se a funcionalidade da interface do *brick* programável e as suas portas/botões, enquanto que na Figura 5 é possível ver o *brick* programável já conectado com os diferentes tipos de sensores e os três servomotores.



Figura 4 - Interface do *brick* programável NXT
Imagem traduzida de (FORD JR, 2011, p.52)

⁸ Dispositivo cujo posicionamento acompanha um sinal de controle, e possuem liberdade de rotação de aproximadamente 180° graus (http://www.societyofrobots.com/actuators_servos.shtml).

⁹ Apesar de ser utilizado amplamente na bibliografia para representar o *brick* programável, daqui em diante o termo 'cérebro' será utilizado para referenciar a parte do agente em Jason, e não o *brick*.

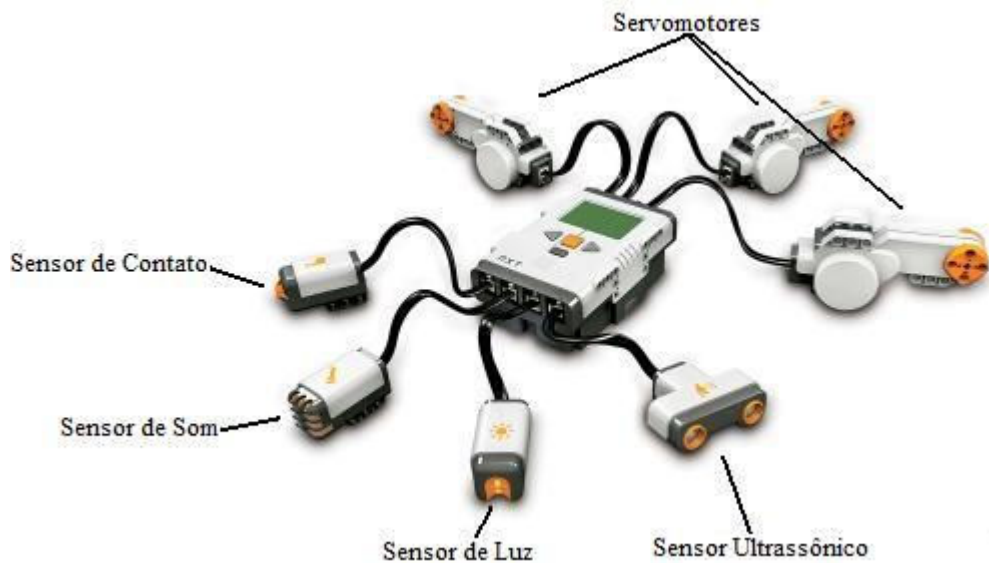


Figura 5 - Visão geral do *brick* programável
 Imagem traduzida de (PANADERO; ROMÁN; KLOOS, 2010, p.504)

Especificações técnicas do *brick* programável NXT (FORD JR, 2011):

- Microcomputador 32-bit ARM7, memória 256 KB Flash, 64 KB RAM;
- Microcontrolador 8-bit AVR, memória 4 KB Flash, 512 bytes RAM;
- *Bluetooth Wireless* (Classe II versão 2.0);
- Porta de comunicação USB 2.0;
- Quatro portas de entrada;
- Três portas de saída;
- Tela LCD monocromática 100x64 pixels;
- Autofalante de 8 kHz de qualidade de som;
- Energia é fornecida por 6 pilhas AA (1,5 Volts) ou por uma bateria recarregável de íon-lítio.

Um dos principais motivos para o sucesso da versão NXT é devido a sua vasta documentação disponibilizada. A LEGO Mindstorms possui em seu site na seção de suporte¹⁰ diversos Kits de desenvolvimento (*Bluetooth*, *Hardware*, *Software*, Aplicações Móveis), além do código fonte aberto de seu *firmware*¹¹, o que possibilitou sua base de fãs a desenvolver diversas linguagens de programação, assunto tratado no subcapítulo 2.1.4.

¹⁰ <http://mindstorms.LEGO.com/en-us/support/files/default.aspx#Advanced>

¹¹ Conjunto de instruções operacionais programadas diretamente no *hardware* de um equipamento eletrônico, armazenado permanentemente em um chip (OPLER, 1967).

2.1.2 Aplicabilidade

Russel e Norvig (2004) listam alguns dos domínios de aplicação da Robótica: indústria e agricultura (linha de montagem); transporte (veículos autônomos); ambientes arriscados (desarme de bombas, limpeza de resíduos nucleares); exploração (Figura 6, fundo do mar); cuidados com a saúde (auxiliar cirurgias); serviços pessoais (robôs serviçais); entretenimento (futebol robótico); ampliação da capacidade humana (membros robóticos artificiais).

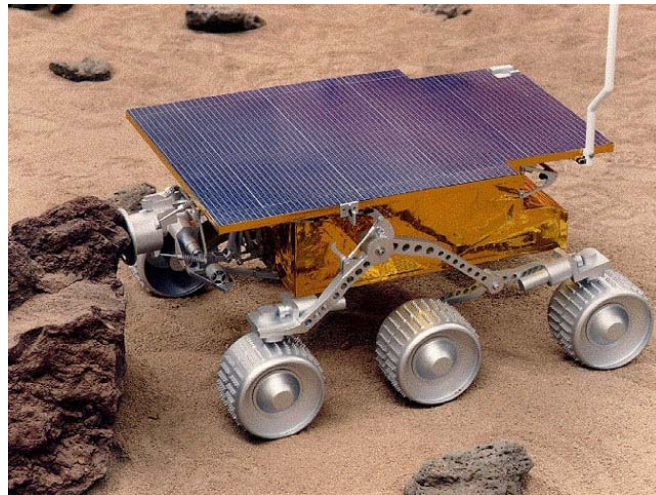


Figura 6 - O Sojourner da NASA, robô que explorou a superfície de Marte (RUSSEL; NORVIG, 2004, p.871)

Seymour Papert, mencionado na introdução do capítulo 2, foi talvez um dos primeiros a sugerir que crianças deveriam programar computadores, uma ideia absurda na época (FORD JR, 2011). Essa ideia levou a formação da teoria de aprendizado de Papert chamada construcionismo, que segundo (RIBEIRO, 2006, p.43): "propõe a ideia de que os seres humanos aprendem melhor quando são envolvidas no planejamento e na construção de objectos ou artefactos que considerem significativos".

Ele acreditava que a dificuldade de entender diversos conceitos é a falta de ferramentas do mundo real que podem demonstrar tal conceito. Robôs programáveis são flexíveis e poderosos o suficiente para demonstrar ideias que anteriormente não possuíam nenhuma analogia no mundo real (ANSORGE, 2006). A linguagem de programação Logo¹², criada por Papert, e o Kit LEGO Mindstorms herdam deste conjunto de conceitos.

O Kit LEGO Mindstorms pode ser utilizado como ferramenta auxiliar no ensino de diversas áreas:

¹² É uma variação da linguagem de programação Lisp, projetada para o auxílio na educação (LOGO, 2011).

- Diversas disciplinas do ensino básico (RIBEIRO, 2006);
- Matemática (FAGUNDES et al, 2005);
- Engenharia (BEHRENS et al, 2010);
- Disciplinas introdutórias de programação (MOTA, 2007) e (LUI; CHEUNG; GURUNG, 2010);
- Linguagem de programação *Assembly* (JIPPING et al, 2007);
- Linguagem de programação Java (GANDY et al, 2010);
- Desenvolvimento avançado de *software* (LEW; HORTON; SHERIFF, 2010);
- Inteligência Artificial (KUMAR, 2004) e (PANADERO; ROMÁN; KLOOS, 2010), área presente nesse trabalho.

Registra-se também o uso do Kit LEGO Mindstorms em processos de modelagem e simulação, por conta de sua modularidade e da facilidade na sua utilização. Como por exemplo, na simulação de um ambiente de coleta de lixo hospitalar utilizando protótipos do Kit LEGO Mindstorms (PFEIFER et al, 2006), representados na Figura 7.

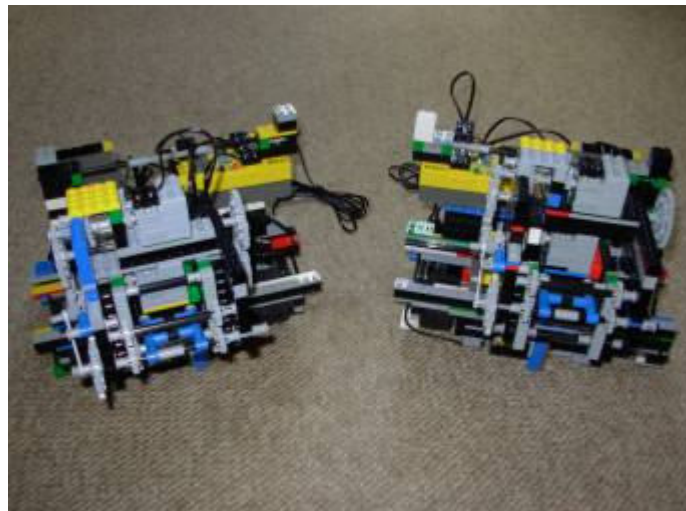


Figura 7 - Robôs RCX utilizados na simulação da coleta de lixo hospitalar (PFEIFER et al, 2006, P.78)

2.1.3 Comunicação

O *brick* programável NXT dispõe de duas maneiras para se comunicar com outro dispositivo: utilizando o cabo USB incluído no Kit LEGO Mindstorms, ou utilizando a tecnologia *Bluetooth* para comunicação sem fio. Porém para usufruir da conexão sem fio, o

dispositivo deve possuir *Bluetooth* embutido, ou então possuir um adaptador. Neste trabalho foi utilizado um adaptador para a comunicação sem fio.

Bluetooth é uma forma de comunicação sem fio de curto alcance entre dispositivos. Requer que ambos dispositivos se identifiquem um ao outro, e que uma senha¹³ seja permutada para conceder permissão aos dispositivos à se comunicarem (KELLY, 2010). Por pertencer a classe II, como visto no subcapítulo 2.1.1, possui um alcance aproximado de até 10 metros.

Enquanto que a comunicação por USB é mais rápida, ela também limita muito a utilidade do robô, que estará preso ao dispositivo por um cabo. Usando *Bluetooth*, o robô pode se mover livremente, dentro do alcance da conexão, mas também proporciona um consumo maior de energia, exaurindo a bateria mais rapidamente.

2.1.4 Linguagens de Programação

Por conta da documentação e dos códigos fontes disponibilizados pela LEGO, programadores tem a sua disposição uma variedade de linguagens de programação, como: NBC (*Next Byte Codes*), NXC (*Not eXactly C*), ROBOTC, pbLua, URBI (*Universal Robot Body Interface*), leJOS NXJ, e finalmente, o *software* padrão da LEGO NXT-G, incluído no Kit LEGO Mindstorms.

NBC (NBC/NXC, 2011) é uma linguagem de código aberto, que utiliza a sintaxe da linguagem *assembly*. NXC (NBC/NXC, 2011) é uma linguagem de alto nível, também de código aberto, semelhante a C, baseado no compilador da NBC. Ambas linguagens utilizam a IDE (*Integrated Development Environment*)¹⁴ BricxCC (*Bricx Command Center*).

ROBOTC (ROBOTC, 2011) é uma linguagem de programação baseada em C, com ferramentas de depuração embutidas, modelos padrão de código para novos documentos, e a capacidade de realizar operações trigonométricas. Seu *firmware* é otimizado, o que melhora a performance do NXT, além de comprimir os seus arquivos para que caiba uma quantidade maior de programas no *brick*.

pbLua (PBLUA, 2011) é uma linguagem de programação baseada na linguagem Lua, escrita em C, e possui uma extensa documentação disponível online. URBI (URBI, 2011) é uma linguagem baseada em eventos, possui paralelismo, interfaces para diversas linguagens

¹³ Também chamada de *passcode*.

¹⁴ Conjunto de ferramentas que auxiliam o desenvolvimento de aplicações, como por exemplo ver os erros enquanto está digitando, automatizar tarefas repetitivas, compilar códigos, etc. (<http://java.sun.com/developer/technicalArticles/tools/intro.html>).

(C, C++, Java), reconhecimento de voz, reconhecimento facial, mapeamento e localização simultâneos.

leJOS NXJ (LEJOS, 2011) é uma linguagem de programação de alto nível baseada na linguagem Java. Programadores mais experientes podem usufruir de reconhecimento de voz, GPS, e tecnologias de mapeamento.

NXT-G (MINDSTORMS, 2011) é a linguagem de programação presente no Kit LEGO Mindstorms, baseado na IDE LabVIEW da *National Instruments*¹⁵. É uma linguagem altamente visual, consistindo de blocos que executam as diversas funções, e possui um ambiente interativo de "arrastar e soltar" tais blocos. Existem diferentes versões da IDE para a linguagem NXT-G: a versão 1.0 que vem com o Kit LEGO Mindstorms NXT 1.0; a versão 2.0 que vem com o Kit LEGO Mindstorms NXT 2.0; e a versão educacional que vem com o Kit LEGO Mindstorms Education NXT.

Comparado com as outras linguagens de programação, programas em NXT-G tendem a ser maiores (em relação ao tamanho que ocupa no disco rígido), e por este motivo necessitam de mais espaço de armazenamento, além de deixarem o processo de transferência de código entre um dispositivo e um *brick* programável mais lento (FORD JR, 2011). Percebe-se, através do Quadro 1, que apenas as linguagens de programação de código aberto (*software* livre) são compatíveis com o sistema operacional GNU/Linux.

Linguagem	Sistema Operacional	Trocar <i>Firmware</i>	<i>Software</i> Proprietário
NBC	GNU/Linux, Windows, Mac OS X	Não	Não
NXC	GNU/Linux, Windows, Mac OS X	Não	Não
ROBOTC	Windows	Não	Sim
pbLua	GNU/Linux, Windows, Mac OS X	Sim	Não
URBI	GNU/Linux, Windows, Mac OS X	Sim	Algumas funcionalidades
leJOS NXJ	GNU/Linux, Windows, Mac OS X	Sim	Não
NXT-G	Windows, Mac OS X	Não	Sim

Quadro 1 - Algumas características das linguagens de programação para o NXT

¹⁵ Companhia americana produtora de equipamento de teste automatizado e *software* de instrumentação virtual (<http://www.ni.com/>).

2.2 LINGUAGEM DE PROGRAMAÇÃO LEJOS NXJ

A linguagem de programação leJOS NXJ implementa uma pequena máquina virtual Java (JVM)¹⁶. Porém, por conta do *hardware* presente no *brick* programável NXT ser muito limitado, não foram incluídas todas as classes da linguagem Java (JENSEN, 2010). Ela possui *plugins*¹⁷ para as IDEs Eclipse e Netbeans, além de permitir a execução e a compilação diretamente pela linha de comando.

2.2.1 Características

Entre suas principais funcionalidades, encontram-se (LEJOS, 2011):

- Linguagem orientada a objetos (Java);
- Threads excludentes (tarefas);
- Vetores multidimensionais;
- Recursividade;
- Sincronização;
- Exceções;
- Tipos de dado Java (float, long, String);
- Grande parte das classes java.lang, java.util, e java.io;
- Vasta documentação (LEJOSAPI, 2011) da API (Application Programming Interface)¹⁸.

Finalmente, a partir da base teórica sobre o robô LEGO Mindstorms NXT 1.0 e a linguagem de programação leJOS NXJ, pode-se entrar na área de Agentes Inteligentes, assunto do próximo capítulo.

¹⁶ Máquina virtual é uma arquitetura de computador. O compilador Java produz código para JVM, que então é normalmente executado por um programa interpretador da JVM (TANENBAUM, 2003).

¹⁷ Conjunto de componentes de *software* que adicionam funções específicas para um outro *software* (PC_MAGAZINE, 2011).

¹⁸ Traduzido como Interface de Programação de Aplicação, é um conjunto de regras e padrões que podem ser utilizados por programas de *software* para se comunicarem entre si. Criados com a finalidade de definir seus vocabulários e recursos (PC_MAGAZINE, 2011).

3 TEORIA DE AGENTES

No capítulo que se segue, são discutidos tipos, arquiteturas e modelos de agentes. Além disso, algumas linguagens e ambientes de programação são abordados, inclusive a metodologia de modelagem de agentes a ser utilizada no trabalho proposto.

Agentes Inteligentes fazem parte da subárea da Ciência da Computação chamada Inteligência Artificial. Agentes são, segundo (REZENDE, 2005, p.269), "personagens computacionais que atuam de acordo com um *script* definido, direta ou indiretamente, por um usuário. Eles podem atuar isoladamente ou em comunidade formando sistemas multiagentes".

Outro conceito, proposto por Russel e Norvig (2004, p.33), se relaciona melhor com a proposta do trabalho, "Um agente é tudo o que pode ser considerado capaz de perceber seu ambiente por meio de sensores e de agir sobre esse ambiente por intermédio de atuadores".

Devido a grande quantidade de conceitos sobre agentes, os pesquisadores da área não possuem uma definição de consenso para o que é um agente (HÜBNER; BORDINI; VIEIRA, 2004). Para um aprofundamento sobre o tema, recomenda-se a leitura de (WOOLDRIDGE; JENNINGS, 1995) e (FRANKLIN; GRAESSER, 1997).

O foco de pesquisa em Sistemas Multiagentes (SMA)¹⁹ encontra-se nos meios com os quais os agentes irão comunicar e cooperar entre si para resolver um problema, quando este for apresentado à sociedade de agentes, e não diretamente na solução do problema em questão (REZENDE, 2005).

3.1 PROPRIEDADES

Na Figura 8, o agente é capaz de perceber alterações no ambiente (percepções), e agir (ações) para transformar o ambiente de seu estado atual para o estado desejado (estado objetivo) pelo agente. O que gera novas alterações no ambiente, ou seja, novas percepções.

Caso o agente pertença a uma sociedade, ele deve ser capaz de se comunicar com os outros agentes que compartilham o mesmo ambiente para coordenar suas ações. Também, é necessário que o agente possua uma representação de estados do ambiente (motivação) que o

¹⁹ Do inglês Multi-Agent System (MAS), suas principais características, problemas, e aplicações são encontradas em (BOISSIER, 1993), (SICHTMAN, 1995), (ALVARES; SICHTMAN, 1997), (BORDINI; VIEIRA; MOREIRA, 2001), (DEMAZEAU; MÜLLER, 1990), (FERBER, 1999), (JENNINGS; WOOLDRIDGE, 1998), (WEIß, 1999) e (WOOLDRIDGE, 2009).

agente deseja alcançar, para que ele aja por iniciativa própria para alcançar tais estados (HÜBNER; BORDINI; VIEIRA, 2004).

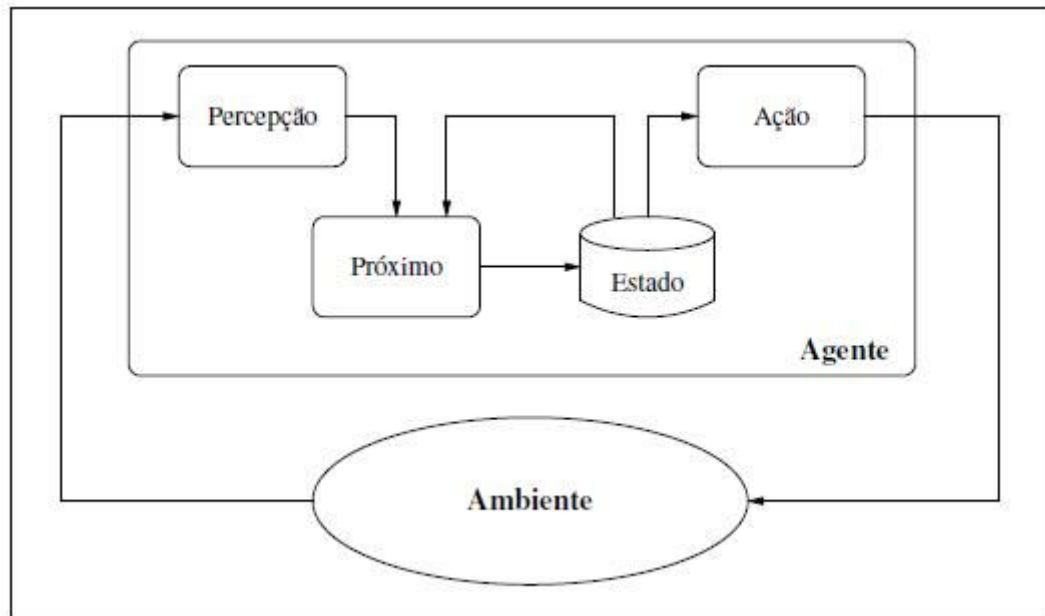


Figura 8 - Modelo geral de um agente
Traduzido de (WOOLDRIDGE, 1999) por (HÜBNER; BORDINI; VIEIRA, 2004)

Logo, dada a motivação do agente e a representação do estado atual do ambiente, o agente deve ser capaz de decidir dentre os possíveis estados seguintes (deliberação). Para aumentar o desempenho na escolha de estados, é possível implementar um sistema de raciocínio e aprendizagem, técnicas da Inteligência Artificial (HÜBNER; BORDINI; VIEIRA, 2004).

3.2 TIPOS DE AGENTES

Os agentes podem ser classificados basicamente em cognitivos ou reativos. Em um agente reativo, a escolha da ação está relacionada com um conjunto de eventos percebidos do ambiente, através dos seus sensores ou ainda por mensagens de outros agentes. Para um agente cognitivo, sua ação é escolhida por uma função de utilidade, e realizada a partir de um plano e de uma representação simbólica do ambiente. Ideia abstraída de (BORDINI; VIEIRA, 2003).

Aplicando esta divisão para SMAs, tem-se que em SMAs reativos a inteligência parte da interação de um grande número de agentes reativos, ideia influenciada pela entomologia

(estudo dos insetos, em analogia com formigueiros, colméias, etc.). Já em SMAs cognitivos a quantidade de agentes é menor, por conta de que cada agente cognitivo ser um sistema sofisticado e computacionalmente complexo (HÜBNER; BORDINI; VIEIRA, 2004).

3.2.1 Modelo BDI

Hübner, Bordini, e Vieira (2004, p.8) apresentam "as mais importantes arquiteturas de agentes deliberativos são baseadas em um modelo de cognição fundamentado em três principais atitudes mentais que são as crenças, os desejos, e as intenções". Por isso a sigla BDI - *Beliefs* (crenças) *Desires* (desejos) *Intentions* (intenções), onde:

- Crenças: o que o agente sabe sobre o ambiente, e dos agentes presentes naquele ambiente;
- Desejos: os estados desejados;
- Intenções: sequências de ações que um agente deve executar para atingir os seus objetivos.

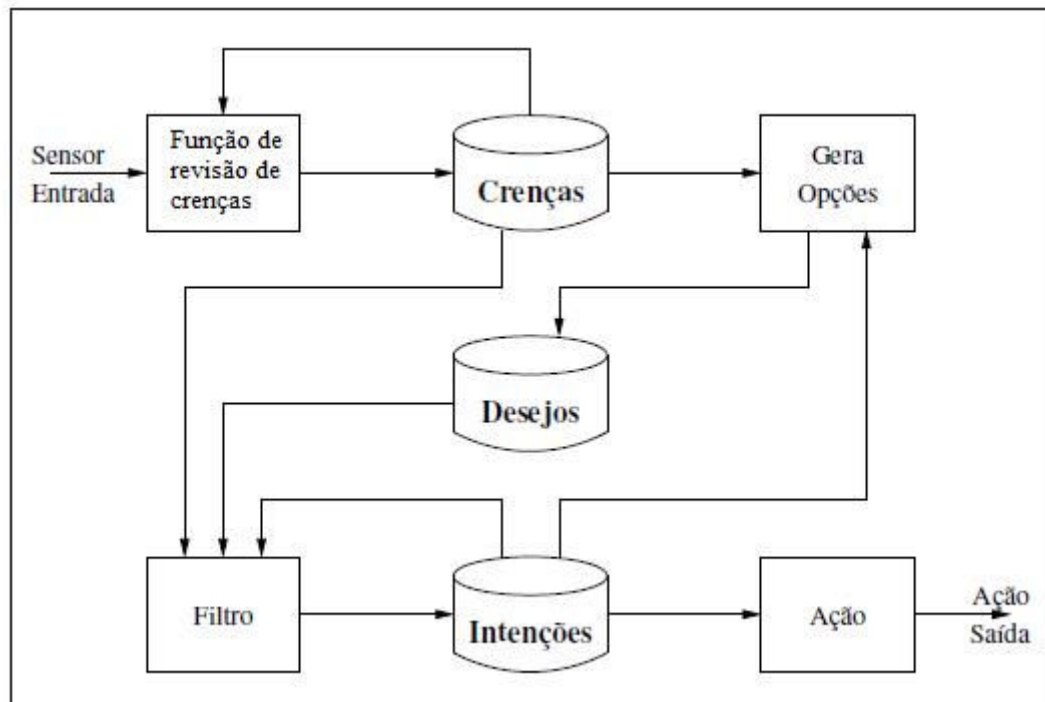


Figura 9 - Esquema genérico do modelo BDI
Adaptado de (WOOLDRIDGE, 1999)

Com base na Figura 9, a função de revisão de crenças recebe a informação de entrada dos sensores, que as atualiza após consultar as crenças anteriores do agente. Devido a essa

atualização, é possível gerar novas opções, ou seja, novos estados a serem atingidos com base nas intenções particulares do agente em questão. O Filtro atualiza as intenções do agente, a partir das crenças e desejos já atualizados e das suas intenções atuais. Finalmente, com as intenções atualizadas, é realizada a escolha da ação a ser executada pelo agente.

3.3 LINGUAGENS E AMBIENTES DE PROGRAMAÇÃO

PRS (*Procedural Reasoning System*) (GEORGEFF; LANSKY, 1986) é um *framework*, baseado no modelo BDI, para construção de sistemas de raciocínio em tempo real que são capazes de realizar tarefas complexas em ambientes dinâmicos. As linguagens e ferramentas presentes no Quadro 2 são baseadas no PRS (WOBCKE, 2007).

Linguagem	Observação
JADEX	Ambiente de programação (JADEX, 2011) em que o desenvolvimento de agentes é baseado em objetivos, programados em XML e Java, além de possuir uma integração com o <i>framework</i> JADE (POKAHR; BRAUBACH; LAMERSDORF, 2003).
JACK	<i>Framework</i> em Java para o desenvolvimento de sistemas multiagentes, foi a base para a criação da companhia AOS (<i>Agent Oriented Software</i>) (AOS, 2011), (BUSETTA et al, 1999).
GOAL	Linguagem de programação para a implementação de agentes racionais, possui uma IDE própria para desenvolvimento (GOAL, 2011).
AgentSpeak(L)	Linguagem de programação orientada a agentes, baseada em lógica (BORDINI; VIEIRA, 2003), (RAO, 1996).
Jason	Ambiente de programação baseado em uma extensão da linguagem AgentSpeak(L) (HÜBNER; BORDINI; VIEIRA, 2004), (JASON, 2011).

Quadro 2 - Algumas linguagens/ambientes de programação BDI

É possível perceber, com o Quadro 2, algumas das ferramentas e linguagens baseadas no modelo BDI que ainda são foco de estudos atuais. A grande diferença entre a linguagem AgentSpeak(L) em relação as outras, é que ela possui uma maior semelhança com a teoria original de BDI (HÜBNER; BORDINI; VIEIRA, 2004).

3.3.1 AgentSpeak(L)

Primeiramente apresentada em (RAO, 1996), a linguagem AgentSpeak(L) foi projetada para a programação de agentes utilizando o modelo BDI, na forma de sistemas de planejamento reativos, ou seja, sistemas que estão permanentemente em execução (BORDINI; VIEIRA, 2004).

Segundo Hübner, Bordini, e Vieira (2004, p.10) "Um agente AgentSpeak(L) corresponde à especificação de um conjunto de crenças que formarão a base de crenças inicial e um conjunto de planos".

Um plano, conforme (HÜBNER; BORDINI; VIEIRA, 2004), é composto por um evento ativador interno ou externo (que denota o propósito do plano), seguido de suas crenças que representam um contexto (a consequência do conjunto de crenças do agente no momento em que o evento é selecionado pelo agente).

O restante do plano é formado por ações básicas que o agente deve executar, ou subplanos (objetivos intermediários, necessários para o sucesso do plano). Para que um plano possa entrar em execução, é necessário a satisfação de uma pré-condição, além da ocorrência de um evento ativador (HÜBNER; BORDINI; VIEIRA, 2004).

```
+show(A,L) : gosta(A)
  <- !reserva_ingressos(A,L);

+!reserva_ingressos(A,L) : ¬ocupado(número)
  <- chamar(número);
...;
!escolher_poltrona(A,L);
```

Figura 10 - Exemplo de planos em AgentSpeak(L)
Traduzido de (HÜBNER; BORDINI; VIEIRA, 2004)

Na Figura 10 é apresentado um exemplo de planos em AgentSpeak(L), onde o primeiro plano (show) informa que ao ser anunciado um show (ativador externo) por um artista A, em um local L, deve ser acrescentada a crença show(A,L) como consequência da

percepção do ambiente. Caso o agente goste do artista A, então seu objetivo será a reserva de ingressos para esse show. O que resulta no subplano de reserva de ingressos (ativador interno), onde caso a linha telefônica não esteja ocupada, o agente ira executar a ação de entrar em contato telefônico, seguido de um protocolo de reserva de ingressos (representado por '...'), e finalmente, terminando por um subplano para a escolha das poltronas para aquele show.

A especificação de um agente em AgentSpeak(L) deve ser feita de acordo com a gramática representada na Figura 11.

```

agente ::= crenças planos
crenças ::= crença1 ... crençaK
planos ::= plano1 ... planoN
plano ::= evento : contexto → corpo
evento ::= + crença | - crença | +! objetivo
contexto ::= crença | not(crença) | contexto & contexto | true
corpo ::= objetivo | ?objetivo | +crença | -crença | ação | corpo; corpo
crença ::= atomo
objetivo ::= atomo
atomo ::= predicado(termo1 , ... termoM)

```

com $K, N \geq 1, M \geq 0$

Figura 11 - Sintaxe da linguagem AgentSpeak(L)
Adaptado de (HÜBNER; BORDINI; VIEIRA, 2004).

3.3.2 Jason

Jason é um interpretador para uma extensão da linguagem AgentSpeak(L), que inclui comunicação entre agentes baseada na teoria de atos de fala (diretivas de comunicação). A ferramenta Jason é implementada em Java (multiplataforma), e disponibilizada como código aberto sob a licença GNU LGPL (JASON, 2011).

Além de interpretar a linguagem AgentSpeak(L) original, Jason possui as seguintes características (JASON, 2011):

- Negação forte: tanto sistemas *closed-world* (mundo-fechado) quanto *open-world* (mundo-aberto) são possíveis;
- Tratamento de falha em planos;
- Comunicação baseada em atos de fala;
- Anotações em identificadores de planos, utilizadas na elaboração de funções personalizadas para seleção de planos;

- Suporte para o desenvolvimento de ambientes (o ambiente é programado em Java);
- Possibilidade de executar o SMA distribuídamente em uma rede;
- Possibilidade de especializar em Java as funções de seleção de planos, as funções de confiança, e toda a arquitetura do agente.
- Possui uma biblioteca básica de 'ações internas';
- Possibilita a extensão da biblioteca de ações internas;
- Possui uma IDE, utilizando o jEdit, ou em forma de um *plugin* para a IDE Eclipse. Tem um 'inspetor mental', que auxilia na depuração.

3.4 MODELAGEM DE AGENTES

Uma variedade de metodologias para sistemas orientados a objetos tem sido extensivamente estudadas e planejadas ao longo do tempo, porém não é possível utilizá-las eficientemente no desenvolvimento de sistemas orientados a agentes. Apesar de agentes e objetos compartilharem algumas similaridades, as diferenças entre eles são muito significantes (PADGHAM; WINIKOFF, 2004).

A Engenharia de *Software* Orientada a Agentes (AOSE) vem sendo descrita como um novo paradigma para o campo de pesquisa na Engenharia de *Software*. Para que isso se torne realidade é necessário o desenvolvimento de metodologias e ferramentas robustas e fáceis de serem utilizadas (TVEIT, 2001).

De acordo com Girardi (2004), "De uma forma similar ao que aconteceu com o desenvolvimento orientado a objetos, uma grande quantidade de técnicas e metodologias tem sido propostas para a Engenharia de *software* baseada em agentes e ainda não existe um padrão reconhecido que reúna as vantagens de cada uma delas".

Algumas das metodologias atuais da AOSE encontradas são: AUML (AUML, 2011); Tropos (TROPOS, 2011); Moise+ (MOISE, 2011); MaSE (MASE, 2011); Prometheus (PADGHAM; WINIKOFF, 2004), (PADGHAM; THANGARAJAH; WINIKOFF, 2007), (PDT, 2011), (PROMETHEUS, 2011).

3.4.1 Metodologia Prometheus

Prometheus é uma metodologia da AOSE desenvolvida com a intenção de ser utilizada tanto no meio acadêmico quanto em aplicações comerciais. Apresenta uma abordagem mais didática em seus modelos e estruturas (PADGHAM; WINIKOFF, 2004). Em 2007, seus autores desenvolveram uma ferramenta, PDT (Prometheus *Design Tool*), para a modelagem de sistemas orientados a agentes (PADGHAM; THANGARAJAH; WINIKOFF, 2007).

Consiste de três etapas, descritas na Figura 12, e resumidas na sequência (PADGHAM; WINIKOFF, 2004):

- Especificação do Sistema: concentra-se em identificar os objetivos e as funcionalidades básicas do sistema, além das entradas (percepções) e saídas (ações);
- Projeto Arquitetural: utiliza as saídas da etapa anterior para determinar quais os tipos de agentes que o sistema irá conter, e como eles irão interagir entre si;
- Projeto Detalhado: tem como objetivo modelar e detalhar a arquitetura interna dos agentes, e como eles deverão cumprir as suas tarefas.

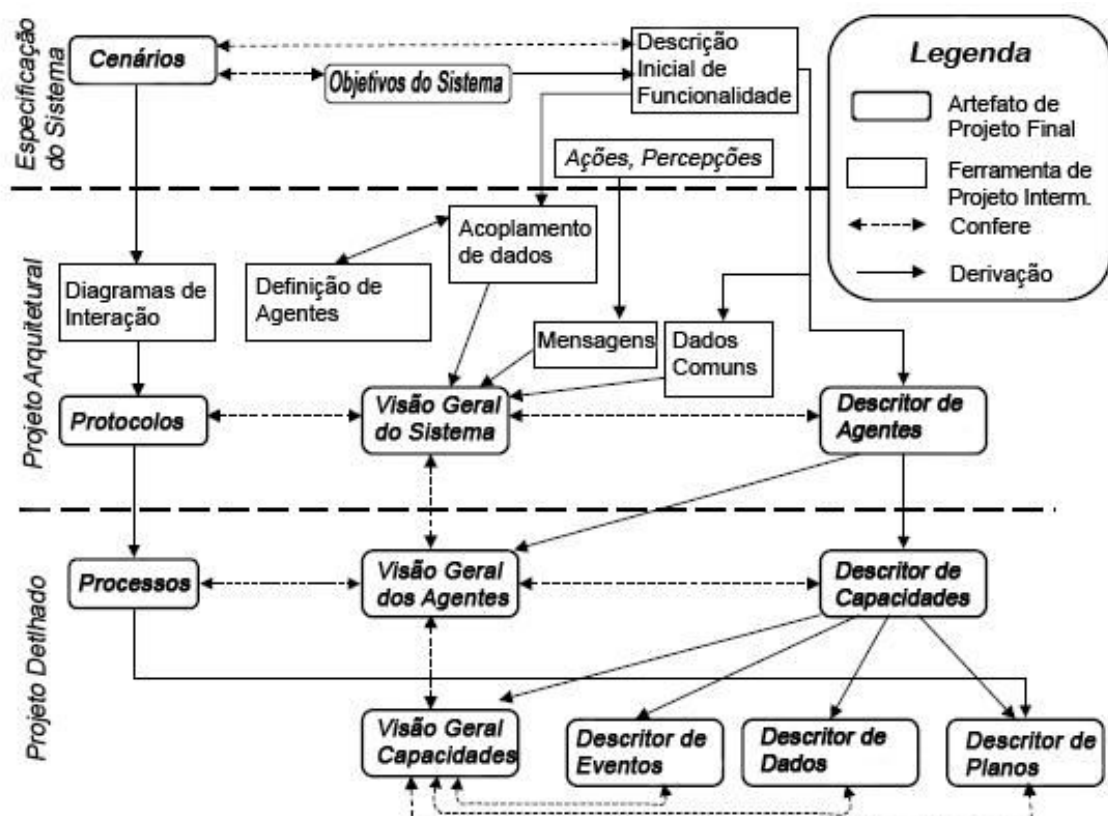


Figura 12 - Esquema com as três fases de desenvolvimento da metodologia Prometheus
Traduzido de (PROMETHEUS, 2011)

Os componentes da metodologia Prometheus utilizados neste trabalho podem ser observados na Figura 13. Alguns dos principais aspectos da metodologia Prometheus incluem (PADGHAM; THANGARAJAH; WINIKOFF, 2007):

- Projeto detalhado das ações internas de agentes, assim como as interações entre os agentes como um sistema. Suporta (porém não é limitada a) agentes do modelo BDI;
- Tenta manter um equilíbrio entre estruturas definidas, e diagramas que permitem aos projetistas alguma liberdade;
- Abrange (de certa forma) todas as fases de desenvolvimento: especificação, projeto, implementação, teste e depuração;
- É projetada para escalar à grandes projetos.

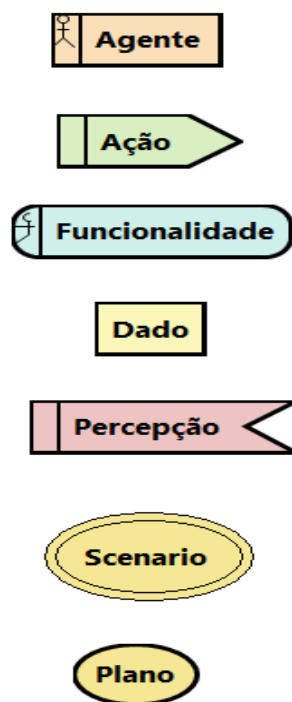


Figura 13 - Legenda dos componentes utilizados na modelagem deste trabalho

Com as informações deste capítulo, em conjunto com as informações do capítulo anterior, foi trabalhado o referencial teórico, e uma visão geral das ferramentas a serem utilizadas na proposta de trabalho.

4 TRABALHOS CORRELATOS

Neste capítulo são apresentados, sumariamente, alguns trabalhos científicos que envolvem a tecnologia LEGO Mindstorms NXT e a área de Inteligência Artificial.

Jensen (2010) propõe uma maneira de combinar o *framework Jason* com robôs LEGO Mindstorms, possibilitando a programação de agentes inteligentes para robôs físicos, atingindo seu objetivo ao estender partes do ciclo de raciocínio do Jason. Apesar de ter obtido sucesso a falta de documentação dificulta a reprodução do trabalho, e nota-se também a ausência da modelagem nos casos de estudo propostos.

Em (BENEDETTELLI et al, 2009) é apresentado um modelo experimental, baseado na tecnologia LEGO Mindstorms, composto de quatro veículos controlados por um sistema centralizado de supervisão. O propósito é construir um ambiente que permita a implementação e a comparação de algoritmos para uma variedade de problemas em sistemas multiagentes, como controle de formação, coordenação de movimento e problemas de cobertura. Contudo, esse trabalho foi especificado e programado na ferramenta MATLAB, inviabilizando maiores estudos e aproveitamentos para o trabalho proposto.

Kroustis e Casey (2008) implementam um *framework*, no LEGO Mindstorms NXT, que combina um algoritmo adaptativo de alto nível com heurísticas de baixo nível. O robô é projetado para encontrar uma fonte de luz em um ambiente desconhecido, e através dessa implementação é demonstrado como a plataforma NXT pode ser utilizada para o desenvolvimento de algoritmos complexos, utilizando comandos remotos via *bluetooth*. No entanto, é um trabalho que não aborda questões de sistemas multiagentes, nem mesmo a teoria de agentes, pois utiliza algoritmos adaptativos, outra área da Inteligência Artificial.

Já em (LEMKE; LAIDLAW; PEDERSEN, 2007), foram construídos quatro robôs NXT equipados com sensores de luz e sensores de contato. Em seguida, implementaram um sistema multiagente híbrido capaz de resolver o seguinte problema: limpar uma sala com diversos brinquedos, que pertencem a um menino e a uma menina, ambos possuem um número de robôs próprios para ajudá-los a limpar a sala, de modo que os robôs devem coletar os brinquedos de seus respectivos donos e os retornar ao seu devido lugar, caso um robô ache um brinquedo que não pertence ao seu dono, ele deve avisar os outros robôs. E, finalmente, implementaram uma versão do sistema mais avançado com agentes BDI.

Finalmente, no Centro Universitário Franciscano - UNIFRA, já foram realizados dois trabalhos envolvendo a tecnologia LEGO Mindstorms NXT, (PASQUALIN, 2009) e (PEREIRA, 2009). O primeiro propôs a detecção e o tratamento de falhas em sensores do

robô através da implementação de um sensor virtual, e o tratamento de falhas motoras através da sincronização de seus motores. O segundo desenvolveu um *software* para realizar o controle remoto de robôs LEGO Mindstorms NXT por meio da biblioteca NXT-Python²⁰, baseada na linguagem de programação Python. Entretanto, nenhum dos dois especificou ou implementou seus sistemas utilizando a teoria de agentes BDI, foco deste trabalho de conclusão de curso.

Neste capítulo foram apresentados alguns trabalhos relacionados, destacando-se o trabalho de Jensen (2010), que foi o único trabalho encontrado onde o *framework* Jason foi utilizado para a programação de agentes inteligentes em robôs LEGO Mindstorms NXT 1.0.

²⁰ Disponível em: <http://code.google.com/p/nxt-python/>.

5 PROPOSTA DE TRABALHO

Baseando-se no contexto teórico apresentado nos capítulos anteriores, é proposto o planejamento de ações e percepções de agentes inteligentes em robôs reais (físicos). O que é realizado através da interação entre a linguagem de programação leJOS NXJ para robôs LEGO Mindstorms NXT, e o ambiente de programação para sistemas multiagentes Jason.

O planejamento de ações e percepções é representado por duas partes neste trabalho: o código em Java rodando no robô LEGO Mindstorms NXT, através do *firmware* leJOS, que é responsável pelo envio das percepções coletadas pelos sensores do agente sobre o ambiente; e o código em AgentSpeak(L) rodando em um computador, através do *framework* Jason, que é responsável pela escolha de um plano a partir das percepções recebidas.

A tecnologia LEGO Mindstorms NXT foi escolhida pelo seu caráter educacional, por ser de fácil manuseio e pela sua disponibilidade na instituição. A linguagem de programação leJOS foi escolhida por ser baseada em Java, o que é necessário para a comunicação por *bluetooth* com o ambiente em Jason, que também é em Java.

O *framework* Jason foi escolhido por proporcionar uma maneira intuitiva de implementar sistemas multiagentes avançados. O dispositivo utilizado para rodar os códigos em Jason foi um computador que, através da tecnologia *bluetooth*, permite a comunicação sem fio entre o robô e o computador. A IDE em código aberto Eclipse²¹ foi escolhida pelo fato de possuir *plugins* para o desenvolvimento em leJOS, Jason e PDT.

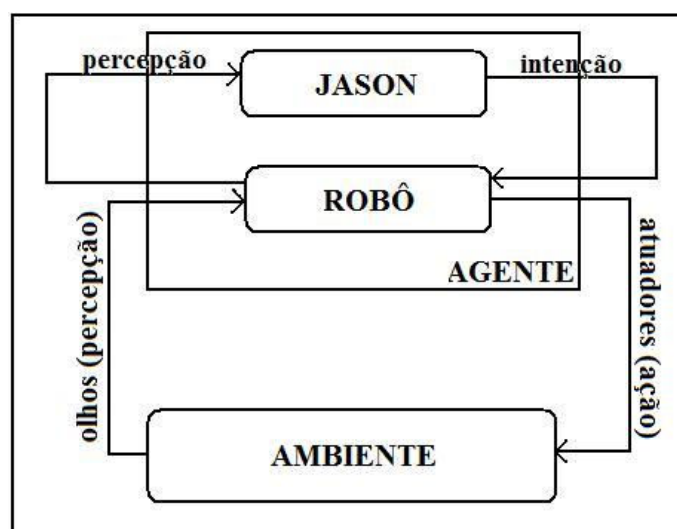


Figura 14 - Esquema do Agente Robótico

²¹ Disponível em: <http://www.eclipse.org/>.

A interação entre o robô e o computador resulta no agente, esquematizado na (Figura 14), que é composto por 'cérebro', 'atuadores' e 'olhos'. O código em Jason representa o 'cérebro' do agente, via crenças e planos, executando em um computador. Esse código faz a comunicação com os 'atuadores' (motores) e 'olhos' (sensores) do robô LEGO Mindstorms NXT, por meio do seu *brick* programável que está executando o código em leJOS.

Os sensores do robô captam as mudanças no ambiente e enviam essas percepções para o 'cérebro' do agente no computador, onde está todo o planejamento de seu comportamento, que por sua vez envia a intenção resultante do plano escolhido ao robô, finalmente realizando a sua execução.

Para testar a interação entre leJOS e Jason, foi projetado uma pista de corrida com obstáculos, ilustrado na Figura 15. O agente deverá perceber e desviar dos obstáculos no caminho, ao mesmo tempo que se mantém dentro do circuito delimitado por diferentes cores de linha, onde:

- a linha em preto informa que o agente está no limite da pista, e caso a encontre deverá virar para a direita;
- a linha em cinza escuro informa que o agente está no limite da pista, e caso a encontre deverá virar para a esquerda;
- a linha em cinza claro determina a chegada, é o critério de parada e o objetivo do agente.

Os retângulos representam os obstáculos que podem ser encontrados no caminho, e não possuem uma posição necessariamente pré-determinada. Eles podem ser movidos para uma posição diferente da que se encontram na Figura 15, desde que o caminho entre a partida e a chegada não se torne impossível (obstáculos muito próximos que impossibilitam a passagem do robô).

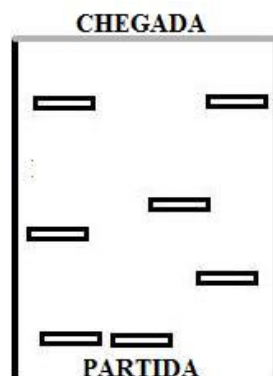


Figura 15 - Modelo da pista a ser percorrida pelo agente

Projeta-se que a estrutura do robô para este circuito é composta por dois motores, cada um responsável por uma roda, um sensor de luz (A) para detectar as linhas, e um sensor ultrassônico (B) para a detecção dos obstáculos, como representado na Figura 16. Nota-se que foi removido o cabo que conecta o sensor de luz a porta 1 para melhor visualização.



Figura 16 - Estrutura do robô para a tarefa proposta

Dentre as metodologias citadas no subcapítulo 3.4, a metodologia Prometheus aparenta ser a mais desenvolvida, com breves comparações entre Prometheus e as demais, presentes na obra (PADGHAM; WINIKOFF, 2004). Por este motivo, foi a metodologia escolhida para modelar o sistema proposto neste trabalho. No subcapítulo seguinte, é apresentada a modelagem do sistema composto pelo agente e o circuito.

5.1 MODELAGEM

Para auxiliar na modelagem do sistema com a metodologia Prometheus, foi utilizado o *plugin* PDT (PDT, 2011) para a IDE Eclipse. Na instalação foi utilizado o guia (QUICKSTART, 2011), no entendimento da ferramenta (PDTMANUAL, 2011), e como modelo os exemplos encontrados em (TUTORIALPDT, 2011) e (PADGHAM; WINIKOFF, 2004). A legenda dos componentes se encontra no subcapítulo 3.4.1, Figura 13.

5.1.1 Especificação do Sistema

Planos (*Goals*) do sistema: São identificados com base na funcionalidade desejada pelo sistema. Os requisitos são transformados em planos e sub-planos no sistema a seguir, e ilustrados na Figura 17.

- Cruzar a linha de chegada
 - Desviar de um obstáculo
 - Permanecer dentro do circuito

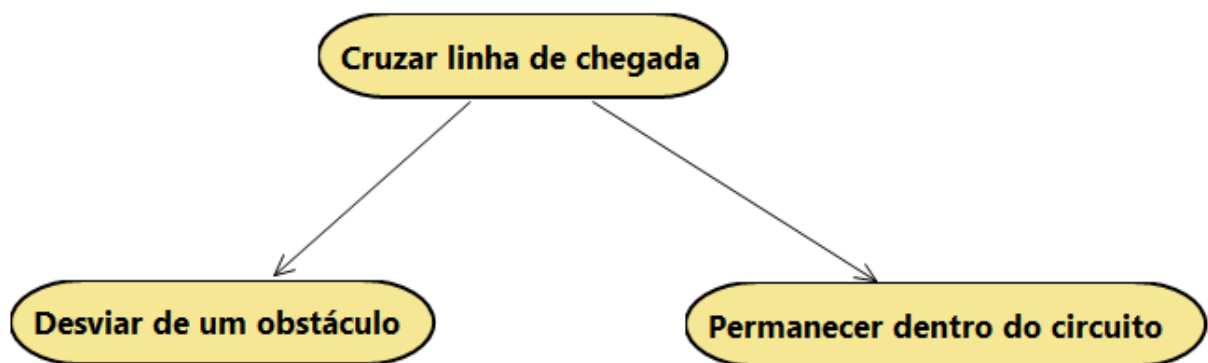


Figura 17 - Diagrama da visão geral de planos

Cenários (Figura 18):

1. Quando o agente precisa desviar de um obstáculo.
2. Quando o agente precisa permanecer dentro do circuito.
3. Quando o agente cruza a linha de chegada.



Figura 18 - Diagrama de cenários

Passos referentes aos cenários:

Desvio de obstáculo:

1. Detectar um obstáculo a frente (percepção)

2. Parar motores (plano)
3. Virar para a esquerda/direita (plano)
4. Mover para frente (ação)
5. Virar para a esquerda/direita (plano)
6. Mover para frente (ação)

Permanecer dentro do circuito:

1. Detectar a linha de cor cinza escuro (percepção) ou Detectar a linha de cor preta (percepção)
2. Parar motores (plano)
3. Virar para a esquerda/direita (plano)
4. Mover para frente (ação)

Cruzar a linha de chegada:

1. Detectar a linha de cor cinza claro (percepção)
2. Parar motores e encerrar o sistema (ação)

Funcionalidades (Figura 19):

- Desviar obstáculo
- Permanecer no circuito
- Cruzar linha chegada

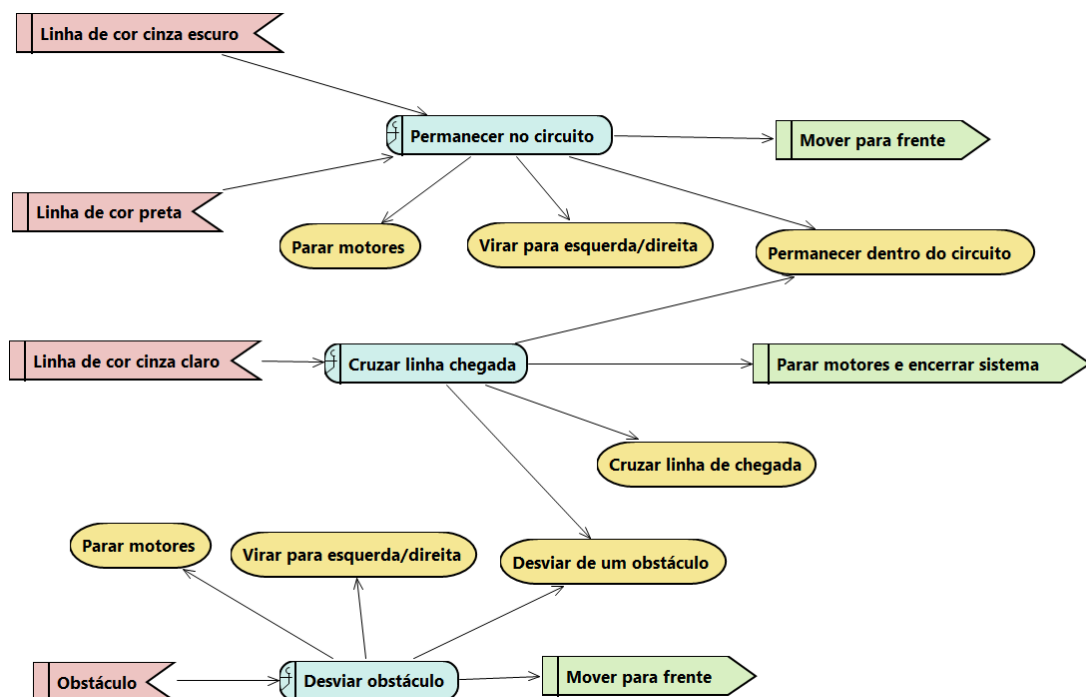


Figura 19 - Diagrama das funcionalidades do sistema

5.1.2 Projeto Arquitetural

Agora é necessário determinar quais os dados que devem ser armazenados no sistema como crenças, através na análise das funcionalidades, e resultando no diagrama da Figura 20.

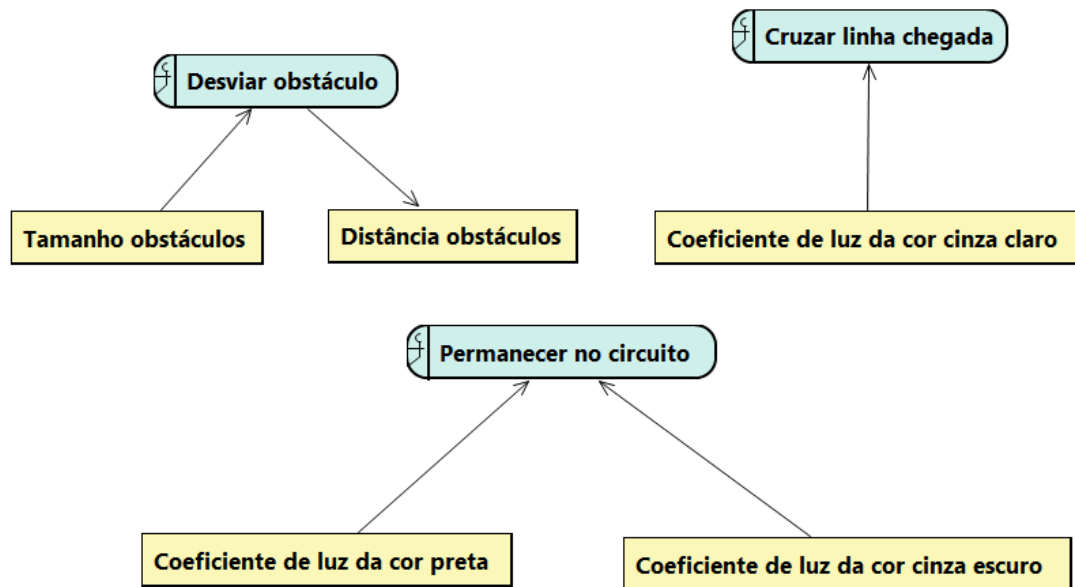


Figura 20 - Diagrama de acoplamento de dados

O próximo passo é identificar os agentes encarregados das funcionalidades do sistema, Figura 21.

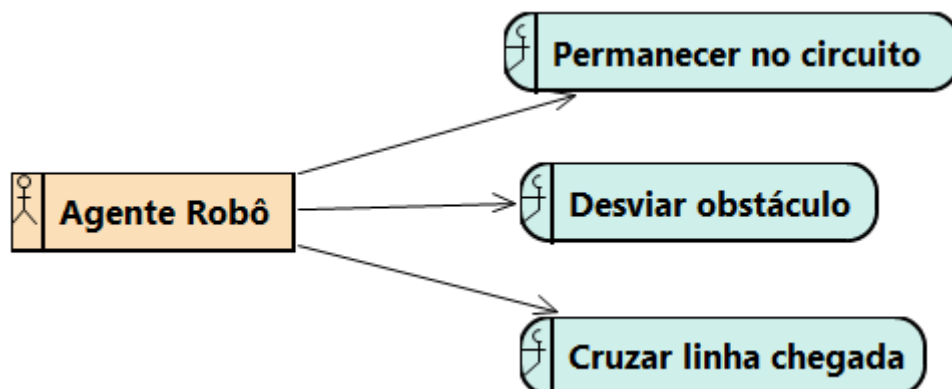


Figura 21 - Diagrama de acoplamento de funcionalidades dos agentes

Por fim, o último passo consiste em gerar o diagrama de visão geral do sistema, Figura 22.

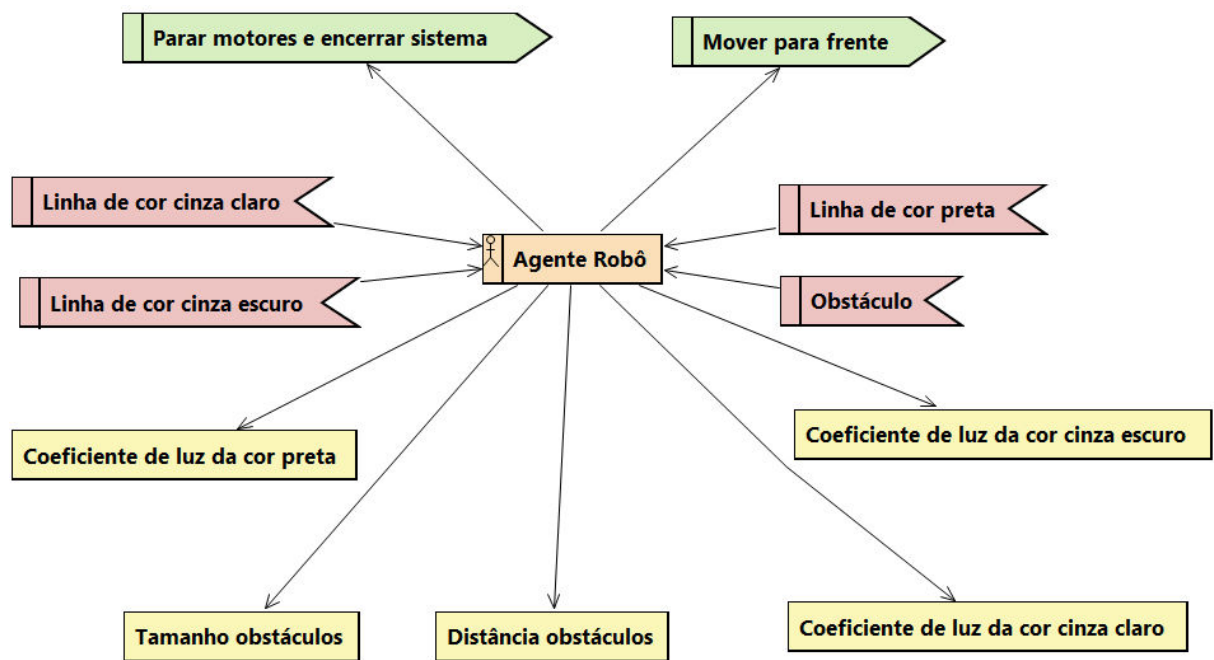


Figura 22 - Diagrama de visão geral do sistema

5.1.3 Projeto Detalhado

Nessa etapa final são definidos os detalhes das capacidades de cada agente, que no caso do sistema proposto por este trabalho, o sistema possui apenas um agente, apresentado em detalhes na sequência pelas figuras Figura 23, Figura 24, Figura 25, e Figura 26.

Agente Robô:

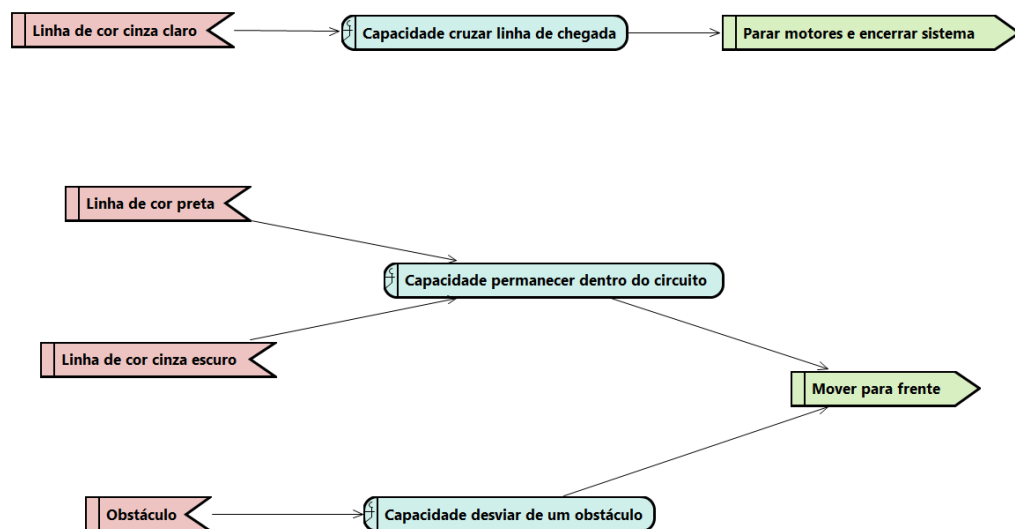


Figura 23 - Diagrama de visão geral do Agente Robô

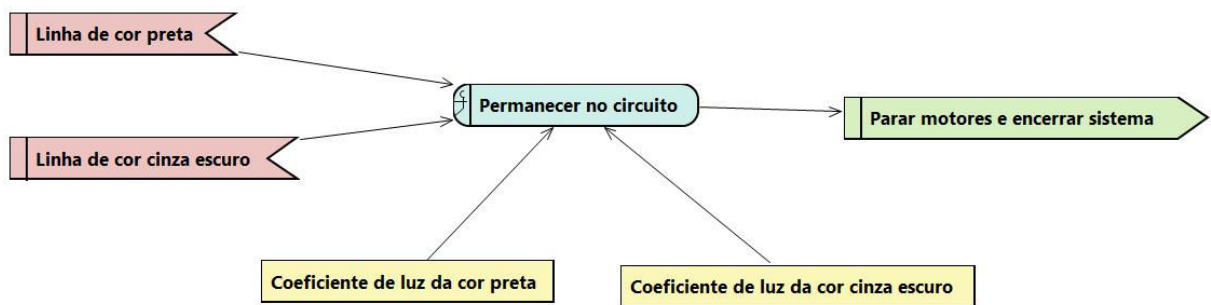


Figura 24 - Diagrama da Capacidade permanecer dentro no circuito



Figura 25 - Diagrama da Capacidade cruzar linha de chegada

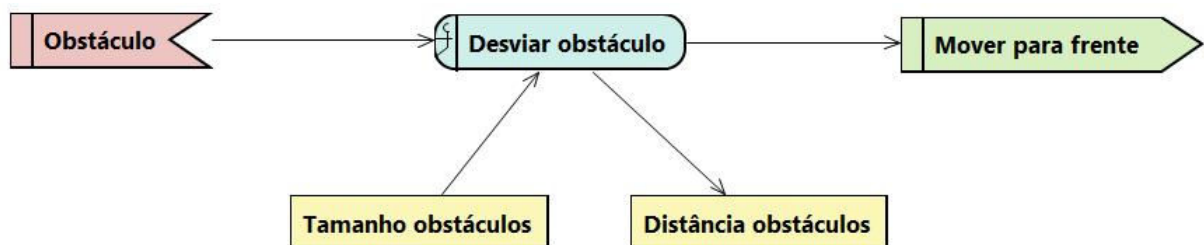


Figura 26 - Diagrama da Capacidade desviar de um obstáculo

Pelo fato do sistema não ser multiagente, não foi possível demonstrar todas as funcionalidades e os diagramas presentes na metodologia Prometheus. Entretanto, a modelagem apresentada deve facilitar a compreensão da proposta de trabalho, além de servir como um exemplo básico da capacidade dessa metodologia.

5.2 IMPLEMENTAÇÃO

O trabalho é baseado na implementação realizada por Jensen (2010), que através da extensão do ciclo de raciocínio do *framework* Jason, permite a programação de agentes inteligentes em robôs LEGO Mindstorms NXT. A implementação é estruturada em duas

partes: uma pertinente aos códigos à serem executados no computador (diretório 'jason'); e a outra aos códigos à serem executados no robô (diretório 'nxt'). A estrutura pode ser observada na Figura 27 e uma breve descrição de cada arquivo se encontra no Anexo B.

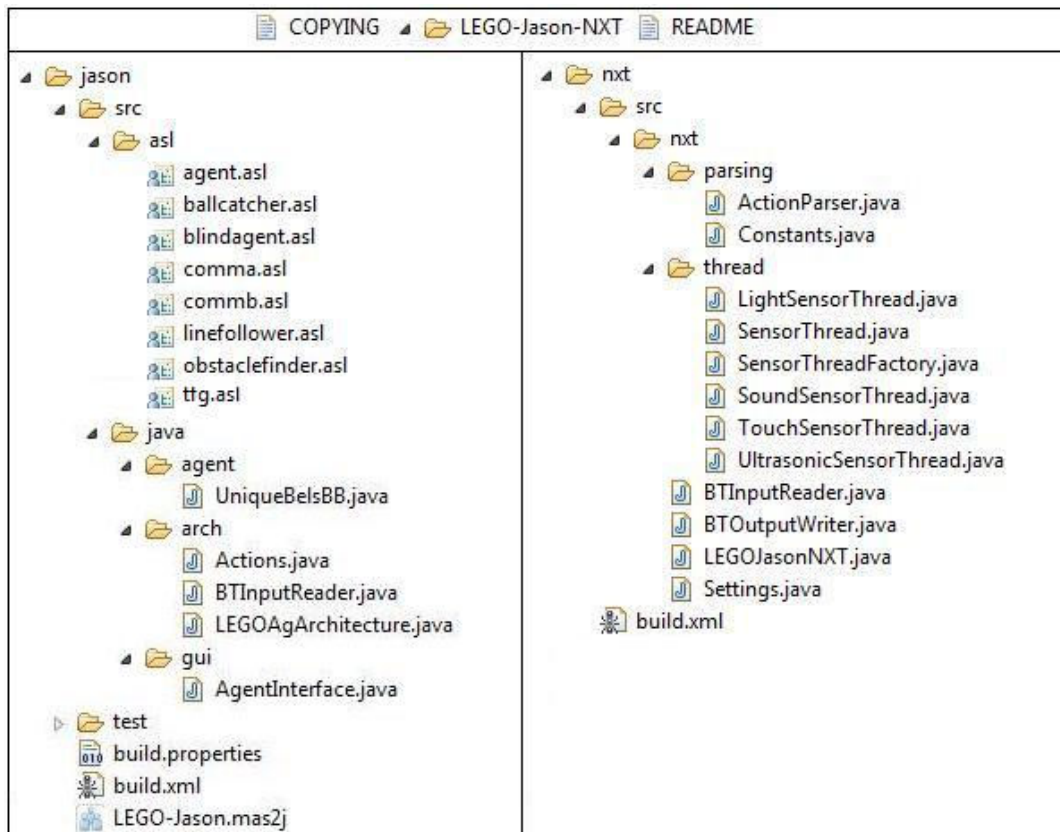


Figura 27 - Estrutura de diretórios

Os códigos estão disponíveis no link: <http://www.alexandrez.org/projetos/download/tfg2RafaelCardoso.zip>. Um detalhamento das ações e percepções que podem ser utilizadas pela extensão da sintaxe do *framework* Jason pode ser encontrado no Anexo A. Para um tutorial na instalação dos componentes necessários na execução dos códigos sugere-se a leitura do Apêndice A. Apesar do projeto ser portátil, os testes foram realizados apenas no sistema operacional Windows e, portanto, o tutorial é disponibilizado apenas para esta plataforma. Para a plataforma Linux faça o *download* dos respectivos componentes.

Dos arquivos citados é relevante destacar para este trabalho o 'LEGO-Jason.mas2j' e o 'tfg.asl'. Na Figura 28 percebe-se que a declaração do agente 'tfg.asl' é realizada no arquivo 'LEGO-Jason.mas2j'. Todo o agente que será executado deve ser declarado neste arquivo, e essa declaração segue o padrão especificado na Figura 29 e no Quadro 3. Os parâmetros

'agentArchClass' e 'beliefBaseClass' devem seguir o padrão, o primeiro trata da comunicação entre a parte do agente em Jason com a parte do agente em leJOS e o segundo garante que em um dado momento se tenha no máximo uma percepção por sensor, de forma a evitar que a base de crenças cresça exponencialmente [JENSEN, 2010].

```

1 MAS legojason {
2   infrastructure: Centralised
3
4   agents:
5
6     tfg tfg.asl
7     [btname="NXT", btaddress="0016530358ED", motora="true", motorb="true",
8      motorc="false", sensor1="light", sensor2="none", sensor3="ultrasonic", sensor4="none"]
9     agentArchClass arch.LEGOAgArchitecture
10    beliefBaseClass agent.UniqueBelsBB("light(port,_)", "sound(port,_)",
11     "obstacle(port,_)", "touching(port,_)");
12
13    aslSourcePath: "src/asl";
14 }

```

Figura 28 - Exemplo de um projeto em Jason-NXT

```

nomeAgente nomeArquivoAgente.asl
[btname="...", btaddress="...",
 motora="...", motorb="...", motorc="...",
 sensor1="...", sensor2="...", sensor3="...", sensor4="..."]
agentArchClass arch.LEGOAgArchitecture
beliefBaseClass agent.UniqueBelsBB("light(port,_)", "sound(port,_)",
 "obstacle(port,_)", "touching(port,_)");

```

Figura 29 - Padrão de declaração de agentes

Argumento	Valor
btname	Nome do <i>brick</i> NXT programável
btaddress	Endereço <i>Bluetooth</i> do NXT, composto por 12 caracteres
motor{a,b,c}	<i>true</i> (verdadeiro) ou <i>false</i> (falso) se o motor está conectado ao <i>brick</i> ou não
sensor{1,2,3,4}	O nome do sensor conectado ao <i>brick</i> . O nome pode ser { <i>touch</i> , <i>light</i> , <i>sound</i> , <i>ultrasonic</i> , <i>none</i> }, significando respectivamente contato, luz, som, ultrassônico, e <i>none</i> no caso de nenhum sensor estar conectado naquela porta

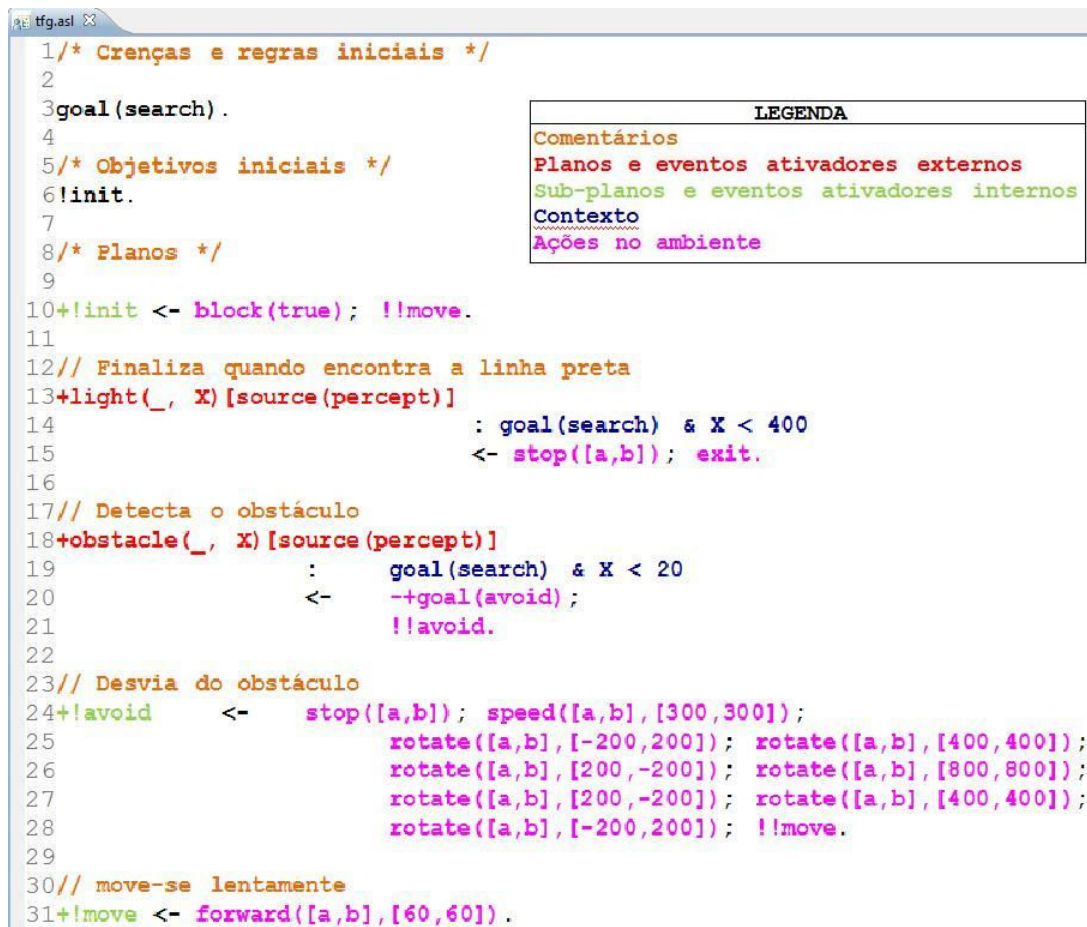
Quadro 3 - Valores que podem ser assumidos pelos argumentos

Logo, da declaração do agente 'tfg.asl' podemos extrair os valores dos argumentos para a construção do Quadro 4, onde as portas dos motores (a, b, c) e as portas dos sensores (1, 2, 3, 4) são as mesmas representadas na Figura 4. Como projetado no capítulo 5, o robô possui dois motores (conectados nas portas 'a' e 'b', um para cada roda), um sensor de luz (conectado na porta '1') para identificar as linhas, e um sensor ultrassônico (conectado na porta '3') para identificar os obstáculos.

Argumento	Valor
btname	NXT
btaddress	0016530358ED
motora	<i>true</i>
motorb	<i>true</i>
motorc	<i>false</i>
sensor1	<i>light</i>
sensor2	<i>none</i>
sensor3	<i>ultrasonic</i>
sensor4	<i>none</i>

Quadro 4 - Valores assumidos nos argumentos do agente 'tfg.asl'

A Figura 30 mostra a codificação da tarefa modelada no subcapítulo 5.1, a pista de corrida com obstáculos. Observa-se que o valor necessário para identificar a linha de chegada foi configurado como sendo menor que 400 (pode variar com o material da linha, a luminosidade do ambiente, e outros fatores), e os obstáculos devem ser da mesma largura para que as manobras funcionem corretamente. O agente assume como regra inicial a busca pela linha de chegada, e ao iniciar move-se lentamente até encontrar a linha ou um obstáculo a menos de 20 centímetros. Caso encontre a linha finaliza-se o agente, e caso encontre um obstáculo é realizado o desvio.



```

1/* Crenças e regras iniciais */
2
3goal(search).
4
5/* Objetivos iniciais */
6!init.
7
8/* Planos */
9
10+!init <- block(true); !!move.
11
12// Finaliza quando encontra a linha preta
13+light(_, X)[source(percept)]
14      : goal(search) & X < 400
15      <- stop([a,b]); exit.
16
17// Detecta o obstáculo
18+obstacle(_, X)[source(percept)]
19      : goal(search) & X < 20
20      <- +goal(avoid);
21          !!avoid.
22
23// Desvia do obstáculo
24+!avoid <- stop([a,b]); speed([a,b],[300,300]);
25          rotate([a,b],[-200,200]); rotate([a,b],[400,400]);
26          rotate([a,b],[200,-200]); rotate([a,b],[800,800]);
27          rotate([a,b],[200,-200]); rotate([a,b],[400,400]);
28          rotate([a,b],[-200,200]); !!move.
29
30// move-se lentamente
31+!move <- forward([a,b],[60,60]).

```

LEGENDA
Comentários
Planos e eventos ativadores externos
Sub-planos e eventos ativadores internos
Contexto
Ações no ambiente

Figura 30 - Códificação do agente 'tfg.asl'

Para auxiliar na observação do comportamento do agente, Jensen (2010) desenvolveu uma interface gráfica (Figura 31), onde podem ser visualizados os argumentos iniciais na configuração do robô (Quadro 4) presentes no arquivo de configuração de projeto 'LEGO-Jason.mas2j'. Nota-se o uso de '*Sensor Sleep*', que indica o tempo de espera até o sensor captar a próxima percepção e enviar ao computador, o valor padrão é 50 milissegundos e pode ser passado como argumento (após o '*sensor4=...*', '*sleep=...*'), ou alterado diretamente no código fonte. Além dos argumentos, são apresentadas as percepções captadas e enviadas pelos sensores do robô, e as ações enviadas pelo computador ao robô para serem executadas.

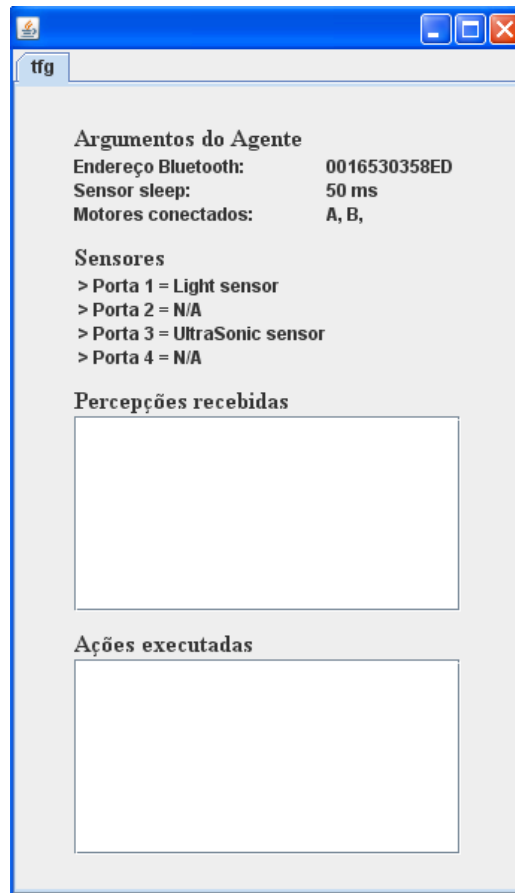


Figura 31 – Interface gráfica do usuário (traduzida)

Durante a etapa de testes foram encontrados diversos problemas na execução dos códigos de Jensen (2010), sendo o que demonstrou mais dificuldade em sua solução foi o representado na Figura 32. A depuração de erros da linguagem leJOS não é tão precisa como na linguagem Java e portanto este erro poderia significar diversos problemas. Foi focado o problema que parecia ser o mais provável segundo pesquisas realizadas no fórum²² da leJOS, pois estava sendo incluída a biblioteca 'classes.jar' nos códigos executados no computador e portanto, causando o erro já que ela deve ser usada apenas para os códigos executados no robô. A correção foi realizada no arquivo 'build.xml' presente no diretório 'jason', indicado pela Figura 33.

Este erro pode ter sido originado pelas diferenças nas versões da linguagem de programação leJOS NXJ, já que Jensen (2010) não especificou qual foi utilizada em seu trabalho. As versões de todos os componentes utilizados neste trabalho de conclusão de curso podem ser analisadas no Apêndice A.

²² <http://lejos.sourceforge.net/forum/viewtopic.php?f=7&t=2216>

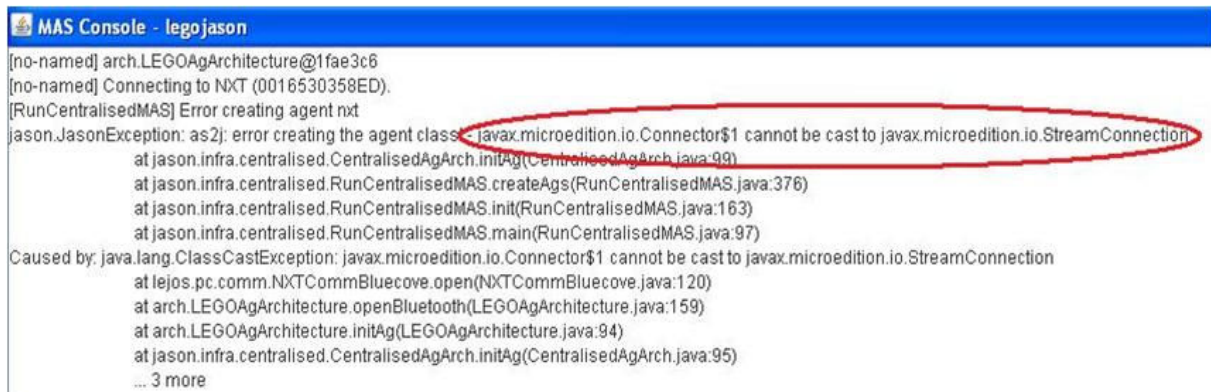


Figura 32 - Erro encontrado durante a implementação

```
<fileset dir="${nxj}/lib" > <include name="*.jar" /> </fileset>
↓
<fileset dir="${nxj}/lib" > <include name="jtools.jar" />
                             <include name="pccomm.jar" />
                             <include name="pctools.jar" /> </fileset>
```

Figura 33 - Solução para o erro da Figura 32

Além disso, foi encontrada uma limitação do sensor de luz do robô LEGO Mindstorms NXT 1.0 onde não é possível detectar três cores distintas com o sensor, pois ele aceita apenas valores maiores ou menores que um coeficiente de luz informado. Por exemplo, ele seria capaz de detectar duas das linhas de cores diferentes da tarefa, mas não detectaria uma terceira linha de cor diferente. Isso pode ser resolvido com a versão 2.0 do Kit LEGO Mindstorms que introduz o sensor de cor, porém não foi possível obter nem a versão 2.0 do Kit, nem um sensor de cor a tempo da conclusão deste trabalho. Logo, a estrutura da pista de corrida foi modificada, removendo-se os limites laterais da pista e mantendo-se apenas a linha de chegada (linha preta) como pode ser visto na Figura 34.

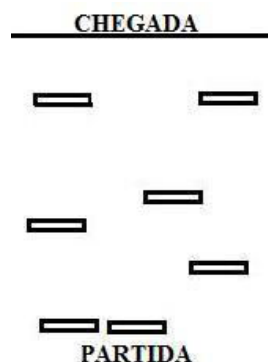


Figura 34 – Pista de corrida alterada

Ao final deste capítulo apresentou-se uma tarefa simples, com a sua modelagem e a sua implementação, para um agente inteligente programado com a extensão do *framework* Jason, criada por Jensen (2010), para robôs LEGO Mindstorms NXT rodando o *firmware* leJOS para linguagem de programação leJOS NXJ. No capítulo seguinte é apresentada uma retomada geral do trabalho, relacionando os objetivos assumidos no início com os resultados obtidos, e algumas ideias para trabalhos futuros.

6 CONCLUSÃO

Ao final deste trabalho de conclusão de curso, tendo como base o ambiente de programação para sistemas multiagentes Jason em conjunto com o robô LEGO Mindstorms NXT, foi realizada a modelagem e a implementação de um agente Jason em um sistema, com o objetivo de demonstrar a utilização de agentes inteligentes em robôs reais.

Retoma-se então, os objetivos específicos assumidos no início do trabalho: i) modelar o comportamento do agente segundo a metodologia Prometheus; ii) pesquisar e testar a programação de agentes utilizando Jason; iii) pesquisar e testar a linguagem de programação leJOS para o robô LEGO Mindstorms NXT; iv) pesquisar e testar a comunicação entre Jason e leJOS.

O objetivo i) foi atingido no subcapítulo 5.1 deste trabalho, porém não foi possível usufruir de todas as funcionalidades e diagramas da metodologia Prometheus pelo fato do sistema projetado não ser um sistema multiagente (a instituição possui apenas um robô disponível). Contudo, a modelagem apresentada deve facilitar a compreensão da proposta de trabalho, além de servir como um exemplo básico da capacidade dessa metodologia.

O objetivo ii) foi alcançado durante a pesquisa, programação, e testes no diretório 'jason', onde foi necessário conhecimentos em AgentSpeak(L) para a programação de agentes, além da sintaxe utilizada pelo Jason para a construção do arquivo de projeto. Não obstante o pequeno conjunto de ações e percepções disponíveis, ver Anexo A, a programação de agentes Jason para robôs LEGO Mindstorms NXT demonstrou ser funcional, sendo possível realizar a maioria das tarefas desejadas.

Já o objetivo iii) encontra-se no diretório 'nxt', onde foi realizada a programação de robôs LEGO Mindstorms NXT com a linguagem de programação leJOS. Apesar da programação do agente (ações, percepções e crenças) se encontrar inteiramente no diretório 'jason', a programação em leJOS atua como o ambiente em Jason para o agente, transmitindo as percepções encontradas pelos sensores do robô para o computador.

Finalmente o objetivo iv) surge da interação entre os códigos localizados em 'jason' e em 'nxt' através da comunicação por *Bluetooth*. Essa comunicação se torna possível por causa do trabalho de Jensen (2010), em que a extensão do ciclo de raciocínio do Jason permite a um robô LEGO Mindstorms NXT com a linguagem de programação leJOS NXJ enviar percepções para um computador com Jason, e receber as ações resultantes a serem executadas com base nos planos e objetivos do agente.

Em um primeiro momento, surgiu a ideia de incluir o *framework* Jason diretamente no robô, porém, conforme Jensen (2010), a máquina virtual Java implementada pelo *firmware* leJOS não possui todas as classes Java necessárias para a compilação dos códigos em Jason. Tornando-se então necessário rodar os códigos Jason em um computador, capaz de interagir com o robô. Isto resultou em um tempo de resposta do agente consideravelmente alto, tornando os casos em que há uma alta atividade por parte dos sensores, ou a necessidade de um tempo de resposta rápido, inviáveis.

O atraso no tempo de resposta se dá pelo fato de que as percepções captadas pelos sensores do robô devem ser tratadas e enviadas para o computador, que por sua vez modificam a base de crenças do agente, e resultam em uma intenção que é tratada e enviada ao robô, que finalmente a transforma em uma ação e realiza a sua execução. O processo em si se torna lento pois é necessário utilizar a comunicação *Bluetooth* entre dois dispositivos com linguagens diferentes.

Com o tempo espera-se que as novas versões do robô LEGO Mindstorms se tornem cada vez mais aptas a essa tarefa, caso ocorra um aprimoramento no *hardware*, aumentando a capacidade do *brick* programável. Eventualmente será possível a implementação da máquina virtual Java completa, o que permitiria a implementação do *framework* Jason diretamente no robô, ocasionando em um aumento considerável no tempo de resposta e possibilitando a programação de tarefas mais complexas.

A ideia original do trabalho era implementar um sistema multiagente, o que não foi possível devido ao fato da instituição possuir apenas um robô disponível. Porém, a programação de sistemas multiagentes em robôs LEGO Mindstorms NXT é possível, e sugere-se que seja explorada. Alguns exemplos demonstrados por Jensen (2010) expõe o potencial no uso de SMA em robôs LEGO Mindstorms NXT, entretanto foram utilizados apenas dois robôs, pode-se expandir tal ideia e verificar os resultados na implementação de uma sociedade com um maior número de agentes robóticos, trabalhando em conjunto para completar tarefas.

Por fim, espera-se que este trabalho auxilie trabalhos futuros na implementação de agentes inteligentes e sistemas multiagentes em robôs LEGO Mindstorms NXT através da extensão do *framework* Jason.

REFERÊNCIAS BIBLIOGRÁFICAS

AGULLÓ, Miguel et al. **LEGO® MINDSTORMS™ Masterpieces: Building and Programming Advanced Robots**, Syngress Publishing Inc., 2003.

ALVARES, Luis Otavio; SICHMAN, Jaime Simão. **Introdução aos sistemas multiagentes**. In Medeiros, C. M. B., editor, Jornada de Atualização em Informática (JAI'97), chapter 1, p. 1-38. UnB, Brasília, 1997.

ANSORGE, John A. **WEBBOT: A PROJECT TO INSPIRE INTEREST IN ROBOTICS DISCOVERY**. 2006. 51f. A thesis Presented to the Faculty of The Graduate College at the University of Nebraska In Partial Fulfillment of Requirements For the Degree of Master of Arts. Major: Teaching, Learning & Teacher Education. Lincoln, Nebraska, July 2006.

AOS. **AOS Autonomous Decision-Making Software**. Disponível em: <http://aosgrp.com/>. Acesso em maio, 2011.

AUML. **Agent UML Web Site**. Disponível em: <http://www.auml.org/>. Acesso em maio, 2011.

BEHRENS, A. et al. **MATLAB Meets LEGO Mindstorms: A Freshman Introduction Course Into Practical Engineering**. Institute of Imaging & Computer Vision - RWTH Aachen University, Germany. IEEE Transactions on Education, vol. 53, nº 2, p.306-317, may 2010.

BENEDETTELLI, Daniele et al. **A LEGO Mindstorms experimental setup for multi-agent systems**. In: Proc IEEE Control Applications CCA Intelligent Control ISIC, p.1230-1235, 2009.

BENEDETTELLI, Daniele. **Creating Cool MINDSTORMS® NXT Robots**. Apress, 2008.
BOISSIER, Olivier. **Problème du contrôle dans un système intégré de vision**, Utilisation d'un système Multi-Agents. Thèse de Doctorat, Institut National Polytechnique de Grenoble, France, Janvier, 1993.

BORDINI, R. H.; VIEIRA, R. **Linguagens de programação orientadas a agentes: uma introdução baseada em AgentSpeak(L)**. Revista de Informática Teórica e Aplicada, X(1):7–38. Instituto de Informática da UFRGS, Brasil, 2003.

BORDINI, Rafael Heitor; VIEIRA, Renata; MOREIRA, Álvaro Freitas. **Fundamentos de sistemas multiagentes**. In Ferreira, C. E., editor, Jornada de Atualização em Informática (JAI'01), volume 2, chapter 1, p.3-44. SBC, Fortaleza, Brasil, 2001.

BULL, Glen. **Children, computers, and powerful ideas.** Contemporary Issues in Technology and Teacher Education, 5(3/4), p.349-352, 2005.

BUSETTA, Paolo et al. **JACK Intelligent Agents** - Components for Intelligent Agents in Java. AgentLink News Issue 2, 1999.

DEMAZEAU, Yves; MÜLLER, Jean-Pierre. **Decentralized Artificial Intelligence.** Elsevier, Amsterdam, 1990.

DIGOIA, A. M.; KANADE, T.; WELLS, P. **Final report of the second International workshop on robotics and computer assisted medical interventions.** Computer Aided Surgery, 2, p. 69-101, 1996.

FAGUNDES, Carlos Artur Nepomuceno et al. **Aprendendo Matemática com Robótica.** Insituto de Matemática - Universidade Federal do Rio Grande do Sul, CINTED, V.3 N° 2, Novembro de 2005.

FERBER, Jacques. **Multi-Agent Systems:** An Introduction to Distributed Artificial Intelligence. Addison-Wesley, London, 1999.

FORD JR., Jerry Lee. **LEGO® MINDSTORMS® NXT 2.0 for Teens.** Course Technology, 2011.

FRANKLIN, Stan; GRAESSER, Art. **Is It an Agent, or Just a Program?:** A Taxonomy for Autonomous Agents. Intelligent Agents III - Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages (ATAL - 96), ECAI'96 Workshop, Budapest, Hungary, number 1193 in Lecture Notes in Artificial Intelligence, pages 21-35, Springer-Verlag, Berlin, 1997.

GANDY, Elizabeth A. et al. **The use of LEGO Mindstorms NXT Robots in the Teaching of Introductory Java Programming to Undergraduate Students.** ITALICS, Volume 9, Issue 1, p.2-9, february 2010.

GASPERI, Michael; HURBAIN, Philippe; HURBAIN, Isabelle. **Extreme NXT: Extending the LEGO® MINDSTORMS® NXT to the Next Level.** Apress, 2007.

GEORGEFF, M. P.; LANSKY, A. L. **Procedural Knowledge.** In: Proceedings of the IEEE 74(10), p.1383-1398, 1987.

GIRARDI, Rosario. **Engenharia de Software baseada em Agentes**. In: IV Congresso Brasileiro de Ciência da Computação - CBCOMP. Itajaí - SC, ed. 39, 2004.

GOAL. **The GOAL Agent Programming Language**. Disponível em: <http://mmi.tudelft.nl/trac/goal>. Acesso em maio, 2011.

HÜBNER, J. F.; BORDINI, R. H.; VIEIRA, R. **Introdução ao desenvolvimento de sistemas multiagentes com Jason**. XII Escola de Informática da SBC - Paraná. Guarapuava, PR: Editora da UNICENTRO. Capítulo 2, 51 - 89, 2004.

JADEx. **Jadex BDI Agent System**. Disponível em: <http://jadex-agents.informatik.uni-hamburg.de>. Acesso em maio, 2011.

JASON. **A Java-based interpreter for an extended version of AgentSpeak**. Disponível em: <http://jason.sourceforge.net/Jason/Jason.html>. Acesso em maio 2011.

JENNINGS, Nicholas R.; WOOLDRIDGE, Michael J. **Agent Technology: foundations, applications, and markets**. Springer Verlag, London, 1998.

JENSEN, Andreas Schmidt. **Implementing LEGO Agents Using Jason**. Publicado em: <http://arxiv.org/abs/1010.0150>. Outubro, 2010.

JIPPING, Michael J. et al. **Teaching Students Java Bytecode Using LEGO Mindstorms Robots**. Proceedings of the 38th SIGCSE technical symposium on Computer Science education, p.64-ff, march 2007.

KELLY, James Floyd. **LEGO® MINDSTORMS® NXT-G: Programming Guide**. 2.ed. Apress, 2010.

KROUSTIS, Constantinos A.; CASEY, Matthew C. **Combining heuristics and Q-learning in an adaptive light seeking robot**. Report, Department of Computing, University of Surrey, Guildford - Surrey, Reino Unido, 2008.

KUMAR, Amruth N. **Three Years of Using Robots in an Artificial Intelligence Course: Lessons Learned**. Ramapo College of New Jersey. ACM Journal on Educational Resources in Computing, vol. 4, nº 3, article 1, september 2004.

LEGO. **LEGO Mindstorms: User Guide**. 2006.

LEJOS. **leJOS, Java for LEGO Mindstorms**. Disponível em: <http://leJOS.sourceforge.net/>. Acesso em março, 2011.

LEJOSAPI. **Overview (leJOS NXJ API documentation)**. Disponível em: <http://lejos.sourceforge.net/nxt/nxj/api/index.html>. Acesso em maio, 2011.

LEMKE, Anders; LAIDLAW, Johan; PEDERSEN, Lars Zilmer. **Developing Multi-Agent LEGO Robotics**. Polytechnical Midterm Project, Technical University of Denmark, Kongens Lyngby, Dinamarca, 2007.

LEW, Michael W.; HORTON, Thomas B.; SHERIFF, Mark S. **Using LEGO MINDSTORMS NXT and LEJOS in an Advanced Software Engineering Course**. Department of Computer Science, University of Virginia. Conference on *Software Engineering Education and Training*.

LOGO. **Logo Foundation**. Disponível em: <http://el.media.mit.edu/logo-foundation/logo/index.html>. Acesso em maio, 2011.

LUI, Andrew K.; CHEUNG, S. C. Ng. Yannie H. Y.; GURUNG, Prabhat. **Facilitating Independent Learning with LEGO Mindstorms Robots**. *Acrs Inroads*, vol. 1, nº 4, p.49-53, december 2010.

MASE. **Organisation-base Multiagent Systems Engineering**. Disponível em: <http://macr.cis.ksu.edu/O-MaSE>. Acesso em maio, 2011.

MICHAELIS. **Moderno Dicionário da Língua Portuguesa**. Disponível em <http://michaelis.uol.com.br/>. Acesso em maio, 2011.

MINDSTORMS. **LEGO Mindstorms NXT**. Disponível em: <http://mindstorms.LEGO.com/>. Acesso em março, 2011.

MOISE. **Moise organisational model**. Disponível em: <http://moise.sourceforge.net/>. Acesso em maio, 2011.

MOTA, Martha Isela Garduno. **Work In Progress - Using LEGO Mindstorms and Robolab as A Mean To Lowering Dropout and Failure Rate In Programming Course**. Universidad Autónoma de Baja California - Facultad de Ciencias Químicas e Ingeniería Tijuana, BC, Mexico. 37th ASEE/IEEE Frontiers in Education Conference, Session F4A-2, october 2007.

NBC/NXC. **Next Byte Codes and Not eXactly C.** Disponível em: <http://bricxcc.sourceforge.net/nbc/>. Acesso em maio, 2011.

OPLER, Ascher. **Fourth-Generation Software.** Datamation 13 (1): p.22–24, janeiro de 1967.

PADGHAM, Lin; THANGARAJAH, John; WINIKOFF, Michael. **The Prometheus Design Tool - A Conference Management System Case Study.** In: *Agent-Oriented Software Engineering VII: 8th International Workshop*, p.197-211, Honolulu - HI - EUA, 2007.

PADGHAM, Lin; WINIKOFF, Michael. **Developing Intelligent Agent Systems: A practical guide.** Editora John Wiley & Sons, Melbourne - Australia, 2004.

PANADERO, Carmen Fernández; ROMÁN, Julio Villena; KLOOS, Carlos Delgado. **Impact of Learning Experiences Using LEGO Mindstorms® in Engineering Courses.** IEEE EDUCON Education Engineering - 2010 - The Future of Global Learning Engineering Education. IEEE, Madrid - SPAIN, p.503-512, 2010.

PAPERT, Seymour. **Mindstorms: Children, computers, and powerful ideas.** Basic Books, 1980.

PASQUALIN, Douglas Pereira. **Tratamento de falhas em robô LEGO mindstorms com arquitetura reativa.** 2009. 48f. Trabalho Final de Graduação (Bacharelado em Sistemas de Informação) - Curso de Sistemas de Informação, Centro Universitário Franciscano, Santa Maria, 2009.

PBLUA. **pBLua.** Disponível em: <http://www.hempeldesigngroup.com/LEGO/pblua/>. Acesso em maio, 2011.

PC_MAGAZINE. **PC Magazine Encyclopedia.** Disponível em: <http://www.pcmag.com/encyclopedia/>. Acesso em maio, 2011.

PDT. **RMIT Agents Group - Prometheus & PDT.** Disponível em: <http://www.cs.rmit.edu.au/agents/pdt/>. Acesso em maio, 2011.

PDTMANUAL. **Prometheus Design Tool User Manual.** Disponível em: <http://www.cs.rmit.edu.au/agents/pdt/docs/PDT-Manual.pdf>. Acesso em maio, 2011.

PEREIRA, Henrique Gabriel Gularte. **Desenvolvimento de uma aplicação para o controle remoto de LEGO Mindstorms NXT.** 2009. 33f. Trabalho Final de Graduação (Bacharelado

em Sistemas de Informação) - Curso de Sistemas de Informação, Centro Universitário Franciscano, Santa Maria, 2009.

PFEIFER, Erick et al. **Coleta de Lixo Médico Utilizando Protótipos de LEGO**. Anais do XXVI Congresso da SBC - EnRI, III Encontro de Robótica Inteligente, 14 a 20 de julho de 2006, Campo Grande - MS, p. 75-82, 2006.

POKAHR, Alexander; BRAUBACH, Lars; LAMERSDORF, Winfried. **Jadex: Implementing a BDI-Infrastructure for JADE agents**. Published in EXP - in search of innovation (Special Issue on JADE), p.76-85, 2003.

PROMETHEUS. **Developing Intelligent Agent Systems: A practical guide**. Disponível em: <http://www.cs.rmit.edu.au/agents/Prometheus/>. Acesso em maio, 2011.

QUICKSTART. **Quick Start Manual for Eclipse-based PDT**. Disponível em: <http://code.google.com/p/pdt-plugin/wiki/QuickStartManual>. Acesso em maio, 2011.

RAO, Anand S. **AgentSpeak(L): BDI Agents speak out in a logical computable language**. In Van de Velde, W. e Perram, J., editors, Proceedings of the Seventh Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAA-MAW'96), 22-25 January, Eindhoven, The Netherlands, number 1038 in Lecture Notes in Artificial Intelligence, p.42-55, London, Springer-Verlag, 1996.

REZENDE, Solange Oliveira. **Sistemas Inteligentes: Fundamentos e Aplicações**. Editora Manole, Barueri - SP, 2005.

RIBEIRO, Célia Rosa. **RobôCarochinha: Um Estudo Qualitativo sobre a Robótica Educativa no 1º ciclo do Ensino Básico**. 2006. 191f. Mestrado em Educação, Tecnologia Educativa - Universidade do Minho, Instituto de Educação e Psicologia, Braga, outubro de 2006.

ROBOTC. **ROBOTC, a C Programming Language for Robotics**. Disponível em: <http://www.robotc.net/>. Acesso em maio, 2011.

RUSSEL, Stuart; NORVIG, Peter. **Inteligência Artificial**. Tradução da 2ª edição. Editora Campus, Florianópolis 2004.

SICHMAN, Jaime Simão. **Du Raisonment Social Chez les Agents: Une Approche Fondée sur la Théorie de la Dépendance**. Thèse (doctorat), Institut National Polytechnique de Grenoble, 1995.

TANENBAUM, Andrew S. **Sistemas Operacionais Modernos**. Tradução da 2ª edição. Editora Prentice Hall, São Paulo, 2003.

TROPOS. **Tropos Methodology**. Disponível em: <http://www.troposproject.org/>. Acesso em maio, 2011.

TUTORIALPDT. **Prometheus Design Tool Tutorial**. Disponível em: <http://www.cs.rmit.edu.au/agents/pdt/docs/Tutorial.pdf>. Acesso em maio, 2011.

TVEIT, Amund. **A survey of Agent-Oriented Software Engineering**. In: Proceedings of the First NTNU Computer Science Graduate Student Conference. Norwegian University of Science and Technology, may 2001.

URBI. **Urbi Open Source**. Disponível em: <http://www.gostai.com/products/urbi/>. Acesso em maio, 2011.

WEIß, Gerhard. **Multiagent Systems: A modern approach to distributed artificial intelligence**. MIT Press, London, 1999.

WOBCKE, Wayne. **Reasoning about BDI Agents from a Programming Languages Perspective**. In: Proceedings of the AAAI 2007 Spring Symposium on Intentions in Intelligent Systems. 2007.

WOOLDRIDGE, Michael. **An Introduction to MultiAgent Systems**. Second Edition, published by John Wiley & Sons, 2009.

WOOLDRIDGE, Michael. **Intelligent Agents**. In Weiß, G., editor, *Multiagent Systems - A Modern Approach to Distributed Artificial Intelligence*, chapter 1, p.27-77. MIT Press, Cambridge, MA, 1999.

WOOLDRIDGE, Michael; JENNINGS, Nicholas R. **Intelligent agents: theory and practice**. The Knowledge Engineering Review, Vol. 10:2, p.115-152, 1995.

ANEXO A - Extensões da sintaxe do Jason

Neste anexo, apresenta-se as ações e percepções que podem ser utilizadas na programação de agentes Jason para robôs LEGO Mindstorms NXT. Traduzido de [JENSEN, 2010].

Ações

***forward*(Motores, Velocidade)** Aciona uma lista de motores a mover para frente, a uma dada lista de velocidades.

Motores: Lista de motores.

Velocidade: Lista de velocidade correspondente ao motor de mesmo índice da lista de motores.

Exemplos:

- *forward*([a, b], [300, 300]): Os motores A e B irão mover para frente com uma velocidade de 300.
-

***backward*(Motores, Velocidade)** Aciona uma lista de motores a mover para trás, a uma dada lista de velocidades.

Motores: Lista de motores.

Velocidade: Lista de velocidade correspondente ao motor de mesmo índice da lista de motores.

Exemplos:

- *backward*([a, b], [300, 300]): Os motores A e B irão mover para trás com uma velocidade de 300.
-

***rotate*(Motores, Ângulo)** Rotaciona uma lista de motores, dada uma lista de ângulos.

Motores: Lista de motores.

Ângulo: Lista de ângulo correspondente ao motor de mesmo índice da lista de motores.

Exemplos:

- *rotate*([a, b], [300, 300]): Os motores A e B irão rotacionar 300.

- *rotate*([a, b], [100, -100]): O motor A irá rotacionar 100 graus, e o motor B irá rotacionar -100 graus.

***reverse*(Motores)**

Reverte a direção de uma lista de motores.

Motores: Lista de motores.

Exemplos:

- *reverse*([a, b]): Os motores A e B irão reverter a sua direção, individualmente. Se o motor A estava indo para frente e o motor B para trás, agora o motor A irá para trás e o motor B para frente.

***speed*(Motores, Velocidade)**

Ajusta a velocidade para cada motor da lista.

Motores: Lista de motores.

Velocidade: Lista de velocidade correspondente ao motor de mesmo índice da lista de motores.

Exemplos:

- *speed*([a, b], [200, 300]): Ajusta a velocidade do motor A para 200 e a velocidade do motor B para 300.

***stop*(Motores)**

Para os motores da lista.

Motores: Lista de motores.

Exemplos:

- *stop*([a, b]): Para os motores A e B.

***block*(Bloquear)**

Determina se o comando de rotação deve retornar o controle para a thread imediatamente ou bloquear.

Bloquear: Valor booleano, *true* bloqueia e *false* retorna.

Exemplos:

- *block*(*true*): Bloqueia as próximas ações do robô até que a rotação seja concluída. Não possui efeito imediato, mas influencia a ação *rotate*(Motores, Ângulo).

exit

Finaliza o agente.

Exemplos:

- *exit*

Percepções

***light*(Porta, Valor)**

Percepção de um sensor de luz.

Porta: Porta em qual o sensor está conectado. Pode ser utilizada para distinguir sensores do mesmo tipo.

Valor: Valor de luz lido pelo sensor.

Exemplos:

- *light*(1, 360): O sensor de luz na porta 1 leu um valor de 360.

***obstacle*(Porta, Valor)**

Percepção de um sensor ultrassônico.

Porta: Porta em qual o sensor está conectado. Pode ser utilizada para distinguir sensores do mesmo tipo.

Valor: Distância em centímetros em que o sensor percebe o obstáculo.

Exemplos:

- *obstacle*(1, 40): O sensor ultrassônico na porta 1 encontrou um obstáculo a 40 centímetros.

***touching*(Porta, Valor)**

Percepção de um sensor de contato.

Porta: Porta em qual o sensor está conectado. Pode ser utilizada para distinguir sensores do mesmo tipo.

Valor: Verdadeiro ou falso, está em contato ou não.

Exemplos:

- *touching*(1, *true*): O sensor de toque na porta 1 detectou contato.
- *touching*(1, *false*): O sensor de toque na porta 1 não detectou contato.

***sound*(Porta, Valor)**

Percepção de um sensor de som.

Porta: Porta em qual o sensor está conectado. Pode ser utilizada para distinguir sensores do mesmo tipo.

Valor: Valor do som lido pelo sensor.

Exemplos:

- *sound*(1, 55): O sensor de som na porta 1 leu um valor de 55.

ANEXO B - Estrutura dos códigos fontes

Encontra-se no diretório 'jason' a seguinte estrutura:

- Diretório 'src': contém os códigos fontes necessários para a execução do agente no computador;
 - Diretório 'asl': códigos em Jason, onde cada arquivo representa um agente (todos os agentes, com a exceção do 'tfg.asl', são de autoria de [JENSEN, 2010] e foram incluídos como exemplos, mas não são utilizados neste trabalho);
 - Arquivo 'ballcatcher.asl': procura um objeto para agarrar;
 - Arquivo 'blindagent.asl': agente cego, que segue os passos do 'obstaclefinder.asl';
 - Arquivo 'comma.asl': gira até tocar 'commb.asl', e então para;
 - Arquivo 'commb.asl': comunica a 'comma.asl' que foi tocado;
 - Arquivo 'linefollower.asl': verifica se o robô está sob uma linha;
 - Arquivo 'obstaclefinder.asl': capaz de detectar obstáculos, e informar ao agente 'blindagent.asl' aonde se encontram;
 - Arquivo 'tfg.asl': construído para a tarefa proposta neste trabalho.
 - Diretório 'java': códigos na linguagem de programação Java referentes à execução do agente no computador.
 - Diretório 'agent': códigos referente aos agentes;
 - Arquivo 'UniqueBelsBB.java': base de crenças customizada, para garantir que em um dado momento se tenha no máximo uma percepção por sensor.
 - Diretório 'arch': códigos que compõem a arquitetura dos agentes;
 - Arquivo 'Actions.java': extensão responsável por identificar as ações dos agentes e verificar se os formatos estão corretos;
 - Arquivo 'BTInputReader.java': responsável por ler as percepções recebidas por *Bluetooth*;
 - Arquivo 'LEGOAgArchitecture.java': interface que controla os parâmetros e a inicialização do agente.

- Diretório 'gui': arquivos da interface gráfica.
 - Arquivo 'AgentInterface.java': interface gráfica do usuário (*Graphic User Interface*).
- Arquivo 'build.xml': utilizado para facilitar a compilação/execução dos códigos, contém os caminhos e as bibliotecas necessárias.
- Arquivo 'LEGO-Jason.mas2j': especificações do projeto em Jason, declaração dos agentes a serem utilizados.

Encontra-se no diretório 'nxt' a seguinte estrutura:

- Diretório 'src': contém os códigos fontes necessários para a execução do agente no robô;
 - Diretório 'nxt': códigos que serão executados no robô.
 - Diretório 'parsing': códigos de interpretação;
 - Arquivo 'ActionParser.java': interpreta as ações recebidas do computador, para comandos da linguagem leJOS;
 - Arquivo 'Constants.java': possui algumas constantes utilizadas para facilitar a interpretação;
 - Diretório 'thread': threads que controlam os diferentes tipos de sensores;
 - Arquivo 'LightSensorThread.java': sensor de luz;
 - Arquivo 'SensorThread.java': superclasse herdada pelas demais classes de sensores;
 - Arquivo 'SensorThreadFactory.java': responsável pela instanciação e a configuração inicial de cada thread;
 - Arquivo 'SoundSensorThread.java': sensor de som;
 - Arquivo 'TouchSensorThread.java': sensor de contato;
 - Arquivo 'UltrasonicSensorThread.java': sensor ultrassônico.
 - Arquivo 'BTInputReader.java': leitura de dados de entrada por *Bluetooth*;
 - Arquivo 'BTOutputReader.java': envio de dados de saída por *Bluetooth*;

- Arquivo 'LEGOJasonNXT.java': espera a conexão por *Bluetooth*, e então inicializa os motores e os sensores;
- Arquivo 'Settings.java': configurações iniciais da velocidade dos motores.
- Arquivo 'build.xml': utilizado para facilitar a compilação/execução dos códigos, contém os caminhos e as bibliotecas necessárias.

APÊNDICE A - Tutorial para a instalação dos componentes

Este apêndice contém um breve tutorial para a instalação dos diversos componentes necessários para o planejamento de percepções e ações de agentes Jason no robô LEGO Mindstorms NXT. Observa-se que pelo fato destes componentes serem projetos ainda em desenvolvimento, é possível que os códigos disponibilizados neste trabalho não funcionem com novas versões desses componentes. Portanto é especificada a versão utilizada de cada componente, de maneira a facilitar a instalação e minimizar os possíveis erros que venham a ocorrer, se possível escolha a versão indicada neste tutorial para *download*.

Jason

Site: <<http://jason.sourceforge.net/Jason/Jason.html>>

Versão utilizada: 1.3.4

Instruções: Faça o *download* do arquivo compactado (é necessário a versão compactada e não o plugin) e o extraia para um diretório de sua preferência, tenha em mente que este diretório será utilizado mais a frente no ajuste das variáveis de ambiente.

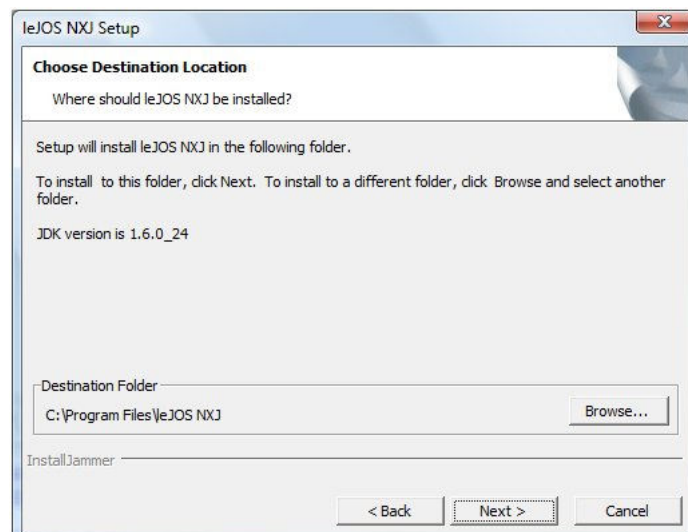
leJOS

Site: <<http://lejos.sourceforge.net/>>

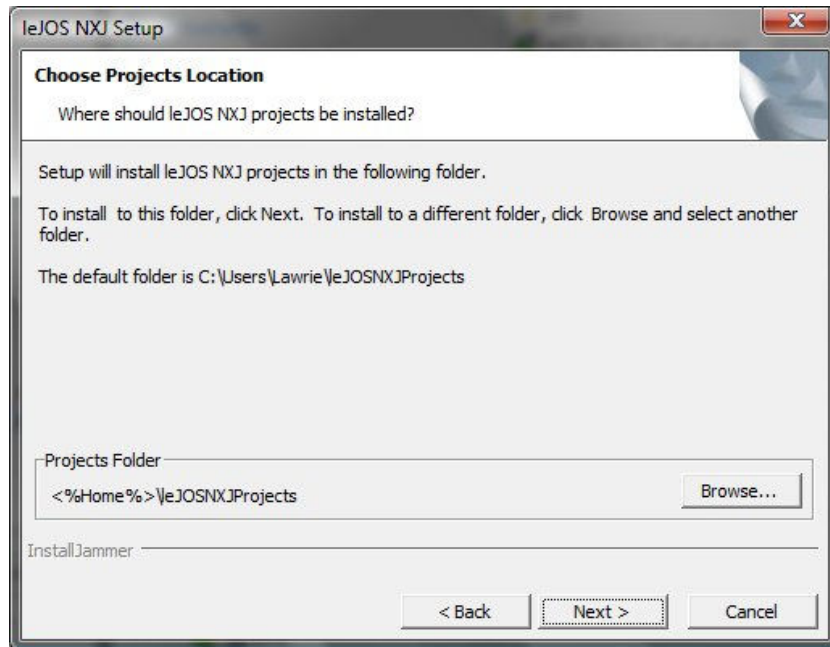
Versão utilizada: 0.8.5 NXJ

Instruções: Faça o *download* do instalador (é necessário a versão com o instalador e não o plugin), e execute o instalador.

Avance até chegar na seguinte tela:

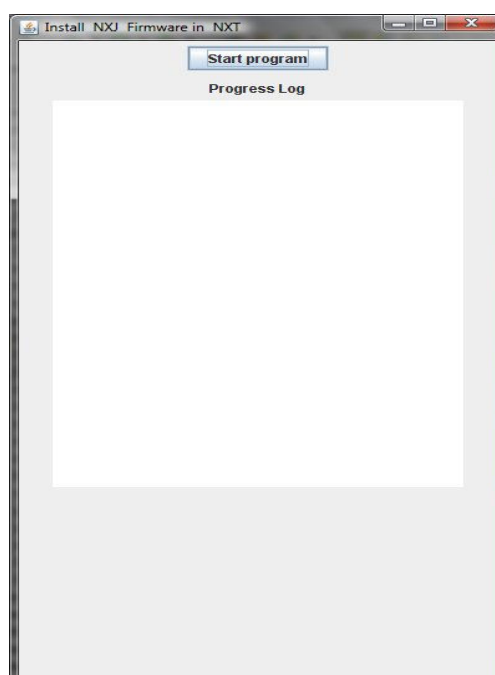


Aqui é possível selecionar o diretório para a instalação selecionando '*Browse*', tenha em mente que este diretório será utilizado mais a frente no ajuste das variáveis de ambiente. Para continuar selecione '*Next*'.



Aqui é possível selecionar o diretório para os projetos selecionando '*Browse*', <%Home%> no Windows 7 e no Windows Vista equivale a C:\Users\usuário e no Windows XP a C:\Documents and Settings\usuário. Para continuar selecione '*Next*'.

Caso possua alguma versão do leJOS já instalada no diretório escolhido pressione '*Next*' para desinstalar. Após a desinstalação prossiga com a instalação até encontrar a seguinte tela:



Caso queira atualizar o *firmware* do robô neste instante, ligue o robô e o conecte no computador, e então selecione '*Start program*' e siga os passos até ser perguntado se deseja atualizar o *firmware* novamente. Caso possua mais robôs para atualizar selecione '*Yes*'. Se não quiser atualizar nenhum robô no momento feche a aplicação. É possível atualizar o *firmware* a qualquer momento digitando 'nxjflash' na linha de comando, ou ainda executando a interface gráfica 'nxjflashg.bat', vista na imagem anterior, presente no diretório 'bin' do diretório de instalação do leJOS.

Apache Ant

Descrição: Apache Ant é uma ferramenta de linha de comando, uma biblioteca em Java que simplifica a compilação e execução de *softwares* através de arquivos *build.xml*.

Site: <<http://ant.apache.org/>>

Versão utilizada: 1.8.2

Instruções: Faça o *download* do arquivo compactado e o extraia para um diretório de sua preferência, tenha em mente que este diretório será utilizado mais a frente no ajuste das variáveis de ambiente.

Java JDK

Descrição: JDK contém ferramentas para o desenvolvimento e teste de programas escritos na linguagem de programação Java que rodam na plataforma Java.

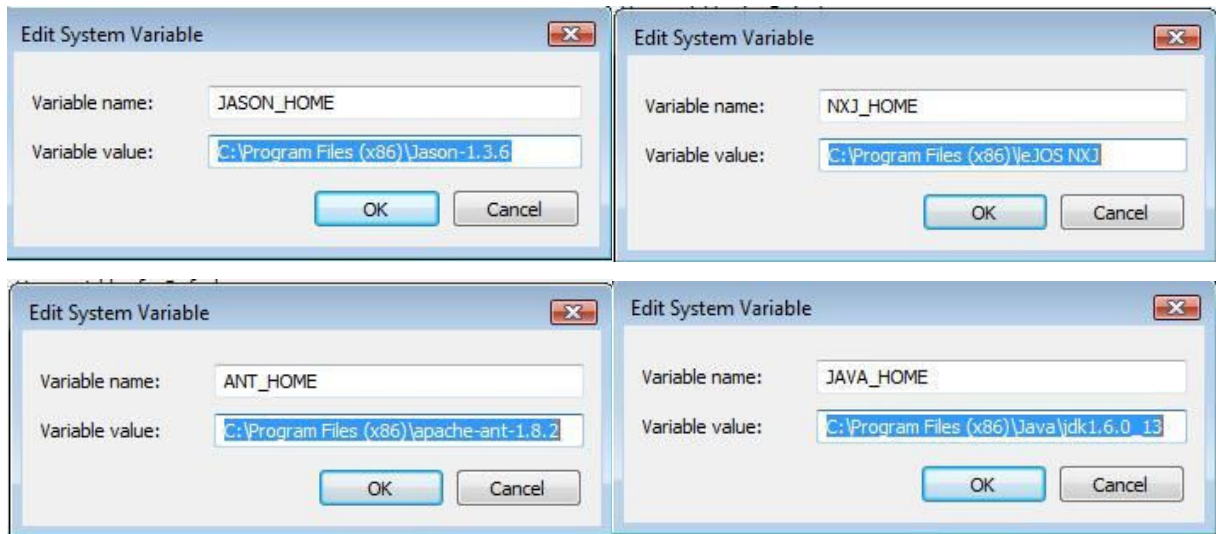
Site: <<http://www.oracle.com/technetwork/java/index.html>>

Versão utilizada: 1.6.0_13

Instruções: Faça o *download* do instalador, execute-o e siga os passos, nota-se que o diretório escolhido para instalação será utilizado mais a frente no ajuste das variáveis de ambiente.

Variáveis de ambiente

Instruções: Clique com o botão direito do mouse em 'Meu Computador' e selecione propriedades > Configurações avançadas do sistema > Variáveis de ambiente. Adicione as seguintes variáveis em variáveis de sistema (substituindo o '*Variable value*' pelo diretório que foi escolhido durante as instalações):



Agora é necessário adicionar os diretórios 'bin' de cada componente a variável 'path'. Se ela já existe, edite-a e adicione a seguinte linha (coloque 'ponto e vírgula' após cada expressão). Caso contrário crie-a.

DISCO:\DiretórioDeInstalação\Jason-1.3.6\bin;DISCO:\DiretórioDeInstalação\leJOS

NXJ\bin;DISCO:\DiretórioDeInstalação\apache-ant-1.8.2\bin;DISCO:\DiretórioDeInstalação\jdk1.6.0_13\bin

Execução

Site com os códigos: <site>

Instruções: Faça o *download* do arquivo compactado e extraia para um diretório de sua preferência. Abra a linha de comando (Iniciar > executar > cmd). Navegue (comando cd) até o diretório extraído. Utilize os seguintes comandos para compilar os códigos (necessariamente nessa ordem):

- cd nxt
- ant jar
- cd ..
- cd jason
- ant jar
- cd ..

Agora para realizar o *upload* do código para o robô LEGO Mindstorms NXT, conecte-o no computador e ligue-o. Após digite os seguintes comandos:

- cd nxt

- `ant upload`
- `cd ..`

Finalmente, para iniciar a execução digite:

- `cd jason`
- `ant run`

Nota-se que sempre que um dos códigos for alterado, todos os passos devem ser refeitos.