UNIVERSIDADE DE SÃO PAULO

ESCOLA DE ENGENHARIA DE SÃO CARLOS

Departamento de Engenharia Elétrica

**Ambiente de Desenvolvimento de Aplicações
para Robôs Móveis**

Jean Miler Scatena

Tese apresentada à Escola de Engenharia de São Carlos da Universidade de São Paulo, como parte dos requisitos para a obtenção do título de Doutor em Engenharia Elétrica.

Orientadora: Prof^a. Dra. Maria Stela Veludo de Paiva

São Carlos
2008

Dedico este trabalho a todos que acreditaram em mim e me apoiaram em todos os momentos difíceis e que nunca me deixaram desistir.

Agradecimentos

À todas as pessoas presentes em todos os momentos da minha vida e especialmente aqueles que se mantiveram ao meu lado quando mais precisei.

À Profa. Dra. Maria Stela Veludo de Paiva por toda a paciência e compreensão além da disposição e atenção dispendida comigo durante todo o projeto.

Aos meus bons e velhos amigos Henrique e Leandro, por todas as “guerras”, “conflitos” e viagens que passamos juntos.

Ao meu orientador do mestrado Prof. Dr. Eduardo Marques, o qual foi responsável pela ideia inicial deste projeto e dentro de suas limitações burocráticas me guiou nesta empreitada até o momento que isto não fosse mais possível.

A minha noiva Aline, que hoje é minha noiva, porém no dia desta defesa, se nada ocorrer de errado, já seja minha esposa.

À uma grande amiga, Dona Magdalena, que por mais ríspida que seja suas palavras, são sempre verdadeiras e sinceras, abrindo sempre meus olhos quando mais precisava.

Aos meus pais e irmãs que sempre me incentivaram a continuar, indiferente das dificuldades encontradas e por todo amor, carinho e confiança depositados em mim.

Além das pessoas acima citadas, o meu ex-professor o José Pacheco, além do agradecimento, um grande abraço a um amigo único, que foi responsável por mais essa conquista. Um amigo que acreditou no meu potencial e apostou nisso.

Resumo

SCATENA, J.M. Ambiente de Desenvolvimento de Aplicações para Robôs Móveis. 2008. 131 f. Tese (Doutorado) – Escola de Engenharia Elétrica de São Carlos, Universidade de São Paulo, São Carlos, 2008.

Recentes avanços na tecnologia de robôs móveis tem sido direcionados para um novo tipo de sistema robótico o qual é denominado de robô pessoal. Esses robôs realizam tarefas caseiras cotidianas, além de interagir e auxiliar pessoas que necessitam de cuidados especiais. Para realizar tais tarefas são necessárias medidas de interação e programação entre o robô e o ser humano. Essa interface é discutida em diversas pesquisas que evidenciam a necessidade de um ambiente de desenvolvimento de aplicações para robôs com alto nível de abstração e que possua uma interface amigável. Com isso, o presente trabalho vem contribuir com essa nova área de pesquisa através da proposta de um ambiente de desenvolvimento de aplicações para robôs com alto nível de abstração utilizando blocos de tarefas, que se destina tanto a programadores experientes quanto a usuários com pouco conhecimento de programação. Para realizar o interfaceamento com esses dois tipos de usuários, o ambiente provê um módulo de programação avançado que se destina à criação de novas tarefas, e um módulo de programação específico para usuários com pouco conhecimento em programação, com uma interface amigável. No módulo avançado as novas tarefas utilizam a estrutura de uma biblioteca de alto nível de abstração (*Java for Robotics*), também proposta nesse trabalho. Além das interfaces, o ambiente contém um módulo intermediário, apresentado como *Middleware*, que manipula todas as informações do ambiente de programação com o sistema robótico ou com o simulador de robôs. Essa proposta de ambiente foi validada utilizando o simulador de robôs *Player/Stage* com o sistema robótico Khepera.

Palavras-chave: Ambiente de desenvolvimento. Robôs Móveis. Robôs Pessoais. Robôs Domésticos

Abstract

SCATENA, J.M. Environment for Development Applications for Mobile Robots. 2008. 131 f. PhD. Thesis - Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2008.

Recent advances in the technology of mobile robots have developed for a new type of robotic system which is named personal robot. These robots are able to realize domestic daily tasks, interact and help people with special cares. In order to perform such tasks, interaction and programming measurements between the robot and the human being are necessary. This interface is discussed in several research that put in evidence the necessity of a development environment for applications or manipulation for the personal robots has a friendly interface. In this way, the present work contributes to this new research field with the proposal of an environment for robots applications development (framework), with high abstraction level with tasks blocks, designed for both experienced programmers and non experienced users. To make possible the interface between the two kinds of users, the environment provides an advanced programming module and a specific one for beginners. The advanced module allows the creation of new applications or tasks for the robotic system using a high abstraction level considering the robot programming. On the other hand, the module for beginners uses tasks blocks with a friendly interface. Based in an intermediate module, named *Middleware*, which management the whole data of environment programming language and provides an output to the robotic system or to the robot simulator, the environment becomes able to support several robotic systems. The proposal of the environment was validate using the simulator *Player/Stage* with Khepera's system.

Keywords: Development Environment. Mobile Robots. Personal Robots. Domestic Robots.

Lista de Figuras

Figura 3.1 - Stage em Simulação.....	22
Figura 3.2 - Stage Simulando o Ambiente de um Hospital.....	22
Figura 3.3 - Gazebo no Ambiente em 3D	23
Figura 3.4 - Gazebo e a visualização do alcance do LaserFinder.	24
Figura 3.5 - Ambiente de Simulação FLAT 2D.	25
Figura 3.6 - Controlador do Robô - FLAT 2D.....	25
Figura 3.7 - Ambiente de Programação - MobotSim.	27
Figura 3.8 - WeBots - Sony Aibo e LEGO Mindstorms.	28
Figura 3.9 - Judô Robot e Humanoid robot.	29
Figura 3.10 - Pioneer 2 robô e Simulação partida de Futebol com detalhes realísticos ..	29
Figura 3.2 - Khepera robot e Khepera II robot.	29
Figura 3.3 - Koala robot e Magellan robot.	30
Figura 3.43 - Arquitetura do Saphira.....	33
Figura 3.5 - Conexão do Cliente Saphira.....	34
Figura 3.6 - Diagrama da estrutura do ARIA.....	36
Figura 4.1 - Roomba.	44
Figura 4.2 - Robô Bender.	45
Figura 4.3 - Robô Maggie	46
Figura 4.4 - Robô MAT	47
Figura 4.5 - Robô ASKA e sua Estrutura	48
Figura 4.6 - The Care-O-Bot II - Utilidades	49

Figura 4.7 - The Care-O-Bot II – Modelo e Arquitetura	49
Figura 4.8 - Mowbot.....	50
Figura 4.9 - Papero.	52
Figura 5.1 - Interface do Pyro.	55
Figura 5.2 - Gazebo simulando uma aplicação desenvolvido no Pyro.	56
Figura 5.3 - Arquitetura ROCI.....	57
Figura 5.4 - Partes do Sistema de Programação do Robô.....	58
Figura 5.5 - Camada do Framework.....	58
Figura 5.6 -Sistema de Programação do Robô.....	61
Figura 5.7 -Sistema de Programação utilizando RobotFlow.....	62
Figura 5.9 - Estrutura de Programação Miro.....	64
Figura 5.8 - A estrutura do sistema básico do MRROC++.....	65
Figura 5.9 - Partes do Sistema de Programação do Robô.....	67
Figura 6.1 -Arquitetura do Ambiente de Programação.	74
Figura 6.2 - Diagrama de USE CASE	77
Figura 6.3 - Diagrama de Classes	79
Figura 6.4 - Diagrama de Sequência do Usuário.....	81
Figura 6.5 - Diagrama de Sequência do Programador	83
Figura 7.1 - Sequência de Desenvolvimento dos Módulos	92
Figura 7.2 - Subdiretório Settings.....	94
Figura 7.3 - Expert Editor	97
Figura 7.4 - Wizard Editor	99
Figura 7.5 - Editor da Biblioteca J4R	101

Figura 7.6 –Conversor para o Sistema Robótico	104
Figura 7.7 –Conversão para o Sistema Robótico.....	105
Figura 7.8 –Conversor para o Simulador	107
Figura 8.1 - Hierarquia de Execução de Aplicações no Sistema Robótico	119

Lista de Tabelas

Tabela 3.1 Comparação entre os softwares de simulação de robôs móveis	37
Tabela 8.1 Comparação entre os Ambientes de Desenvolvimento	117

Lista de Abreviações

ADARM – Ambiente de Desenvolvimento de Aplicações para Robôs Móveis

ARIA - Advanced Robotics Interface for Applications

API - Application Programming Interface

CORBA - Common Object Request Broker Architecture

DRP – Dinamically Reconfigurable Personal Robot

EAP - Environment Application Development

EAP4MR - Environment Application Development for Mobile Robots

FPGA – Field Programmable Gate Array

HRI – Human-Robot Interaction

IA – Inteligência Artificial

IDL - Interface Definition Language

J4R – Java For Robotics

JVM – Java Virtual Machine

LCR – Laboratório de Computação Reconfigurável

OOP – Object-Oriented Programming

PbD – Programming by Demonstration

PIC – Peripheral Interface Controller

RDC – Robot Description Code

SO – Sistema Operacional

UML – Unified Modeling Language

Wi-Fi – Wireless Fidelity

Sumário

Capítulo 1 - Introdução	1
1.1. Motivação e Relevância da Tese	1
1.2. Objetivo.....	4
1.3. Justificativa.....	6
1.4. Organização da Tese	10
Capítulo 2 - Robôs	11
2.1. Introdução	11
2.2. Robôs Móveis.....	12
2.3. Robôs Móveis Embarcados	13
2.4. Robôs Domésticos (Domestic Robots) ou Robôs Pessoais (Personal Robots)	15
2.5. Considerações sobre o Capítulo.....	18
Capítulo 3 - Simuladores de Robôs Móveis	19
3.1. Introdução	19
3.2. Simulador Player/Stage.....	20
3.2.1. Interface de dispositivos de robô.....	21
3.2.2. Simulador de múltiplos de robôs.....	22
3.2.3. Simulador de multi-robôs 3D	23
3.3. Simulador Flat 2D	24
3.4. Simulador MobotSim.....	25
3.5. Simulador WeBots.....	27
3.6. Simulador Saphira.....	30
3.7. Considerações sobre o Capítulo.....	36
Capítulo 4 - Estado da Arte em Robótica Doméstica ou Robôs Pessoais.....	39
4.1. Introdução	39

4.2. Exemplos de Robôs Domésticos	41
4.3. Considerações Finais do Capítulo	52
Capítulo 5 - Estado da Arte de Ambientes de Desenvolvimento para Sistemas Robóticos 53	
5.1. Introdução	53
5.2. Principais Ambientes de Desenvolvimento para Sistemas Robóticos	53
5.3. Considerações sobre o Capítulo.....	66
Capítulo 6 - Proposta de um Ambiente de Desenvolvimento de Aplicações para Robôs Móveis 71	
6.1. Introdução	71
6.2. Objetivo.....	72
6.3. Arquitetura.....	73
6.4. Modelo Formal do Ambiente de Desenvolvimento de Aplicações para Robôs Móveis	76
6.5. Outras considerações sobre o ADARM	84
6.5.1. Escolha da Linguagem	84
6.5.2. Biblioteca Java for Robotics (J4R)	85
Capítulo 7 - Construção do Ambiente de Desenvolvimento de Aplicações para Robôs Móveis 89	
7.1. Introdução	89
7.2. Metodologia	92
7.2.1. Interface de Usuário.....	95
7.2.2. Biblioteca J4R – Java For Robotics.....	100
7.2.3. Camada Middleware	102
7.2.4. Módulo de Conversão para o Ambiente de Simulação.....	106
7.3. Considerações sobre o Capítulo.....	108

Capítulo 8 - Resultados e Discussões	109
8.1. Introdução	109
8.2. Resultados	109
8.2.1. Avaliação do ambiente implementado (ADARM).....	110
8.2.2. Comparação do ADARM com outros ambientes.....	113
8.3. Outras considerações.....	118
Capítulo 9 - Conclusões Gerais e Trabalhos Futuros.....	121
9.1. Conclusões	121
9.2. Contribuições.....	123
9.3. Propostas para trabalhos futuros.....	124
Referências Bibliográficas	127

Capítulo 1 - Introdução

1.1. *Motivação e Relevância da Tese*

Recentes avanços na tecnologia de robôs móveis tem se desenvolvido para um novo tipo de sistema robóticos, os quais são chamados de robôs pessoais (SALEMI et al., 2005). Esses robôs têm como objetivo a realização de tarefas da vida cotidiana, como por exemplo, passar roupa, lavar louça, cuidar do jardim, etc. (PARK et al., 2008; FRAUNHOFER, 2003). Além disso, devem interagir com os seres humanos e com os objetos encontrados no seu percurso. (FALCONE et al., 2003; MAZER et al., 1991).

Atualmente as aplicações de robôs domésticos vem ganhando espaço em proporções consideráveis, a ponto de ser cogitado o possível desenvolvimento desses sistemas em aplicações comerciais, aproveitando assim, as inúmeras possibilidades a serem exploradas no ramo doméstico, que incluem desde simples tarefas cotidianas até tarefas mais complexas, como a vigilância interna de residências, com o monitoramento da casa e o acionamento de alarme via telefone (residencial ou celular), além de informar via internet, a situação de cada cômodo da casa (PARK et al., 2008; FALCONE et al., 2003).

Outra aplicação interessante dos robôs domésticos, que é evidenciada pelas pesquisas da área, como no trabalho de Balaguer et al.(2006) , está vinculada ao auxílio a pessoas idosas e a pessoas enfermas, que não possuem condições de se locomover ou realizar tarefas cotidianas (PARK et al., 2008; FRAUNHOFER, 2003; FALCONE et al., 2003).

Algumas aplicações para robôs domésticos podem ser encontradas em livros de ficção científica como os de Isaac Asimov, onde são descritas as utilidades dos robôs dentro do ambiente doméstico e a alta interação entre os robôs e os seres humanos (ASIMOV, 1939).

Para que os robôs existentes hoje consigam se aproximar dos contos de Isaac Asimov (ASIMOV, 1939) é necessário um desenvolvimento árduo e conjunto das áreas de sistemas robóticos e de inteligência artificial (IA), aliados ao desenvolvimento de hardware e ambientes para os desenvolvimentos destas tecnologias. Isso já vem ocorrendo como pode ser comprovado pela quantidade de pesquisas envolvendo essas áreas de forma interdisciplinar, onde estão sendo unidos os interesses da robótica com os da área de inteligência artificial, cujo objetivo tem sido o de tentar reproduzir características humanas (visão, fala, audição, locomoção, etc.). Isso torna possível o desenvolvimento de sistemas robóticos com características adequadas para aplicação doméstica (robôs domésticos) (PARK et al., 2008; CORADESHI et al., 2006; ANDROID, 2005; FALCONE et al., 2003; BREAZEL; SCASELLATI, 2002).

No entanto, além da junção das duas áreas, robótica e Inteligência Artificial (IA), o robô doméstico deve possuir características que permitam, além da capacidade de “tomar decisões”, a interação entre o sistema robótico e o usuário, possibilitando a realização de tarefas sem a supervisão do usuário (PARK et al., 2008; FALCONE et al., 2003; SCATENA, 2002).

Um exemplo de robô doméstico pode ser visto em Coradeschi et al. (2006), onde é apresentado um estudo sobre as características humanas, como cognição, reconhecimento de voz e comportamentos, que podem ser adicionados a humanóides para que eles interajam com os seres humanos, tornando-se máquinas mais humanizadas.

Um sistema como esse possui alta complexidade para seu desenvolvimento, tanto de hardware quanto de software. Contudo, uma vez implementado o sistema básico de interação do sistema robótico com o usuário, pode-se evoluir o ambiente de desenvolvimento de acordo com a necessidade de cada usuário, sendo o usuário responsável pelo desenvolvimento de novas tarefas para esse sistema (BRÄUNL, 2003; FALCONE et al., 2003).

Segundo Kramer e Scheutz (2007), um ambiente de desenvolvimento para robótica é extremamente importante para as pesquisas em robótica e em particular para o desenvolvimento de arquiteturas de robôs móveis.

Esses ambientes de desenvolvimento possuem características que podem ser discriminadas com base nas tecnologias existentes atualmente e outras ainda terão que ser aperfeiçoadas ao longo do tempo, como por exemplo, a interação do sistema robótico com o ser humano e a interface homem/robô (PARK et al., 2008; BRÄUNL, 2003; FALCONE et al., 2003; ZIELINSKI, 1991).

Na interface homem/robô deve-se levar em consideração a interface de comunicação do sistema robótico com o usuário, sendo de suma importância esta interface, uma vez que quanto maior a facilidade de interação com o sistema robótico, melhor será a aceitação deste sistema como produto final (CÔTÉ et al., 2004; FREUND et al., 2001;

ZIELINSKI, 1991). Esta interface deverá ser intuitiva e objetiva, como pode ser visto em Dario et al. (2001) que a descreve como “[...] sendo envolvente para o usuário, maximizando a utilização, auxiliando o usuário e sendo prazerosa [...]”.

Até o presente momento diversos ambientes de desenvolvimento de aplicações para robôs móveis tem sido desenvolvidos, porém todos eles dedicados a programadores experientes (PARK et al., 2008; DRUMWRIGHT, 2007; KRAMER; SCHEUTZ, 2007; ZIELINSKI; SZYNKIEWICS; WINIARSKI, 2005; CÔTÉ et al., 2004; FALCONE et al., 2003; ZIELINSKI, 1991).

O presente trabalho contribui com a área de desenvolvimento de interface de programação através da proposta de uma arquitetura de desenvolvimento de aplicações para robôs móveis que engloba interfaces de alto nível de abstração para a programação. Essa abstração esta vinculada a programadores experientes, assim como usuários com pouco conhecimento de programação.

1.2. *Objetivo*

A proposta deste trabalho baseou-se nos artigos de Drumwright (2007), Côté et al. (2004) e Falcone et al. (2003) que mencionam que um ambiente de desenvolvimento de aplicações para robôs móveis poderia conter recursos de forma a abranger desde usuários que possuam pouco conhecimento de programação até programadores experientes. Adicionou-se a essa proposta as idéias encontradas nos trabalhos Gerkey, Vaughan e Howard (2003), Scatena (2002) e Interesse e Distante (1988) relacionadas às características de um ambiente de desenvolvimento, que funcionaria como uma plataforma de tra-

balho, onde os usuários desenvolvem as aplicações para os robôs e o ambiente transforma essa aplicação em um código intermediário, que pode ser portado tanto para um simulador de robôs (*Player/Stage*), como para robôs reais.

Baseando-se então nos artigos mencionados, o presente trabalho propõe um ambiente de programação de aplicações para robôs móveis (*ADARM*) que contenha interfaces para programação de alto nível de abstração com blocos de tarefas, destinando-se a dois tipos de utilizadores: programadores experientes e usuários com pouco conhecimento em programação.

No caso de programadores experientes, a programação é realizada com a linguagem *Java2* associada à biblioteca *Java for Robotics (J4R)*. A biblioteca *J4R* é uma proposta deste trabalho e contém as ações (tarefas) utilizadas em robôs móveis

Para os usuários com pouco conhecimento em programação é disponibilizada uma interface de programação com um editor que cria aplicações através de blocos de ações (blocos de tarefas), baseando-se na estrutura denominada *Robot/Flow*. Essa ferramenta provê uma interface de programação rápida de forma a auxiliar também os programadores experientes que desejam implementar aplicações simples para testes.

Além do alto nível de abstração, o presente trabalho propõe também um módulo de vinculação das aplicações desenvolvidas para os diversos sistemas robóticos, sem que o usuário tenha que se preocupar com qual estrutura robótica ele irá utilizar.

Com isso, esse ambiente abrange um grau de liberdade maior no desenvolvimento de aplicações para sistemas robóticos, sem ficar preso aos fabricantes de dispositivos.

O que torna este trabalho uma contribuição relevante para a área de robôs móveis é o fato de unir em um único ambiente, as estruturas de uma linguagem de alto nível (*Java2*) e as tarefas da biblioteca *J4R*, juntamente com o editor de programas para usuários que possuem pouco conhecimento em programação e o suporte a diversos tipos de sistemas robóticos.

1.3. *Justificativa*

Tendo em vista todos os aspectos já analisados, o presente trabalho abrange como foco principal a área de robôs domésticos que é uma das áreas que vem se destacando atualmente (PARK et al., 2008 ; FALCONE et al., 2003).

A arquitetura de desenvolvimento de aplicações tal como descrita na seção 1.2 é justificada por diversos trabalhos podendo-se citar:

- ✓ Drumwright (2007), Kim e Kim (2004), Chaimowicz et al. (2003), Dario, Guglielmelli e Laschi (2001) que enfocam a necessidade de um ambiente de desenvolvimento para aplicações de robô. Em Kramer e Scheutz (2007) é mencionado que “ [...] os ambientes de desenvolvimento facilitam as pesquisas em robôs autônomos e auxilia os arquitetos a gerenciarem a complexidade dos robôs [...]”. Já em Shepherd (1993), “[...] os métodos de programação de robôs móveis são considerados uma barreira significativa para o crescimento de aplicações robóticas” e ainda ressalta “[...] que é necessário um ambiente de programação para robôs com alto nível de abstração [...]”;

- ✓ Ambiente de desenvolvimento, onde os trabalhos de Drumwright (2007), Kim e Kim (2004), Chaimowicz et al. (2003), Dario, Guglielmelli e Laschi (2001), enfocam a necessidade de um ambiente de desenvolvimento, como Kramer e Scheutz (2007) que diz “ [...] os ambientes de desenvolvimento facilitam as pesquisas em robôs autônomos e auxilia os arquitetos a gerenciar a complexidade dos robôs [...]” e segundo Shepherd (1993), “[...] a programação de robôs móveis é uma barreira significativa para o crescimento de aplicações robóticas” e ainda ressalta “[...] que é necessário um ambiente de programação para robôs, com alto nível de abstração [...]”;
- ✓ Com relação aos editores de programação para usuários iniciantes, Dario, Guglielmelli e Laschi (2001) ressaltam que “[...] para usuários do nível ‘iniciante’ a interface deve prover acessos a tarefas pré-definidas, já para o nível ‘avançado’ é fornecido uma interface que provê comandos mais complexos, tal como, tele-operação.”;
- ✓ Com relação ao módulo de programação em blocos, Kramer e Scheultz (2007) e Côté et al. (2004), fazem referência ao *RobotFlow* que representam blocos semelhantes à estrutura *FlowDesigner* (CÔTÉ et al., 2004), que são utilizados no presente trabalho para representar as tarefas programadas pela linguagem *Java2+J4R*.

Os itens acima citados são a base do ambiente de desenvolvimento do presente projeto, sendo esse ambiente complementado com as características seguintes:

- ✓ Editor de Programação Avançado (DARIO; GUGLIELMELLI; LASCHI, 2001; FREUND et al., 2001) que é utilizado por programadores avançados. Através da linguagem *Java2*, esse módulo permite a criação de rotinas avançadas para os sistemas robóticos, melhorando assim, a abstração da programação utilizada;
- ✓ Editor de Programação para Iniciantes (KRAMER, 2007; PARK et al., 2006; CÔTÉ et al., 2004; FALCONE et al., 2003) que pode ser utilizado sem dificuldades, por usuários com pouco conhecimento de programação. Como mencionado por Côté et al.(2004), “Um ambiente de programação com *RobotFlow*, provê um ambiente modular e gráfico, que auxilia a visualização e compreensão do que ocorre nos atuadores, sensores e controle do robô”. O editor utilizado no presente trabalho, baseou-se na estrutura de blocos denominada *FlowDesigner*, citada em Côté et al. (2004), representando uma contribuição para os demais trabalhos nesta área, utilizando em sua estrutura as funções de alta abstração que são desenvolvidas no editor de programação avançado, convertidas em tarefas e adicionadas pelo editor *Java for Robotics (J4R)*;

- ✓ Simplificação de Programação utilizando Tarefas que se baseiam nos códigos de programação para robôs móveis existentes (ALEOTTI; CASELLI; REGIANNI, 2004; CÔTÉ et al., 2004; SHEPHERD, 1993; INTERESSE; DISTANTE, 1988). Essas tarefas possuem a função de facilitar a programação de robôs móveis através de sub-rotinas de fácil manipulação gerando uma interface de programação para a linguagem do robô. Esse módulo do presente trabalho, contribui com as pesquisas relacionadas à criação de tarefas, codificando a linguagem *Java2* em tarefas para robôs, que são armazenadas em uma biblioteca denominada *J4R*;
- ✓ Conversores de Sistemas e Ambiente de Simulação que possui a capacidade de converter os programas vindos do ambiente de programação para o sistema robótico ou para o ambiente de simulação (ALEOTTI; CASELLI; REGIANNI, 2004; CÔTÉ et al., 2004; FREUND et al., 2001).

Resumindo, o presente trabalho contribui com a área de desenvolvimento de aplicações para robôs através da proposta de um ambiente de desenvolvimento, com a linguagem de programação com alto nível de abstração associado à criação de tarefas e com o editor de programação que utiliza a estrutura *RobotFlow* como estrutura principal de desenvolvimento.

1.4. *Organização da Tese*

Nesta seção é feita uma breve introdução sobre o conteúdo abordado em cada capítulo.

No capítulo 2 são apresentados os aspectos gerais sobre Robôs Móveis e algumas variações desse sistema, como por exemplo, Robôs Embarcados e Robôs Pessoais.

No capítulo 3 aborda os simuladores de robôs móveis encontrados na literatura atualmente e uma avaliação sobre esses sistemas com relação ao custo x benefício da utilização dos mesmos.

No capítulo 4, um resumo é apresentado sobre o avanço das pesquisas relacionadas aos sistemas robóticos domésticos e sobre os robôs pessoais.

No capítulo 5 é apresentada um resumo sobre as pesquisas relacionadas aos ambientes de desenvolvimento de aplicações para robôs móveis.

No capítulo 6 é apresentada a proposta para um ambiente de desenvolvimento de aplicações para robôs móveis.

Já no capítulo 7 é abordado a implementação do ambiente de desenvolvimento de aplicações para robôs móveis proposto no capítulo anterior.

No capítulo 8 é realizada a apresentação dos resultados e discussões obtidos nos testes do ambiente de desenvolvimento para robôs móveis.

No capítulo 9 é abordada as conclusões gerais e uma projeção de trabalhos futuros para o projeto atual.

Capítulo 2 - Robôs

2.1. *Introdução*

Desde os primórdios dos tempos, o homem vem aprimorando o conceito sobre um conjunto de máquinas, as quais desenvolveriam tarefas que os seres humanos não conseguiriam desenvolver, e tarefas repetitivas que não exigiam nenhum esforço intelectual. A esse conjunto de máquinas independentes denominou-se robô (SCATENA, 2002; KOTAY, 1998).

Com o desenvolvimento das pesquisas na área de robótica, objetivando principalmente sua aplicação na indústria, houve a necessidade do aperfeiçoamento dos robôs visando à melhoria da qualidade da produção industrial, além de mais eficiência, confiabilidade, redução de custos e da mão-de-obra. A essas vantagens adiciona-se ainda a capacidade dos robôs de realizarem tarefas para as quais os humanos teriam grandes dificuldades, como por exemplo, tarefas em ambientes perigosos e tarefas repetitivas, que exigem pouco ou nenhum esforço intelectual (SCATENA, 2002; INTERESSE; DISTANTE, 1988).

Seguindo a evolução dos robôs industriais, surgiu uma nova geração de sistemas robóticos, ampliando-se a sua atuação a níveis superiores ao da área industrial, de forma a permitir a interação entre os robôs e os seres humanos. Como exemplo pode-se citar os sistemas ALVINN, TESEO e FINALLE (SCATENA, 2002).

As próximas seções apresentam essa nova geração de sistemas robóticos, e sua evolução até os tempos atuais.

2.2. *Robôs Móveis*

Em uma nova etapa de evolução dos sistemas robóticos, diversos pesquisadores têm concentrado grandes esforços na construção de robôs móveis, introduzindo nestas máquinas as capacidades de mobilidade e autonomia para reagir adequadamente em um ambiente, o que abre um vasto campo de novas aplicações e conseqüentemente muitos desafios (SCATENA, 2002).

Os robôs móveis podem ser classificados conforme sua independência a usuários externos, da forma como segue:

- ✓ Robôs móveis controlados - são robôs capazes de se movimentarem através da interação com usuários externos, isto é, o robô só consegue se movimentar através de um controle remoto, através do qual o ser humano irá decidir qual a localização e qual tarefa de navegação será realizado pelo robô. Como exemplo pode-se citar o robô *throwable*, desenvolvido pela Universidade Carnegie Mellon em conjunto com o exército americano, com o objetivo de navegar por terrenos de alta periculosidade, enviando informações importantes para os soldados em campo. Este robô foi enviado ao Iraque para auxiliar os soldados americanos, sendo comandado pelos soldados através de um controle remoto (COHEN, 2008; REMOTE-CONTROLLED, 2004);

- ✓ Robôs Móveis autônomos - são robôs que possuem a capacidade de se movimentarem dentro de ambientes abertos ou fechados, sem a intervenção de nenhum usuário externo. O sistema robótico navega totalmente independente da intervenção externa, tomando as decisões necessárias para a navegação sem colisão (COHEN, 2008).

Dentre algumas possibilidades de uso de robôs móveis pode-se citar o transporte de materiais, combate a incêndios, desativação de explosivos, vigilância de armazéns, viagens submarinas, aeroespaciais e prestação de serviços, onde haja interação com pessoas para o auxílio de tarefas rotineiras.

2.3. *Robôs Móveis Embarcados*

Com a evolução dos sistemas digitais e com o desenvolvimento dos sistemas embarcados e sistemas de tempo crítico, os robôs móveis acompanharam essa evolução, passando a ser implementados com os sistemas embarcados, gerando assim um novo sistema robótico com capacidade de processamento elevada e baixo custo de desenvolvimento (BRÄUNL, 2003; SCATENA, 2002; KOTAY, 1998).

Muitos desses novos sistemas passaram a utilizar *FPGAs* em sua implementação, devido ao baixo custo desta tecnologia, associada à maleabilidade de programação e ao alto desempenho obtido com esses dispositivos (BRÄUNL, 2003; SCATENA, 2002; KOTAY, 1998).

Para uma melhor execução das tarefas, os robôs móveis podem ter a capacidade de interagir com o ambiente, modificando as suas estruturas internas. Esses tipos de robôs são denominados de robôs auto-reconfiguráveis.

Os robôs auto-reconfiguráveis são robôs compostos de muitos módulos físicos conectados, os quais podem modificar sua configuração estrutural para suportar múltiplas funcionalidades. Os módulos podem ser robôs completos capazes de executar algumas tarefas sem cooperação, ou podem ser unidades que são funcionais somente quando uma quantidade mínima dos módulos está presente. Os módulos criam estruturas unindo-se aos objetos estáticos ou unindo-se a si próprios. Os robôs auto-reconfiguráveis são mais versáteis, extensíveis e mais confiáveis do que os robôs de arquitetura fixa (SCATENA, 2002; KOTAY, 1998).

Aos robôs auto-reconfiguráveis pode-se atribuir as funções dos robôs móveis, de forma que este novo sistema robótico torne-se habilitado em navegação e exploração de ambientes, possuindo a capacidade de reestruturar suas funções internas (módulos), tornando-se assim mais eficazes (ROH et al., 2004; SCATENA, 2002; KOTAY, 1998).

O foco do presente trabalho é o desenvolvimento de um ambiente de desenvolvimento para robôs móveis, os quais possuirão a capacidade de se auto-reconfigurar conforme a necessidade do ambiente de interação. Para isso é necessário ter conhecimento das habilidades desse sistema robótico para uma melhor estruturação do ambiente de desenvolvimento. Algumas de suas habilidades são:

- ✓ O robô deve ter a capacidade de captar informações sobre o ambiente com o qual está interagindo;
- ✓ Deverá trabalhar meses ou até mesmo anos sem a intervenção humana;
- ✓ Navegar de um ponto A para um ponto B, sem assistência humana;
- ✓ Auto reparar-se sem assistência externa;
- ✓ Adaptar-se às circunstâncias do ambiente sem intervenção externa;
- ✓ Identificar situações de risco, tanto para o robô quanto para os seres humanos que estarão se inter-relacionando no mesmo ambiente (KOTAY, 1998).

Tendo como base essas habilidades, os robôs móveis autônomos baseados em sistemas reconfiguráveis são considerados robôs auto-suficientes, podendo interagir com ambientes estáticos e dinâmicos, ajustando-se a todas as situações possíveis sem que haja nenhuma intervenção (SCATENA, 2002; KOTAY, 1998).

2.4. *Robôs Domésticos (Domestic Robots) ou Robôs Pessoais (Personal Robots)*

Com o aperfeiçoamento dos robôs móveis autônomos, o próximo passo no desenvolvimento de robôs visa à produção de robôs capazes de realizar tarefas domésticas independentes do usuário. Estes novos modelos de robôs são denominados de *domestic robots* (FALCONE et al., 2003; ASIMOV, 1939). Como mencionado por Ruiz-del-Solar (2007) “Um robô pessoal é uma subclasse dos serviços de robôs móveis desenvolvidos para interagir com humanos como parceiros, provendo entretenimento, companhia e

interface de comunicação”, podemos visualizar a pretensão das pesquisas em relação a esse tipo de sistema robótico.

Porém, os primeiros modelos de *domestic robots* realizam apenas tarefas domésticas simples, como cortar grama (robô Roomba) e lembrar as tarefas pré-programadas aos idosos. Em modelos mais avançados estes robôs ajudarão pessoas idosas em tarefas de maior complexidade, como medir os sinais vitais e enviar ao médico, além de reconhecer faces e vozes de seus donos, tendo assim uma interface mais intuitiva e interativa com os seres humanos (PARK et al., 2008; FALCONE et al., 2003; DARIO; GUGLIELMELLI; LASCHI, 2001).

Um modelo desenvolvido pela empresa Mitsubishi-Heavy, denominado *Wakamuru*, é um robô doméstico que quando conectado a um telefone celular pode monitorar situações à distância, como roubos ou pessoas doentes em casa, além de que o dono do aparelho poderá agendar tarefas e pedir que o robô o acorde pela manhã e o lembre de eventos (MAZER et al., 1991).

Outro modelo interessante de robô pessoal é o robô Bender (RUIZ-DEL-SOLAR, 2007) que interaje com o ser humano através do reconhecimento de voz, além de realizar pesquisas na internet para o seu usuário. A pretensão para o robô Bender, segundo Ruiz-del-Solar (2007) é que “futuramente ele seja utilizado com propósitos educacionais”.

Tendo como base os exemplos apresentados acima, em um futuro não muito distante, os robôs domésticos além de reconhecerem comandos de voz mais avançados, possuirão capacidade de reprogramar as suas tarefas (ROH et al., 2004), e o proprietário do robô poderá criar novas tarefas que esses robôs poderão realizar (FALCONE et al., 2003; BISHOFF, 2000).

Os fatos apresentados anteriormente são comprovados através dos trabalhos sobre sistemas robóticos apresentados abaixo:

- ✓ Robô DRP I (Dinamically Reconfigurable Personal Robot) (ROH, 2004);
- ✓ Robô MOVAID (DARIO; GUGLIELMELLI; LASCHI, 2001);
- ✓ Robô Roomba (ANDROID, 2005);
- ✓ Robô Bender (RUIZ-DEL-SOLAR, 2007);
- ✓ Robô ASKA (IDO et al., 2002);
- ✓ Robô Lino (KRÖSE et al., 2003; VAN-BREEMEN, 2003);
- ✓ Robô The Care-O-Bot II (GRAF; HANS; SCHRAFT, 2006; FRAUNHOFER, 2003);
- ✓ Robô Milo (SALEMI et al., 2005);
- ✓ Robô Papero (NEC, 2005);
- ✓ Robô Khepera (ÉCOLE, 2005);
- ✓ Robô MATS (BALAGUER et al., 2006);
- ✓ Robô MowBot (FRIENDLY, 2008);

Alguns dos robôs apresentados anteriormente nesta seção serão descritos no capítulo 4.

2.5. *Considerações sobre o Capítulo*

Os robôs apresentados nesse capítulo comprovam que, cada vez mais, o ser humano está avançando na tecnologia de desenvolvimento de robôs a fim de criar sistemas mais inteligentes e independentes da manipulação humana, para que possam realizar tarefas cotidianas e caseiras. Com isso, os dispositivos robóticos poderão realizar tarefas como cuidar de pessoas idosas ou de servir de auxiliares em outras profissões, sendo que algumas versões iniciais desses sistemas robóticos já podem ser encontradas comercialmente, como os robôs dos modelos TMSUK IV (IDO et al., 2002) e Care-O-Bot II (GRAF; HANS; SCHRAFT, 2006; FRAUNHOFER, 2003).

O grande problema desses primeiros sistemas robóticos é o alto custo. Porém, com a evolução constante dos dispositivos de hardware e das pesquisas na área de software de controle e desenvolvimento dessa nova área, o preço desses robôs deverá chegar a um patamar onde todos poderão adquirir um sistema robótico doméstico.

Capítulo 3 - Simuladores de Robôs Móveis

3.1. Introdução

Com o desenvolvimento das pesquisas na produção de robôs móveis, surgiu a necessidade de validação dos aplicativos desenvolvidos para os sistemas robóticos, antes que esses aplicativos fossem utilizados nos sistemas reais. Isso se deve ao alto custo do hardware dos robôs e ao fato de que, os robôs possam sofrer não somente danos mecânicos mas sim danos estruturais também.

Devido aos problemas apresentados de hardware, foram desenvolvidos softwares de simulação que são utilizados para testar as aplicações do sistema robótico sem fazer o uso do sistema físico do robô.

O software de simulação representa as características do mundo real, como se os dados dos sensores fossem extraídos do próprio sistema robótico. Esses dados podem variar desde o tipo de piso, como chão escorregadio ou irregular, além das falhas de leitura de sonares e as interferências em outros sensores.

Dentre os simuladores já desenvolvidos, alguns modelos são apresentados a seguir:

- ✓ *Player/Stage* (KRAMER; SCHEULTZ, 2007; GERKEY; VAUGHAN; HOWARD, 2003);
- ✓ *Saphira - Aria* (KRAMER; SCHEULTZ, 2007; ACTIVMEDIA, 2005);
- ✓ *The Flat 2D Robot Simulator* (NGUYEN, 1999);

- ✓ MobotSim (MOBOTSOFT, 2005);
- ✓ Simulador WeBots(CYBERBOTICS, 2005);

3.2. *Simulador Player/Stage*

A função de um simulador é simplesmente a de replicar as características e imperfeições de um ambiente externo ou mesmo o funcionamento de um simples dispositivo, tudo isso dentro de um software. Porém, a complexidade de um simulador de robôs vai além da simulação do ambiente externo, atingindo o patamar de replicar a reação aplicada nesse ambiente quando o robô que será simulado interagir com esse ambiente externo. Devido a essas interações e à problemática de transformar o ambiente externo em cálculos matemáticos, nem todos os simuladores conseguem ter uma alta fidelidade de simulação do ambiente real (KRAMER; SCHEULTZ, 2007; GERKEY; VAUGHAN; HOWARD, 2003).

O simulador *Player/Stage*, além de que possuir uma alta fidelidade com o mundo real é gratuito e utilizado por diversas instituições ao redor do mundo, como *Robotics and Autonomous System Group Universidad del País Vasco, School of Computer Science and Information Technology, University of Nottingham, Intel Corporation, Laboratório de Computação Reconfigurável, Universidade de São Paulo*, os quais possuem altas qualificações e uma vasta experiência em sistemas robóticos (KRAMER; SCHEULTZ, 2007; GERKEY; VAUGHAN; HOWARD, 2003).

O simulador *Player/Stage* é constituído pelos seguintes módulos: o módulo *Player* o qual é a interface de robôs, o módulo *Stage* que é o simulador de robôs e o módulo *Gazebo* que é o simulador de robôs em 3D. Será apresentada a seguir, a descrição desses módulos e seus respectivos objetivos dentro do contexto da simulação (GERKEY; VAUGHAN; HOWARD, 2003).

3.2.1. Interface de dispositivos de robô

O módulo *Player* fornece uma interface para uma variedade de robôs e sensores. O modelo cliente/servidor do *Player* permite que programas de controle de robôs possam ser escritos em qualquer linguagem de programação e executados em qualquer computador que tenha uma conexão de rede com o robô. O *Player* suporta múltiplas e concorrentes conexões clientes para os dispositivos, criando novas possibilidades para controle e sensoramento colaborativo e distribuído. Suporta também uma grande variedade de robôs móveis e acessórios (KRAMER; SCHEULTZ, 2007; GERKEY; VAUGHAN; HOWARD, 2003).

3.2.2. Simulador de múltiplos de robôs

O módulo *Stage* simula uma população de robôs móveis, movendo e analisando um ambiente bidimensional (*bitmapped*). Vários modelos de sensores são fornecidos, incluindo sonar, *laser scanner*, câmera panorâmica com zoom, detecção de cores e odometria, como podem ser vistos nas figuras 3.1 e 3.2 (GERKEY; VAUGHAN; HOWARD, 2003).

Figura 3.1 - Stage em Simulação.

Fonte: (GERKEY; VAUGHAN; HOWARD, 2003)

Figura 3.2 - Stage Simulando o Ambiente de um Hospital.

Fonte: (GERKEY; VAUGHAN; HOWARD, 2003)

O *Stage* apresenta uma interface com o servidor *Player*, de forma que, para utilizar a aplicação do simulador no hardware do robô não é necessário nenhuma modificação da aplicação. O *Stage* foi projetado para suportar sistemas autônomos multi-agente, e fornece populações virtuais de dispositivos para o *Player* (GERKEY; VAUGHAN; HOWARD, 2003).

3.2.3. Simulador de multi-robôs 3D

O módulo Gazebo é um simulador multi-robô desenvolvido para ambiente externo (*outdoor*). Como o *Stage*, o Gazebo é capaz de simular uma população de robôs, sensores e objetos, mas com uma implementação de mundo tridimensional. Gera interação plausível entre os objetos físicos e os retornos dos sensores, sendo totalmente realístico. Apresenta uma interface padrão para o *Player* em adição à sua própria interface nativa. Os controladores escritos para o simulador *Stage* podem ser utilizados com o Gazebo, sem modificação (GERKEY; VAUGHAN; HOWARD, 2003).

Algumas imagens do Gazebo são apresentadas nas figuras 3.3 e 3.4.

Figura 3.3 – Gazebo no Ambiente em 3D .
Fonte: (GERKEY; VAUGHAN; HOWARD, 2003)

Figura 3.4 - Gazebo e a visualização do alcance do LaserFinder.
Fonte: (GERKEY; VAUGHAN; HOWARD, 2003)

3.3. *Simulador Flat 2D*

O Flat é um simulador 2D (duas dimensões) com alta qualidade para simular sensores do tipo sonar e laser *rangefinder*. Nesse simulador é possível definir os mundos onde o robô navegará, incluindo mundos com vários níveis, nos quais o robô poderá navegar de um andar para dentro de uma sala e vice-versa. Os sensores incluem modelos de ruídos opcionais, possuindo assim uma simulação real dos sensores reais (NGUYEN, 1999).

O funcionamento do simulador baseia-se na conexão do programa do sistema robótico conectado ao programa *Flat*, que controlará os movimentos do robô. O *Flat* simula os movimentos do robô no mundo virtual, e fornece um controlador para a entrada dos sensores. O controlador é responsável pela velocidade e direção do robô. Este simulador suporta a programação LISP e o programa cliente *Flat* fornece todos os comandos de baixo-nível necessários para controlar o robô, ler as entradas dos sensores e mostrar as saídas do sistema (NGUYEN, 1999).

O monitor *Flat* mostra as saídas dos sensores, o robô, o mundo, além de permitir que seu controlador (ou qualquer outro programa) desenhe na tela qualquer saída ou informação sobre o robô, como pode ser visto nas figuras 3.5 e 3.6 (NGUYEN, 1999).

Figura 3.5 - Ambiente de Simulação FLAT 2D.

Fonte: (NGUYEN, 1999)

Figura 3.6 - Controlador do Robô - FLAT 2D.

Fonte: (NGUYEN, 1999)

3.4. *Simulador MobotSim*

O *MobotSim* é um software de simulação em duas dimensões (2D) para robôs móveis. Esse software provê uma interface gráfica, que representa o ambiente no qual se pode criar, configurar, editar robôs facilmente (MOBOTSOFT, 2005). Este simulador fornece um editor baseado na linguagem BASIC, através do qual o usuário pode escre-

ver macros, fazendo uso de funções específicas para capturar informações sobre as coordenadas do robô e dados dos sensores, além de configurar a velocidade e os dados para a navegação do robô, além de utilizar a linguagem de programação BASIC (MOBOTSOFT, 2005).

O simulador *MobotSim* foi desenvolvido pensando nos pesquisadores, estudantes e pessoas que desejam projetar, testar e simular robôs móveis e tópicos relacionados a pesquisas de técnicas de navegação autônoma, desvio de obstáculos, inteligência artificial, integração de sensores, etc. (MOBOTSOFT, 2005).

Esse simulador pode ser visto na figura 3.7 e algumas de suas características são:

- ✓ Número de robôs e obstáculos ilimitados;
- ✓ Formas severas de obstáculos (linha, retângulo, arco, elipse, etc) e formas feitas à mão livre;
- ✓ Simulação de Sensores de varredura;
- ✓ Configuração flexível do *Mobots*, como o diâmetro da plataforma, diâmetro das rodas, distância entre as rodas, número de sensores, entre outros;
- ✓ Configuração das medições dos sensores, como cone radial, porcentagem de perda de leitura, entre outros;
- ✓ Fácil integração com controles ActiveX e arquivos dll. Suporta lógica fuzzy, algoritmos genéticos, redes neurais, etc. (MOBOTSOFT, 2005)

*Figura 3.7 - Ambiente de Programação - MobotSim.
Fonte: (MOBOTSOFT, 2005)*

3.5. Simulador WeBots

O *WeBots* é um software de simulação de robôs móveis profissional, contendo uma ferramenta de prototipação rápida, que permite ao usuário criar mundos virtuais em três dimensões (3D) com propriedades físicas, como a massa de repartição, junções, coeficientes de fricção, etc. O usuário pode adicionar um simples objeto inerte ou objetos ativos chamados robôs móveis. Estes robôs podem ter diferentes esquemas de locomoção (rodas, pernas, asas). Porém, ele é equipado com um grande número de dispositivos, como sensores de distância, rodas motorizadas, câmeras, servos, sensores de to-

que, emissores, receptores, etc. Finalmente o usuário pode programar cada robô individualmente para mostrar o comportamento desejado (CYBERBOTICS, 2005).

O *WeBots* contém um vasto número de modelos de robôs e exemplos de programas de controle que podem ajudar usuários iniciantes. Além disso, ele conta também com um grande número de interfaces para robôs móveis reais, sendo o comportamento dessas interfaces semelhante aos componentes dos robôs reais, sendo possível transferir um programa de controle para o robô real sem que haja a necessidade de modificação ou adaptação do programa de controle (CYBERBOTICS, 2005).

Algumas das interfaces suportadas pelo *WeBots* são mostradas nas imagens das figuras 3.8, 3.9, 3.10, 3.11 e 3.12 (CYBERBOTICS, 2005).

Figura 3.8 – WeBots - Sony Aibo e LEGO Mindstorms.
Fonte: (CYBERBOTICS, 2005)

*Figura 3.9 - Judô Robot e Humanoid robot.
Fonte: (CYBERBOTICS, 2005)*

*Figura 3.10 - Pioneer 2 robô e Simulação partida de Futebol com detalhes realísticos
Fonte: (CYBERBOTICS, 2005)*

*Figura 3.2 - Khepera robot e Khepera II robot.
Fonte: (CYBERBOTICS, 2005)*

Figura 3.3 - Koala robot e Magellan robot.

Fonte: (CYBERBOTICS, 2005)

3.6. *Simulador Saphira*

O sistema *Saphira* é uma arquitetura desenvolvida para controle de robôs móveis, sendo também uma ferramenta para desenvolvimento de ambientes para vários tipos de plataforma de robôs. Ele foi originalmente desenvolvido para as pesquisas do robô Flakey 2 do SRI Internacional, que é um instituto sem fins lucrativos, tendo, depois de 10 anos, sua arquitetura completamente modificada, passando a suportar uma vasta variedade de aplicações para a programação de robôs móveis e em diversas plataformas (ACTIVMEDIA, 2005; SCATENA, 2002).

O *Saphira* é uma integração de um conjunto de rotinas de comunicação que controla um robô através de um computador hospedeiro (*host*). Este sistema é projetado para facilitar a definição das aplicações do robô, o qual está associado ao programa cliente. Por causa disto, esta arquitetura é uma arquitetura aberta onde os usuários podem escrever seu próprio sistema de controle de robôs móveis e reutilizar as vantagens das micro-tarefas e propriedades do refletor de estados, o qual gerencia as informações in-

ternas do robô (sensores, motores e garras) (ACTIVMEDIA, 2005; SCATENA, 2002).

A forma de operação do *Saphira* é através de uma plataforma cliente/servidor, sendo que suas bibliotecas são rotinas para a construção de clientes, realizando assim todo o trabalho de comunicação e gerenciamento dos dados para o robô servidor (ACTIVMEDIA, 2005; SCATENA, 2002).

O cliente *Saphira* conecta-se a um robô servidor, com os componentes básicos para o robô obter as informações necessárias do ambiente e navegar, ou seja, as informações da posição dos *encoders*, a leitura dos sensores e manipulação do motor e das rodas. Já o servidor manipula as rotinas de mais baixo nível, como enviar e responder aos comandos do cliente, através de um protocolo de comunicação, gerenciar a direção e os sensores do robô (ACTIVMEDIA, 2005; SCATENA, 2002).

Este software suporta funções de alto nível para o controle do robô e interpretação dos sensores, incluindo o sistema de planejamento reativo, navegação baseada em mapa, sistema de registro e controle de comportamento *fuzzy* (ACTIVMEDIA, 2005; SCATENA, 2002).

O *Saphira* possui um software de simulação do robô físico e de seu ambiente. Estas características foram incluídas para melhorar a análise das aplicações em um computador, sem ter que utilizar o *hardware* do robô para validar os sistemas (ACTIVMEDIA, 2005; SCATENA, 2002).

Este simulador implementa as principais características de um ambiente real, como distorções para as leituras dos sensores e dos *encoders* das rodas. Com isso a interface de comunicação é a mesma do robô físico, fazendo com que não seja necessária a modificação ou ajuste do software para funcionar no robô (ACTIVMEDIA, 2005; SCATENA, 2002).

Este simulador constrói os modelos do mundo real ou do seu ambiente em modelos de duas dimensões, sendo estes modelos chamados de mundos. Os modelos de mundo são abstrações de um mundo real, com segmentos lineares representando as superfícies verticais dos corredores, passagens das portas e os objetos presentes nele (ACTIVMEDIA, 2005; SCATENA, 2002).

Na versão 6.x do *Saphira* foi incorporada uma linguagem semelhante à linguagem C, chamada de COLBERT, a qual é utilizada para desenvolver programas de controle de robô. Com o COLBERT, os usuários podem desenvolver e depurar procedimentos de controle, denominados de *activities*. Os *activities* possuem semânticas de estado-finito, as quais conseguem produzir representações procedimentais de conhecimentos de uma sequência de ações. *Activities* podem iniciar e parar ações diretas do robô, comportamento de baixo nível, e outros *activities*. Os *activities* são utilizados pelo executor de COLBERT, o qual suporta o processamento concorrente entre diversos *activities* (Figura 3.13) (ACTIVMEDIA, 2005; SCATENA, 2002).

Figura 3.43 - Arquitetura do Saphira.
Fonte: (ACTIVMEDIA, 2005; SCATENA, 2002)

O COLBERT vem com um ambiente de avaliação em tempo de execução, através do qual os usuários podem visualizar a interação de seus programas, editá-los e executá-los, e agregá-los em um código padrão de linguagem C. Muitos usuários programam interativamente com o COLBERT, o qual pode desenvolver todas as funções da *API* (*Application Programming Interface*) do *Saphira* avaliadas no ambiente (ACTIVMEDIA, 2005; SCATENA, 2002).

O *Saphira* possui um sistema operacional básico para o controle do robô. A Figura 3.14 mostra a estrutura de uma aplicação típica no *Saphira*. As rotinas do *Saphira* estão em azul turquesa e as rotinas do usuário em amarelo. Todas as rotinas do *Saphira* são executadas a cada ciclo *Saphira*, ou seja, a cada 100ms, sendo que estas rotinas são invocadas pelo Sistema Operacional (SO) do *Saphira*, que constrói estas micro-tarefas.

Esse SO também manipula os pacotes de comunicação com o robô, a construção do estado do robô, entre outras tarefas mais complexas como navegação e interpretação de sensores (ACTIVMEDIA, 2005; SCATENA, 2002).

A Figura 3.15 mostra o funcionamento da conexão do robô com o cliente *Saphira*.

Figura 3.5 - Conexão do Cliente Saphira
Fonte: (ACTIVMEDIA, 2005; SCATENA, 2002)

Recentemente a empresa *activmedia robotics* aperfeiçoou o software simulador *Saphira*, resultando uma nova versão denominada de ARIA (ActivMedia Robotics Interface Application). Este novo simulador implementa com mais qualidade a programação orientada a objetos, utilizando a linguagem C++. O simulador ARIA comunica-se com o robô, da mesma forma que o *Saphira*, via comunicação cliente/servidor, usando uma conexão serial ou conexão TCP/IP. Além da comunicação, o simulador ARIA é capaz de executar múltiplas ou simples tarefas (Figura 3.15) (ACTIVMEDIA, 2005).

Algumas das principais características do ARIA são:

- ✓ Controle dinâmico de velocidade, de leituras e leituras relativas dos sensores;
- ✓ Alta flexibilidade;
- ✓ Facilidade de programar as ações para o sistema robótico.

Além destas características, o simulador ARIA implementa ainda uma vasta gama de sensores e dispositivos de varredura de objetos (laser *rangefinder*) e câmeras, para um melhor aproveitamento do sistema robótico, mantendo a arquitetura principal do simulador *Saphira* (KRAMER; SCHEULTZ, 2007; ACTIVMEDIA, 2005).

Figura 3.6 - Diagrama da estrutura do ARIA.
 Fonte: KRAMER; SCHEULTZ, 2007; ACTIVMEDIA, 2005

3.7. Considerações sobre o Capítulo

A tabela 1 resume as principais características de cada simulador apresentado nas seções anteriores.

Os simuladores de interesse para esse trabalho são os que apresentam o diferencial de serem gratuitos e possuírem código aberto, destacando-se nessa categoria os simuladores *MOBOTSIM* e *Player/Stage*.

O simulador escolhido para realizar as validações do ambiente de desenvolvimento de aplicações para robôs móveis é o *Player/Stage*. Esse simulador foi escolhido pelo fato de ser muito utilizado nas pesquisas com robôs móveis em universidades de todo mundo, além de possuir credibilidade e suporte para o desenvolvimento de novas características que representam sensores ainda não incluídos no simulador *Player/Stage*.

Tabela 3.1 Comparação entre os softwares de simulação de robôs móveis

Simulador	Custo	Open Source	SO	Client/ Server	Mapa	Real-time	Fidelidade Hardware
<i>Player/Stage</i>	Gratuito	Sim	L	Sim	2D/3D	Sim	Sim
Flat2D	Gratuito	Não	L/J	Sim	2D/3D	Sim	Sim
MobotSim	\$290	Não	W	Sim	2D	Sim	Sim
MobotSim	Gratuito	Sim	L	Sim	2D/3D	Sim	Sim
Khepera	\$1.860	Não	W/L	Sim	2D/3D	Sim	Sim
Aria	\$2.000	Não	W/L	Sim	2D/3D	Sim	Sim
Sysquake	\$2.500	Sim	W/M	Não	2D	Sim	Sim

Legenda: J: Java2 - L: Linux - M: Machintosh - W: Windows

Esse simulador possui também uma ótima estabilidade e uma alta fidelidade com o mundo real, simulando as problemáticas do mundo real nos dispositivos do robô móvel.

Pelo fato de ser distribuído gratuitamente e de possuir a contribuição de toda a comunidade de robótica, este simulador possui uma flexibilidade de desenvolvimento tendo um aumento considerável no desenvolvimento de novas aplicações ou *drivers* para os dispositivos mais utilizados. Esta flexibilidade é suportada devido ao fato do simulador utilizar linguagens de programação de grande uso no mundo da informática que são as linguagens C, C++ e Java. Dessa forma qualquer programador de robôs móveis pode criar suas situações ou até mesmo descrever as características do mundo para o simulador.

Capítulo 4 - Estado da Arte em Robótica Doméstica ou Robôs Pessoais

4.1. Introdução

Com o grande avanço na área tecnológica, principalmente na área de hardware, que tem como grande destaque os sistemas embarcados, os sistemas robóticos vem se tornando cada vez mais parte do nosso dia-a-dia. Isso se deve à inclusão de novas funcionalidades, desenvolvidas por pesquisadores, nos sistemas robóticos móveis, criando assim os conhecidos *Domestic Robots* (PARK et al., 2008, FALCONE et al., 2003).

Como já mencionado nos capítulos anteriores, os *Domestic Robots* possuem diversas aplicações que fazem parte do nosso dia-a-dia, que vão desde um simples cortador de grama autônomo (VR-4000 da LG) (LG, 2007), até tarefas mais complexas como a de vigilância, onde o robô percorre a casa o tempo todo ou mesmo um *Robot Care* (FRAUNHOFER, 2003) que cuida de pessoas idosas, executando tarefas tais como lembrar os horários dos medicamentos e especificar as doses das receitas, como também monitorar os sinais vitais de uma pessoa, e caso ela necessite de cuidados especiais o sistema robótico entra em contato com um médico através de uma ligação telefônica (Care-o-Bot II, TMSUK-IV e o ASKA) (ANDROID, 2005; GRAF; HANS; SCHRAFT, 2006; BREAZEAL; SCASSELLATI, 2002; IDO et al., 2002).

Uma das principais características dos sistemas robóticos apresentados acima é a sua forma de programação, ou seja, a forma de como serão criadas as aplicações que serão utilizadas nos robôs. Na maioria dos sistemas robóticos essa forma de programação é realizada através de um ambiente de programação ou através de uma linguagem de programação específica para o robô (GOROSTIZA et al., 2006; DOMÍNGUES-BRITO et al., 2005).

A forma de programar o sistema robótico é de suma importância para os desenvolvedores de aplicações para robôs, pois quanto maior a facilidade de programar, maior será a produtividade no desenvolvimento da aplicação (DARIO, GUGLIELMELLI e LASCHI, 2001; BROOKS, 1999).

Além da importância para os programadores, o ambiente de desenvolvimento tem um papel fundamental quando o foco está nos sistemas robóticos domésticos, pois os usuários destes sistemas, muitas vezes, não possuem conhecimento de programação e têm pouco conhecimento na área de robótica (BALAGUER et al., 2006; DOMÍNGUES-BRITO et al., 2005; BROOKS, 1999).

Porém, para que esses ambientes sejam viáveis, os sistemas robóticos deverão ser capazes de receber diversos tipos de programação, ou a realização de diversas tarefas. Essas características são encontradas em alguns robôs como o robô *TMSUK IV* (IDO et al., 2002), *MATS* (BALAGUER et al., 2006), *MILO* (SALEMI et al., 2005), *Bender* (RUIZ-DEL-SOLAR, 2007), sendo a quantidade de robôs com essas características vem crescendo, devido ao fato das pesquisas na área de robôs domésticos serem muito recentes e estarem sofrendo modificações constantes e de forma rápida (RUIZ-DEL-SOLAR,

2007; DOMÍNGUES-BRITO et al., 2005).

Alguns dos sistemas robóticos e seus ambientes encontrados serão apresentados nas seções a seguir.

4.2. *Exemplos de Robôs Domésticos*

Visando integrar a tecnologia atual com os conceitos em robôs móveis, algumas empresas estão iniciando uma nova era na robótica criando os chamados *Personal Robots* (FALCONE et al., 2003; GRAEFE; BISHOFF, 2002; BISHOFF, 2000).

Alguns desses *Personal Robots* já estão disponíveis em alguns modelos, os quais realizam nessa fase inicial de criação, tarefas simples ou tarefas únicas, que já vem configurada de fábrica, como no exemplo do robô *MowBot* da empresa *Friendly Machines*, desenvolvido para cortar grama do jardim sem que haja qualquer manipulação humana.

Uma tendência para os *Personal Robots* é a criação de sistemas de robôs móveis com a forma humana, sendo esses robôs chamados de robôs humanóides (RUIZ-DELSOLAR, 2007; GOROSTIZA et al., 2006; BREAZEAL; SCASSELLATI, 2002; ATKESON, 2000). O principal objetivo no desenvolvimento de robôs humanóides é a criação de um sistema robótico que tenha as mesmas capacidades de movimentação e realização de tarefas dos seres humanos, além de prover uma fisionomia mais aceitável para a interação com o ser humano (PARK et al., 2008). Em alguns casos esses sistemas robóticos são utilizados na tentativa de compreender a cognição do cérebro humano e em outros

casos ele é utilizado para a representação das sensações humanas como sorrir, reagir ao toque, demonstrar alegria, entre outras características (GOROSTIZA et al., 2006; SALICHS et al., 2006).

Um exemplo desse sistema pode ser visto através do robô *Maggie* que possui a capacidade de interação social e que segundo Salichs et al. (2006)

[...] o robô será habilitado a desenvolver muitas tarefas da sociedade humana. Estas tarefas podem incluir, sem limitações, manipulação de vários deveres de casa, provendo cuidados médicos para pessoas idosas, supervisionando pessoas com problemas motores ou cognitivos, entretendo educacionalmente, informando direções em locais públicos, etc. [...].

Essa interação social é possível devido aos sensores presentes no robô, como por exemplo, sensores de toque, infra-vermelhos, câmeras entre outros dispositivos, que além de captar todas as informações em volta do robô, envia essas informações via internet para um computador que as processa e retorna o comportamento que o robô assumirá diante de tais informações (GOROSTIZA et al., 2006; SALICHS et al., 2006).

O desenvolvimento e as pesquisas sobre os robôs humanóides estão tão avançados que já existe uma conferência internacional organizada sobre esta modalidade e há previsão para 2015, da primeira disputa de futebol entre homens e humanóides (BOTTAZZI; FONSECA, 2005).

Outro modelo de robô que atenderia as exigências dos robôs domésticos são os robôs auto-reconfiguráveis, que possuem a capacidade de reprogramar suas tarefas sem que haja intervenção humana, ou a parada do sistema para a troca de tarefas. Um modelo de robô com essas características é o DRP I, que é estruturado modularmente, permitindo assim, um melhor interfaceamento tanto de hardware quanto software (ROH et al., 2004).

Tendo em vista os exemplos citados acima, torna-se possível comprovar o interesse nesse novo ramo de pesquisa e desenvolvimento, podendo-se citar diversos outros *Domestic Robots*:

- ✓ Robô Roomba, apresentado na figura 4.1, é utilizado como aspirador de pó autônomo. Esse robô tem três tipos de limpeza, os quais são utilizados duas escovas rotativas e um aspirador que remove as sujeiras do chão enquanto as escovas limpam os cantos e paredes. O roomba ainda possui sensores infravermelho que informa sua posição atual, evita que ele caia das escadas e informa a localização da sujeira para o sistema. Além disso, o próprio robô retorna a sua base quando termina a limpeza ou quando sua bateria está fraca (FORLIZZI; DISALVO, 2006; ANDROID, 2005);

Figura 4.1 - Roomba.

Fonte: (FORLIZZI; DISALVO, 2006)

- ✓ Robô Bender apresentado na figura 4.2, foi desenvolvido para auxiliar as pessoas. Esse sistema robótico interage com os humanos através do reconhecimento de gestos e voz, tendo como principal característica o reconhecimento de um usuário (perfil) que já interagiu com o sistema. Esse robô é interligado com a internet e fornece ao usuário conexão visual com a rede através de um monitor que tem acesso aos principais sites (RUIZ-DEL-SOLAR, 2007). Segundo Ruiz-del-Solar(2007) “Devido a grande interação com o usuário e com a internet esse sistema robótico poderá ser utilizado fins educacionais [...]”;

Figura 4.2 - Robô Bender.
Fonte: (RUIZ-DEL-SOLAR, 2007)

- ✓ Robô Maggie desenvolvido com o objetivo de interagir com os seres humanos e possui alto grau de autonomia. Essa interação é realizada através de diversos sensores acoplados ao sistema, como sensores de toque, câmeras e reconhecimento de voz. Com isso, o robô Maggie, apresentado na figura 4.3, possui algumas habilidades, como saudação, reconhecimento de face e identificação de usuário, diálogo, interação audiovisual e expressão visual não-verbal (SALICHS et al., 2006). Devido a todas essas características, o robô Maggie, segundo Salichs et al. (2006) “[...] pode ser considerado como parceiro de um humano e não meramente uma ferramenta [...]”;

Figura 4.3 - Robô Maggie
Fonte: (GOROSTIZA et al., 2006)

- ✓ Robô MATS é um sistema robótico similar a um braço mecânico que segundo Balaguer et al. (2006) “tem a função de auxiliar pessoas deficientes, manipulando pratos, garrafas e elementos existentes em uma cozinha, além escovar os dentes, barbear, vestir e manipular coisas em diversos ambientes [...]”. Esse robô ao realizar as tarefas acima pode auxiliar uma pessoa tetraplégica nas tarefas cotidianas, como pode ser visto na figura 4.4. Além disso, o robô pode ser acoplado em uma cadeira de rodas ou a um suporte mecânico com rodas que permite que o sistema todo possa se locomover independentemente, aumentando assim a versatilidade do robô (SALICHS et al., 2006);

Figura 4.4 - Robô MAT
Fonte: (BALAGUER et al., 2006)

- ✓ Robô ASKA é um robô doméstico atendente, que possui a capacidade de reconhecer a face das pessoas e interagir com os usuários através do reconhecimento de voz. O ASKA possui 27 graus de liberdade e pode ser controlado de diversas formas, sendo uma delas o controle através de uma rede de telefone móvel público com 64kbps de velocidade de transmissão de dados, como a rede PHS existente no Japão, que transmitirá o comando de voz através desta rede (IDO et al., 2002). Esse sistema robótico e sua estrutura podem ser visualizados através da figura 4.5;

Figura 4.5 – Robô ASKA e sua Estrutura

Fonte: (IDO et al., 2002)

- ✓ Robô The Care-O-Bot II, apresentado nas figuras 4.6 e 4.7, é um robô desenvolvido para cuidar de pessoas doentes ou idosas. Ele executa trabalhos diários como arrumar a mesa, operar com microondas e realizar tarefas de limpeza. Além das tarefas diárias o robô controla a infra-estrutura da casa, como por exemplo, controle das luzes, ar condicionado, janelas, porta, alarmes, etc. O robô tem também a utilidade de cuidar de pessoas, através de um módulo de controle de sinais vitais das pessoas, juntamente com a comunicação total com o corpo médico (GRAF; HANS; SCHRAFT, 2006; FRAUNHOFER, 2003);

Care-O-Bot – Guia

Care-O-Bot – Assistente Médico

Figura 4.6 - The Care-O-Bot II – Utilidades

Fonte: (GRAF; HANS; SCHRAFT, 2006; FRAUNHOFER, 2003)

Figura 4.7 - The Care-O-Bot II – Modelo e Arquitetura.

Fonte: (GRAF; HANS; SCHRAFT, 2006; FRAUNHOFER, 2003)

- ✓ Robô *MowBot* (figura 4.8) foi desenvolvido para realizar a função de cortador de gramas. Para o seu funcionamento é necessária apenas a configuração do método de cortar a grama que será utilizado. Possui sensores que detectam os perímetros (cantos) do jardim, fazendo assim com que o robô não saia da região onde deverá realizar o seu trabalho. Para a tarefa de cortar grama, ele utiliza um motor de 5.5 HP movido à gasolina. (FRIENDLY, 2008);

Figura 4.8 - Mowbot.
Fonte: (FRIENDLY, 2008)

Dentre todos os robôs mencionados até aqui, um robô em especial merece uma atenção diferenciada. Esse robô é o *Papero*, desenvolvido pelos japoneses para servir de companheiro caseiro (SATO; SUGIYAMA; OHNAKA, 2006; NEC, 2005).

O *Papero*, robô desenvolvido pela NEC, foi projetado para auxiliar em tarefas domésticas, além de ser utilizado como auxiliar nos cuidados com crianças, idosos ou pessoas com mobilidade limitada. O *Papero* tem um visual bastante diferente, mais parecendo um brinquedo do que um robô, dotado de sensores e reconhecimento de voz e imagem. Ele também não se move por pernas, mas através de três rodas, numa estrutura semelhante ao *R2D2*.

O robô *Papero* destaca-se dos demais, através da interface com o ser humano, pois possui sensores que permitem que o robô ouça, saiba de onde vem o som e reconheça pessoas. Com duas câmeras CCD que permitem ao robô reconhecer pessoas que se aproximem dele, através do rastreamento de rostos na imagem captada pela câmera. A mesma câmera serve para identificar obstáculos e mesmo gravar cenas como uma filmadora comum. O sentido de audição emprega quatro microfones. Três deles servem para detectar a origem do som, através da comparação das diferenças de tempo que o som leva para atingir cada um deles. O quarto microfone é utilizado para reconhecimento de voz (SATO; SUGIYAMA; OHNAKA, 2006; NEC, 2005).

Atualmente o *Papero* já reconhece 650 palavras, mas pode pronunciar mais de 3.000. Não há limitações técnicas para esses números que podem crescer rapidamente devido a interação com ele.

Os proprietários do *Papero* poderão obter arquivos, via Internet, para incrementar seu vocabulário. Outros sensores, de toque e proximidade, tornam o robô capaz de detectar o usuário quando alguém toca sua cabeça ou mesmo levanta-o e tente carregá-lo. Ao ser tocado, o *Papero* identifica a pessoa por meio de sua câmera e faz saudações diferenciadas conforme o interlocutor. Ele conversará de forma diferente quando se tratar de uma criança ou de um adulto. Cinco sensores ultra-sônicos tornam o *Papero* capaz de circular livremente pela casa, sem o risco de colisões com móveis ou pessoas. Não haverá também o risco de que ele fique encalhado em algum desnível da casa ou caia escada abaixo.

Atendendo a comandos de voz, o *Papero* pode acionar equipamentos por controle remoto, como TVs, aparelhos de som, além de responder data e hora e fazer coisas divertidas como dançar ou jogar vídeo-game (SATO; SUGIYAMA; OHNAKA, 2006; NEC, 2005).

Para facilitar e reduzir o custo da produção, o *Papero* funciona baseado na arquitetura do microcomputador PC. As plataformas distribuídas são baseados num Pentium 4 Mobile – 1.6 Ghz, executado no Windows XP e pode ser visualizado através da figura 4.9 (SATO; SUGIYAMA; OHNAKA, 2006; NEC, 2005).

Figura 4.9 - Papero.

Fonte: (SATO; SUGIYAMA; OHNAKA, 2006; NEC, 2005)

4.3. Considerações Finais do Capítulo

Nesse capítulo foram apresentados diferentes tipos de *Domestic Robots*, ilustrando dessa forma as interfaces de interação homem/robô, juntamente com as características física e programáveis dos sistemas robóticos.

Capítulo 5 - Estado da Arte de Ambientes de Desenvolvimento para Sistemas Robóticos

5.1. *Introdução*

Nesse capítulo são descritos os ambientes de desenvolvimento para sistemas robóticos mais relevantes para este trabalho. Esses ambientes são utilizados para agilizar o desenvolvimento de aplicações bem como para simplificar a forma de programar o sistema robótico (BOTTAZZI; FONSECA, 2005; DOMÍNGUES-BRITO et al., 2005).

5.2. *Principais Ambientes de Desenvolvimento para Sistemas Robóticos*

Para que seja possível a implementação de um *framework* é necessário a especificação formal e a metodologia de implementação do ambiente. Para uma melhor compreensão, os *frameworks* são geradores de aplicações e devem conter os seguintes componentes (ZIELINSKI; SZYNKIEWICS; WINIARSKI, 2005; ZIELINSKI, 1999; MAZER et al., 1991):

- ✓ Biblioteca dos módulos de software que contém os blocos de construção com o qual o sistema robótico é programado;
- ✓ Metodologia para o desenvolvimento de novos módulos, os quais podem ser adicionados nas bibliotecas citadas anteriormente;

Os principais trabalhos relacionados a ambiente de desenvolvimento para aplicações robóticas e que atendem aos requisitos acima citados, serão apresentados a seguir.

O ambiente de desenvolvimento robótico *Pyro* (PYRO, 2007) é um dos mais recentes e destaca-se por ser compatível com diversos robôs móveis e simuladores de robôs móveis. O *Pyro* é portátil e abrange todas as camadas de desenvolvimento necessárias ao funcionamento de um sistema robótico. Além disso, ele é distribuído gratuitamente (*OpenSource*) (KRAMER; SCHEULTZ, 2007; PYRO, 2007).

Esse ambiente utiliza arquivos de configurações, os quais contém as características dos sistemas robóticos e definem os comandos e ações que o ambiente usa como *API* (PYRO, 2007).

Juntamente com as definições dos arquivos de configuração, o *Pyro* contém suporte a diversos módulos, desde redes neurais artificiais e algoritmos evolutivos, até a simulação de um cérebro baseado em Inteligência Artificial (IA). Essas características são utilizadas como ferramentas educacionais, provendo ao *Pyro* uma grande variação de aplicações para robôs móveis, facilitando o trabalho do desenvolvedor do robô móvel (PYRO, 2007).

A figura 5.1 apresenta a interface do *Pyro* executando a interface do robô *Pioneer*, mostrando a leitura dos sensores (sonares) do robô enquanto este navega pelo ambiente.

*Figura 5.1 - Interface do Pyro.
Fonte: (PYRO, 2007).*

Na figura 5.2 é apresentado o funcionamento de uma aplicação desenvolvida no ambiente *Pyro* e validado no simulador 3D denominado *Gazebo*. Todas as informações da aplicação são simuladas pelo *Pyro* e os resultados dessa simulação são apresentados pelo cliente *Gazebo* (PYRO, 2007).

Figura 5.2 - Gazebo simulando uma aplicação desenvolvido no Pyro.

Fonte: (PYRO, 2007).

O ambiente de desenvolvimento *ROCI (Remote Objects Control Interface)*, consiste em uma plataforma de desenvolvimento que suporta projetos de aplicações distribuídas e programação com componentes reutilizáveis.

O *ROCI* é descrito como um *framework* para o desenvolvimento de aplicações para robôs móveis e sistemas de automação de robôs industriais.

O processo de programação desse *framework* baseia-se em blocos de estruturação de aplicações, sendo esses blocos a especificação das tarefas. Cada tarefa é composta de módulos, que são executados em nós. Esse processo é ilustrado na figura 5.3 (COWLEY; CHAIMOWICZ; CAMILLIO, 2006; CHAIMOWICZ et al., 2003).

Figura 5.3 - Arquitetura ROCL.

Fonte: (COWLEY; CHAIMOWICZ; CAMILLIO, 2006; CHAIMOWICZ et al., 2003).

Além da estruturação das tarefas em módulos, o *framework* utiliza o paradigma orientado a objetos e tem a codificação dos seus módulos nas linguagens C, C++, C#, Java, VB, entre outras. Com isso, a programação das aplicações utiliza uma linguagem de alto nível, estimulando os programadores a utilizar o *framework* (COWLEY; CHAIMOWICZ; CAMILLIO, 2006; CHAIMOWICS et al., 2003).

No trabalho de Parker et al. (2006) é implementado um *framework* que utiliza blocos interconectáveis, de forma que esses blocos agilizam a criação de uma aplicação para um sistema robótico, devido à característica visual apresentada pelos blocos. Esses blocos são configurados conforme a necessidade do programador, podendo ser visualizado na figura 5.4.

Figura 5.4 - Partes do Sistema de Programação do Robô.
 Fonte: (PARK et al., 2006)

O trabalho apresentado por Domínguez-Brito et al. (2005) é descrito um ambiente de desenvolvimento (*framework*) para múltiplos robôs. Este *framework* está direcionado para trabalhar com grupo de robôs móveis reais que operam no mesmo ambiente e interagindo em grupo. Ele utiliza estrutura de camadas que são apresentadas na figura 5.5.

Figura 5.5 - Camada do Framework.
 Fonte: (DOMÍNGUES-BRITO et al., 2005)

Na camada de coordenação (*Coordinating Layer*) é realizada a comunicação com o grupo de robôs, aonde as coordenadas são gerenciadas e o comportamento de cooperação de cada robô é gerado. Já a camada de execução (*Performing Layer*) é responsável pelos movimentos de cada robô individualmente, sendo que essa camada realiza o tratamento de erros e o gerenciamento das coordenadas de cada robô individualmente, de forma que, possam ser detectados erros de coordenadas do sistema (DOMÍNGUES-BRITO et al., 2005).

No topo da hierarquia das camadas deste *framework*, têm-se a camada de aplicação que é responsável pelo ambiente de desenvolvimento. Essa camada tenta minimizar todo o processo de desenvolvimento de aplicações para o grupo de robôs, pois as demais camadas implementam as funções necessárias à navegação, gerando assim, um nível superior de abstração (DOMÍNGUES-BRITO et al., 2005).

Já no trabalho de Bottazzi e Fonseca (2005) é apresentada uma ferramenta de programação baseada no paradigma de programação orientado a objetos, destinada a robôs industriais (braços mecânicos). Essa ferramenta tem como base um modelo abstrato para programação de robôs, sendo esse modelo encapsulado em classes da linguagem *Java2*. Com isso, o ambiente provê a reutilização dos códigos desenvolvidos para os diversos robôs que o ambiente suporta (BOTTAZZI; FONSECA, 2005; UTZ et al., 2002).

A utilização da estrutura de classes na programação auxilia os desenvolvedores na criação de aplicações para robôs industriais, pois eles não necessitam de conhecimento específico sobre o robô móvel, para o qual eles estão desenvolvendo a aplicação. Este aspecto colabora para aumentar o número de desenvolvedores, pesquisadores e até

mesmo estimula a criação de novas tarefas e novas aplicações para os sistemas robóticos (BOTTAZZI; FONSECA, 2005; UTZ et al., 2002).

No trabalho de Aleotti, Caselli e Reggiani (2004) é apresentado um *framework* que utiliza um novo paradigma de programação chamado de *Programming by Demonstration* (*PbD*). Esse paradigma trabalha com técnica de programação explícita através de blocos em 3D, que representam as operações de manipulação básica do sistema robótico. Esses blocos em 3D são utilizados em um ambiente virtual como meio de criar virtualmente a seqüência de tarefas que o robô tem que executar. Para realizar o reconhecimento das tarefas, o sistema reconhece a sequência de ações de alto nível, através do movimento das mãos do usuário e convertendo-as em comandos para manipular o robô. Uma vez reconhecidas, as tarefas são testadas e validadas no simulador (ALEOTTI; CASELLI; REGGIANI, 2004).

A figura 5.6 apresenta a programação utilizando o movimento das mãos que é uma das características do paradigma *PbD*.

Figura 5.6 – Sistema de Programação do Robô.

Fonte: (ALEOTTI; CASELLI; REGGIANI, 2004)

No trabalho de Côtê et al. (2004) é proposto um ambiente de programação que utiliza uma estrutura de blocos organizados em um fluxo de execução, essa estrutura de blocos denominada como *RobotFlow/FlowDesigner*.

O objetivo do *RobotFlow/FlowDesigner* é prover um ambiente gráfico e modular que auxilie os usuários a visualizar e compreender como as operações de controle, sensoriamento e atuadores funcionam. Para isso o *RobotFlow/FlowDesigner* apresenta essas funcionalidades através do fluxo de tarefas representados pelos blocos de ações do robô (CÔTÊ et al. 2004).

O ambiente de programação *FlowDesigner* foi desenvolvido para visualizar blocos de decisão, reutilizar os blocos construídos e auxiliar no desenvolvimento e na estruturação de sistemas robóticos. Contudo, não especifica em seus blocos de tarefas algoritmos inteligentes e complexos para robôs móveis autônomos (CÔTÊ et al. 2004).

A figura 5.7 apresenta o ambiente de desenvolvimento proposto por Côtê et al. (2004), ilustrando os blocos das ações que podem ser utilizados na programação de uma aplicação.

*Figura 5.7 –Sistema de Programação utilizando RobotFlow.
Fonte: (CÔTÊ et al. 2004)*

No trabalho de MacDonald et al. (2003) é descrito um ambiente de desenvolvimento estruturado em três camadas. A principal característica deste ambiente é o suporte a diversos sistemas robóticos, juntamente com uma interface de ligação a diversos dispositivos de entrada, saída e armazenamento. Inclui também uma linguagem de programação semelhante à linguagem BASIC, que é utilizada na estruturação dos comandos do ambiente de programação.

No ambiente de desenvolvimento apresentado por Utz et al. (2002), chamado de *MIRO*, foi implementada uma camada intermediária denominada *middleware*, que se baseia no paradigma de programação orientada a objetos, com a finalidade de possibilitar o desenvolvimento de aplicações para robôs móveis de modo fácil e rápido, mantendo a portabilidade e a manutenibilidade do software para robôs.

Segundo os autores Utz et al. (2002),

[...] a alta qualidade do software de robôs, juntamente com a melhoria do processo de desenvolvimento de software serão os fatores chaves tanto para um avanço nas pesquisas em ambas as áreas como em aplicações na área industrial e no mercado de consumo [...].

A característica que torna esse *middleware* viável é que o programador pode escrever de modo fácil seus códigos, além da reutilização dos seus códigos, pois o código desenvolvido é padrão, ou seja, a linguagem utilizada é padronizada para todos os tipos de robôs suportados pelo *middleware* (KRAMER; SCHEULTZ, 2007; UTZ et al., 2002; MAZER et al., 1991).

O *middleware* MIRO foi estruturado em três camadas, como pode ser visto na figura 5.9 (KRAMER; SCHEULTZ, 2007; UTZ et al., 2002):

- ✓ Camada de Dispositivos, que provê uma interface de abstrações orientada a objetos para todos os sensores e atuadores do robô;
- ✓ Camada de Serviços, que é responsável pela abstração dos serviços ativos que são utilizados pelos sensores e atuadores. Essa abstração é feita através da Interface de Definição de Linguagem (*IDL – Interface Definition Language*) vinda

da language CORBA(*Common Object Request Broker Architecture*). Essa interface implementa os serviços como sendo objetos de uma rede, os quais são transparentes à plataforma onde serão aplicados.

- ✓ Camada *Framework*, que é responsável pelos módulos funcionais mais utilizados no sistema robótico, tais como módulos para mapeamento, auto-localização, geração de comportamentos, planejamento de caminhos, etc.

Além das características citadas acima, o *middleware MIRO* implementa um simulador para a realização dos testes das aplicações, sem que haja a necessidade de carregar a aplicação para o robô físico (KRAMER; SCHEULTZ, 2007; UTZ et al., 2002).

Figura 5.9 - Estrutura de Programação Miro.

Fonte: (UTZ et al., 2002)

O trabalho apresentado por Zielinski (1999) descreve o ambiente de desenvolvimento *MRROC++*, cuja estrutura é mostrada na figura 5.8.

Esse ambiente é implementado utilizando programação orientada a objetos, linguagem C++ e possui sensores virtuais, que facilitam a utilização do ambiente pelos programadores (ZIELINSKI; SZYNKIEWICS; WINIARSKI, 2005; ZIELINSKI, 1999).

Figura 5.8 - A estrutura do sistema básico do MRROC++.
Fonte: (ZIELINSKI; SZYNKIEWICS; WINIARSKI, 2005; ZIELINSKI, 1999)

Apesar de sua estruturação, o projeto *MRROC++* visa o desenvolvimento de um ambiente de programação para robôs industriais ou braços mecânicos, não tendo como foco o desenvolvimento de aplicações para robôs móveis (ZIELINSKI; SZYNKIEWICS; WINIARSKI, 2005; ZIELINSKI, 1999). A contribuição relevante do *MRROC++* para a proposta de um ambiente de desenvolvimento de aplicações para robôs móveis é fato

de utilizar tarefas para a sua programação.

5.3. *Considerações sobre o Capítulo*

Os trabalhos apresentados nesse capítulo enfocam a criação de ambientes de desenvolvimento, os quais apresentam uma interface de programação de alto nível de abstração. Essa abstração permite aos programadores experientes desenvolverem aplicações de forma rápida e eficiente. Com isso, a perspectiva desses ambientes é a de que o programador não necessita ter um conhecimento aprofundado sobre o hardware, para o qual será desenvolvida a aplicação. Outra característica importante é que esses ambientes são totalmente *off-line*, isto é, a ferramenta desenvolve as aplicações sem que haja a necessidade do sistema robótico estar conectado ao ambiente (ALEOTTI; CASELLI; REGGIANI, 2004; UTZ et al., 2002; MAZER et al., 1991).

A figura 5.9 ilustra como o ambiente de desenvolvimento deve se comportar em relação ao sistema robótico.

Figura 5.9 - Partes do Sistema de Programação do Robô.

Fonte: (MACDONALD, 2003)

Nesse caso o sistema é dividido em duas partes distintas, sendo elas (DRUMWRIGHT, 2007; ALEOTTI; CASELLI; REGGIANI, 2004; MACDONALD et al., 2003; MAZER et al., 1991):

- ✓ A camada de interação homem-robô que descreve uma ou mais linguagens de programação, as quais farão conexão com a camada de infra-estrutura. Esse tipo de programação é realizada em linguagem de alto nível de abstração, visando facilitar e agilizar o processo de desenvolvimento de aplicações para o sistema robótico. Essa camada ainda provê, através de comandos específicos do robô (*API*), acesso direto à camada de infra-estrutura (DRUMWRIGHT,

2007; ALEOTTI; CASELLI; REGGIANI, 2004; MORRIS, 2006; MAZER et al., 1991);

- ✓ A camada de infra-estrutura é composta por todas as características físicas do sistema robótico, apresentando também a programação de baixo nível para a programação do robô. Essa a camada faz a ligação principal das tarefas ou códigos da programação com os dispositivos. Essa ligação é feita através da conversão do código de programação para comando ou seqüência de comandos da API (MAZER et al., 1991, DRUMWRIGHT, 2007, ALEOTTI; CASELLI; REGGIANI, 2004).

Além disso, todos os trabalhos analisados têm em comum um ponto principal, que é a utilização do paradigma orientado a objetos como forma de criação de uma linguagem de alto nível de abstração. Em alguns trabalhos é apresentada uma metodologia de desenvolvimento de aplicações através de componentes gráficos, porém todos os trabalhos, mesmo os que possuem implementação através de características gráficas, utilizam a estrutura do paradigma orientado à objetos para especificar os componentes do robô, para somente então, codificá-los na linguagem específica para cada sistema robótico.

Outra característica presente nesses trabalhos é que todos eles mencionam que para o ambiente alcançar um grau de abstração interessante aos programadores, o *framework* deve ser implementado em três camadas, pois assim, cada camada representaria um nível de abstração do sistema (Nível Programador, Nível de Especificação de Dispositivos, Nível de Hardware).

Juntamente com as características apresentadas pelos ambientes de programação (*frameworks*) de sistemas robóticos citados acima, a revisão bibliográfica desses ambientes contribui para definição de dois níveis de editores para o presente projeto, sendo: um dedicado à programadores experientes, e o segundo nível para usuários com pouco conhecimento de programação, que pretendem somente criar as ações que seus robôs realizarão ou para a validação de idéias ou teorias.

Além desses editores é necessária a definição de uma linguagem de alto nível de abstração baseada no paradigma orientado a objetos.

Capítulo 6 - Proposta de um Ambiente de Desenvolvimento de Aplicações para Robôs Móveis

6.1. Introdução

Esse capítulo apresenta a proposta de um ambiente de desenvolvimento de aplicações para robôs móveis, denominado *ADARM*, baseando-se nas características analisadas nos trabalhos discutidos no capítulo 5.

Os trabalhos do capítulo 5 que apresentaram maior contribuição para a proposta do presente projeto foram Park et al. (2006), Côtê et al. (2004) e MacDonald et al. (2003).

Os trabalhos de Park et al. (2006) e Côtê et al. (2004) utiliza em seus respectivos ambientes blocos de ações para simplificar a programação das aplicações. Essa característica é utilizada como base do editor *Wizard* proposto no projeto *ADARM*. A principal diferença entre o projeto *ADARM* e o trabalho desses autores é que no editor *Wizard* do *ADARM* os blocos representam tarefas de alto nível de abstração e não comandos ou ações com nível de abstração mais baixo, como nos trabalhos de Park et al. (2006) e Côtê et al. (2004).

O ambiente proposto por MacDonald (2003) é estruturado em três camadas. Essas camadas são utilizadas para dividir o ambiente em estruturas: programação, interação homem/robô e infraestrutura. Essas estruturas servem de base para a criação das camadas de interface, programação e *middleware* do ambiente ADARM.

Nas seções seguintes serão apresentados o objetivo, a arquitetura e o modelo formal do projeto ADARM (Ambiente de Desenvolvimento de Aplicações para Robôs Móveis).

6.2. *Objetivo*

O objetivo deste projeto de doutorado é propor um ambiente de desenvolvimento de aplicações para robôs móveis, que possibilita o uso de blocos de tarefas como base de sua programação, que representam tarefas de alto nível de abstração. O uso desses blocos permite que usuários com pouco conhecimento de programação possam desenvolver aplicações robóticas. Além disso, possibilita a implementação de aplicações rápidas para testes dos sistemas robóticos. Esse ambiente também inclui a proposta de uma biblioteca de subrotinas de alto nível de abstração denominada *Java for Robotics (J4R)*.

As contribuições esperadas para esse ambiente de desenvolvimento incluem o aumento da abstração de programação, a geração de uma interface de programação para usuários com pouco conhecimento em programação além de criar uma interface de comunicação mais simples entre o usuário/programador com o sistema robótico.

6.3. *Arquitetura*

O ambiente de desenvolvimento de aplicações para robôs móveis foi estruturado em camadas e interfaces. Essa estrutura de camadas foi utilizada para facilitar o desenvolvimento, além de que cada camada pode ser estendida para implementação de novas funcionalidades.

Uma vez estruturado em camadas independentes, o ambiente se fundamenta através das interfaces de ligação entre as camadas e as interfaces com os usuários.

Essa organização em camadas mostra que o funcionamento do ambiente segue a estrutura de uma hierarquia definida. Essa hierarquia inicia-se na camada *User Interface* e percorre todas as demais camadas até a finalização da aplicação quando é realizada a conversão para o simulador ou para o robô. Todas essas camadas estruturadas hierarquicamente, interagindo entre si e dentro de um mesmo ambiente gera uma arquitetura, a qual pode ser visualizada na figura 6.1.

Figura 6.1 –Arquitetura do Ambiente de Programação.

As principais camadas dessa arquitetura são:

- ✓ Camada de Interface com o usuário (HRI), que são as interfaces utilizadas por todos os usuários do ambiente de programação. Essas interfaces permitem a seleção de usuário 'Iniciante' (Wizard) ou 'Avançado' (Expert);
- ✓ Camada de Desenvolvimento, que é a camada responsável por todos os editores encontrados no ambiente de desenvolvimento. Esses editores correspondem aos editores de programação (Wizard e Expert Editor), editor de linguagem do sistema robótico (RDC) e o editor da biblioteca Java for Robotics (J4R);

- ✓ Camada *Middleware (Kernel)*, que é responsável pelo gerenciamento dos códigos vindos do ambiente de programação (*J4R* e *Wizard Editor*) e a conversão desses códigos para linguagens intermediárias (*C*, *J4R*, *Robot Description Code*, *Player/Stage*, etc.). Essa camada integra os dispositivos de hardware ao ambiente de desenvolvimento, sendo responsável pela descrição funcional dos sistemas robóticos utilizados pelo ambiente de desenvolvimento. Para cada hardware robótico, é utilizada uma biblioteca específica de conversão de hardware (KRAMER; SCHEULTZ, 2007; CÔTÉ et al., 2004; FALCONE et al., 2003; FREUND et al., 2001);
- ✓ Camada de Saída, que é a camada que direciona a saída do ambiente para o simulador ou para o sistema robótico.

Essa organização foi definida utilizando como camada principal a interface com o usuário e como camada final a conversão para o sistema robótico ou simulador. Isso ocorreu devido ao nível de abstração encontrado na proposta do *framework*, que se inicia com o nível de abstração mais alto, reduzindo o nível dessa abstração até chegar à camada da aplicação final (simulador ou robô), conforme apresentado nos trabalhos de Park et al. (2008), Kramer e Scheultz (2007), Park et al. (2006), Botazzi e Fonseca (2005), Aleotti, Caselli e Reggiani (2004), Côté et al. (2004) e MacDonald et al. (2003).

Embora tenha sido adotada essa forma de organização, a arquitetura poderia ter sido estruturada utilizando outras regras. Porém a estrutura definida no projeto segue a dos trabalhos já apresentados anteriormente, os quais fundamentam a estrutura proposta para o ambiente *ADARM*.

Na seção seguinte é descrito o modelo formal dessa arquitetura tendo como base a *Unified Modeling Language (UML)*.

6.4. Modelo Formal do Ambiente de Desenvolvimento de Aplicações para Robôs Móveis

As camadas da arquitetura citadas na seção 6.3 são analisadas juntamente com as demais estruturas do ambiente nas figuras 6.2, 6.3 e 6.4 , que mostram o projeto do ambiente desenvolvido em *UML*. Essa ferramenta (*UML*) foi utilizada para descrever tanto as ações que cada tipo de usuário irá realizar dentro da arquitetura quanto os processos internos que se relacionam entre si.

Figura 6.2 – Diagrama de USE CASE

A figura 6.2 apresenta o diagrama de caso de uso (*Use Case*) da proposta do ambiente ADARM. Esse diagrama informa a interação dos atores programador e usuário do sistema, destacando as funções que os atores podem requisitar do sistema.

Analisando separadamente os atores, tem-se as seguintes interações para o usuário:

- ✓ O ator pode realizar desde a criação de novos projetos que descrevem os detalhes do sistema robótico, até programar aplicações utilizando o editor *Wizard*;
- ✓ O ator pode solicitar a validação de uma aplicação através do validador do sistema, além de converter a aplicação para o sistema robótico ou para o simulador.

No contexto do programador tem-se:

- ✓ a criação de projetos com o detalhamento do sistema robótico;
- ✓ a inserção de linguagens que são utilizadas pelos sistemas robóticos;
- ✓ o desenvolvimento de novas tarefas através do editor *J4R*, que armazena essas tarefas na biblioteca *J4R*;
- ✓ o desenvolvimento de aplicações para o sistema robótico utilizando o editor *Expert*, o qual usa a linguagem *Java2* em conjunto com as tarefas da biblioteca *J4R*;
- ✓ a validação das aplicações que são convertidas para o sistema robótico ou para o simulador.

As tarefas apresentadas acima estão englobadas dentro de todas as camadas mencionadas no início desta seção.

Na figura 6.3 é apresentado o diagrama de classes. Esse diagrama utiliza a estrutura de classes para apresentar, de forma mais detalhada, as ações vindas do diagrama de *Use Case*, além de exibir as interações e correlações existentes entre essas classes.

Figura 6.3 – Diagrama de Classes

Nesse diagrama de classes é possível identificar as camadas principais do projeto através dos pacotes que encapsulam as classes presentes no diagrama. A camada de interface é identificada com o pacote *EDITORS*, já a camada de *Middleware* é identificada pelo pacote *KERNEL* e a camada de saída é representada como uma extensão do pacote *KERNEL* através das classes *CONVERSOR* e *PLAYERSIM*.

Como pode ser visto, até o presente momento, as classes estão relacionadas através da estruturação em camadas que foi adotada na proposta do ambiente. Contudo o diagrama de classes apresenta, além de sua estruturação, a relação existente entre essas classes.

Tendo em vista essas relações, as classes do ambiente *ADARM* estão correlacionadas tendo como classe principal a classe *JMFJ4R* que é a principal interface de acesso dos usuários ao sistema, sendo responsável pelo acesso aos principais módulos do ambiente. Devido a isso, a classe *JMFJ4R* esta associada diretamente às classes: *SETPROJ*, *XPEDITOR*, *WIZEDITORV2*, *HARDDDESC*, *LANGUAGEEDITOR*, *J4REDITOR*, *CONVERSOR* e *PLAYERSIM*.

Além das associações diretas das classes com a classe *JMFJ4R*, tem-se ainda as relações entre as classes adjacentes à classe *JMFJ4R*, que podem ser vistas a seguir:

- ✓ A classe *XpEditor* realiza uma chamada à classe *Conversor* que, após a criação da aplicação através do editor, realiza a conversão para o sistema robótico;
- ✓ A classe *Conversor* utiliza a classe *Analyser* para realizar a análise dos códigos desenvolvido nos editores ,e extraí as características da aplicação para a conversão para o sistema robótico;
- ✓ A classe *PlayerSim* utiliza diretamente a classe *Analyser* para realizar a extração dos códigos desenvolvidos para converter para o simulador *Player/Stage*;
- ✓ A classe *WizEditorV2* usa a classe *J4RCreate* que realiza a conversão dos blocos vindos da aplicação do editor *Wizard* para a linguagem *Java2+J4R*;
- ✓ Após a interação anterior, a classe *J4RCreate* faz uma chamada para a classe *J4RCompiler*, para a verificação do código gerado pela classe *J4RCreate*;

- ✓ A classe *J4RCompiler* é invocada pelas classes *XpEditor*, *J4REditor* e *J4RCreate*, de forma que, as classes invocadoras utilizam essa classe para verificar os códigos *Java2+J4R*;

Após o detalhamento do diagrama de classes, o próximo passo da modelagem é apresentar como os atores (usuários e programadores) interagem diretamente nos módulos do sistema. Essas interações são apresentadas nas figuras 6.4 e 6.5 .

Figura 6.4 – Diagrama de Sequência do Usuário

A figura 6.4 apresenta a interação direta do usuário com pouca experiência em programação com o ambiente de desenvolvimento de aplicações para robôs móveis (ADARM). O usuário, nesse caso, vai acessar a classe *JMFJ4R* que representa a interface de comunicação primária do ambiente, ou seja, essa classe é o menu de opções que o usuário pode acessar diretamente.

Uma vez acessado o menu, o usuário pode acessar os módulos: *SetProj*, *WizEditorV2*, *PlayerSim* e *Conversor*.

No caso do módulo *SetProj*, esse módulo é responsável pela criação de projetos, criando um arquivo com os dados do sistema robótico que serão utilizados pelo ambiente. A execução desse módulo não possui interdependência com outros módulos, pois o *SetProj*.

Caso o projeto já esteja criado, o usuário poderá acessar o editor *WizEditorV2* para criar as aplicações através dos blocos de tarefas. Utilizando esse mesmo editor, o usuário pode converter essa aplicação de blocos de tarefas em uma aplicação em *Java2+J4R* e posteriormente convertê-la para o simulador *Player/Stage* ou sistema robótico que irão utilizar o módulo *Analyser* para realizar a conversão.

Figura 6.5 – Diagrama de Seqüência do Programador

Na figura 6.5 é apresentado o diagrama de seqüência, com as ações que o programador poderá selecionar no ambiente *ADARM*. Essas ações são baseadas nas funções apresentadas pelo menu *JMFJ4R*, e são semelhantes às do usuário com pouco conhecimento em programação. A única diferença é que nesse caso o programador irá utilizar o *XpEditor* como forma de criar a aplicação para o sistema robótico.

Uma vez que o programador escolha o *XpEditor*, ele poderá validar suas aplicações através do módulo de compilação, e após a validação poderá converter essa aplicação para o sistema robótico.

Além do *XpEditor*, o programador poderá selecionar o módulo de projeto *SetProj*, além de converter a(s) aplicação(ões) para o sistema robótico ou para o simulador *Player/Stage* sem que seja necessário acessar o *XpEditor*.

Juntamente com os módulos mais utilizados, tanto por programadores quanto por usuários, tem-se ainda os seguintes módulos:

- ✓ *HardDesc*, utilizado para a inclusão de novos dispositivos no ambiente;
- ✓ *J4REditor*, utilizado para a inclusão de novas tarefas e funções na biblioteca *J4R*;
- ✓ *LanguageEditor*, é um editor que possui a finalidade de incluir novas linguagens de programação que o ambiente *ADARM* irá converter para o sistema robótico.

Os módulos acima citados são utilizados por programadores experientes, pois esses programadores são capacitados para gerenciar tais características do ambiente de desenvolvimento.

6.5. Outras considerações sobre o *ADARM*

As seções a seguir discute da escolha da linguagem de implementação do *framework* e descreve as características da biblioteca *Java for Robotics (J4R)*.

6.5.1. Escolha da Linguagem

A linguagem *Java2* foi utilizada no presente projeto como linguagem de programação do ambiente e linguagem para a criação do ambiente, sendo que a escolha da linguagem baseou-se nas características apresentadas e suportadas por essa linguagem. Para a escolha da linguagem a ser utilizada, foram analisadas as seguintes linguagens: linguagem C++, *Phyton*, C# e *Java2*. Dentre essas linguagens, a linguagem *Java2* destacou-se no recurso de modularidade em relação às linguagens C++ e *Phyton*, pois a aplicação em *Java2* pode ser executada em qualquer plataforma que contenha uma *Java*

Virtual Machine (JVM).

No caso da linguagem C#, apesar de possuir uma portabilidade semelhante ao *Java2*, essa linguagem utiliza um *framework* proprietário, apresentando um custo alto para o desenvolvimento do ambiente, enquanto que o *framework* da linguagem *Java2* é totalmente gratuito.

6.5.2. Biblioteca Java for Roboticas (J4R)

A biblioteca *Java for Robotics (J4R)*, proposta nesse projeto, é uma extensão de alto nível da programação de *Java2*, e encontra-se localizada entre o ambiente de programação e o hardware final. Foi estruturada para conter sub-rotinas de alto nível de abstração, as quais constituem as tarefas relacionadas aos blocos do *Wizard Editor*. Essas sub-rotinas descrevem tarefas complexas, baseadas nos conhecimentos adquiridos sobre robôs móveis, de forma que os programadores experientes não precisam reescrevê-las toda vez que seja necessário utilizá-las. Como exemplo de tarefas pode-se citar a detecção de objetos, a navegação por um prédio, entre outras.

A finalidade dessas tarefas é simplificar o desenvolvimento de macro-tarefas, como por exemplo, o desenvolvimento de uma aplicação em que o robô pessoal percorrerá os cômodos da casa para levar o remédio para o paciente, dentro do horário estipulado (ZIELINSKI; SZYNKIEWICS; WINIARSKI, 2005; FREUND et al., 2001; SHEPHERD, 1993; MAZER et al., 1991; ZIELINKSI, 1991).

A biblioteca *J4R* inclui tarefas que podem ser utilizadas por qualquer sistema robótico, ou seja, para um robô do tipo *Pioneer* existe uma biblioteca de configuração com as características peculiares do robô, e que se relaciona com a biblioteca *J4R*, fazendo com que essa biblioteca possa ser interpretada por qualquer sistema robótico. Nesse caso, essas bibliotecas de configuração possuem sub-rotinas, que podem ser ajustadas conforme a necessidade do hardware. Por exemplo, numa sub-rotina de navegação genérica, pode-se selecionar os tipos de sensores a serem utilizados.

Juntamente com todas as funcionalidades já descritas, a biblioteca *J4R*, em associação com o módulo *Middleware*, tem uma sub-rotina de tradução que converte todas as funções da aplicação desenvolvida pela linguagem *Java2* com *J4R*, na linguagem específica para o sistema robótico. Essa tradução é feita através de bibliotecas que descrevem as características de programação dos sistemas robóticos.

O módulo de desenvolvimento da biblioteca *J4R* implementa um tradutor da linguagem *Java2*, que verifica o código gerado pelo editor. O editor, por sua vez, verifica se ocorreu erros na criação da tarefa.

Uma vez criada e verificada, a tarefa pode ser incluída no módulo *Wizard Editor* para ser utilizada posteriormente por um usuário leigo.

Todas as funções e estruturas apresentadas possuem modularidade suficiente para que não haja necessidade de grandes mudanças no desenvolvimento das tarefas do robô. Assim, uma vez selecionado o hardware, o ambiente analisa as características desse hardware, e então disponibiliza as informações de manipulação desse hardware para o programador, e reajusta os comportamentos básicos e de navegação do robô doméstico

(ZIELINSKI; SZYNKIEWICS; WINIARSKI, 2005; FALCONE et al., 2003; FREUND et al., 2001; SHEPHERD, 1993; MAZER et al., 1991; ZIELINKSI, 1991).

Capítulo 7 - Construção do Ambiente de Desenvolvimento de Aplicações para Robôs Móveis

7.1. Introdução

Nesse capítulo será apresentada a implementação da proposta do ambiente de desenvolvimento de aplicações para robôs móveis (*ADARM*), e será detalhada a metodologia utilizada para o desenvolvimento dos seus módulos.

A implementação do *ADARM* baseou-se na arquitetura e modelagem apresentadas no capítulo 6, sendo a arquitetura, apresentada na seção 6.3, desmembrada em módulos, os quais são interligáveis e possuem alto grau de independência, de forma a tornar o ambiente maleável e modular.

Para atender os quesitos do ambiente *ADARM* especificados na modelagem (seção 6.4) foram desenvolvidos os seguintes módulos:

- ✓ Módulos de Interface com o usuário (*HRI*), que são os módulos utilizados por todos os usuários do ambiente de programação. Esses módulos permitem a seleção de usuário ‘Iniciante’ (*Wizard*) ou ‘Avançado’ (*Expert*). Os usuários do tipo ‘Avançado’ podem desenvolver novas aplicações e tem a liberdade para modificar os sistemas existentes ou até mesmo criar sistemas novos. Já na opção ‘Iniciante’, o usuário segue uma série de passos, que especifica o tipo de

robô que é utilizado, e quais as tarefas que o usuário deseja. No modo ‘Iniciante’ o usuário tem o suporte de tarefas através de blocos, que se assemelham ao *RobotFlow* que é descrito a seguir (CÔTÉ et al., 2004; FALCONE et al., 2003; FREUND et al., 2001; BISHOFF, 2000; MAZER et al., 1991);

- ✓ Módulo de Estruturação em Blocos (*RobotFlow*) baseado em *FlowDesigner* (KRAMER; SCHEULTZ, 2007; CÔTÉ et al., 2004). Esse módulo utiliza modelos de blocos para representar as atividades (tarefas) que o robô deverá desempenhar, sendo esses blocos colocados em ordem de execução pelo ambiente de desenvolvimento (ALEOTTI; CASELLI; REGGIANI, 2004; JOHNSON, 1998; SHEPHERD, 1993). Além disso, esse módulo trabalha diretamente com o editor *Wizard* cuja principal característica é aplicar as funcionalidades desse módulo e validar a aplicação;
- ✓ Módulo de Linguagem de Programação de alto nível, que se baseia na linguagem de programação *Java2* (*J4R – Java for Robotics*), sendo utilizado para a criação de aplicações e tarefas para robôs móveis. Essas tarefas podem ser adicionadas ao módulo de estruturação de blocos;
- ✓ Módulo de Configurações que tem como função a especificação das características do robô que é utilizado no projeto. Este módulo possui a linguagem de programação correlacionada com a linguagem do ambiente de desenvolvimento, que é utilizada para a conversão das aplicações desenvolvidas para o sistema robótico (*Robot Description Code - RDC*);

- ✓ Módulo *Middleware (Kernel)*, que é responsável pelo gerenciamento dos códigos vindos do ambiente de programação (*J4R* e *Wizard Editor*) e a conversão desses códigos para linguagens intermediárias (*C*, *J4R*, *Robot Description Code*, *Player/Stage*, etc.). Esse módulo integra os dispositivos de hardware ao ambiente de desenvolvimento, sendo responsável pela descrição funcional dos sistemas robóticos utilizados pelo ambiente de desenvolvimento. Para cada hardware robótico, é utilizada uma biblioteca específica de conversão de hardware (KRAMER; SCHEULTZ, 2007; CÔTÉ et al., 2004; FALCONE et al., 2003; FREUND et al., 2001);
- ✓ Módulo de Conversão para o Simulador que é utilizado para testar e validar as aplicações desenvolvidas pelo ambiente de desenvolvimento. Este módulo é responsável pela conversão da aplicação desenvolvida, para um determinado sistema robótico, em uma aplicação a ser simulada no simulador de robôs *Player/Stage*.

A seqüência de interação de tais módulos é apresentada na figura 7.1

Figura 7.1 - Seqüência de Desenvolvimento dos Módulos

7.2. Metodologia

A estruturação da plataforma de desenvolvimento de aplicações para robôs móveis (*framework*), apresentada na figura 7.1, é do tipo *top-down* e foi desenvolvida utilizando a linguagem *Java2* em conjunto com o *framework* *NetBeans*. A interação entre os módulos foi organizada tendo como módulos iniciais as interfaces com os diferentes tipos de usuário, que criam um projeto contendo todas as características necessárias ao sistema de forma que, o *framework* consiga codificar a aplicação desenvolvida no ambiente, em aplicação para o robô ou para o simulador *Player/Stage*.

As especificações das interfaces são direcionadas ao formato de linguagem que é usada pelos sistemas robóticos, como por exemplo, *C*, *C++*, *Java2*, etc., e também pela forma de comunicação utilizada no simulador *Player/Stage* (*C*, *C++* ou *Java2*) e pela localização dos arquivos do projeto e da biblioteca *J4R*.

A estruturação do módulo *Robot Description Code* (*RDC*) baseou-se nos trabalhos Bottazzi e Fonseca (2005) e Utz et al. (2002).

Os módulos subseqüentes são a biblioteca *J4R* e o *Middleware*.

O módulo *J4R* foi estruturado como uma biblioteca e seu desenvolvimento é descrito na seção 6.5.2. Essa biblioteca contém as tarefas de uso comum aos sistemas robóticos, as quais podem ser implementadas ou modificadas de forma a criar e melhorar as tarefas já existentes.

Já no módulo *Middleware* é realizado todo o processo de transformação da aplicação desenvolvida em *Java2* e *J4R*, para o sistema robótico ou para o simulador. Essa conversão baseia-se nas configurações encontradas no projeto criado e nos arquivos de configuração do *framework*.

Todos os módulos do projeto trabalham em conjunto e são ligados diretamente ou indiretamente uns aos outros, sendo que os módulos principais de entradas de dados do sistema são os editores *Expert Editor* e *Wizard Editor* que conduzem ou requisitam dados para os demais módulos, conforme a ilustração de seqüência de execução da figura 6.6.

Para o funcionamento desses módulos foi criado um subdiretório denominado *settings*, onde foram alocados todos os arquivos necessários para a configuração do ambiente de programação. Esse subdiretório é visto na figura 7.2

Figura 7.2 – Subdiretório Settings

Esse subdiretório é um repositório dos dados que o ambiente utiliza para seu funcionamento, estando ele dividido nas seguintes subpastas:

- ✓ *J4Rlib* que contém os arquivos das tarefas desenvolvidas e validadas pelo editor *J4R*;
- ✓ *Languages* que contém os arquivos com a descrição das linguagens de programação utilizadas pelos sistemas robóticos;
- ✓ *Player/Stage* é utilizado para armazenar as configurações das linguagens de programação que o *Player/Stage* suporta;
- ✓ *Robots* contém os arquivos que compõe e descrevem as características dos sistemas robóticos;

✓ *J4RPath.lib* é o arquivo com a localização da pasta *J4RLib*.

Associado às características apresentadas até aqui, tem-se, juntamente com o ambiente de desenvolvimento, um módulo de configuração das características do robô, isto é, o ambiente de programação contém um *menu* de configuração do sistema robótico que se destina à configuração da linguagem ou dispositivos do robô (DRUMWRIGHT, 2007; JOHNSON; MARSH, 1998).

Resumidamente, na implementação da proposta do *ADARM* houve a preocupação não só de seguir as especificações apresentadas no capítulo 6, mas também prover um ambiente de desenvolvimento amigável e flexível, destinando-se não só a profissionais de programação como também a usuários de computadores com pouco conhecimento de informática (DRUMWRIGHT, 2007; ZIELINSKI; SZYNKIEWICS; WINIARSKI, 2005; ALEOTTI; CASELLI; REGGIANI, 2004).

Nas próximas seções serão apresentadas, com mais detalhes, as características dos principais módulos do presente projeto.

7.2.1. Interface de Usuário

A camada de interação homem/robô deve ser uma interface de fácil manipulação pelo usuário, e que ao mesmo tempo permita a utilização dos recursos do sistema robótico, sem que haja a necessidade de grandes conhecimentos na área de robótica.

Para essa condição ser alcançada na implementação da *ADARM*, a interface de usuário foi construída com as seguintes características :

- ✓ Ambiente interativo e amigável, o qual possui todas as funcionalidades e itens de configuração do sistema robótico, bem como todos os dispositivos de acesso do robô (DRUMWRIGHT, 2007; ALEOTTI; CASELLI; REGGIANI, 2004; FALCONE et al., 2003; MACDONALD et al., 2003);
- ✓ Linguagem de Programação para Robôs, linguagem de alto nível de abstração contendo sub-rotinas básicas e avançadas para o funcionamento de um robô móvel (DRUMWRIGHT, 2007; ZIELINSKI; SZYNKIEWICS; WINIARSKI, 2005; JOHNSON; MARSH, 1998; ZIELINKSI, 1991);
- ✓ A interação homem/robô é caracterizada pela maleabilidade de se utilizar diversos dispositivos e arquiteturas de hardware sem que haja a necessidade de se alterar o código da aplicação do robô doméstico, além de prover as informações necessárias para a conversão do software em aplicação final para o hardware ou aplicação final para o simulador de robôs (DRUMWRIGHT, 2007; FALCONE et al., 2003; SHEPHERD, 1993; MAZER et al., 1991).

Além das características mencionadas, para que a interface pudesse ser utilizada tanto por programadores como por usuários, o *framework* foi dividido em dois tipos de editores:

- ✓ *Expert Editor* que é o editor utilizado por programadores que farão uso do sistema para programar suas tarefas. Nesse editor o programador utiliza a linguagem *Java2* para programar as aplicações do sistema robótico, utilizando também as tarefas desenvolvidas na biblioteca *J4R*. Para simplificar o desenvolvimento das aplicações, a linguagem *Java2* é utilizada de forma simplificada e objetiva, deixando de lado alguns dos seus recursos, pois a maior parte das linguagens de desenvolvimento de sistemas robóticos não utiliza tais recursos (interfaces gráficas, alguns conceitos de orientação à objetos, entre outros). Essa interface é apresentada na figura 7.3;

Figura 7.3 – Expert Editor

- ✓ *Wizard Editor* é o editor desenvolvido por usuários com pouco conhecimento de programação, porém possui conhecimento sobre as tarefas que queiram que o sistema robótico desenvolva. Nesse editor são encontradas também as características do robô que estão descritas no projeto e o seu funcionamento é semelhante a um conjunto de blocos os quais possuem os nomes das tarefas que podem ser utilizadas pelo sistema robótico do usuário. Esses blocos contêm a descrição da tarefa e a implementação desta tarefa não fica visível a esse tipo de usuário, sendo que juntamente com a implementação desta tarefa existe a descrição das dependências existentes a essa tarefa (ZIELINSKI; SZYNKIEWICS; WINIARSKI, 2005; ALEOTTI; CASELLI; REGGIANI, 2004; CÔTÉ et al., 2004; MAZER et al., 1991; ZIELINKSI, 1991). A interface desse editor é mostrada na figura 7.4, onde pode ser vista a seleção de uma tarefa já desenvolvida e validada na biblioteca *J4R*, podendo ser utilizada pelo *Wizard Editor*.

Figura 7.4 – Wizard Editor

Os editores acima mencionados constituem o principal meio de manipulação do ambiente de desenvolvimento. Aliado a esses editores existe também o editor do projeto que tem como essência os detalhes do sistema robótico, como por exemplo tipo de linguagem que o robô e o simulador utilizam, bem como a funcionalidade de incluir novos sistemas robóticos ainda não adicionados ao modelo atual do ambiente de desenvolvimento.

Para complementar os editores, tem-se ainda o *Robot Description Code* (código de descrição do robô) que tem a função de criar uma tabela de correlação entre a biblioteca *J4R* e a linguagem do robô, sendo que o processo de correlação gera uma lista que é utilizada pela camada *Middleware* para a conversão final.

Para que os módulos acima interajam entre si, foram criados diversos arquivos de configuração, os quais contêm diversas informações sobre o projeto, como por exemplo, o arquivo de configuração que determina as funções validadas da biblioteca *J4R* que servem tanto para a utilização na biblioteca *J4R*.

Essa implementação de arquivos de configuração foi desenvolvida a fim de validar todas as interações entre os módulos do *framework*.

7.2.2. Biblioteca *J4R* - Java For Robotics

Tendo como base as características da implementação da biblioteca *J4R* apresentadas na seção 6.5.2, pode-se visualizar a interface gráfica para esse módulo na figura 7.5, a qual exibe o editor *J4R*.

Figura 7.5 - Editor da Biblioteca J4R

7.2.3. Camada Middleware

Baseando-se nas funcionalidades da biblioteca *J4R*, é observado que para o funcionamento completo de um sistema robótico, além do ambiente de programação, é necessário um sistema de validação das aplicações desenvolvidas, de forma que, essas aplicações possam ser enviadas para o sistema robótico ou para o simulador de robôs sem erros de conversão ou interpretação de comandos (DRUMWRIGHT, 2007).

Esta tarefa é executada pela camada *Middleware*, que provê a conversão do código para o simulador de robôs, e também é responsável por diversas funções dentro do contexto do *framework*. Essas funções são descritas a seguir (DRUMWRIGHT, 2007):

- ✓ Gerenciar todos os tipos de arquivos dentro do ambiente;
- ✓ Gerenciar os arquivos de configuração, fazendo a requisição e distribuindo essas informações entre os demais módulos.
- ✓ Converter o código desenvolvido no ambiente, em um software específico para um determinado hardware ou sistema robótico;
- ✓ Converter o código desenvolvido no ambiente, em uma aplicação que seja executada diretamente no simulador de robô *Player/Stage*, sendo que esse simulador contém todas as características do robô que é utilizado pelo projeto de desenvolvimento;
- ✓ Analisar a sintaxe dos programas e validar os códigos feitos pelos programadores nos editores *Wizard Editor*, *Expert Editor* e *J4R Editor*, e validar as tarefas da biblioteca *J4R*;

- ✓ Gerenciar os dados vindos do *Robot Description Code* e distribuí-los entre os demais módulos;

A camada *Middleware* utiliza diretamente as características implementadas pelo *Robot Description Code (RDC)*, que contém a transcrição dos códigos do *J4R* para o sistema robótico ou para o simulador.

Dessa forma, a camada *middleware* consegue prover uma interface flexível entre o sistema final (robô ou simulador) e o ambiente de desenvolvimento, tornando os módulos do ambiente expansíveis, de forma que, qualquer modelo de robô possa utilizar esse ambiente de desenvolvimento. Essa flexibilidade é possibilitada pela descrição dos comandos das linguagens utilizadas pelo sistema robótico ou simulador, as quais podem ser usadas pelo ambiente, através do arquivo descritor (*RDC*), que auxilia no processo de validação e conversão dos códigos gerados pelo ambiente.

Algumas das funções apresentadas anteriormente podem ser vistas na interação de alguns módulos do sistema nas figura 7.6 e 7.7. A figura 7.6 ilustra o momento da conversão de um código feito por um programador experiente (figura 7.3) para a linguagem do sistema robóticos, que foi configurado no arquivo do projeto. A figura 7.7 mostra o processo de conversão, que é habilitada após a validação, desde que não seja encontrado nenhum erro em seu código original (programa *Java2+J4R*).

Figura 7.6 –Conversor para o Sistema Robótico

Figura 7.7 – Conversão para o Sistema Robótico

7.2.4. Módulo de Conversão para o Ambiente de Simulação

O módulo de conversão para o ambiente de simulação refere-se ao ponto de saída do *framework* para o simulador de robôs móveis, que é utilizado para validar as aplicações desenvolvidas pelo ambiente de desenvolvimento. Normalmente este módulo deverá ser muito utilizado pelos usuários do ambiente de desenvolvimento, devido ao fato de que toda aplicação desenvolvida deverá ser testada antes de ser enviada para o robô físico, pois se houver algum erro de codificação ou programação, a estrutura do robô poderá ser comprometida ou até mesmo danificada. Como a estrutura física do robô possui um alto valor agregado, deve-se evitar os erros de codificação, ou como no caso utilizado aqui, deve-se fazer uma validação através do simulador de robôs móveis.

Este módulo atende as necessidades dos sistemas robóticos utilizados nesse ambiente de desenvolvimento, pois usa o simulador *Player/Stage*, que consegue abranger grande parte dos sistemas robóticos, e tem a capacidade de se adaptar aos novos sistemas robóticos.

A figura 7.8 apresenta o conversor de aplicações do ambiente de desenvolvimento para o simulador *Player/Stage*.

Figura 7.8 –Conversor para o Simulador

A escolha do *Player/Stage* foi feita através dos resultados obtidos com uma avaliação realizada para testar e averiguar as características dos simuladores existentes, tanto no mercado como na área acadêmica. Nesta pesquisa foi constatado que por melhor que seja o simulador, ele sempre apresentará limitações tanto para o sistema robótico como para as necessidades do usuário. Exceção feita aos simuladores *Player/Stage* e *Webots*, que conseguem atender as necessidades dos programadores, principalmente no que se refere à confiabilidade, pois ambos possuem identificação fiel aos hardwares utilizados pelos seus sistemas robóticos emulados, isto é, quando um programador desenvolve uma aplicação para o robô, a simulação é fiel às características do robô que será utilizado para aquela aplicação.

Com relação ao custo, o simulador *Player/Stage* tem a vantagem de ser completamente gratuito. Essa característica aliada ao fato dele possuir seu código de desenvolvimento aberto, além de ser um dos simuladores mais confiáveis, levou à escolha desse simulador.

Para a implementação do módulo de conversão para o sistema de simulação foi verificada a codificação que o simulador *Player/Stage* utiliza para criar ambientes de simulação para os sistemas robóticos. Essa codificação é uma linguagem de programação que descreve quais as tarefas e características que o simulador deverá cumprir para um determinado robô ou conjunto de robôs (GERKEY; VAUGHAN; HOWARD, 2003). O formato de programação dessa linguagem baseia-se nas linguagens C, C++ ou *Java2*, o que a torna mais fácil de programar e compreender (GERKEY; VAUGHAN; HOWARD, 2003). Essa linguagem foi então utilizada, nesse módulo do ambiente *ADARM*, para criar aplicações de validação do sistema robótico no simulador *Player/Stage*.

7.3. *Considerações sobre o Capítulo*

O presente capítulo apresentou com detalhes a implementação do *ADARM* juntamente com a descrição dos módulos existentes no mesmo, sendo essa implementação fundamentada de acordo com a modelagem e diagrama de seqüência detalhados no capítulo 6.

O próximo capítulo aborda os resultados e discussões dessa implementação.

Capítulo 8 - Resultados e Discussões

8.1. *Introdução*

O presente capítulo visa apresentar os resultados obtidos com a implementação da proposta do ambiente de desenvolvimento de aplicações de robôs móveis (*ADARM*), discutir esses resultados e comparar o ambiente implementado com alguns ambientes descritos em trabalhos apresentados no capítulo 5.

8.2. *Resultados*

Para a avaliação da implementação do *ADARM* foram adicionadas ao ambiente as características dos sistemas robóticos *KheperaIII* e do *Korebot*, sendo que suas especificações foram extraídas da *API* dos próprios fabricantes. Em seguida, os módulos do ambiente implementado foram analisados, primeiramente de forma individual e posteriormente em conjunto, considerando os dois tipos de usuários especificados, o programador experiente e o usuário com pouco conhecimento de programação.

Outros sistemas robóticos não foram incluídos pelo fato de grande parte dos fabricantes não informar a *API* de seus sistemas robóticos sem a compra dos mesmos. Porém, com a inclusão desses dois sistemas robóticos foi possível realizar a depuração do ambiente desenvolvido e identificar suas limitações, que poderão ser utilizadas como base para novas pesquisas na área.

A avaliação do ambiente *ADARM* para esses dois sistemas robóticos foi feita através do simulador *Player/ Stage*, pelo fato de não se ter os robôs, e essa avaliação é apresentada na seção 8.2.1. A comparação do *ADARM* com outros ambientes é apresentada na seção 8.2.2.

8.2.1. Avaliação do ambiente implementado (*ADARM*)

Considerando a avaliação por módulos, com relação aos resultados apresentados, pode-se concluir, de forma resumida, que:

- ✓ *Expert Editor* - nesse sub-módulo foi identificada uma limitação vinda da própria linguagem *Java2*, relacionada à inexistência de heranças múltiplas. Esse recurso permite que o programador crie uma classe, que herda as características de mais de uma classe ao mesmo tempo, comportando assim, diversos métodos de programação distintos. Devido à falta deste recurso (herança múltipla), o programador avançado deverá implementar diversas classes, de forma que cada uma herde as características de uma única classe, e no final da herança seja criada um classe que herde, em cascata, todas as outras classes. Essa limitação pode ser solucionada através de bibliotecas e de pacotes, sendo que qualquer uma das soluções apresentadas geraria um novo problema: ao se criar novas funcionalidades essas bibliotecas teriam de ser modificadas e tanto os pacotes quanto as bibliotecas necessitariam ser atualizadas. Apesar dessa limitação, esse editor possui uma interface semelhante aos editores de programas encontrados em diversos ambientes de programação encontrados comercial-

mente;

- ✓ *Wizard Editor* - possui a sua base de estruturação na biblioteca *J4R*, e apresentou uma estrutura estável, de forma que, a sua ligação com a biblioteca *J4R*, não altera as tarefas já existentes. Nesse editor, a maior dificuldade encontrada foi com relação à ligação das estruturas desenvolvidas pela biblioteca *J4R* com a linguagem *Java2*, e apesar dessa dificuldade inicial, o editor mostrou-se funcional e a integração de suas estruturas apresentou resultados satisfatórios com relação às conversões realizadas;
- ✓ O editor da biblioteca *J4R* - possui uma estrutura semelhante ao *Expert Editor* com algumas simplificações, como por exemplo, a inexistência da classe principal e do corpo do programa. Nesse caso, o programador implementa direto sua tarefa, compila e a adiciona no *Wizard Editor*. Além disso, essa biblioteca é referenciada a um arquivo contendo os comandos de alto nível, que poderão ser utilizadas na criação das tarefas. Os comandos de alto nível citados anteriormente são vinculados à linguagem do robô, através do editor *RDC* (*Robot Description Code*), o qual relaciona os comandos da linguagem *Java2+J4R* com a linguagem do robô;

- ✓ Módulo *RDC* - nessa interface foi identificada uma inconsistência, relativa à ligação de novas tarefas desenvolvidas para a biblioteca *J4R* com a linguagem do sistema robótico ou simulador. Essa inconsistência ocorre quando o programador cria uma nova tarefa no editor *J4R*. Essa nova tarefa, ao ser validada, é incluída automaticamente no arquivo *base.lng* e com isso, essa nova tarefa não possui correlação com a linguagem do sistema robótico e nem com a linguagem do simulador. Dessa forma, após criar uma nova tarefa, o programador deverá entrar no *RDC* e correlacionar essa tarefa com as linguagens do sistema robótico e do simulador. Devido a essas características do *RDC*, é necessário uma especificação detalhada tanto da linguagem *Java2+J4R* quanto a linguagem do sistema robótico;
- ✓ Módulo *Middleware* - conjunto de pequenos módulos responsáveis pela interconexão de todos os módulos do ambiente, fazendo com que sejam possíveis as conversões, validações e modificações realizadas em qualquer fase do desenvolvimento da aplicação. Esse módulo, apesar de sua complexidade de desenvolvimento, atingiu um funcionamento estável, podendo ser aperfeiçoado para comportar funções adicionais, como, pacotes de bibliotecas da linguagem *Java2+J4R*, herança, polimorfismo, vinculação das tarefas do editor *J4R* ao *Wizard Editor* sem a passagem pelo editor *J4R*, entre outras.

Com relação à interconexão dos módulos verificou-se que a interação entre as interfaces e o *middleware* ocorreu conforme o projetado, de forma que, as interfaces conseguiram acessar as tarefas dispostas pela camada *middleware* corretamente. No contexto geral, os seguintes problemas foram identificados:

- ✓ Dificuldade de uso dos recursos do sistema para o usuário: o usuário necessita sair do módulo que está, para o acionamento de outros módulo do ambiente;
- ✓ Abertura da estrutura do projeto da aplicação: nesse caso, o usuário ou programador, ao utilizar o ambiente, necessita abrir o projeto do robô para informar ao ambiente qual o sistema robótico ou linguagem de programação será convertida a aplicação. A solução ideal seria que já fosse solicitado o projeto ao usuário/programador quando o ambiente fosse aberto, e esse projeto ficaria na memória até a finalização da aplicação ou a troca de projeto;
- ✓ Problemas na criação de tarefas no *J4R*, pois quando é criada uma nova tarefa o programador deve adicionar essa tarefa manualmente no *RDC*.

Pode-se concluir que a implementação do ambiente *ADARM* conseguiu atingir a maioria dos requisitos especificados na modelagem apresentada no capítulo 6.

8.2.2. Comparação do *ADARM* com outros ambientes

Dentre os ambientes de desenvolvimento para sistemas robóticos mencionados no capítulo 5, destacam-se para comparação com o *ADARM* aqueles descritos nos trabalhos de Kramer e Scheultz (2007), Park et al. (2006), Botazzi e Fonseca (2005), Aleotti, Caselli e Reggiani (2004), Côté et al. (2004) e MacDonald et al. (2003). Esses trabalhos

foram os que apresentaram uma maior parcela de contribuição para a evolução e concretização da proposta do *ADARM*, e na comparação apresentada a seguir será destacada apenas a parcela de contribuição de cada trabalho .

No trabalho de Kramer e Scheultz (2007) é descrito o ambiente *Pyro* como um ambiente utilizado para fins educacionais, podendo ser usado também como ferramenta de desenvolvimento. A principal característica do *Pyro* é demonstrar o funcionamento de alguns métodos de *IA* e controle de robôs móveis. Em relação à proposta do projeto *ADARM*, o ambiente *Pyro* implementa funcionalidades de alto nível de abstração, porém não utiliza interface gráfica de programação e nem blocos de tarefas pré-programadas.

Já no trabalho de Bottazi e Fonseca (2005) o ambiente apresentado utiliza a linguagem *Java2* no desenvolvimento de aplicações. Essa trabalho, juntamente com o de Utz et al. (2002), contribuíram para a estruturação do módulo de descrição da linguagem do robô móvel (*RDC*) do *ADARM*. O *ADARM* também utiliza a linguagem *Java2* em sua programação, porém sua proposta destina-se a robôs móveis e robôs domésticos, gerando códigos específicos para o simulador *Player/Stage* ou para o sistema robótico, enquanto o ambiente de Bottazzi e Fonseca (2005) destina-se a robôs industriais, e gera código em *assembly* para as três linhas de robôs industriais configurados no ambiente.

Nos trabalhos apresentados por Park et al. (2006) e Côté et al. (2004), ambos apresentam em seus ambientes programação utilizando blocos de ações, para simplificar a programação das aplicações. No caso do trabalho de Park et al. (2006), os blocos descrevem os comandos de programação, de forma que o programador apenas configura as

características para o funcionamento do comando. Assim um conjunto de blocos de comandos configurados geram uma aplicação. Em relação ao trabalho de Côtê et al. (2004) são criados blocos de ações que podem ser estruturados através de um fluxo de seqüência para gerar a aplicação robótica.

As peculiaridades apresentadas por Park et al. (2006) e Côtê et al. (2004) são utilizadas como base do editor *Wizard* proposto no presente projeto. A principal diferença entre o projeto e o trabalho apresentado é que no editor *Wizard* os blocos representam tarefas de alto nível de abstração e não comando ou ações com nível de abstração mais baixo.

No trabalho de Aleotti, Caselli e Reggiani (2004) é apresentado um ambiente de desenvolvimento que utiliza um ambiente virtual e blocos de programação em três dimensões para a programação de robôs. A principal característica desse ambiente é utilizar as representações de gestos do usuário, e transforma-los em simulação para o ambiente. Com relação ao presente projeto, a proposta do *ADARM* utiliza como característica, uma interface gráfica de forma a simplificar a programação dos sistemas robóticos.

O ambiente proposto por MacDonald (2003), é estruturado em três camadas, que são utilizadas para dividir o ambiente em estruturas: programação, interação homem/robô e infraestrutura. Essas estruturas serviram de base para a criação das camadas de interface, programação e *middleware* do ambiente *ADARM*.

Outros ambientes que podem ser comparados com o *ADARM* são o *Pyro* (PYRO, 2007), o *Miro* (UTZ et al., 2002), o *RobotFlow* (CÔTÊ et al. 2004) e o *Aria* (KRAMER; SCHEULTZ, 2007; ACTIVMEDIA, 2005), cujas características são apresentadas na tabela 8.1, juntamente com as características da *ADARM*. Dentre as características apresentados na tabela 8.1, as que enfocam diretamente as discussões realizadas sobre ambientes de desenvolvimento de aplicações para robôs móveis, e mais relevantes, são as seguintes:

- ✓ Linguagem de alto nível de abstração, que é utilizada para facilitar e agilizar o desenvolvimento de aplicações;
- ✓ Interface Gráfica, que serve como meio de comunicação dos usuários com o ambiente;
- ✓ Blocos de Comandos, que são comandos estruturados em blocos configuráveis, utilizados para facilitar a programação;
- ✓ Blocos de Tarefas, são tarefas estruturadas em blocos não configuráveis que servem para a criação rápida de uma aplicação. As tarefas utilizadas como blocos são estruturadas através de diversos comandos associados, criando assim uma pequena aplicação gerando um nível de abstração mais alto.

Observa-se na tabela 8.1 que o *ADARM* é o único que apresenta bloco de tarefas. Além disso, permite a programação em linguagem de alto nível como os demais ambientes permitem (exceto o *Robot Flow*). Somente o *ADARM* e o *Pyro* possibilitam o suporte a diversos robôs. Somente o *ADARM* e o *RobotFlow* apresentam interface gráfica.

Tabela 8.1 Comparação entre os Ambientes de Desenvolvimento

<i>RDE</i>	<i>Pyro</i>	<i>Miro</i>	<i>Robot Flow</i>	<i>Aria</i>	<i>ADARM</i>
<i>Sistema Operacional</i>	U,W	U,W	L	U,W	J
<i>Suporte a diversos robôs</i>	Sim	Parcialmente	Não	Não	Sim
<i>Linguagem Programação</i>	Python	C++	BC	C++	Java2+ J4R
<i>Linguagem Alto nível</i>	Sim	Sim	BC	Sim	Sim
<i>Interface Gráfica</i>	Não	Parcialmente	Sim	Não	Sim
<i>Blocos de Comandos</i>	Não	Não	Sim	Não	Não
<i>Blocos de Tarefas</i>	Não	Não	Não	Não	Sim
<i>Data</i>	2007	2002	2004	2005	2008

Legenda: U = Unix; W = Windows; L = Linux; J = Java2; BC = Bloco de Comandos; J4R = Java for Robotics

8.3. *Outras considerações*

A implementação da *ADARM* foi realizada de forma hierárquica, pois de acordo com Park et al. (2008), Park et al. (2006) e Freund (2001), em todo sistema robótico há uma hierarquia estrutural e uma hierarquia de execução. No caso de robôs móveis essa hierarquia também existe, pois o robô deve realizar as tarefas atribuídas a ele, desde que ele não desrespeite as definições pré-programadas de um sistema básico, como por exemplo, um cortador de grama cuja função é a de cortar a grama, deve parar essa atividade se ele detectar a presença de algum objeto que ele não consiga identificar. Com isso, todo robô doméstico possuirá a capacidade de executar as funções de emergência (funções reativas) e funções pré-determinadas (funções deliberativas). Esta hierarquia é apresentada na Figura 8.1.

Figura 8.1 - Hierarquia de Execução de Aplicações no Sistema Robótico

A importância desta hierarquia em um sistema robótico, principalmente nos sistemas robóticos domésticos, é prover a segurança dos usuários, uma vez que, quando o robô doméstico for ativado, deverá realizar suas tarefas programadas sem provocar danos às pessoas que estejam à sua volta. Esta segurança visa proteger também a integridade estrutural do robô, pois ao analisar o ambiente, o robô poderá definir quais as melhores ações a serem tomadas sem que haja risco ao usuário e à sua estrutura.

Esse sistema de segurança que é representado pelas camadas Sistemas Reativos e Sistemas Pró-Ativos estão presentes nos sistemas robóticos através do sistema operacional (SO), o qual controla e gerencia os recursos e aplicações que serão executados (PARK et al., 2008; FREUND et al., 2001). Apenas em sistemas que não possuem SO para controlar os dispositivos e aplicações, o programador deverá se preocupar com a segurança dos usuários.

Capítulo 9 - Conclusões Gerais e Trabalhos Futuros

9.1. *Conclusões*

Esse trabalho apresentou a proposta de um ambiente de desenvolvimento de aplicações para robôs móveis (*ADARM*), estruturado de forma hierárquica (*top-down*), que possibilita o uso de blocos de tarefas como base de sua programação, que representam tarefas de alto nível de abstração. O uso desses blocos permite que usuários com pouco conhecimento de programação possam desenvolver aplicações robóticas, além de possibilitar a implementação de aplicações rápidas para testes dos sistemas robóticos. A proposta inclui também uma biblioteca de subrotinas de alto nível de abstração denominada *Java for Robotics (J4R)*.

Para validação da proposta, o ambiente foi implementado de acordo com as especificações de sua arquitetura e modelagem apresentadas no capítulo 6. O ambiente implementado foi avaliado através dos sistemas robóticos *KheperaIII* e *Korebot*, em aplicações geradas para o simulador *Player/Stage*. Essa avaliação verificou o funcionamento do ambiente em relação à proposta apresentada no capítulo 6, não levando em consideração desempenho e otimização, pois a proposta discute a implementação de um ambiente de desenvolvimento de aplicações para robôs móveis, de forma a comprovar a viabilidade desse ambiente com características de alto nível de abstração na programação e utilização de blocos de tarefas.

Os resultados da avaliação do ambiente implementado evidenciaram que a proposta contribui com o desenvolvimento de aplicações de robôs em dois aspectos principais. O primeiro refere-se à simplificação de programação através do módulo *J4R*, que melhora a abstração da programação criando subrotinas de alto nível de abstração, que serão utilizadas no momento da criação da aplicação para o robô. O segundo refere-se à interface desenvolvida para os usuários com pouco conhecimento de programação (*Wizard Editor*), que utiliza blocos de tarefas vindos das subrotinas do módulo *J4R*.

As duas características principais do presente trabalho podem ser vistas como consequência de diversos trabalhos já realizados tanto na área de robôs industriais como robôs móveis, sendo sua validação encontrada nos trabalhos de (PARK et al., 2008; ANDROID, 2005; ZIELINKSI, 2005; CÔTÉ et al., 2004; BRÄUNL, 2003; FALCONE et al., 2003; FRAUNHOFER, 2003; BREAZEAL; SCASSELLATI, 2002; SCATENA, 2002; ZIELINKSI, 1999).

Além dessas duas características, o presente trabalho inclui recursos de integração de diversas plataformas robóticas, sendo que algumas dessas características são encontradas em alguns simuladores de robôs móveis, como por exemplo, o simulador *Player/Stage* e o *WeBots*.

Com base na avaliação do ambiente implementado foram identificadas as seguintes limitações no ambiente desenvolvido:

- ✓ O editor *J4R* não correlaciona as tarefas criadas automaticamente com a linguagem do sistema robótico;

- ✓ A linguagem *Java2* possui restrições a diversas características da programação orientada a objetos;
- ✓ A interface *RDC* suporta apenas linguagens de programação estruturadas como modelo de correlação;
- ✓ O editor *Wizard* não permite que os blocos utilizados sejam manipuláveis, ou seja, a estrutura criada no editor segue a sequência inserida pelo usuário.

Essas limitações podem ser utilizadas como propostas de estudos futuros, de forma a evoluir o ambiente *ADARM* para um projeto mais robusto. Apesar de suas limitações de implementação, o ambiente *ADARM* está estruturado de forma flexível, facilitando assim, a sua evolução.

9.2. *Contribuições*

As contribuições relevantes do ambiente *ADARM* para a área de robôs móveis são:

- ✓ Linguagem de alto nível de abstração, que possui subrotinas direcionadas às tarefas de robôs móveis;
- ✓ Editor de aplicações, que utiliza blocos de tarefas para a criação das aplicações.

Esse editor destina-se a usuários com pouco conhecimento em programação ou para programadores que buscam uma ferramenta de desenvolvimento rápido;

Além das contribuições mencionadas nas conclusões (Seção 8.1) referente ao módulo *J4R* e ao módulo *Wizard*, as seguintes contribuições podem ser citadas:

- ✓ A conversão dos blocos do editor *Wizard* em ações existentes na biblioteca *J4R*, provendo uma convergência a nível da linguagem do ambiente e desconsiderando as características do sistema robótico, permitindo assim, a utilização da mesma tarefa em diversos robôs;
- ✓ A criação de bibliotecas contendo as correlações entre a linguagem utilizada pelo ambiente (*Java2+J4R*) com a linguagem do robô.

9.3. *Propostas para trabalhos futuros*

As limitações mencionadas na seção 8.1 podem ser exploradas como temas futuros de pesquisa. Essas propostas estão formulados com segue:

- ✓ O *Wizard editor* possui uma restrição na interação do usuário com os blocos do sistema. Além dessa alteração estrutural, o ambiente poderia ficar mais “amigável” aos usuários, utilizando técnicas avançadas de estruturas gráficas em *Java2*;
- ✓ No *Expert Editor* a melhoria proposta seria na apresentação do editor, melhorando a interface, de forma a conter os recursos comumente encontrados em outros editores. Estruturalmente, uma proposta interessante a este projeto seria uma melhoria das características utilizadas pela linguagem *Java2*, como o uso de herança múltipla ou uma estrutura de vinculação de todas as bibliotecas em uma única classe do sistema robótico;

- ✓ O editor *J4R*, pelo fato de criar tarefas aos sistemas, requer uma reestruturação na interconexão com o módulo *RDC*, pois, quando novas tarefas são adicionadas ao sistema, o editor *J4R* realiza as inclusões necessárias para essas tarefas, porém se essas tarefas não forem descritas nas linguagens do robô e do simulador, o ambiente não conseguirá converter a aplicação;
- ✓ Editor *RDC* é o módulo responsável pelas conversões do sistema. Esse editor foi estruturado para suportar linguagens de programação seqüenciais ou estruturadas, pois essas linguagens são semelhantes à linguagem *Java2*. Devido a isso, esse editor poderia ser reestruturado para suportar o paradigma orientado a objetos sem restrições, suportar linguagens de programação não sequenciais, como por exemplo, Prolog, e alertando o programador quando for incluída uma nova tarefa (Editor *J4R*), solicitando ao mesmo, a correlação daquela tarefa às linguagens do sistema robótico e simulador.

Além das melhorias propostas nos principais módulos, existem melhorias que estão relacionadas ao sistema em geral que podem ser interessantes, como por exemplo, a execução do simulador *Player/Stage* com os códigos gerados no ambiente e o *download* da aplicação convertida para o sistema robótico.

Referências Bibliográficas

ActivMedia Robotics. 2005. Aria Simulator. *ActivRobots*. [Online] ActivMedia Robotics, Novembro 14, 2005. [Visitado: Novembro 14, 2005.] <http://www.activrobots.com/SOFTWARE/aria.html>.

Aleotti, J., Caselli, S. e Reggiani, M. 2004. Leveraging on a Virtual Environment for Robot Programming by Demonstration. *Robotics and Autonomous Systems*. May, 2004, 47.

Android World. 2005. Domestic Robots or Androids. *Android World*. [Online] Android World, 09 de Julho de 2005. [Visitado em: 19 de Julho de 2005.] <http://www.androidworld.com/prod27.htm>.

Asimov, I. 1939. *Asimov The Robot Series - I Robot*. s.l. : New Ed edition Voyager, 1939.

Atkeson, C. G., et al. 2000. Using Humanoid Robots to Study Human Behavior. *IEEE Intelligent Systems*. 2000, Vol. 15, July/August 2000.

Balaguer, C., et al. 2006. The MATS robot: service climbing robot for personal assistance. *IEEE Robotics & Automation Magazine*. 2006, Vol. 13, 1.

Bishoff, R. 2000. *Towards the Development of 'Plug-and-Play' Personal Robots*. Neubiberg : s.n., 2000.

Bottazzi, V. S. e Fonseca, J. F. C. 2005. Off-Line Robot Programming Framework. *ICAS 2005 - IEEE*. 19/12/2005, 2005, 2005.

Breazeal, C. e Scassellati, B. 2002. Robots that Imitate Humans. *Trends in Cognitive Sciences*. 2002, Vol. 6, 11.

Brooks, R. A., et al. 1999. *The Cog Project: Building a Humanoid Robot*. Cambridge : IEEE ICRA, 1999.

Chaimowicz, L., et al. 2003. ROCI: A Distributed Framework for Multi-Robot Perception and Control. *IEEE/RSJ Intl. Conference on Intelligent Robots and Systems*. October, 2003.

- Cohen, R. e Peleg, D. 2008.** Local Spreading Algorithms for Autonomous Robot Systems. *Science Direct - Theoretical Computer Science*. 2008, 399.
- Coradeschi, S., et al. 2006.** Human-Inspired Robots. *IEEE Intelligent Systems*. 2006, Vol. 21, 4.
- Côté, C., et al. 2004.** Code Reusability Tools for Programming Mobile Robots. *IEEE/RSJ International Conference on Intelligent Robots and Systems*. October, 2004, 06/04.
- Cowley, A., Chaimowicz, L. e Camillio, J. T. 2006.** Design Minimalism in Robotics Programming. *International Journal of Advanced Robotic Systems*. 2006, Vol. 3, 1.
- Cyberbotics.** WebBots Simulator. *CyberBotics*. [Online] CyberBotics. [Visitado em: 10 de Outubro de 2005.] <http://www.cyberbotics.com/products/webots/webots5.pdf>.
- Dario, P., Guglielmelli, E. e Laschi, C. 2001.** Humanoids and Personal Robots: Design and Experiments. *Journal of Robotic Systems - Special Issue: Biorobotics and Humanoid Robotics*. 2001, Vol. 18, 12.
- Domínguez-Brito, A. C., et al. 2005.** Another Component Base Programming Framework for Robotics. *IEEE/RSJ International Conference on Intelligent Robots and Systems*. August, 2005.
- Drumwright, E.** The Task Matrix: A Robot-Independent Framework For Programming Humanoids. *Ph.D Thesis*. University of Southern California, 2007.
- École Polytechnique Fédérale de Lausanne.** Khepera Simulator. *Khepera Simulator*. [Online] [Visitado em: 15 de Agosto de 2005.] <http://diwww.epfl.ch/lami/team/michel/khep-sim>.
- Bräunl, T. 2003.** *Embedded Robotics - Mobile Robot Design and Applications with Embedded Systems*. Junho, s.l. : Springer, 2003. ISBN:3-540-03436-6.
- Falcone, E., et al. 2003.** The Personal Rover Project: The Comprehensive Design of a Domestic Personal Robot. *Robotics and Autonomous Systems*. March, 2003, Vol. 42, 4.
- Forlizzi, J. e DiSalvo, C. 2006.** Service Robots in the Domestic Environment: a Study of the Roomba Vacuum in the Home. *1st ACM SIGCHI/SIGART Conference on Human-Robot Interaction*. 2006.

Fraunhofer IPA. 2003. Care-O-bot Tasks. *Care-O-bot*. [Online] Fraunhofer IPA, Setembro de 11 de 2003. [Visitado em: 14 de Novembro de 2005.] <http://www.care-o-bot.de/english/Tasks.php>.

Freund, E., et al. 2001. Creating the Architecture of a Translator Framework for Robot Programming Languages. *International Conference on Robotics & Automation*. May, 2001, 05/01.

Friendly Robotics Corporation. Robomow Lawn Mower. *Friendly Robotics*. [Online] Friendly Robotics. [Visitado em: 10 de 07 de 2007.] <http://www.friendlyrobotics.com/robomow/>.

Gerkey, B. P., Vaughan, R. T. e Howard, A. 2003. The *Player/Stage* Project: Tools for Multi-Robot and Distributed Sensor Systems. *IEEE International Conference on Advanced Robotics - ICAR*. July, 2003.

Gorostiza, J. F., et al. 2006. Multimodal Human-Robot Interaction Framework for a Personal Robot. *The 15th IEEE International Symposium on Robot and Human Interactive Communication*. September, 2006.

Graefe, V. e Bishoff, R. 2002. *Three Examples of Learning Robots*. Neubiberg: Bundeswehr University Munich, Institute of Measurement Science, 2002.

Graf, B., Hans, M. e Schraft, R. D. 2004. Care-O-bot II—Development of a Next Generation Robotic Home Assistant. *Autonomous Robots Journal*. March, 2004, Vol. 16, 2.

Ido, J., et al. 2002. A Robotic Receptionist ASKA: A Research Platform for Human-Robot Interaction. *IEEE Robot and Human Interactive Communication*. 11, 2002.

Interesse, M. e Distanti, A. 1988. A Flexible Robots Programming Environment With Simulation Capabilities. *IEEE International Workshop on Intelligent Robots*. November, 1988.

Johnson, C. G. e Marsh, D. 1998. A Robot Programming Environment Based on Free-Form CAD Modelling. *International Conference on Robotics & Automation*. May, 1998, 05/98.

Kim, J. H. e Kim, J. O. 2004. A Task Management Design for Task-Based Control Architecture for Personal Robots. *IEEE Industrial Electronics Society - IECON*. 2004, Vol. 3.

Kotay, K. D. Self-reconfigurable Robot Systems - Ph.d Thesis. *Ph.D Thesis*. Dartmouth College - New Hampshire, 1998.

Kramer, J. e Scheutz, M. 2007. Development Environments for Autonomous Mobile Robots: A Survey. *Autonomous Robots*. 2007, Vol. 22, 2.

Kröse, B. J.A., et al. 2003. *Lino, the User-Interface Robot*. Berlin / Heidelberg : Springer Berlin / Heidelberg, 2003. Vol. 2875/2003. 978-3-540-20418-3.

LG Eletronics. LG V-R4000. *LG Global Site*. [Online] LG Eletronics. [Visitado em: 30 de Maio de 2007.] <http://www.lge.com/products/model/detail/v-r4000.jhtml>.

MacDonald, B., et al. 2003. Robot Programming Environments. *ENZCon2003 10th Electronics New Zealand Conference*. University of Waikato, Hamilton : ENZCon2003 10th Electronics New Zealand Conference, 2003.

Mazer, E., et al. 1991. ACT: A Robot Programming Environment. *International Conference on Robotics & Automation*. April, 1991, 04/91.

MobotSoft. MobotSim Mobile Robot Simulator. *MobotSoft*. [Online] MobotSoft. [Visitado em: 29 de Novembro de 2005.] <http://www.mobotsoft.com/mobotsim.htm>.

Morris, K. 2006. C to FPGA Altera Accelerate NIOS II. *FPGA and Structured ASIC Journal*. [Online] FPGA and Structured ASIC Journal, 04 de Abril de 2006. [Visitado em: 20 de Julho de 2006.] http://www.fpgajournal.com/articles_2006/20060404_nios.htm.

NEC Laboratories. NEC Laboratories - Papero. *Personal Robot Research Center*. [Online] NEC Laboratories. [Visitado em: 19 de Novembro de 2005.] http://www.incx.nec.co.jp/robot/english/robotcenter_e.html.

Nguyen, H. 1999. The Flat 2D Simulator. *The Flat 2D Simulator*. [Online] 15 de Fevereiro de 1999. [Visitado em: 15 de Agosto de 2005.] <http://www.cs.utexas.edu/users/qr/robotics/flat/>.

Park, J., et al. 2006. Design and Implementation of Service-Oriented Task Model for Autonomous Service Robot. *IEEE International Joint Conference SICE-ICASE*. October, 2006.

Park, K., et al. 2008. A Steward Robot for Human-Friendly Human-Machine Interaction in a Smart House Environment. *IEEE Transactions on Automation Science and Engineering*. January, 2008, Vol. 5, 1.

Pyro Robotics. *Pyro, Python Robotics. Pyro, Python Robotics.* [Online] *Pyro Robotics.* [Visitado em: 24 de Julho de 2007.] <http://Pyrorobotics.org/?page=Pyro>.

Remote-Controlled, 2004. Remote-Controlled, Throwable Robots Developed at Carnegie Mellon in Conjunction with U.S. Marine Corps Are Being Sent to Iraq for Testing. *Carnegie Mellon Press Release: June 23, 2004.* [Online] Carnegie Mellon, 23 de June de 2004. [Visitado em: December de 10 de 2006.] http://www.cmu.edu/PR/releases04/040623_dragonrunner.html.

Roh, S. G., et al. 2004. Development of Dynamically Reconfigurable Personal Robot. *IEEE International Conference on Robotics & Automation*. April, 2004, Vol. 4.

Salemi, B., et al. 2005. MILO: personal robot platform. *IEEE/RSJ International Conference on Intelligent Robots and Systems, 2005. (IROS 2005)*. August, 2005, Vol. 1.

Salichs, M. A., et al. 2006. Maggie: A Robotic Platform for Human-Robot Social Iteration. *IEEE Conference on Robotics, Automation and Mechatronics*. 2006-12-04, 2006.

Sato, M., Sugiyama, A. e Ohnaka, S. 2006. Auditory System in a Personal Robot, PaPeRo. *International Conference on Consumer Electronics, ICCE*. 2006, Vol. 7.

Scatena, J. M. 2002. Implementação de Mapas Topológicos para Navegação de Robôs Móveis baseadas em Computação Reconfigurável - Dissertação de Mestrado. *Implementação de Mapas Topológicos para Navegação de Robôs Móveis baseadas em Computação Reconfigurável*. São Carlos : USP - ICMC, 2002.

- Scatena, J., et al. 2004.** Desenvolvimento de um Ambiente de Programação e Simulação de Robôs Móveis Baseados em Hardware Reconfigurável. *Simpósio Latino Americano em Aplicações de Lógica Programável e Processadores Digitais de Sinais em Processamento de Vídeo, Visão Computacional e Robótica - SLALP*. 2004, Vol. 1.
- Shepherd, B. 1993.** Applying Visual Programming to Robotics. *International Conference on Robotics & Automation*. May, 1993, 05/93.
- Silva, A. L.V.** Arquitetura Compacta Para Projeto de Robôs Móveis Visando Aplicações Multipropósitos. *Dissertação de Mestrado*. Universidade de São Paulo (EESC) - Brasil, 2007.
- Tmsuk Co., Ltd.** Tmsuk Co., Ltd. . *Tmsuk Co., Ltd.* . [Online] [Visitado em: 07 de Setembro de 2005.] <http://www.tmsuk.co.jp/english/robots.html>.
- Utz, H., et al. 2002.** Miro - Middleware for Mobile Robot Applications. *IEEE Transactions on Robotics and Automation*. 2002, Vol. 18, 4.
- van Breemen, A. J.N., et al. 2003.** A User-Interface Roboto for Ambient Intelligent Environments. *1st International Workshop on Advances in Service Robotics - ASER*. March, 2003, Vol. 1.
- Zielinski, C. 1999.** The MRROC++ System. *First Workshop on Robot Motion and Control, RoMoCo*. June, 1999.
- Zielinski, C., Szyrkiewicz, W. e Winiarski, T. 2005.** Applications of MRROC++ Robot Programming Framework. *Fifth International Workshop on Robot Motion and Control*. June, 2005.