UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

FERNANDO HENRIQUE CARDOSO

fwWorkCell: Framework para a Construção de Simuladores Didáticos de Células de Trabalho

Trabalho de Conclusão de Mestrado apresentado como requisito parcial para a obtenção do grau de Mestre em Informática

Prof. Marcelo Soares Pimenta
Orientador

Porto Alegre, junho de 2005.

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Cardoso, Fernando Henrique

fwWorkCell: Framework para construção de Simuladores de Células de Trabalho/ Fernando Henrique Cardoso. – Porto Alegre: PPGC da UFRGS, 2005.

68 f.: il

Trabalho de Conclusão (Mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2005. Orientador: Marcelo Soares Pimenta.

1. Engenharia de Software. 2. *Frameworks*. 3. Braços Robóticos. 4. Células de Trabalho. 5. Simuladores. I. Pimenta, Marcelo Soares. II. Título.

DEDICATÓRIA

Esta obra é dedicada a minha amada esposa, Custódia,
que tanto me incentivou durante esta longa jornada.
Aos meus pais, Luiz e Maurília, que me apoiaram e
acreditaram em todas as minhas decisões.
Aos meus irmãos, Michel e Júnior, que me ensinaram
o verdadeiro significado da palavra amizade.
E a minha avó paterna, Terezinha (*in memorian*), que
sempre esteve ao meu lado em todos os momentos e,
juntamente com meus pais e minha esposa, foi a
pessoa que mais amei na minha vida.

AGRADECIMENTOS

Agradeço a Deus por ter me dado saúde e força para trilhar os caminhos da vida. Ao meu professor, orientador e amigo, Marcelo Pimenta, por ter tido paciência e compreensão, além de me ajudar no desenvolvimento de minhas idéias. Agradeço também a minha família, base tão essencial de minha vida. Aos meus pais, irmãos, avós, cunhada e sobrinha, por tudo que fizeram e ainda farão para tornar minha vida tão feliz. À minha esposa, pessoa tão importante e que está sempre ao meu lado nesta caminhada. À minha segunda família , Vanderlei, Gládis, Renata e Rosana, que me acolheram e deixaram eu me tornar parte de sua família, quando vim aqui para Porto Alegre. Aos meus colegas do SENAI-RS, SICREDI e do curso de mestrado que tanto contribuíram com suas experiências e amizades. Enfim, a todos que de um jeito ou de outro, contribuíram para a conclusão desta minha mais importante obra.

SUMÁRIO

LISTA DE ABREVIATURAS

ACL	*Advanced Control Language*
JVM	*Java Virtual Machine*
NIEd	*Núcleo de Informática Educacional*
SENAI-RS	*Serviço Nacional de Aprendizagem Industrial do Rio Grande do Sul*
XML	*eXtensible Markup Language*
PPP	*Deslizante/Deslizante/Deslizante*
RPP	*Revolução/Deslizante/Deslizante*
RRP	*Revolução/Revolução/Deslizante*
RRR	*Revolução/Revolução/Revolução*
CAD	*Computer Aided Design*
GOF	*Gang of Four*
ORB	*Object Request Broker*
CORBA	*Commom Object Request Broker Architecture*
RML	*Reliable Multicast Library*
TP	*Teach Pendant*

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Braços robóticos articulados são cada vez mais utilizados hoje em dia e consistem de dispositivos mecânicos programáveis, equipados com sensores e atuadores sob o controle de um sistema computacional. Existem atualmente no mercado inúmeros fabricantes e modelos destes braços, cada um adequado a uma determinada utilização ou faixa de mercado.

Para que se saiba operar devidamente este robô é necessário um período de aprendizagem. Essa necessidade pode ser suprida pelo emprego dos simuladores de braços robóticos.Desenvolver um simulador é uma atividade complexa, mas alguns elementos de sua estrutura e de seu comportamento são comuns a vários tipos de simuladores e podem idealmente ser reusados. Permitir reuso de código e de projeto é exatamente um dos principais fatores que motivaram a construção de um *framework.*

Este trabalho descreve a definição e a construção do fwWorkCell , um framework que permita agilizar a construção destes simuladores. Tal agilidade será obtida através da implementação de um ambiente de edição e de classes genéricas para controle, visualização e programação dos robôs. A proposta deste framework inclui definição de classes genéricas e de controle, a construção de todo um ambiente de suporte à manipulação e visualização das células de trabalho e suas simulações e visa dar suporte à construção de uma grande variedade de simuladores. O framework proposto foi utilizado em uma aplicação real: através dele foi feita a migração de um simulador já existente.

Palavras-chave: Simuladores didáticos de braços robóticos, Framework, Robótica, Engenharia de Software.

fwWorkCell: Framework for Building Didatic Simulators of Work Cells

ABSTRACT

In the present days industry needs equipment with essencial aspects as accuracy, efficiency, effectiveness and reliability in order to perform repetitive or dangerous tasks. That is the main reason for articulated robotic arms usage. An articulated robotic arm is a programmable electromechanical device, running under control of a computerized system. Its function is to reproduce highly-complex or dangerous movement sequences to emulate the behaviour of a human arm. There are many robotic arm models and providers: the choice of one specific device depends on the price and the specific goals.

Learning to operate an articulated robotic arm requires an specific training usually involving robotic arm simulators, since it is possible to configure a simulator to simulate any specific robotic arm. Building simulators is a complex activity, but some elements of its structure and behaviour are common to many kinds of simulators and may be reused.

This work describes the definition and implementation of fwWorkCell, a framework whose main goal is to turn easier the process of robotic arm simulators.

The present dissertation contains basic concepts about robotic frameworks and the complete definition for the generic classes and methods of fwWorkCell. It also contains the description of the environment related to this framework. This definition intends to support the construction of a great variety of simulators.

FwWorkCell was applied to the study of a real case: the migration of an existing simulator to this new platform and language.

Keywords: Robotic arm simulators, Framework, Robotics, Software Engineering

1 INTRODUÇÃO

A necessidade da indústria por máquinas de grande precisão, eficiência, qualidade, confiabilidade e repetibilidade e que sejam responsáveis por tarefas que trariam riscos à saúde do ser humano, aliada ao vertiginoso crescimento do potencial da robótica industrial, fez difundir a utilização dos braços robóticos articulados. Uma demonstração disto é o seu crescente uso na indústria automobilística: robôs estão presentes em praticamente toda a linha de montagem, desde o corte das chapas até a pintura dos automóveis.

Braços robóticos articulados são dispositivos mecânicos programáveis, equipados com sensores e atuadores sob o controle de um sistema computacional [ARA 2003]. Sua única função é reproduzir seqüências de movimentos altamente complexos, como agarrar, deslocar, soltar, etc., imitando um membro superior humano [SAB 99].

Existem atualmente no mercado inúmeros fabricantes e modelos destes braços, cada um adequado a uma determinada utilização ou faixa de mercado. Eles podem variar em número de graus de liberdade, tamanho dos elos e peso suportado pelo braço, por exemplo.

Para que estes braços articulados possam realizar suas tarefas é ainda necessário a intervenção humana, responsável por posicioná-lo e programá-lo. Mas haverá a necessidade que se saiba operar devidamente este robô, ou seja, é necessário um período de aprendizagem Essa necessidade pode ser suprida pelo emprego dos simuladores de braços robóticos.

Esses simuladores existem em uma grande diversidade: vão desde simuladores comerciais até os simuladores gratuitos. Simulam de robôs educacionais, utilizados em escolas profissionalizantes, até robôs de grande porte, usados nas linhas de produção de grandes indústrias. Em função disto, a comunidade científica vem produzindo constantemente estudos de técnicas como o desenvolvimento de editores nos quais as células construídas possam ser executadas através de técnicas de animação e criação de ferramentas capazes de receber uma especificação e a partir disto construir automaticamente editores.

Para funcionar de maneira genérica, uma ferramenta para a construção de células de trabalho que podem ser executadas deve receber as especificações do(s) robô(s), tais como quantidade de juntas, cinemáticas direta e inversa, representação gráfica e linguagem de programação suportada. Desenvolver um simulador é uma atividade complexa, mas alguns elementos de sua estrutura e de seu comportamento são comuns a vários tipos de simuladores e podem ser reusados.

Permitir reuso de código e de projeto é exatamente um dos principais fatores que motivaram a construção de um *framework.*

Este trabalho pretende desenvolver um framework que permita agilizar a construção destes simuladores. Tal agilidade será obtida através da implementação de um ambiente de edição e de classes genéricas para controle, visualização e programação dos robôs.

O framework gerado deverá ser genérico o suficiente para permitir suporte a projeto e à implementação de uma grande variedade de simuladores de diversos tipos de braço robótico.

Este trabalho está estruturado como segue:

Os capítulos 2 e 3 apresentam conceitos e fundamentos, respectivamente, de Robótica e Simuladores e de Frameworks, visando criar uma base conceitual e terminológica para a compreensão do restante do trabalho.

O capítulo 4 apresenta uma visão geral, a arquitetura e as características principais do *fwWorkCell*, proposto neste trabalho.

O capítulo 5 descreve uma aplicação do fwWorkCell: sua utilização na implementação do Asimov 1.0, um simulador didático de células de trabalho de um braço robótico, desenvolvido no SENAI-RS.

O capítulo 6 contém a conclusão do trabalho.

2 ROBÓTICA E SIMULADORES: CONCEITOS E FUNDAMENTOS

Neste capítulo serão apresentadas algumas definições e conceitos de robótica e simuladores de forma a permitir aos leitores uma visão geral sobre o tema.

2.1 Robótica

Trabalho forçado – esse é o significado para a palavra de origem tcheca *robot* – em português **robô**. E automatizar tarefas que são executadas pelo homem é o principal objetivo da robótica.

Realizar determinada tarefa através de ações independentes sem ser continuamente supervisionada por um operador humano é o que diferencia os robôs de outras máquinas.

O desenvolvimento tecnológico da microeletrônica, somado com o desenfreado avanço das linguagens e programas de computador, colocam-nos perante a chamada *Inteligência Artificial* já num estágio muito evoluído, permitindo que um computador adquira conhecimentos da sua própria experiência, o que faz parecer que um robô se comporta com inteligência. Contudo, um robô não pode pensar como nós humanos [SER 2003].

Essencialmente, um robô é composto por uma parte mecânica e outra eletrônica. A mecânica é constituída por base fixa ou móvel, por braços e garras (manipuladores) possuindo, em geral, diversos graus de liberdade de movimento de translação e/ou rotação, emulando, não raramente, um braço humano. E a parte eletrônica é um computador constituído, basicamente, por uma unidade central de processamento apoiada numa memória interna que armazena instruções e dados previamente preparados. As portas de entrada e saída estão conectadas a motores, micro-interruptores, sensores de proximidade, transdutores/detectores de temperaturas, de cheiros, sons, pesos, pressões, presença e reconhecimento de imagens [HIS 2003].

2.1.1 História

O conceito de robô já é conhecido desde o início da história, quando os mitos faziam referência a mecanismos que ganhavam vida.

Na civilização grega já eram encontrados modelos com aparência de animais e/ou humanos na qual utilizavam sistemas de pesos e bombas pneumáticas cuja necessidade prática ou econômica era nula.

Atribuir funções úteis às necessidades humanas aos robôs foi o novo e importante conceito acrescentado pelos cientistas árabes.

A criação de bonecos que conseguiam realizar simples ações como escrever ou tocar alguns instrumentos através do movimento das mãos, olhos e pernas só foi possível com o emprego de articulações mecânicas. Articulações estas que surgiram com a extensiva pesquisa da anatomia humana realizada por Leonardo DaVinci.

A palavra robô foi introduzida pelo dramaturgo Karel Capek em sua peça *R.U.R (Rossum's Universal Robots)*. E Isaac Asimov, em 1942, foi o primeiro a usar o termo robótica referindo-se ao estudo e à utilização de robôs, numa pequena história chamada *Runaround*. Logo depois, em 1950, publicou *I Robot*, uma compilação de pequenas histórias. Asimov propôs ainda, três leis aplicáveis à robótica que mais tarde foi acrescida da lei zero [SAN 2003]. São elas:

Lei Zero – Um robô não pode causar mal à humanidade ou, por omissão, permitir que a humanidade sofra algum mal, nem permitir que ela própria o faça.

Lei 1 – Um robô não pode ferir um ser humano ou, por omissão, permitir que um ser um humano sofra algum mal.

Lei 2 – Um robô deve obedecer às ordens que lhe sejam dadas por seres humanos, exceto nos casos que tais ordens contrariem a primeira lei.

Lei 3 – Um robô deve proteger sua própria existência, desde que tal proteção não entre em conflito com a primeira e segunda leis.

Os teares mecânicos da indústria têxtil do começo do século XVIII foram o primeiro esforço para a automatização das operações industriais. Esforço esse que evoluiu com o progresso da revolução industrial. Mas apenas em 1940 com a invenção do computador e de seus sucessivos aperfeiçoamentos que foi possível a criação dos verdadeiros robôs.

No final da década de 50, início da década de 60, George Devol e Joe Engleberger desenvolveram *Unimates*, o primeiro robô industrial. Engleberger pela construção do primeiro robô comercial ficou conhecido como o *pai da robótica* e Devol obteve as primeiras patentes de máquinas transportadoras, robôs primitivos que moviam objetos de um local para outro.

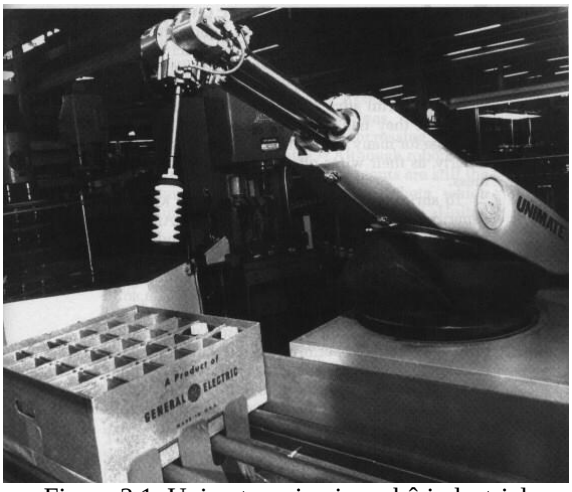

Figura 2.1: Unimate, primeiro robô industrial

O modelo experimental Shakey desenhado para pesquisas em Standford, foi outro dos primeiros robôs a surgir no final da década de 60. E ele ainda é utilizado hoje para pesquisas.

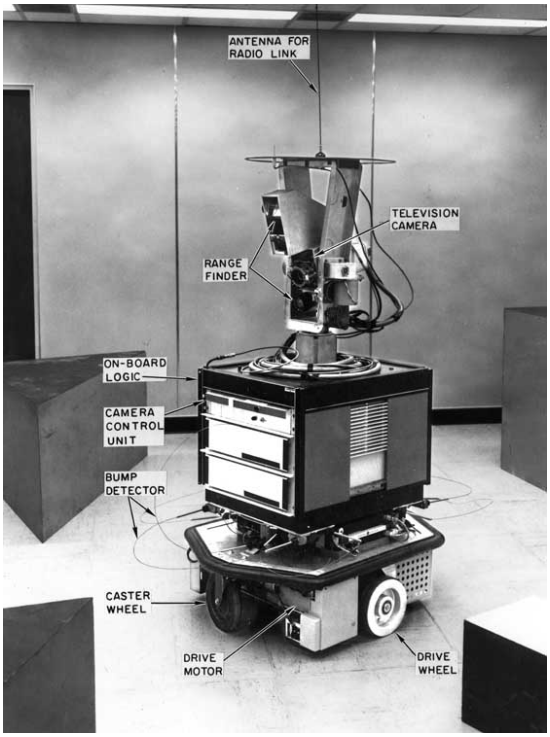

Figura 2.2: Shakey

2.1.2 Classificação geral dos robôs

Devido a várias diferenças em função de características e propriedades, existem diversas classes de robôs que se diferenciam em suas aplicações e formas de trabalhar.

Tipos de robôs:

- *Robôs inteligentes*: manipulados por sistemas multifuncionais controlados por computador, capazes de interagir com o seu ambiente através de sensores e de tomar decisões em tempo real.
- *Robôs com controle por computador*: semelhantes aos inteligentes, mas sem a capacidade de interagir com o ambiente.
- *Robôs de aprendizagem*: limitam-se a repetir uma seqüência de movimentos previamente armazenados em sua memória.
- *Manipuladores*: sistemas mecânicos multifuncionais, cujo sensível sistema de controle, permite governar o movimento de seus membros.

Se for considerada a perspectiva do controle dos movimentos dos robôs, teremos a seguinte classificação:

- *Sem servo-controle*: sem programa que controla o movimento dos diferentes componentes do robô (servo-controle). Realiza-se um posicionamento ponto-a-ponto no espaço.
- *Com servo-controle*: subdivide-se em duas formas de trabalho:
  - Controle dos movimentos dos membros do robô em função de seus eixos. Esses movimentos podem ser realizados ponto-a-ponto ou com uma trajetória continua.
  - Os movimentos se estabelecem da respectiva posição de seus eixos de coordenada e da orientação da ferramenta do robô.

2.1.3 Braço mecânico

Manipulador projetado para realizar e repetir diferentes tarefas através do movimento de suas partes, respeitando um trajeto pré-programado. O responsável pelo controle desses movimentos é um computador que armazena em sua memória, a trajetória que o robô deverá seguir e vai enviando sinais que ativam motores que deslocarão o braço.

Todo braço mecânico é composto por uma série de elos e juntas. Cada junta conecta dois elos, permitindo um movimento relativo entre eles. E é a quantidade desses elos e juntas que determinará a mobilidade do robô. Possui ainda, uma base fixa que tem o primeiro elo preso a ela.

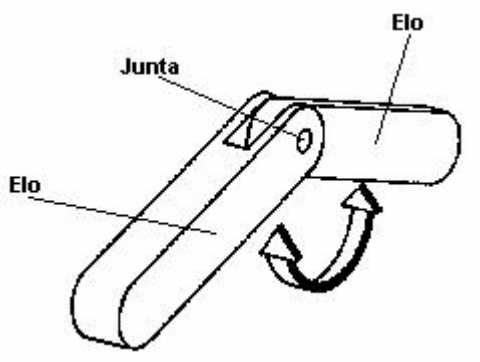

Figura 2.3: Elos e juntas

2.1.3.1 Tipos de juntas

Juntas deslizantes, de rotação e de bola e encaixe, são os três tipos de juntas que podem formar um robô. Sendo que a maioria deles é constituída por juntas deslizantes e de revolução. A seguir, será descrito cada tipo de junta.

2.1.3.1.1 Juntas deslizantes

Composto por dois elos alinhados um dentro do outro, onde o interno escorrega pelo externo, caracterizando um movimento linear.

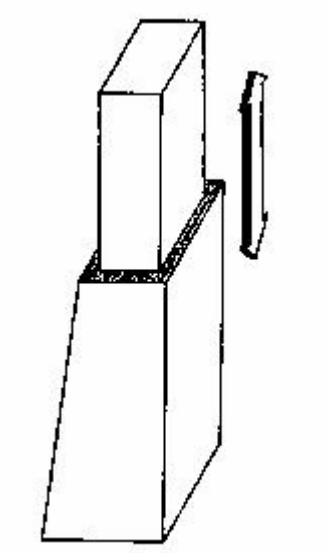

Figura 2.4: Junta deslizante

2.1.3.1.2 Juntas de rotação

Permite movimentos de rotação entre dois elos que são unidos por uma dobradiça comum, com uma parte podendo se mover num movimento cadenciado em relação à outra parte.

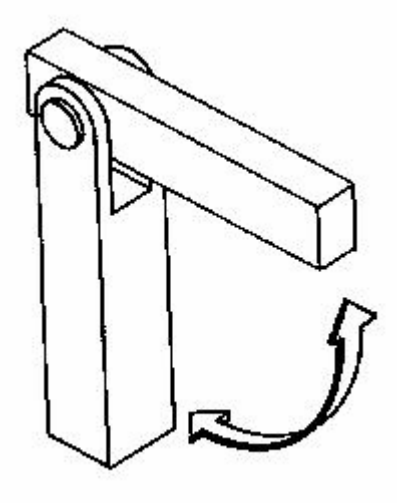

Figura 2.5: Junta de rotação

2.1.3.1.3 Juntas de bola e encaixe

Permite movimentação de rotação em torno dos três eixos através da combinação de três juntas de rotação.

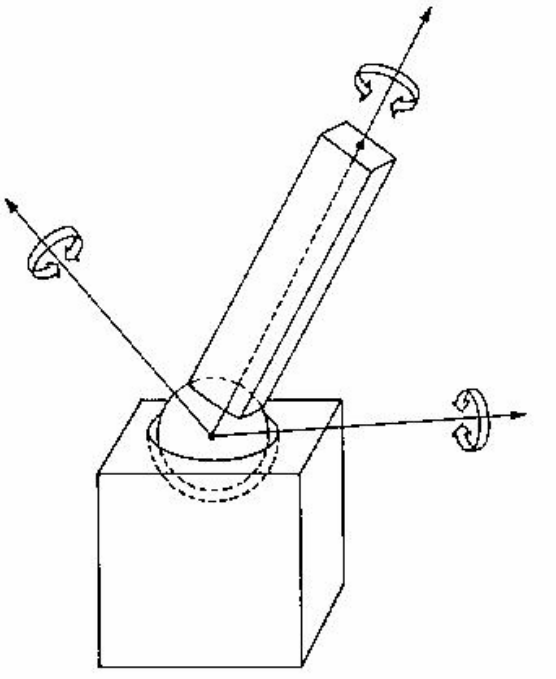

Figura 2.6: Junta de bola e encaixe

Devido à dificuldade de ativação, as juntas de bola e encaixe não são muito utilizadas, sendo substituídas por três juntas rotacionais separadas, cujos eixos de movimentação cruzam-se em um ponto.

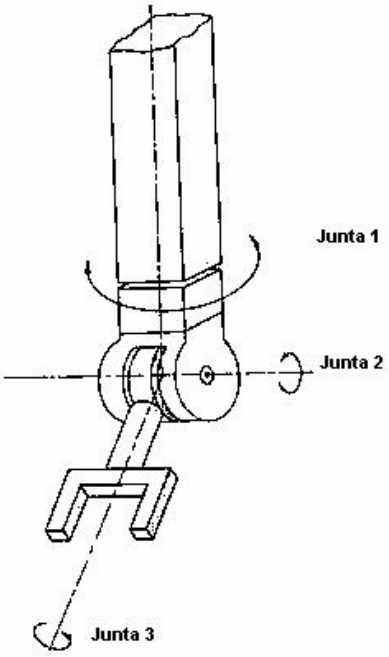

Figura 2.7: Três juntas rotacionais substituindo a junta de bola e encaixe

2.1.3.2 Graus de Liberdade

Quando o movimento relativo da junta ocorre em apenas um eixo, diz-se que esta possui um grau de liberdade. Quando ocorre em dois ou três eixos, o número de graus de liberdade é dois ou três, respectivamente.

O número de graus de liberdade de um robô é a soma das quantidades de graus de liberdade de suas juntas. A maioria dos braços robóticos possuem entre quatro e 6 graus de liberdade, já o homem, do ombro até o pulso, tem 7 graus de liberdade.

2.1.3.3 Classificação dos braços robóticos pelo tipo de articulação

A fim de fornecer informações sobre características dos robôs em categorias como *espaço de trabalho, grau de rigidez* e *extensão de controle sobre o curso de movimento*, classificou-se os braços robóticos de acordo com o tipo das três juntas mais próximas da base do robô.

Abaixo serão citados e descritos os cinco grupos e cada um virá acompanhado de um código de três letras que se referem ao tipo de junta (*R = revolução* e *P = deslizante*) na ordem em que ocorrem, começando pela junta mais próxima à base.

2.1.3.3.1 Robôs cartesianos (PPP)

Possuem três articulações deslizantes que junto com o momento de inércia da carga fixa por toda a área de atuação, permitem um controle simples. Caracterizam-se ainda, pela pequena área de trabalho, elevado grau de rigidez mecânica e grande exatidão.

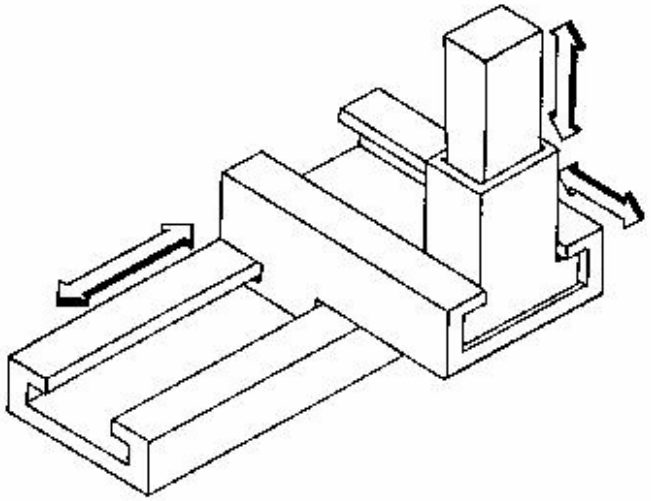

Figura 2.8: Robô cartesiano

2.1.3.3.2 Robôs cilíndricos (RPP)

Possuem uma junta de revolução e duas deslizantes e uma área de trabalho maior que a dos robôs cartesianos, porém uma rigidez mecânica um pouco menor. Seu controle é ligeiramente mais complicado devido a rotação da junta da base e a vários momentos de inércia para diferentes pontos na área de trabalho.

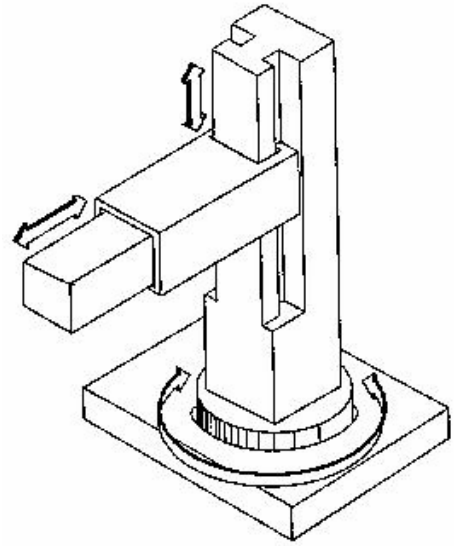

Figura 2.9: Robô cilíndrico

2.1.3.3.3 Robôs esféricos (RRP)

Possuem duas juntas de revolução e uma deslizante e uma área de trabalho maior que os modelos cilíndricos, porém perde também na rigidez mecânica. Os movimentos de rotação deixam o seu controle ainda mais complicado.

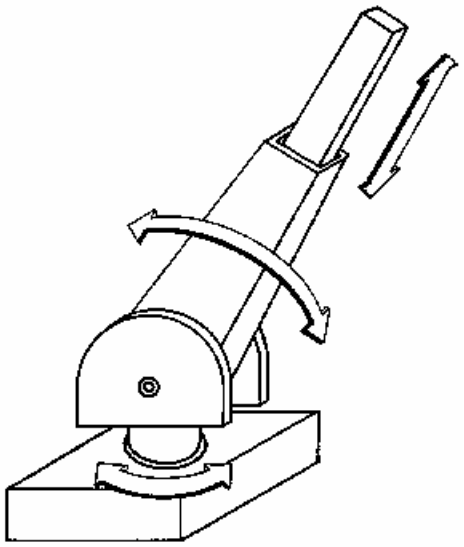

Figura 2.10: Robô esférico

2.1.3.3.4 Robôs com articulação horizontal (RRP)

Possuem duas juntas de revolução e uma deslizante e uma área de trabalho menor que no modelo esférico. É bastante apropriado para operações de montagem pelo movimento linear vertical do seu terceiro eixo.

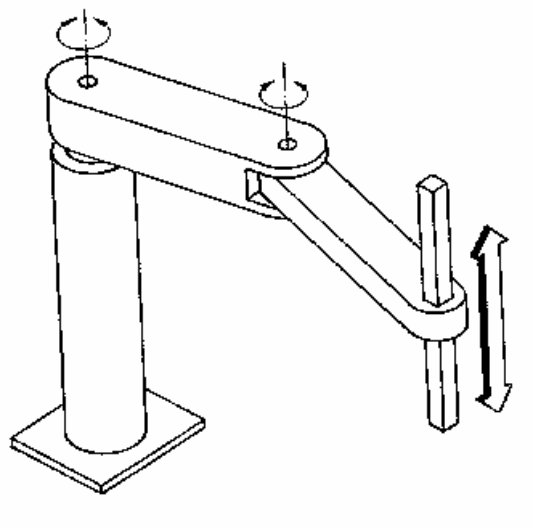

Figura 2.11: Robô com articulação horizontal

2.1.3.3.5 Robôs com articulação vertical (RRR)

Possui três juntas de revolução, a maior área de atuação, baixa rigidez mecânica e um controle complicado e difícil devido às três juntas de revolução e às variações no momento de carga e inércia.

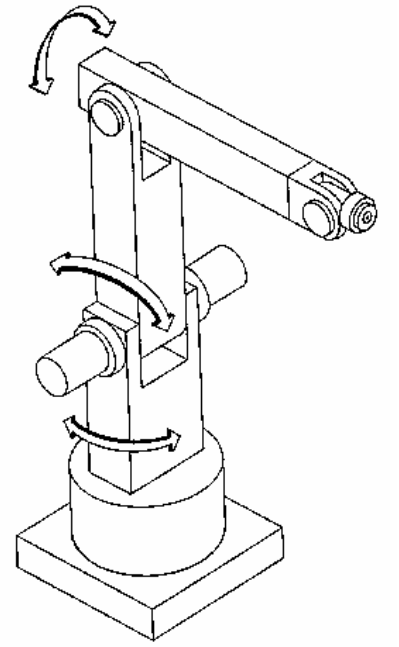

Figura 2.12: Robô com articulação vertical

2.2 Simuladores

Simuladores são ferramentas úteis, interativas e práticas para o treinamento e aprendizado formal que reproduzem virtualmente uma situação, permitindo experimentar os efeitos de um determinado procedimento sem sofrer as possíveis conseqüências de um erro num ambiente real.

A simulação, ao contrário de outras tecnologias, pode ser aplicada a todas as disciplinas. Inúmeros relatórios, teses de mestrado e doutorado, revistas e livros mostram diversos campos de aplicação para a simulação, como negócios, economia, educação, transportes, e muitos sistemas produtivos dos mais diferentes setores da economia.

Em 1940, Von Newman & Ulam ao associar a expressão Análise de Monte Carlo a uma técnica matemática utilizada para solucionar determinados problemas de blindagem em reatores nucleares, originaram a utilização moderna do verbo simular. O tratamento experimental desses problemas seria muito caro e uma abordagem analítica seria muito complicada.

A análise de Monte Carlo, que consiste na amostragem experimental com números randômicos, e a simulação se confundem, pois ambas são técnicas de computação numérica, sendo a primeira é aplicada a modelos estáticos e a segunda a modelos dinâmicos.

Com o aumento da capacidade de processamento dos computadores domésticos e com o aprimoramento das ferramentas de desenvolvimento, é possível simular desde a reação química entre um ácido e uma base até a pilotagem de um Boeing 747. Embora possamos sempre pensar em simulações complexas e grandiosas, existem uma gama de simulações relativamente mais simples, mas igualmente importantes e, muitas vezes, mais úteis.

Um exemplo a ser citado e que a maioria dos usuários de computador conhecem, são os jogos simuladores de vôos. Embora não sejam educativos, em sua maioria, seu princípio básico de funcionamento, interatividade e uso racional de recursos, tornou-o uma referência para a programação de jogos e outros softwares didáticos.

Figura 2.13: Flight Simulator 2002, simulador de vôo

A simulação pode ser identificada como sendo *Terminal* e *Não-Terminal* e a possibilidade de definir um comprimento para a simulação é a diferença entre elas. Se for possível definir um instante início e um instante fim, a simulação é terminal, caso contrário, não-terminal. A produção de uma fábrica que tem um horário de início de turno e um horário de término de turno pode ser citada como um exemplo de sistema terminal, e o sistema de atendimento de emergência de um hospital o exemplo de sistema não-terminal.

Simulação terminal possui, normalmente, um estado transiente e freqüentemente repete o ciclo: inicia vazio, fica ocupado por um período de tempo e termina vazio novamente. Para ele é importante definir quantas corridas devem acontecer para que os resultados estatísticos tenham consistência.

Já a simulação não-terminal, ou simulação em regime, não possui um evento ou instante onde a simulação termina, o que não significa dizer que nunca acabará, mas que, teoricamente, pode transcorrer infinitamente sem afetar as saídas, bastando apenas que o modelista determine o comprimento da corrida de simulação que garanta os resultados consistentes.

2.2.1 Softwares simuladores didáticos

"Os simuladores são recursos didáticos que enriquecem o processo ensino-aprendizagem porque permitem aos alunos a vivência de experiências que se situam entre a manipulação abstrata de conceitos / estratégias e a manipulação direta dos elementos com que se defrontam na prática, criando um ambiente que facilita a articulação entre o conceitual e o concreto dos alunos". [NEA 2004]

Além da utilização para treinamentos e diversão, os simuladores estão sendo bastante utilizados para o ensino regular. Sua principal vantagem consiste na economia de tempo e dinheiro, pois a necessidade de ter laboratórios, equipamentos, técnicos, etc. é bastante reduzida, além de ganhar tempo por não precisar preparar, checar e dar manutenção em equipamentos, evitando assim erros na montagem e operação destes. Por outro lado, todo o conhecimento específico que seria adquirido com esses procedimentos foi perdido. Portanto, é necessário utilizar racionalmente esses simuladores de acordo com os objetivos pré-estabelecidos.

Um exemplo da correta utilização de simuladores educacionais é quando o professor, através de esquemas estáticos, necessita representar situações dinâmicas, como é o caso do estudo da segunda lei de Kepler, relativa aos planetas. Fica claro que é vantajoso o uso de um simulador em vez do desenho em quadro negro (o simulador em questão mostra o movimento do planeta simulando sua variação de velocidade real).

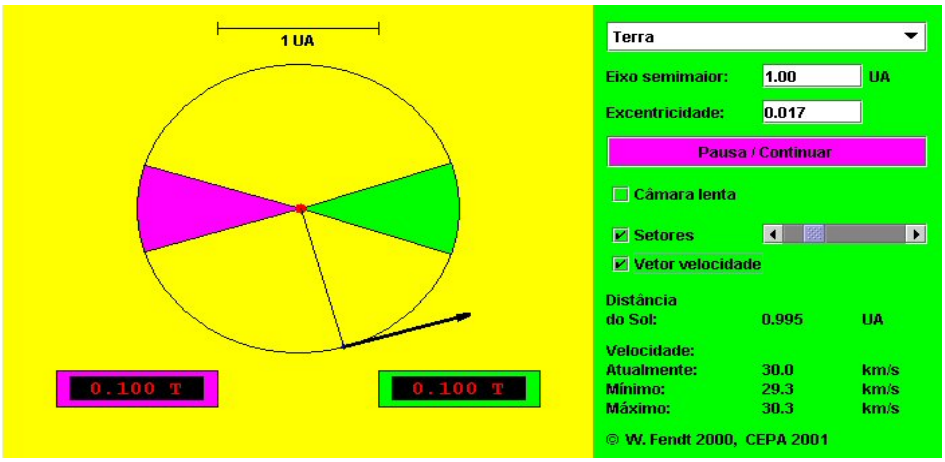

Figura 2.14: Applet Java simulando a segunda lei de Kepler

Ensinar o traçado de retas e círculos utilizando um simulador que faz exatamente isso, seria um exemplo do mau uso dos simuladores educacionais.

2.2.2 Softwares simuladores de sistemas

Técnica tradicional da pesquisa operacional e uma das ferramentas mais importantes e úteis para a análise do projeto e a operação de sistemas complexos. E é utilizada, normalmente, quando é impossível realizar experimentações no sistema real, devido, por exemplo, ao alto custo ou ao longo tempo necessário para realizar o experimento.

A grande vantagem de seu uso é permitir, com velocidade e baixo custo, estudos de sistemas reais sem a necessidade de modificá-los. Alternativas de mudanças podem ser tentadas e estudas de forma sistemática sem interferir no sistema real.

Além de auxiliar na tomada de decisão, eles contribuem para a compreensão do sistema estudado, uma vez que freqüentemente nos enganamos, pensando saber mais do que realmente sabemos sobre determinado assunto, até que tentamos simulá-lo em um computador.

A simulação de sistemas é, portanto, uma metodologia experimental que busca descrever e construir formas de quantificar o comportamento de um sistema, prevendo, assim, o seu comportamento futuro. E sua proposta é auxiliar na explicação, compreensão e melhoria do sistema estudado através da identificação, obtida da análise dos dados produzidos, dos aspectos importantes[1].

2.2.2.1 Elementos da simulação de sistemas

Segundo [COS 2003], todo modelo de simulação possui um ou mais dos seguintes elementos:

i. Componentes – partes integrantes do sistema;
ii. Parâmetros e variáveis – elementos do sistema que recebem valores;
iii. Relações funcionais – apresentadas na forma de relações matemáticas;
iv. Restrições – limitações impostas pelo modelador ou pela natureza do problema;
v. Objetivos – estabelecimento das metas como podem ser avaliadas.

Neste trabalho utilizaremos todos estes elementos.

[1] Mais detalhes e informações sobre Simulação e Simuladores, podem ser encontrados em [ARS 2003] e [COS 2003].

2.2.2.2 Terminologia da simulação de sistemas

De acordo com [COS 2003], a terminologia utilizada não é única, mas há certa tendência geral em aceitar a que é apresentada na tabela 2-1:

Tabela 2.1: Terminologia Básica de Simulação

Terminologia	Descrição
Modelo	Representação de um sistema.
Entidade	Elemento essencial para o modelo.
Atributo	Características de cada entidade.
Atividade	Seqüência de procedimentos que causa mudança no sistema.
Evento	Atividades são iniciadas e terminadas por eventos.
Acumuladores	Variáveis que permitem medir o desempenho do sistema.
Relógio	Variável que marca o tempo da simulação.
Lista de eventos futuros	Estrutura de dados que armazena os eventos previstos para ocorrer no futuro.
Cenário	Experimentação estruturada com um conjunto de configurações de dados e de entidades do sistema.
Replicação	Execução do modelo no computador.
Rodada	Período compreendido entre o início e o fim da replicação.
Variáveis de estado	Conjunto de variáveis que identificam o estado do sistema em um determinado instante de tempo.
Recurso	Entidade estática do modelo que serve entidades dinâmicas.
Filas	Locais de espera onde as entidades dinâmicas esperam sua vez de seguir através do sistema.

2.2.2.3 Etapas do processo de simulação de sistemas

[COS 2003] cita as etapas do processo tradicional da simulação de sistemas que serão mostradas a seguir:

i. Definição do problema;
ii. Plano de estudo;
iii. Coleta de dados;
iv. Representação do problema;
v. Escolha da ferramenta computacional;
vi. Desenvolvimento do modelo computacional;
vii. Verificação e validação do modelo;
viii. Planejamento dos experimentos;
ix. Execução do modelo;
x. Análise das alternativas de ação;
xi. Documentação e implementação.

Estas etapas, em geral, são utilizadas no desenvolvimento de simuladores de braços robóticos.

2.2.3 Softwares simuladores nos projetos

Projetistas, administradores e usuários de sistemas geralmente têm como objetivo o maior desempenho pelo menor custo. Utilizar simulação antes de começá-los é uma boa preparação para eventuais decisões no projeto real, pois ajudará a compreender e a otimizar o desempenho e/ou a confiabilidade do sistema.

A maioria, se não todos, os circuitos digitais integrados feitos atualmente são primeiramente simulados extensivamente para identificar e corrigir erros de projeto. A aplicação da simulação o quanto antes no ciclo do projeto é importante, pois o custo de reparação de erros aumenta drasticamente à medida que se avança.

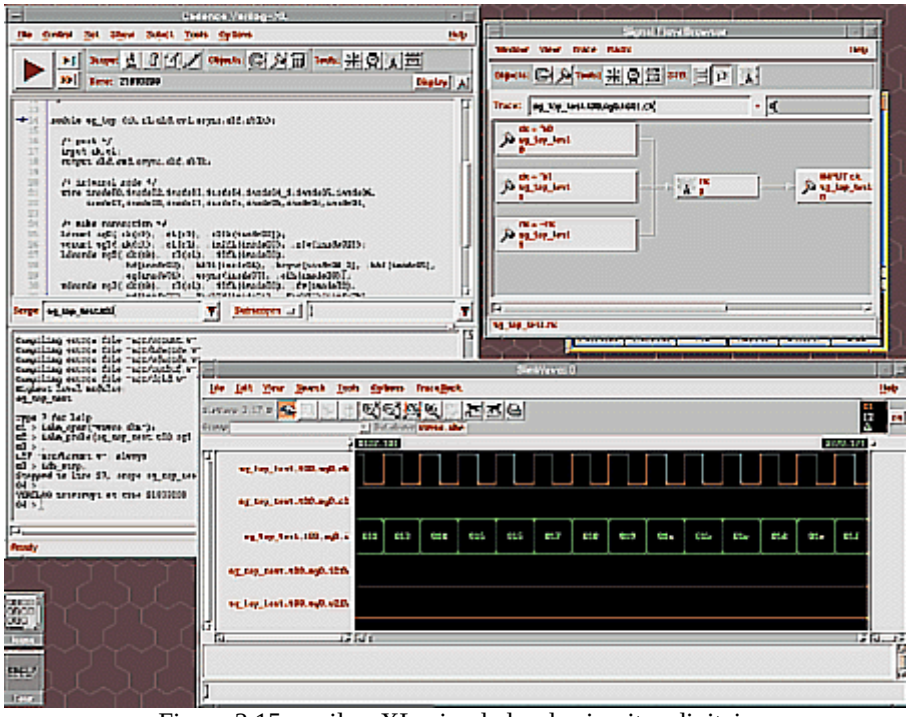

Figura 2.15: verilog-XL, simulador de circuitos digitais

Simular sistemas é imitar uma operação de algo real, tal como a operação cotidiana de um banco, ou na atribuição da equipe de funcionários de um hospital ou de uma companhia de segurança, em um computador. Em vez de modelos matemáticos extensivos feitos por peritos, o software simulador faz o possível para modelar e analisar as operações de um sistema real feito por leigos, que são gerentes e não programadores.

2.2.4 Softwares simuladores de robótica

Os princípios da robótica são ensinados extensivamente no ramo da engenharia, porém um problema comum é a disponibilidade limitada do caro equipamento com que os estudantes podem trabalhar, a fim de adquirir experiência prática. Esta demanda por disponibilidade de equipamento vem sendo suprida pelo uso de simuladores de robótica que possuem um custo relativamente muito inferior e possibilitam maior acesso aos recursos, desenvolvendo as habilidades e permitindo experiências, fatores cruciais para o processo de instrução.

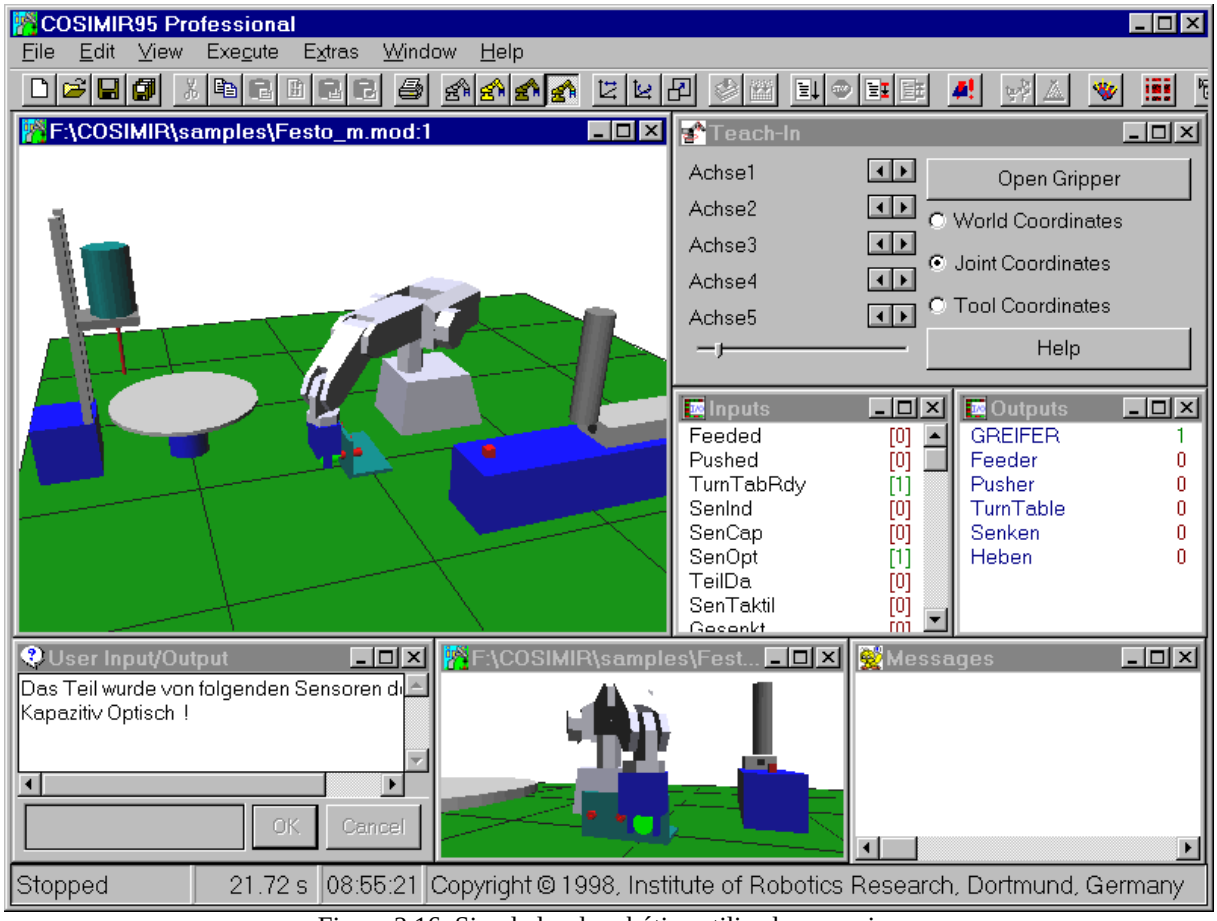

Figura 2.16: Simulador de robótica utilizado no ensino

Uma outra aplicação bastante utilizada para os simuladores está sendo feita pelas indústrias de grande, médio e pequeno porte. Elas necessitam de simulação para programar suas células robóticas off-line testando problemas comuns como colisões, limites de juntas, etc., enquanto a linha de produção está funcionando. Uma vez terminado o processo de programação off-line, o próprio simulador gera o programa que será carregado e executado nos robôs das novas células.

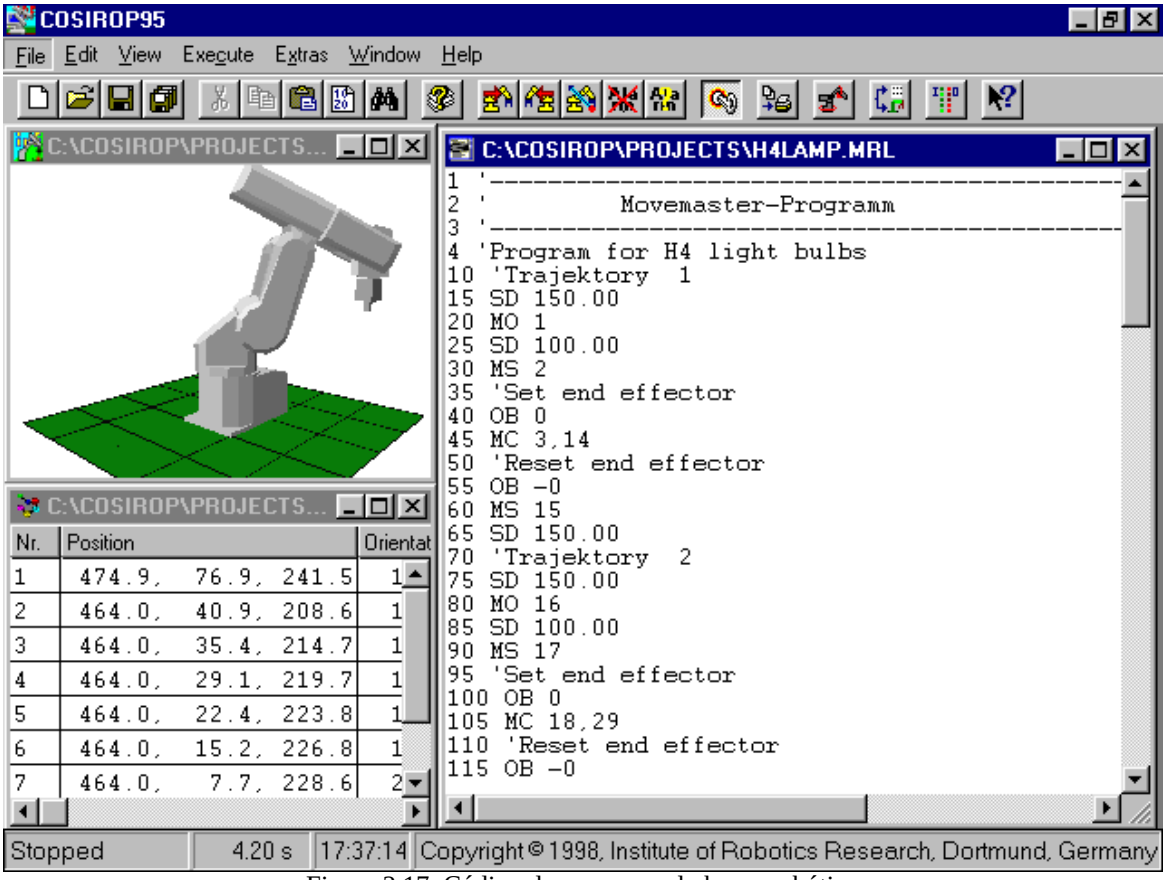

Figura 2.17: Código do programa do braço robótico

Teste de cenário também é outra aplicação para os simuladores na indústria. Perguntas como "*Se girar o robô 90 graus, o tempo de ciclo diminuirá?*" e "*Se utilizar outro modelo de robô, a linha de produção se agilizará?*" podem ser respondidas simplesmente movendo o objeto em questão e refazendo a simulação.

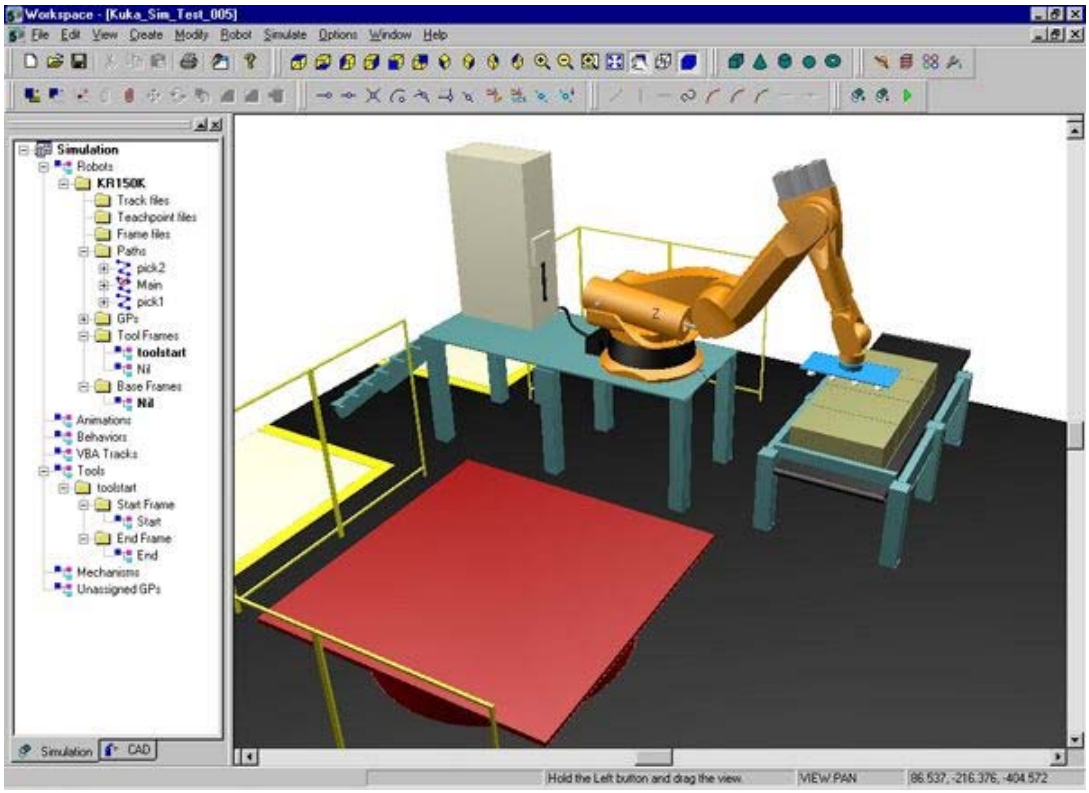

Figura 2.18: Determinando o posicionamento do robô na célula de trabalho

E por último, a simulação também é usada como prova de conceito ou para aprovação de projetos. Robôs e linhas de montagem são difíceis de visualizar em ambiente de duas dimensões como o dos softwares CAD. Com as visualizações tridimensionais dos simuladores de robótica, pode-se ver claramente como a linha de produção se parecerá. Movimentos das máquinas não são sempre óbvios com desenhos 2D, na qual faz com que não se perceba que o chão está interferindo na movimentação. Consequentemente, esta interferência poderá ser muito custosa, envolvendo a mudança e nivelamento de muitas máquinas e paredes. Situação que poderia ter sido prevista na fase de planejamento se os simuladores tivessem sido usados.

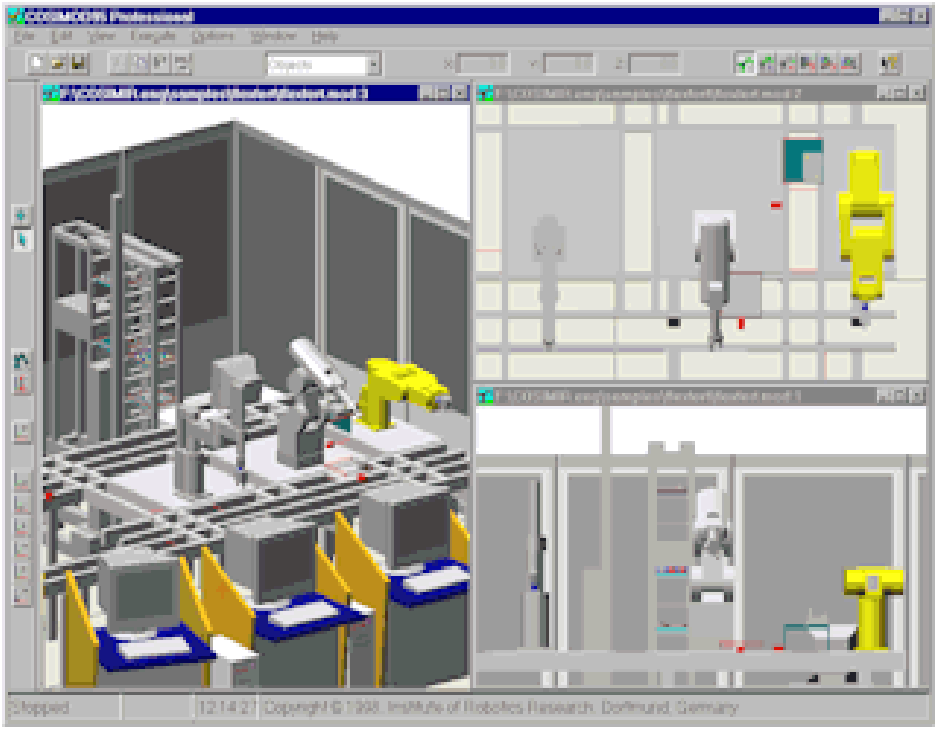

Figura 2.19: Demonstração de uma célula de trabalho projetada

O grande diferencial dos novos simuladores está sendo sua acessibilidade. Antigamente, simulação era para grandes empresas com muitos recursos. Com a simplicidade de utilização e a baixa no custo, as médias e pequenas empresas também podem usufruir dos benefícios que a simulação pode trazer.

3 FRAMEWORK: CONCEITOS E FUNDAMENTOS

Desenvolver sistemas com grande qualidade, pouco esforço e tempo cada vez menores são requisitos necessários na construção de qualquer programa e um dos objetivos principais da Engenharia de Software.

Existem muitas formas de se promover a reutilização em desenvolvimento de software, desde padrões e herança até o código. Dentro deste contexto, os *frameworks* são uma abordagem de projeto e código em um nível de granularidade elevado.

Neste capítulo serão resumidos os fundamentos sobre frameworks e seu projeto. Para construir frameworks que auxiliam no reuso a nível de análise, projeto ou de código muitas vezes utilizam-se padrões de projeto (design patterns) e componentes já desenvolvidos e depurados para diminuir o tempo de desenvolvimento e de testes.

Os *frameworks* possibilitam aumentar a produtividade e a qualidade no desenvolvimento de aplicações. O desenvolvimento parte de uma aplicação pré-implementada, a qual é desenvolvida por projetistas experientes em um domínio de aplicação.

"*Não gastar tempo desnecessário modelando elementos já existentes; Convergir todos os esforços para modelar o que é específico do projeto e; Utilizar frameworks em diferentes projetos, garantindo com isso que as equipes partam da mesma base já conhecida e amplamente testada, são os três benefícios primários na elaboração e utilização de frameworks*" [BOG 2002].

O ciclo de desenvolvimento de um framework geralmente contém três estados: análise, projeto e código. A análise enfatiza ítens como compreensão de requisitos, dos conceitos e das operações. Ítens que podem ser aproveitados dependendo da similaridade de sistemas desenvolvidos anteriormente. Projeto utiliza soluções como diagramas que auxiliam o usuário a entender melhor o domínio e relacionamento das classes. E código é a implementação de tudo o que foi analisado e projetado anteriormente.

Em um sistema não baseado no paradigma da orientação a objetos, nota-se a grande utilização de bibliotecas de funções, classificada como reutilização de rotinas. Já em um sistema orientado a objetos, é possível observar a reutilização de classes de objetos, artefato de software mais complexo que uma rotina isolada. Quando uma classe é reutilizada, é reutilizado também um conjunto de rotinas (métodos), bem como uma estrutura de dados (atributos). Reutilizar classes, portanto, tende a ser mais eficiente que reutilizar rotinas e de mais alto nível de granularidade.

É possível observar que o reuso de classes interligadas com um propósito comum (Frameworks) quando confrontado com a reutilização de classes isoladas, apresenta um grande ganho de produtividade no desenvolvimento de sistemas.

3.1 Definição

Há várias definições para framework,algumas das quais citamos a seguir.

Subsistema extensível para um conjunto de serviços relacionados ou um conjunto coeso de classes concretas e abstratas que colaboram para fornecer serviços para o núcleo invariante de um sistema lógico [LAR 2000].

Conjunto de estrutura de classes que constituem implementações incompletas que quando estendidas, permitem produzir diferentes artefatos de um sistema [SIL 2000].

Esqueleto de implementação de uma aplicação ou de um subsistema de aplicação em um domínio de problema particular. É composto de classes abstratas, concretas e provê um modelo de interação ou colaboração entre as instâncias de classes definidas pelo framework. Um framework é utilizado através de configuração ou conexão de classes concretas e derivação de novas classes concretas a partir de classes abstratas do framework [WIR 91].

Um *framework*, como já definido pelos vários autores acima, é uma aplicação semi-acabada que, muitas vezes pode ser um sistema inteiro ou, às vezes, um subsistema, sempre vinculado ao paradigma da OO. Mais recentemente, o termo passou a ser mais abrangente, significando que um *framework* é qualquer solução incompleta que pode ser completada através da instanciação e, desta forma, possibilitando a geração de mais de uma aplicação dentro do domínio-alvo do *framework* [FON 99].

Os *frameworks* apresentam inúmeras características, sendo que se destacam duas delas: inversão do controle e fornecimento de infra-estrutura e projeto.

A inversão de controle acontece porque os engenheiros de software reutilizam, na maioria das vezes, componentes de uma biblioteca em que o programa principal chama os componentes quando necessário, decidindo quando o componente será chamado e suas interações com os demais componentes. Quando se utiliza a abordagem de *frameworks*, o programa principal é reutilizado e o engenheiro de software decide o que será conectado dentro dele, determinando a estrutura geral e o fluxo de controle dos programas.

Os *frameworks* fornecem infra-estrutura de projeto ao engenheiro de software, reduzindo assim a quantidade de código a ser desenvolvido, testado e depurado com ele. Através dele é definida a arquitetura da aplicação [SIL 2000].

3.2 Projeto

Esta fase tem por objetivo criar uma arquitetura bem estruturada através da utilização de padrões de projetos para construir classes homogêneas e compatíveis entre si, obtendo com isso, fácil integração, portabilidade e consistência.

Uma estrutura orientada a objetos bem projetada facilitará a adaptação de um desenvolvedor ao ambiente de frameworks. Abaixo podemos citar alguns requisitos essenciais para a construção frameworks bem projetados:

- Modularidade (alta coesão e baixo acoplamento) – obtida com a definição de interfaces limpas e consistentes, como também o encapsulamento da implementação de métodos. As interfaces devem ser as mais simples possível e deverão agrupar todos os objetos de mesma linhagem e os aspectos comportamentais do frameworks serão encapsulados para obter-se assim, a qualidade e a funcionalidade desejada;
- Reusabilidade – frameworks devem oferecer um grau muito elevado de reutilização em relação às classes individuais, pois além de herdar as características inerentes às classes-pai, herda-se também o comportamento e o relacionamento destas. Assim, podemos definir frameworks como uma arquitetura de um sistema reutilizável em termos de contratos de colaboração entre classes abstratas e um conjunto de pontos adaptáveis que precisam ser especializados em uma aplicação, também chamados de *hot spots*. O projeto destes pontos adaptáveis é o elemento central na criação de um framework que permitirá construir novas aplicações utilizando componentes genéricos. O uso de um padrão possibilita explorar as similaridades para a obtenção de soluções a outras classes de problemas, garantindo assim, maior produtividade de programação, performance e confiabilidade do programa, promovendo menos esforço e menor probabilidade de erros;

A seguir serão apresentados alguns recursos que os frameworks disponibilizam e permitem ao desenvolvedor um amplo conhecimento e domínio do problema que trabalhará:

- Capacidade de estender funcionalidades a partir de comportamentos gerando subclasses a partir das classes existentes;
- Robustez, capacidade de responder a eventos externos que as vezes não foram previstos na especificação dos requisitos, mantendo o controle da execução da aplicação; e
- Compatibilidade, facilidade na combinação de produtos do sistema.

3.2.1 Elementos do Projeto

Desenvolvedor, definição do domínio e futuras aplicações são os ítens necessários para se obter um projeto de framework e serão descritos a seguir:

- Desenvolvedor – peça primordial ao projeto e tem como incumbência manter, organizar e escolher as classes que farão parte da estrutura do framework;
- Definição do domínio – define o trajeto que o projeto percorrerá. Uma análise sobre que domínio o framework trabalhará é de vital importância, pois é a partir dele que o desenvolvedor decidirá que tipo de ferramenta será possível gerar através da reutilização de classes genéricas do framework;
- Futuras aplicações – capacidade de criar diferentes aplicativos em menor tempo e com esforço reduzido, utilizando-se da mesma base de estrutura com ligações entre classes pré-definidas do framework.

3.3 Metodologias de Desenvolvimento de Frameworks

Conhecer as principais características de algumas metodologias é essencial para quem deseja desenvolver frameworks.

Os ajustes na estrutura das classes são provenientes da apuração cíclica na evolução do desenvolvimento dos frameworks. Esta apuração toma como parâmetro os aspectos de generalidade e extensibilidade, gerando com isso o manuseio de uma grande quantidade de informações.

Pode-se afirmar que o desenvolvimento de um framework é mais complexo que o desenvolvimento de aplicações específicas do mesmo domínio, devido a necessidade de considerar os requisitos de um conjunto significativo de aplicações, de modo a dotar a estrutura de classes do framework de generalidade, em relação ao domínio tratado; a necessidade de ciclos de evolução voltados a dotar a estrutra de classes do framework de alterabilidade e extensabilidade [SIL 2000].

Dentre as diversas metodologias voltadas ao desenvolvimento de frameworks, cabe destacar as seguintes: Projeto dirigido por exemplo [JOH 93] e Projeto dirigido por pontos adaptáveis [PRE 94] que serão descritas logo a seguir:

3.3.1 Projeto dirigido por exemplo

Necessidade de compreender o que será gerado no seu domínio através da análise adequada de exemplos particulares na construção de frameworks. Apesar de não utilizar técnicas de modelagem detalhadas como a análise e projeto orientadas a objetos, esta metodologia usa o conhecimento obtido a partir de diferentes aplicações feitas anteriormente para extrair as diretivas básicas (abstrações) através de uma forma *bottom-up* compondo assim o domínio de aplicação. Classes abstratas que agrupam os aspectos semelhantes adquiridos e armazenam as características gerais do domínio de aplicação, são criadas a partir do resultado obtido da generalização e fornecem às classes concretas de nível hierárquico inferior, a especialização pela flexibilização para satisfazer cada aplicação em particular. Num framework, uma classe abstrata pode conter, propositalmente, métodos incompletos para que sua definição seja feita na geração de uma aplicação.

3.3.2 Projeto dirigido por pontos adaptáveis

Segundo [PRE 94] e [SCH 97], uma aplicação orientada a objetos é completamente definida, ao contrário dos frameworks que contém partes indefinidas, fornecendo flexibilidade e capacidade de se moldar a diferentes sistemas, se implementado a partir de padrões de projetos. A essência da metodologia numa estrutura de classes flexíveis são os pontos adaptáveis e são utilizados para a construção de frameworks.

3.4 Desenvolvimento

A seguir, serão descritos elementos do desenvolvimento de frameworks, os quais montam uma base sólida na definição de sua estrutura geral.

3.4.1 Estrutura

A reutilização de uma estrutura de frameworks, baseada na aplicação de padrões que já foram testados e observados em outros sistemas, deve ser feita através de suas classes e instâncias.

i. O princípio dos frameworks é possibilitar que a partir de uma idéia principal de uma determinada família de aplicações, possa-se construir um sistema começando de um nível superior na implementação do mesmo. Isto é obtido com a repetitiva evolução da estrutura de classes adaptadas da estrutura inicial e se estendendo até os pontos de generalização e flexibillização.

3.4.2 Tipos de Frameworks

Os *frameworks* podem ser categorizados de diferentes formas. Saber *onde* e *como* o framework é utilizado, são os dois critérios simples para desenvolver frameworks. A seguir, será detalhado os tipos de framework seguindo o critério de *como o framework é utilizado*:

i. *Focado na Herança, Caixa Branca ou Direciona a Arquitetura*: uma aplicação deve implementar os métodos que são definidos em interfaces ou classes abstratas. Ligação dinâmica e herança são mecanismos que trabalham juntos através da modelagem orientada a objetos;

ii. *Focado na Composição, Caixa Preta ou Direcionado a Dados*: não é possível visualizar ou alterar a arquitetura do framework, podendo apenas, utilizar as funcionalidades já existentes. São fornecidas e definidas interfaces para os componentes e somente através delas é que são feitas as extensões. Por meio da definição e integração de componentes que se ajustam a uma interface específica, os recursos existentes são reutilizados através de padrões definidos no projeto, tornando o framework orientado a componentes;

iii. *Híbrida*: obtido a partir da união das duas classificações, na maioria dos frameworks, sua classificação tem sido Focada na Herança com algumas funcionalidades prontas Focadas na Composição.

Continuando, a seguir serão apresentados três modelos de framework seguindo o critério de *onde o framework é utilizado*:

i. *Framework de Suporte*: dedica-se a assuntos que envolvam serviços de sistema operacional e não de aplicação, como por exemplo: comunicação, acesso a arquivos, computação distribuída, interfaces gráficas, linguagens de programação, etc;

ii. *Framework de Aplicação*: trata de assuntos que envolvam o projeto da construção de aplicações específicas. Encapsula conhecimento aplicável a uma vasta gama de aplicações, por exemplo: construção de interface gráfica para usuário, telecomunicações, finanças, produção e educação, geoprocessamento, etc;

iii. *Framework de Integração*: responsáveis por integrar aplicações distribuídas e componentes em uma mesma arquitetura. Encapsula conhecimento aplicável a aplicações pertencendo a um domínio particular de problema, por exemplo: ORB, CORBA e RML.

Esta dissertação escolheu a classificação do tipo Caixa Branca focando um Framework de Aplicação.

3.4.3 Framework de Aplicação

Valendo-se do modelo de colaboração pela união das classes e comunicação entre os objetos, geram-se soluções a problemas semelhantes de um nicho, através da forma de reuso da análise, projeto e código. Seu conjunto de classes é flexível e extensível para que seja permitida a construção de várias aplicações, especificando apenas suas particularidades [LAR 2000].

O Framework de Aplicação, exemplificado na figura abaixo, demonstra todo o aproveitamento do sistema, promovendo uma diminuição do esforço necessário para produzir a ferramenta. Uma estrutura de framework pré-definida com seus objetos e conexões, pode ser observada na área hachurada. E, abaixo desta área, existe um complemento de classes que, quando aliado à estrutura do framework, caracteriza um novo aplicativo. A infraestrutura de projeto disponibilizada pelos frameworks, reduz a quantidade de código a ser desenvolvido, testado e apurado e, somada com interconexões pré-estabelecidas, definem a arquitetura da aplicação.

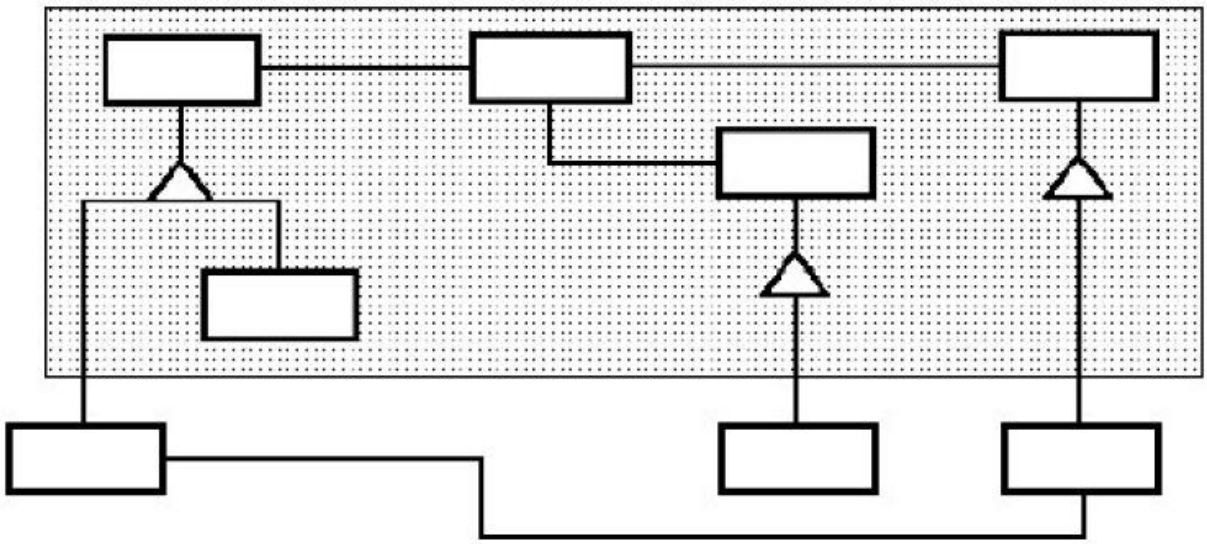

Figura 3.1: Aplicação desenvolvida reutilizando um framework – exemplo genérico

"Os frameworks são estruturas de classes interrelacionadas que permitem não apenas a reutilização de classes, mas minimizam o esforço para o desenvolviimento de aplicações, por conterem o protocolo de controle da aplicação. Eles invertem a óptica do reuso de classes, da abordagem bottom-up para a abordagem top-down: o desenvolvimento inicia com o entendimento do sistema contido no projeto do framework e segue no detalhamento das particularidades da aplicação específica, o que é definido pelo usuário do framework" [SIL 2000].

3.4.4 Arquitetura e Componentes de um *Framework*

Um *framework* possui um *kernel* que contém um conjunto de classes que não são passíveis de adaptação e que permanecem presentes em todas as suas instanciações.

A diferenciação das aplicações que instanciam um mesmo *framework* são as adaptações realizadas nos pontos de flexibilização (*hot-spots*).

Os *hot-spots* dirigem a especialização do *framework*, podendo ser feita tanto por herança como delegação. Os *frozen-spots* são as partes fixas do *framework.*

A figura 3.2 ilustra a arquitetura e os componentes de um *framework*.

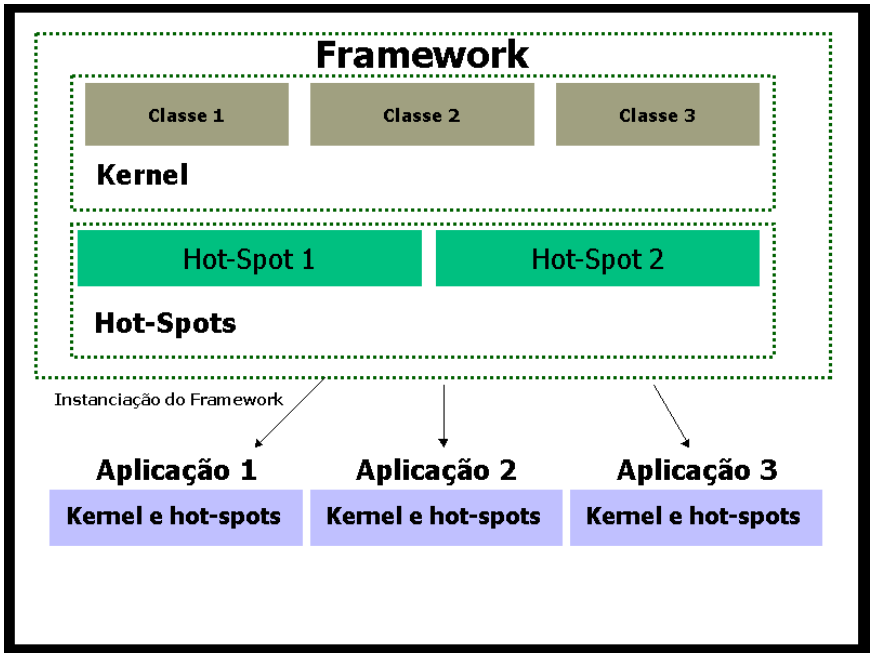

Figura 3.2: Arquitetura e Componentes de um Framework

3.4.5 Vantagens de uso de frameworks

Alguns exemplos das vantagens obtidas por usar um Framework de Aplicação são:

i. A quantidade de linhas de código diminui comparado a uma aplicação iniciada do zero;
ii. Possibilita uma infraestrutura com alto nível de portabilidade para o projeto;
iii. Estabele uma arquitetura da aplicação, já definida pelas interconexões;
iv. Moda a aplicação de áreas onde é possível estendeer um código escrito, a uma necessidade específica, tornando-a particular a outras aplicações;
v. Agrupam diferentes quantidades de classes complexas;
vi. Encapsula o projeto genérico completo para um domínio de aplicação.

No capítulo seguinte, será descrita a definição de classes genéricas e de controle que compõem o framework proposto.

4 FWWORKCELL – FRAMEWORK PARA SIMULADORES DE CÉLULAS DE TRABALHO: VISÃO GERAL, ARQUITETURA E CARACTERÍSTICAS

Neste capítulo é apresentado o *fwWorkCell*: seu objetivo, sua arquitetura e suas características principais. Em particular, será descrita não só a definição de classes genéricas e de controle que o compõem mas também a construção de um ambiente de suporte à utilização do framework visando dar suporte à construção de uma grande variedade de simuladores.

4.1 Definição

O framework *fwWorkCell* é um tipo de Framework de Aplicação desenvolvido na linguagem Java[2] e seu objetivo principal é dar suporte no desenvolvimento de Simuladores de Células de Trabalho.

A escolha de Java como linguagem de programação se deveu a sua portabilidade e ao fato de ser uma linguagem de programação orientada a objetos, possibilitando a flexibilidade e abstração de classes para construir novos modelos, facilidade de estabelecer ligações entre as classes, encapsulamento da implementação de métodos, reusabilidade de código, a extensibilidade através da inserção de novas classes e a robustez no tratamento de exceções.

4.2 Arquitetura

A figura 4.1 demonstra a arquitetura de um ambiente para o desenvolvimento de um simulador baseado no *fwWorkCell*.

Um novo simulador de Célula de Trabalho será obtido com a inclusão/especialização de alguns componentes ao framework fwWorkCell. Nas seções seguintes teremos um detalhamento de cada um desses componentes e do framework em si.

4.2.1 Braço Robótico

Elemento central do Simulador de Células de Trabalho. Sem a especificação de ao menos um braço robótico, o novo simulador não tem motivo para existir.

---

[2] [APA 2004], [DEI 2003], [HAT 2003], [HOR 2001], [SUN 2001] e [TIL 2002] serviram como fontes de pesquisa, estudo e consulta da linguagem Java e aplicativos de apoio.

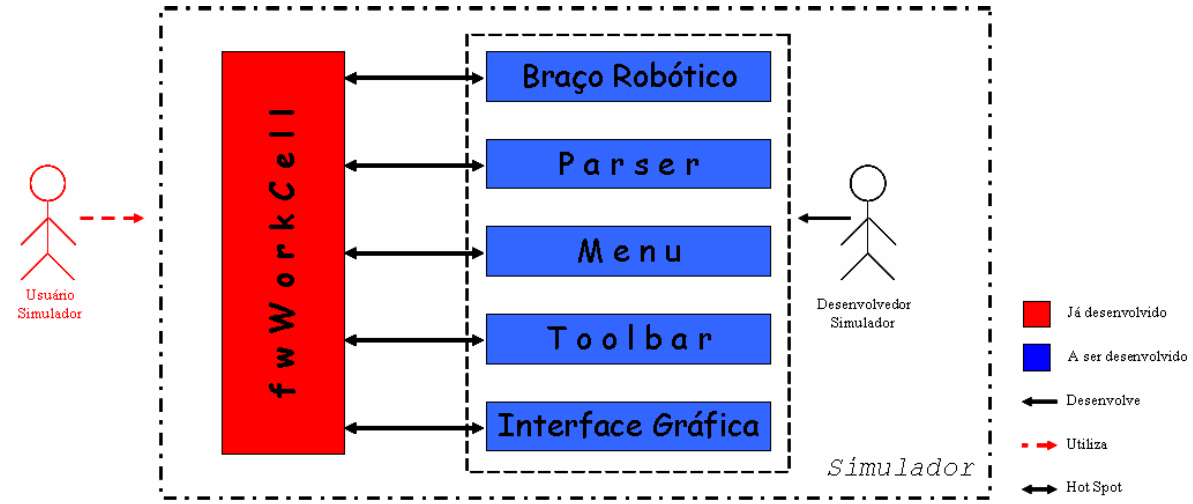

Figura 4.1: Arquitetura do ambiente para desenvolvimento de simuladores

O *fwWorkCell* possibilita a especificação de uma quantidade ilimitada de braços robóticos diferentes e cada célula criada poderá conter vários destes robôs podendo, inclusive, conter vários do mesmo tipo funcionando de maneira independente.

A especificação de um braço robótico no *fwWorkCell* foi dividida em duas partes como pode ser vista na figura abaixo:

Figura 4.2: Componentes internos de Braço Robótico

4.2.1.1 Especificação técnica

Especialização da classe *Robot* do framework. É responsável por toda especificação e controle do braço robótico.

A especificação se dá por meio de um arquivo XML[3], informado no método *Create* da classe, definindo o número de juntas, sua posição inicial e seus limites inferiores e superiores, além da quantidade de entradas e saídas que o mesmo possui.

O controle é feito através da implementação de alguns métodos abstratos da classe pai. São métodos que informam se o braço possui *Pitch*, *Yaw* e/ou *Roll*, além de implementar as cinemáticas direta e inversa, responsáveis pela movimentação e posicionamento do robô.

A separação da especificação da parte de controle permite que se tenha dois braços robóticos muito parecidos implementados pela mesma classe, diferenciando-se apenas na sua especificação. Podemos citar como um exemplo, dois robôs praticamente iguais que se diferenciam apenas pelos limites de suas juntas. Estes seriam implementados pela mesma classe de controle e teriam apenas arquivos XML de especificação diferentes.

---

[3] [DEC 2000], [FED 2004] e [MCL 2001] foram consultados a fim de obter maiores informações para o desenvolvimento de métodos responsáveis pela carga de arquivos XML.

4.2.1.2 Representação Gráfica

Implementação da interface *RobotGraph* do framework. É responsável por toda a parte da representação visual do braço robótico.

Cada braço deverá ter sua classe correspondente implementada e esta será alocada e incluída na classe de controle correspondente para que a mesma possa invocar toda vez que sofrer alteração nos valores de seus atributos.

Por exemplo, o robô X teve o ângulo da junta 2 alterado de 5º para 25,6º. Quando a classe controladora deste robô receber a invocação para que o valor da junta 2 seja alterado, a mesma, após a alteração do valor da junta em questão, invocará à classe de representação gráfica, que altere o posicionamento do robô de maneira a refletir as alterações sofridas.

Vale a pena lembrar que todo este processo acima descrito, está implícito no framework, sendo necessário apenas que a interface de representação gráfica seja implementada e atribuída à classe de controle.

4.2.2 Parser

O Parser é uma especialização da classe *Interpreter* do framework. É responsável por validar programas e interpretar comandos, fazendo com que o braço robótico realize as tarefas que lhe foram definidas.

Toda vez que um programa é escrito e atribuído a um robô, o framework chama o método *syntaxCheck* para verificar se o mesmo foi escrito corretamente e poderá ser executado. Caso informe que está inválido, o programa não poderá ser adicionado nem executado ao/pelo robô.

Se um programa foi escrito corretamente e adicionado ao braço robótico, este poderá ser executado. O fwWorkCell permite que um programa seja executado passo-a-passo, isto é, uma linha do programa por vez, ou pode ser executado continuamente até chegar ao fim do mesmo ou até encontrar um breakpoint. Se este for encontrado, é possibilitada a hipótese novamente de seguir passo-a-passo ou continuamente e segue-se neste ciclo até o programa chegar ao fim.

Cabe salientar que todo este processo de controle está embutido no framework, ficando apenas a validação do programa e a interpretação das linhas de comando de responsabilidade do programador.

4.2.3 Menu

O *fwWorkCell* permite ao programador incluir e customizar o menu de seu aplicativo. Para isso, basta ele criar um arquivo XML de configuração informando o formato do menu desejado. O framework permite que se configure além do texto que aparecerá, a figura, o atalho e a tecla aceleradora. Suporta ainda, a criação de submenus, dando com isso total liberdade ao programador para montar o menu da maneira que melhor lhe convier.

Todo item de menu que possuir uma ação associada, deverá informar qual o nome da classe que implementará esta ação. Esta classe deverá ser especializada da classe Action do framework. Toda vez que o item for selecionado pelo usuário, o framework chamará o método *actionPerformed* da classe indicada. Portanto, toda ação deverá estar implementada neste método dentro da classe.

O framework possui um conjunto de ações previamente criadas que se encontram dentro do pacote *Action*. Ações como fechar janela, criar novo programa e copiar para a área de transferência já foram criadas e estão disponíveis para utilização.

Abaixo um exemplo de menu criado com o auxílio do *fwWorkCell*:

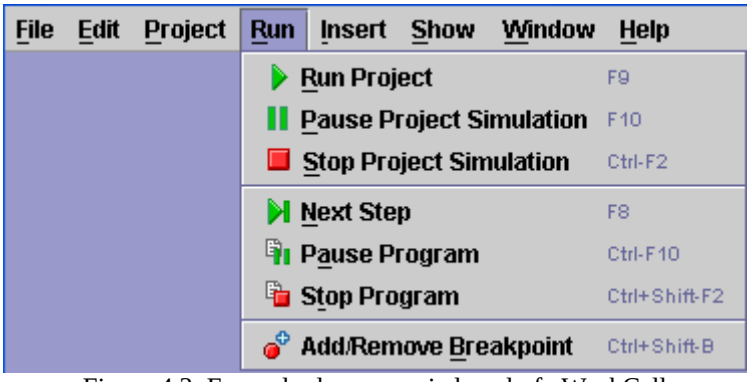

Figura 4.3: Exemplo de menu criado pelo fwWorkCell

4.2.4 Toolbar

Assim como o menu, também é disponibilizada a possibilidade de criar uma barra de ferramentas (toolbar) e sua configuração é similar a do menu, bastando apenas informar quais os botões a serem criados e suas ações correspondentes em um arquivo XML de configuração. O funcionamento é idêntico ao menu, sendo chamado o método *actionPerformed* da classe indicada toda vez que um botão for pressionado.

Abaixo um exemplo de barra de ferramentas criada com o auxílio do fwWorkCell:

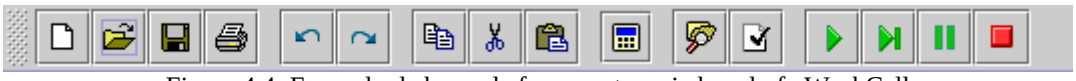
Figura 4.4: Exemplo de barra de ferramentas criado pelo fwWorkCell

4.2.5 fwWorkCell

O framework *fwWorkCell* é a união de vários packages que se interrelacionam entre si para atender as mais diversas solicitações como podemos verificar na figura abaixo:

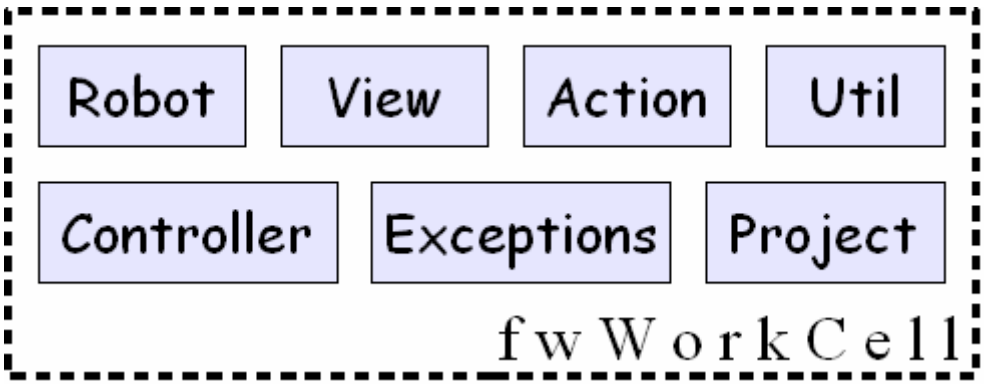

Figura 4.5: Componentes internos de fwWorkCell

A seguir cada package será descrita individualmente, citando suas funções dentro do framework e relacionando as classes disponibilizadas.

4.2.5.1 Exceptions

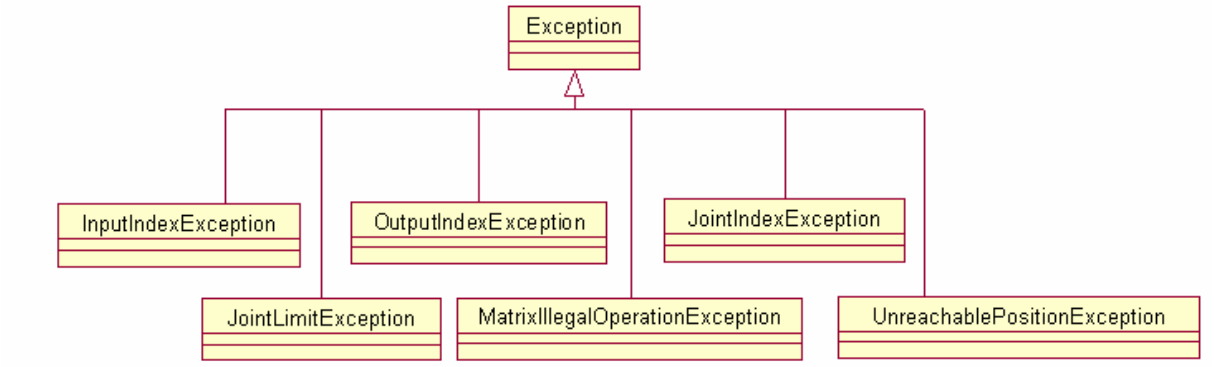

Figura 4.6: Diagrama de classes de Exceptions

Package que agrupa as classes de Exceptions lançadas pelas outras classes do framework. Foram criadas com o objetivo de melhor informar os problemas ocorridos durante a execução de um método. A seguir, serão listadas e explicadas brevemente cada uma das classes pertencentes a package:

i. *InputIndexException* – lançada quando da tentativa de acessar uma posição inexistente do vetor de entradas;
ii. *JointIndexException* – lançada quando da tentativa de acessar uma junta que não existe;
iii. *JointLimitException* – lançada quando da tentativa de colocar uma junta numa posição que extrapola seus limites;
iv. *MatrixIllegalOperationException* – lançada quando uma operação de matriz ilegal for executada;
v. *OutputIndexException* – lançada quando da tentativa de acessar uma posição inexistente do vetor de saídas;
vi. *UnreachablePositionException* – lançada quando da tentativa de colocar o braço robótico em uma posição inalcansável por ele.

4.2.5.2 Project

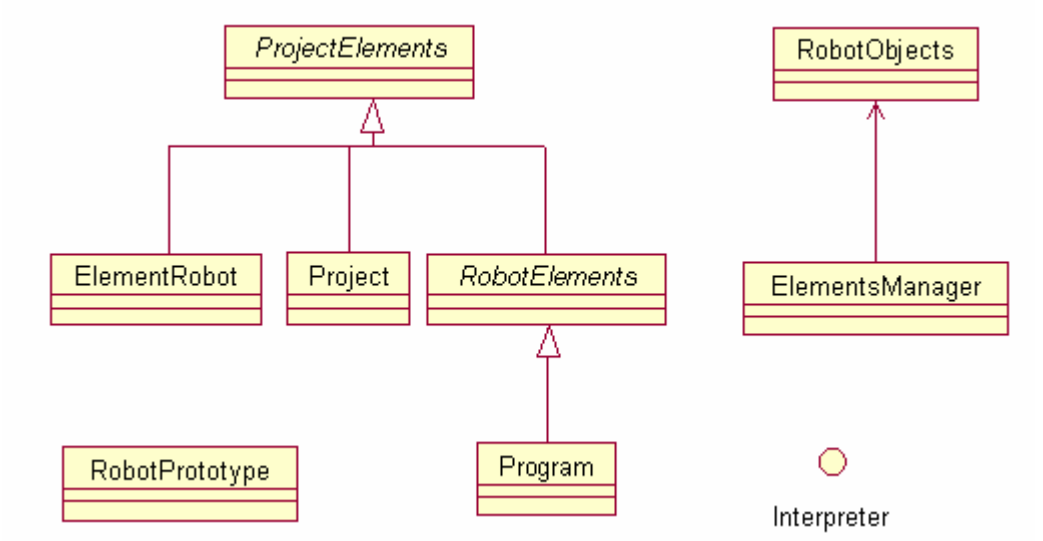

Figura 4.7: Diagrama de classes de Project

Package agrupadora das classes que tratam da parte de projeto do simulador. A seguir, serão listadas e explicadas brevemente cada uma das classes pertencentes a package:

i. *ElementRobot* – classe responsável pela organização dos elementos de um braço robótico. É nesta classe que se adiciona programas, por exemplo;
ii. *ElementManager* – responsável pela organização dos elementos de uma célula de trabalho. É nela que braços robóticos são adicionados a célula de trabalho, por exemplo;
iii. *Interpreter* – interface que deverá ser implementada por toda classe Parser de alguma linguagem;
iv. *Program* – classe responsável por tudo que diz respeito a um programa. É ela que solicita a abertura e a execução de um programa, por exemplo;
v. *Project* – responsável por tudo que diz respeito a um projeto. Como exemplo de sua utilização pode ser citada a abertura e salvamento de um projeto de célula de trabalho;
vi. *ProjectElements* – classe abstrata que todas as classes que farão parte de um projeto deverão herdar.
vii. *RobotElements* – classe abstrata, descendente de *ProjectElements*, que todas as classes que farão parte de um braço robótico deverão herdar;
viii. *RobotObjects* – classe que agrupa todos os objetos pertencentes a um braço robótico na célula de trabalho;
ix. *RobotPrototype* – classe que implementa o p*attern Prototype* e é utilizado para que um robô seja alocado diversas vezes.

4.2.5.3 Action

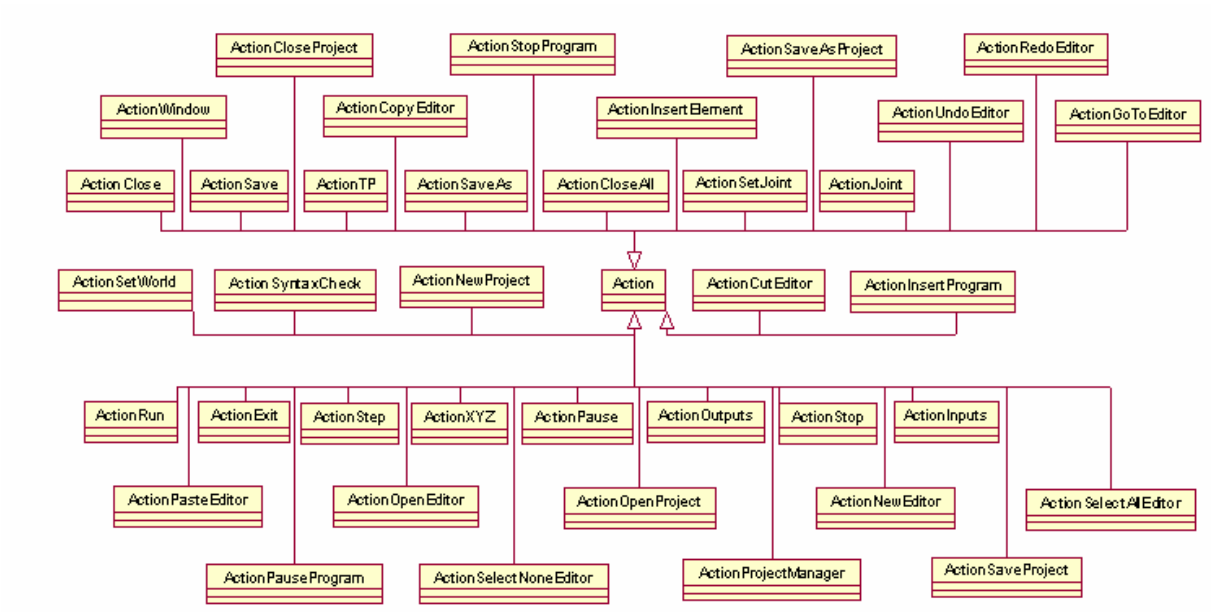

Figura 4.8: Diagrama de classes de Action

Contém a classe Action que é herdada por classes que serão utilizadas nas chamadas das ações dos ítens de menu e da toolbar. Contém, também, algumas classes Action já implementadas e que poderão ser utilizadas. A seguir, serão listadas e explicadas brevemente cada uma das classes pertencentes a package:

i. *Action* – classe responsável pela execução da ação de escolher um item de menu ou pressionar um botão na toolbar. Ela será herdada por todas as classes que implementarão ações de menu ou toolbar;

ii. *ActionClose* – associada a ação de fechar a janela ativa;
iii. *ActionCloseAll* – associada a ação de fechar todas as janelas abertas;
iv. *ActionCloseProject* – associada a ação de fechar o projeto aberto atualmente;
v. *ActionCopyEditor* – associada a ação de copiar o texto selecionado para a área de transferência;
vi. *ActionCutEditor* – associada a ação de recortar o texto selecionado para a área de transferência;
vii. *ActionExit* – associada a ação de sair do programa;
viii. *ActionGoToEditor* – associada a ação de ir até determinada linha do editor de programa;
ix. *ActionInputs* – associada a ação de mostrar a janela com os estados do vetor de entrada;
x. *ActionInsertElement* – associada a ação de inserir um elemento ao projeto atualmente editado;
xi. *ActionInsertProgram* – associada a ação de inserir um programa ao braço robótico;
xii. *ActionJoint* – associada a ação de mostrar a janela com os estados das juntas do braço robótico ativo no momento;
xiii. *ActionNewEditor* – associada a ação de abrir uma janela de texto para a edição de um novo programa;
xiv. *ActionNewProject* – associada a ação de criar um novo projeto;
xv. *ActionOpenEditor* – associada a ação de abrir uma janela com um programa já existente;
xvi. *ActionOpenProject* – associada a ação de abrir um projeto salvo anteriormente;
xvii. *ActionOutputs* – associada a ação de mostrar a janela com os estados do vetor de saída;
xviii. *ActionPasteEditor* – associada a ação de copiar o texto da área de transferência para a janela do editor de texto;
xix. *ActionPause* – associada a ação de parar momentaneamente a execução da simulação de todos os programas;
xx. *ActionPauseProgram* – associada a ação de parar momentaneamente a execução da simulação do programa selecionado;
xxi. *ActionProjectManager* – associada a ação de abrir o gerenciador de projetos;
xxii. *ActionRedoEditor* – associada a ação de refazer a última tarefa desfeita no editor de programa;
xxiii. *ActionRun* – associada a ação de executar a simulação;
xxiv. *ActionSave* – associada a ação de salvar o programa atualmente editado;
xxv. *ActionSaveAs* – associada a ação de salvar como... o programa atualmente editado;
xxvi. *ActionSaveAsProject* – associada a ação de salvar como... o projeto atualmente editado;
xxvii. *ActionSaveProject* – associada a ação de salvar o projeto atualmente editado;
xxviii. *ActionSelectAllEditor* – associada a ação de selecionar todo o texto atualmente editado;
xxix. *ActionSelectNoneEditor* – associada a ação de desfazer a seleção de todo o texto atualmente selecionado;
xxx. *ActionSetJoint* – associada a ação de mostrar a janela responsável por atribuir novos valores às juntas do robô ativo;

xxxi. *ActionSetWorld* – associada a ação de mostrar a janela responsável por atribuir novos valores às coordenadas do robô ativo;
xxxii. *ActionStep* – associada a ação de executar passo-a-passo o programa que está rodando atualmente;
xxxiii. *ActionStop* – associada a ação de parar a execução da simulação de todos os programas;
xxxiv. *ActionStopProgram* – associada a ação de parar a execução da simulação do programa atual;
xxxv. *ActionSyntaxCheck* – associada a ação de validar os programas;
xxxvi. *ActionTP* – associada a ação de mostrar a janela do Teach Pendant;
xxxvii. *ActionUndoEditor* – associada a ação de desfazer a última ação feita no editor de programa;
xxxviii. *ActionWindow* – associada a ação de escolher uma das janelas abertas do projeto;
xxxix. *ActionXYZ* – associada a ação de mostrar a janela com os valores das coordenadas do braço robótico ativo;

4.2.5.4 Controller

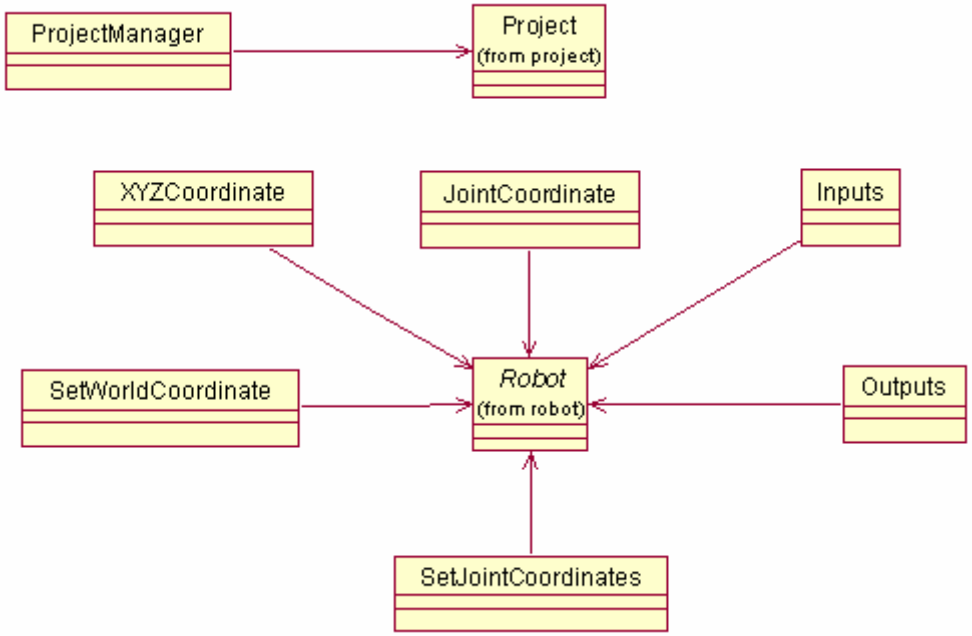

Figura 4.9:Diagrama de classes de Controller

Contém as classes que possuem as informações de dados mostrados pelas classes da package View. Informação e visualização foram separados para possibilitar a alteração da interface e continuar utilizando a mesma classe provedora de informação. A seguir, serão listadas e explicadas brevemente cada uma das classes pertencentes a package:

i. *Inputs* – classe controladora responsável por disponibilizar o estado do vetor de entrada do objeto atualmente selecionado;
ii. *JointCoordinates* – classe controladora responsável por disponibilizar os valores das juntas do braço robótico atualmente selecionado;
iii. *Outputs* – classe controladora responsável por disponibilizar o estado do vetor de saída do objeto atualmente selecionado;
iv. *ProjectManager* – classe controladora responsável por gerenciar os elementos do projeto;

v. *SetJointCoordinates* – classe controladora responsável por alterar os valores das juntas do braço robótico atualmente selecionado;
vi. *SetWorldCoordinates* – classe controladora responsável por alterar os valores das coordenadas do braço robótico atualmente selecionado;
vii. *XYZCoordiantes* – classe controladora responsável por disponibilizar as coordenadas do braço robótico atualmente selecionado.

4.2.5.5 View

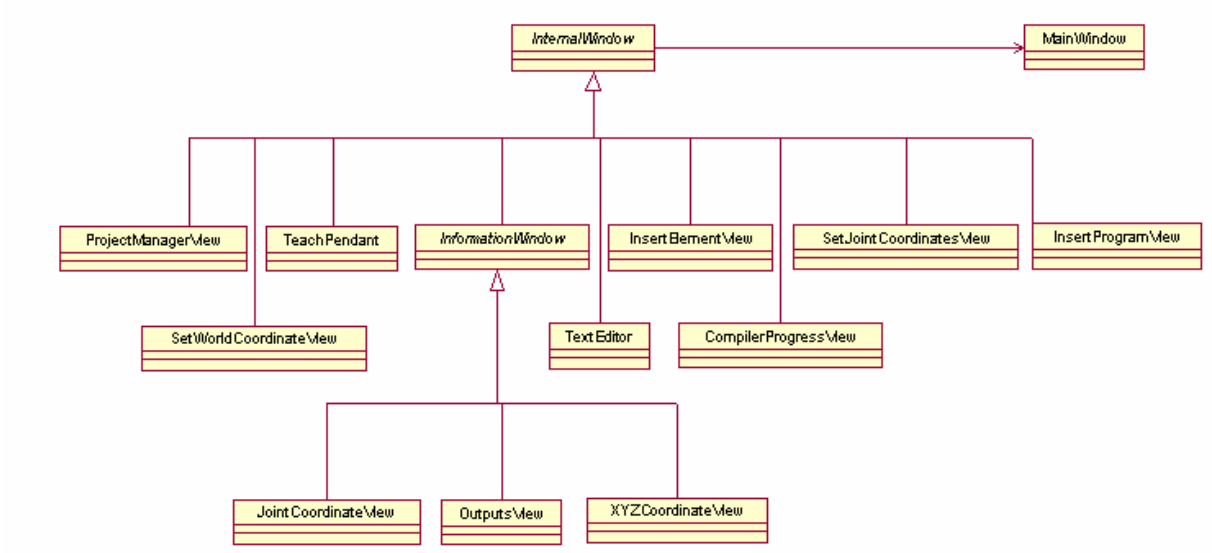

Figura 4.10: Diagrama de classes de View

Agrupa as classes responsáveis pela visualização das informações disponibilizadas pelas classes da package de controle. Além disso, possui também duas classes que todas as janelas da aplicação especializará. A seguir, serão listadas e explicadas brevemente cada uma das classes pertencentes a package:

i. *InternalWindow* – classe que disponibiliza um padrão para qualquer janela interna da aplicação;
ii. *InformationWindow* – classe que disponibiliza um padrão para as janelas que possuem o objetivo único de apenas mostrar valores;
iii. *InputsView* – classe que disponibiliza a visualização dos estados do vetor de entrada do objeto selecionado;
iv. *InsertElementsView* – classe responsável por disponibilizar a janela para a inserção de elementos no projeto;
v. *InsertProgramView* – classe responsável por disponibilizar a janela para a inserção de programas no braço robótico;
vi. *CompilerProgressView* – classe que disponibiliza o progresso da validação dos programas do projeto;
vii. *JointCoordinateView* – classe que disponibiliza a visualização dos valores das juntas do robô selecionado;
viii. *OutputsView* – classe que disponibiliza a visualização dos estados do vetor de saída do objeto selecionado;
ix. *ProjectManagerView* – classe da janela que mostra e permite manipular os elementos do projeto atualmente aberto;
x. *SetJointCoordinatesView* – classe que disponibiliza a interação com a classe controladora de manipulação das juntas do robô selecionado;

xi. *SetWorldCoordinatesView* – classe que disponibiliza a interação com a classe controladora de manipulação das coordenadas do robô selecionado;
xii. *XYZCoordiantesView* – classe que disponibiliza a visualização dos valores das coordenadas do robô selecionado;
xiii. *TeachPendant* – classe da janela do Teach Pendant, responsável por manipular o posicionamento do braço robótico;
xiv. *TextEditor* – classe que disponibiliza o editor de texto para manipular os programas;
xv. *MainWindow* – classe da janela principal do simulador. É nela que estão contidas todas as demais janelas, menu e toolbar.

4.2.5.6 Robot

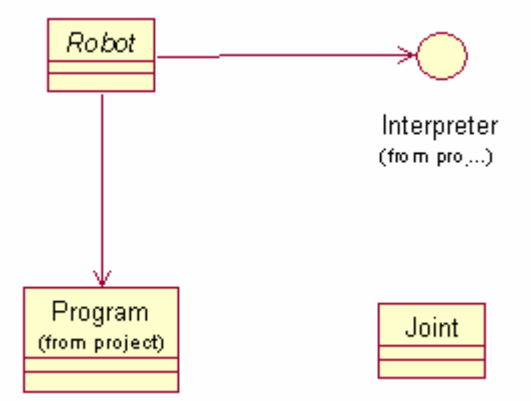

Figura 4.11: Diagrama de classes de Robot

Package que possui as classes referentes ao braço mecânico. A seguir, serão listadas e explicadas brevemente cada uma das classes pertencentes a package:

i. *Joints* – classe responsável pela representação interna das juntas de um braço robótico;
ii. *Robot* – classe abstrata responsável pela implementação do braço robótico. É dela que se deve herdar para se criar os braços robóticos específicos.

4.2.5.7 Util

Package que agrupa classes de função diversas utilizadas pelas outras classes do framework. Elas foram organizadas nas seguintes subpackages: *editor*, *file*, *math*, *view* e *patterns*. A seguir, será descrita cada uma destas subpackages:

4.2.5.7.1 Editor

Package que implementa o editor de texto utilizado para editar e executar os programas no simulador. Este editor não foi implementado totalmente pelo framework, foi obtido o código na internet `[JSO 2004], [JED 2004] e [SJE 2004]` e adaptado para as necessidades do *fwWorkCell*. Todas as suas classes são única e exclusivamente para atender as necessidades do editor de texto, portanto, não vale a pena citá-las nem descrevê-las.

4.2.5.7.2 File

Package que agrupa classes que auxiliam o manuseio de arquivos pelo framework. A seguir, serão listadas e explicadas brevemente cada uma das classes pertencentes a package:

i. *FileDialog* – classe abstrata para a apresentação de dialogs de manipulação de arquivos;
ii. *FileLoader* – classe que carrega um arquivo selecionado para o editor de texto;
iii. *FileSaver* - classe que salva o arquivo presente no editor de texto selecionado;
iv. *OpenDialog* – classe filha de *FileDialog* para selecionar qual arquivo deverá ser aberto e carregado;
v. *SaveDialog* – classe filha de *FileDialog* para selecionar o nome do arquivo a ser salvo.

4.2.5.7.3 Math

Package que agrupa classes que auxiliam no cálculo de funções matemáticas necessárias no framework. A seguir, serão listadas e explicadas brevemente cada uma das classes pertencentes a package:

i. *Generic* – classe para resolução de problemas matemáticos genéricos. Por enquanto, o único método presente é para comparar dois valores em ponto flutuante com um intervalo de tolerância;
ii. *Matrix* – classe para resolução de cálculos envolvendo matrizes;
iii. *Trigonometric* – classe para resolução de cálculos trigonométicos.

4.2.5.7.4 View

Package que agrupa classes com componentes visuais utilizados no framework. A seguir, serão listadas e explicadas brevemente cada uma das classes pertencentes a package:

i. *RScrollBar* – componente visual para a representação de uma barra de scroll;
ii. *StatusBar* – componente visual para a representação de um barra de status;

4.2.5.7.5 Patterns

Package que agrupa subpackages com a implementação dos *patterns* presentes no framework. A seguir, serão listados cada um deles:

i. *Memento* `[GAM 2000]`
ii. *Observer* `[GAM 2000]`
iii. *Prototype* `[GAM 2000]`
iv. *Singleton* `[GAM 2000]`

4.3 Recursos

O framework *fwWorkCell* possui os seguintes recursos:

i. *Interface gráfica* – permitirá a criação de simuladores com a interface totalmente gráfica através da utilização do framework que se responsabilizará por disponibilizar tal interface para o usuário final, cabendo ao desenvolvedor apenas a definição do *menu* e *barra de ferramentas*;
ii. *Portabilidade* – por ter sido escrito em Java, permitirá a criação de simuladores em qualquer plataforma que possui a Java Virtual Machine, possibilitando que o simulador criado seja executado em qualquer plataforma sem que seja necessário alterar uma linha de programa sequer;

iii. *Representação gráfica da célula de trabalho* – a célula de trabalho com o braço robótico poderá ser vista tridimensionalmente, gerando com isso um maior realismo e um melhor entendimento da solução proposta;
iv. *Edição de texto* – o framework disponibilizará recursos de edição de texto como abrir, salvar, copiar, colar e recortar;
v. *Manipulação de projetos* – o framework disponibilizará recursos para trabalhar com projetos, permitindo que se crie, edite e salve projetos;
vi. I*mpressão dos programas* – o programas criados ou abertos no simulador, poderão ser impressos;
vii. *Menus e barra de ferramentas* – permite que se crie menus e barra de ferramentas facilmente apenas criando um arquivo XML de configuração;
viii. *Célula de trabalho com vários tipos de braços robóticos* – permite que numa mesma célula de trabalho, existam vários braços robóticos diferentes que serão totalmente independentes e poderão estar rodando programas distintos;
ix. *Ajuda online* – permite que se crie e atribua um arquivo de ajuda online no simulador criado, dando um melhor suporte ao usuário final;
x. V*alidação e simulação (normal e passo-a-passo) de programas* – permite a validação e simulação dos programas pelos braços robóticos;
xi. *Breakpoints nos programas* – breakpoints poderão ser adicionados nos programas simulados;
xii. *Informações de coordenadas de juntas e/ou cartesianas dos braços robóticos* – o framework disponibiliza janelas contendo informações das coordenadas de juntas e/ou cartesianas do braço robótico ativo. Também permite que se crie novas interfaces para estas janelas através da extensão de suas classes e utilização das classes controladoras das mesmas;
xiii. *Teach Pendant* – uma janela para a manipulação do posicionamento (coordenadas cartesianas e/ou de juntas) dos braços robóticos na célula de trabalho é disponibilizada e assim como as janelas de informações, esta também poderá ser customizada;
xiv. *Informação das entradas e saídas* – o *fwWorkCell* disponibiliza uma janela informando o estado das entradas e das saídas do objeto atualmente selecionado;

4.4 Vantagens e Desvantagens do FwWorkCell

A principal vantagem da utilização do *fwWorkCell* para a criação de um novo simulador é o fato de o ambiente de edição e simulação estarem prontos, necessitando apenas a customização do(s) braço(s) robótico(s). Esta customização compreende a representação gráfica, o(s) parser(s) da(s) linguagem(ns) suportada(s) pelo(s) braço(s) robótico(s), a especialização da classe *Robot*, onde se informará as cinemáticas direta e inversa, além de outros dados pertinentes do robô em questão e a especificação deste(s) braço(s) informando, via arquivo de configuração, o número de juntas, os limites e a posição inicial de cada uma das juntas e o número de entradas e saídas.

Outra vantagem que vale a pena citar seria a possibilidade de inserção de outros braços robóticos em um simulador já criado. Isto é possível devido a arquitetura em que o framework foi criado, permitindo trabalhar com um número ilimitado de objetos diferentes. Uma vez criado, o objeto aparecerá em uma lista onde poderá ser inserido *n* vezes em um projeto. Possibilitando assim, uma outra vantagem que seria o instanciamento de um objeto diversas vezes em uma mesma célula de trabalho e estes objetos que são do mesmo tipo,

porém são independentes entre si, podem executar cada um, uma tarefa diferente ao mesmo tempo, deixando o simulador muito próximo da realidade.

E a última vantagem que cabe citar é a portabilidade tanto do framework quanto do simulador criado. Por ser escrito em Java, o *fwWorkCell* possibilita que uma infinidade de plataformas sejam suportadas, ficando limitado apenas, a disponibilidade da *JVM* para tal plataforma. Ou seja, um simulador baseado no *fwWorkCell* pode ser criado no Linux e, sem nenhuma alteração ou recompilação, ser executado no Windows XP. Garante com isso um grande alcance do aplicativo, sendo um diferencial aos outros concorrentes.

A principal desvantagem do *fwWorkCell* seria a necessidade de grandes recursos de máquina, como maior quantidade de memória RAM e processador. Isto se deve a utilização da linguagem Java que exige isso para poder carregar a *JVM*. Isto não vem a ser uma grande desvantagem, visto que a maioria dos simuladores de células de trabalho também necessitam de grande quantidade de recurso de máquina.

Outra desvantagem seria a necessidade de programação da representação gráfica dos braços robóticos. Talvez, com um pouco mais de pesquisa, o *fwWorkCell* pudesse, a partir de um arquivo de configuração (ou XML ou VRML), criar tal representação desonerando o programador desta tarefa. No seu estágio atual, no entanto, é necessário o seguinte procedimento para cada braço robótico presente no simulador:

i. Criar uma classe que implementa a interface *RobotGraph*;
ii. Implementar os métodos obrigatórios da interface;
iii. Atribuir a classe criada à classe de controle do robô;

Por fim, poder-se-ia citar a falta de alguns recursos tais como a disponibilização de objetos (cubos, cilindros, etc), edição de posições, possibilidade da troca/inserção de ferramentas aos braços robóticos, objetos para interação com os robôs (como mesa giratória, estantes, esteiras, etc), etc.

5 APLICANDO O FWWORKCELL

Para validar o *fwWorkCell* como ferramenta de base para o desenvolvimento de softwares simuladores de Células de Trabalho, ele foi utilizado em um projeto.

Tal projeto consistiu na implementação do *fwAsimov* que nada mais é do que a migração para a linguagem Java do *Asimov 1.0*, um simulador de Célula de Trabalho, contendo um braço robótico, modelo SCORBOT ER-VII da ESHED ROBOTEC, desenvolvido em Borland Delphi pelo NEAD[4] do SENAI-RS que gentilmente cedeu os fontes para que fosse realizada esta migração/validação.

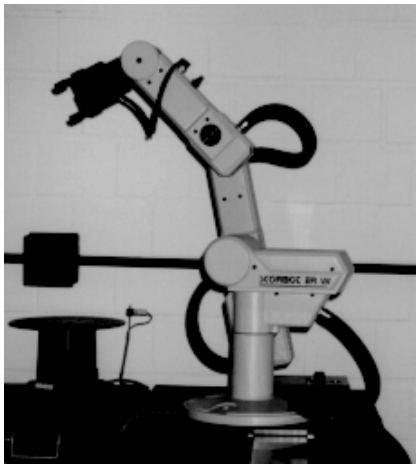

Figura 5.1: Eshed Scorbot ER-VII

O objetivo principal desta validação não consistiu em reproduzir o sistema já existente com todos os seus recursos e, sim, migrar os principais pontos que o framework abordava, validando-o e, também, comprovando sua eficiência tanto na menor utilização de recursos quanto no baixo tempo gasto para seu desenvolvimento.

O *Asimov 1.0* é um simulador de Células de Trabalho com um braço robótico e alguns objetos como esteira, mesa giratória, escaninho e peças móveis. Possui um editor da célula de trabalho responsável pelo posicionamento dos elementos no ambiente, além da configuração e referenciamento entre eles. Possui, também, todos os recursos para edição de projetos e arquivos textos dos programas.

O ESHED ER-VII, braço robótico incluso no *Asimov 1.0*, possui um Teach Pendant para controlar seus movimentos e posições, além de um interpretador para ACL, linguagem responsável por programar tal braço robótico e, um terminal onde é possível

[4] NEAD é uma área do SENAI-RS responsável pela criação de softwares simuladores educacionais para o ensino industrial.

manipular e visualizar os recursos do controlador do braço robótico, tais como variáveis, posições e programas[5].

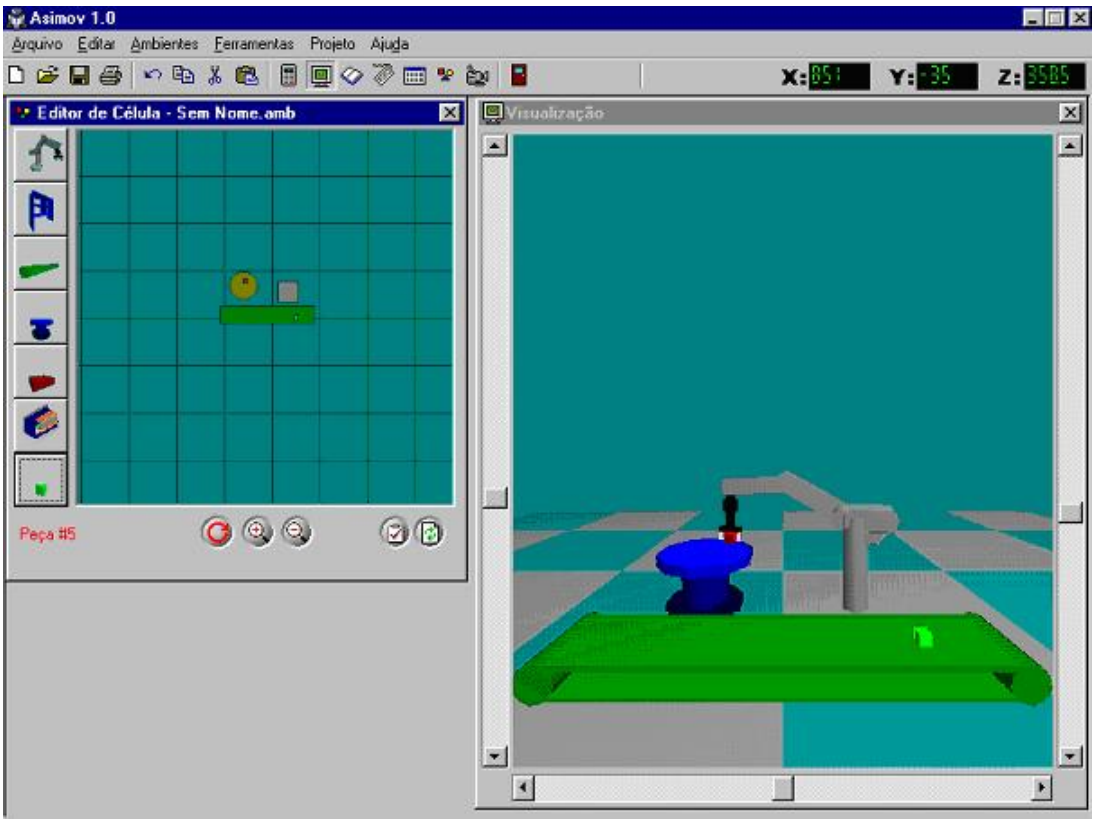

Figura 5.2: Visão geral do Asimov 1.0

5.1 Descrição da Aplicação do fwWorkCell na implementação do fwAsimov

O *fwAsimov* contém uma pequena parte do *Asimov 1.0*. Nem todos os comandos da linguagem ACL foram contemplados e nem os objetos auxiliares (esteira, mesa giratória, escaninho e peças móveis) da Célula de Trabalho foram incluídos. No entanto, seus elementos são definidos de forma suficientemente genérica para suportar a criação de um simulador com todos estes ítens faltantes através da especialização de determinadas classes do *fwWorkCell*.

Nos ítens a seguir, será demonstrado passo-a-passo com foi a implementação do *fwAsimov*.

5.1.1 Criar arquivo XML de configuração do braço robótico

O primeiro passo para o desenvolvimento do simulador foi criar um arquivo XML, indicando todas as configurações do braço robótico.

Na figura abaixo pode-se visualizar o arquivo ERVII.xml, responsável pela configuração do braço robótico (ESHED SCORBOT ER-VII) a ser incluído no simulador:

[5] Mais informações em [NEA 2004].

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Asimov>
  <Juntas>
    <Junta>
      <Limite_Superior>135</Limite_Superior>
      <Limite_Inferior>-135</Limite_Inferior>
      <Valor_Inicial>0</Valor_Inicial>
    </Junta>
    <Junta>
      <Limite_Superior>110</Limite_Superior>
      <Limite_Inferior>-60</Limite_Inferior>
      <Valor_Inicial>0</Valor_Inicial>
    </Junta>
    <Junta>
      <Limite_Superior>112</Limite_Superior>
      <Limite_Inferior>-112</Limite_Inferior>
      <Valor_Inicial>0</Valor_Inicial>
    </Junta>
    <Junta>
      <Limite_Superior>90</Limite_Superior>
      <Limite_Inferior>-90</Limite_Inferior>
      <Valor_Inicial>90</Valor_Inicial>
    </Junta>
    <Junta>
      <Limite_Superior>180</Limite_Superior>
      <Limite_Inferior>-180</Limite_Inferior>
      <Valor_Inicial>0</Valor_Inicial>
    </Junta>
  </Juntas>
  <Entradas>
    <Quantidade>10</Quantidade>
  </Entradas>
  <Saídas>
    <Quantidade>10</Quantidade>
  </Saídas>
</Asimov>
```

Figura 5.3: Arquivo ERVII.xml

Basicamente o arquivo de configuração está dividido em três partes. A primeira, onde é informado a quantidade de juntas e a configuração de cada uma; e a segunda e a terceira, onde são informadas as quantidades de entradas e saídas, respectivamente.

5.1.2 Criar classe de controle do braço robótico

O passo seguinte foi criar a classe *ERVII* responsável pelo controle do braço em questão, através da especialização da classe *fw.robot.Robot* do *fwWorkCell*. Cabe salientar que grande parte destes controles já foram abstraídos e implementados na classe *Robot* e esta extensão serviu apenas para especializar alguns ítens que são específicos de cada robô, tais como suas cinemáticas diretas e inversas.

Na figura abaixo, pode-se visualizar um trecho do arquivo *ERVII.java*, responsável pela implementação da classe *ERVII* a ser incluída no simulador:

```
public class ERVII extends Robot
{
  public static final double A1 = 50;
  private Matrix M = new Matrix(4, 4);
  ...

  public ERVII(String xmlFile) {...}
  public Object clone() {...}
  public double getJoint(int joint) throws JointIndexException {...}
  public double getJointView(int joint) throws JointIndexException {...}
  private double getTheta1(double x0, double y0)
    throws UnreachablePositionException,JointIndexException {...}
  public double getX() {...}
  public double getY() {...}
  public double getZ() {...}
  public boolean hasPitch() {...}
  public boolean hasRoll() {...}
  public boolean hasYaw() {...}
  public void setJoint(int joint, double value) throws
    JointIndexException, JointLimitException  {...}
  public void setPitch(double value) {...}
  public void setRoll(double value) {...}
  public void setX(double value) {...}
  public void setY(double value) {...}
  public void setZ(double value) {...}
  ...
}
```

Figura 5.4: Trecho do arquivo ERVII.java

5.1.3 Criar classe de interpretação da linguagem

O próximo passo foi criar a classe *ACLInterpreter*, responsável pela interpretação dos comandos ACL presentes nos arquivos de programas enviados ao robô. Esta classe é a implementacão da interface *fw.project.Interpreter* do framework.

Na figura abaixo, pode-se visualizar um trecho do arquivo *ACLInterpreter.java*, responsável pela implementação da classe *ACLInterpreter* a ser incluída no simulador:

```
public class ACLInterpreter implements Interpreter
{
  public Object clone() {...}
  public boolean syntaxCheck(File f) {...}
  public Vector getErrors() {...}
  public Vector getWarnings() {...}
  public int interpreter(String programLine, int line) {...}
}
```

Figura 5.5: Trecho do arquivo ACLInterpreter.java

5.1.4 Criar arquivo XML de configuração da barra de ferramentas

Seguindo na criação do simulador, foi necessário criar o arquivo XML de configuração da barra de ferramentas do aplicativo, indicando quais botões deveria conter e também que classes que estes deveriam chamar.

Na figura abaixo pode-se visualizar o arquivo *ToolBar.xml*, responsável pela configuração da barra de ferramentas a ser incluída no simulador:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ToolBar>
  <Button>
    <Hint>New</Hint>
    <URLIcon>fw/images/new.gif</URLIcon>
    <Class>fw.action.ActionNewEditor</Class>
  </Button>
  <Separator/>
  <Button>
    <Hint>Teach Pendant</Hint>
    <URLIcon>fw/images/teachpendant.gif</URLIcon>
    <Class>fw.action.ActionTP</Class>
  </Button>
  …
</ToolBar>
```

Figura 5.6: Trecho do arquivo ToolBar.xml

5.1.5 Criar arquivo XML de configuração do menu

O próximo passo na criação do simulador, foi necessário criar o arquivo XML de configuração do menu principal do aplicativo, indicando quais ítens deveria conter e também que classes que estes deveriam chamar.

Na figura abaixo pode-se visualizar o arquivo *Menu.xml*, responsável pela configuração do menu principal a ser incluído no simulador:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<MenuBar>
  <Menu>
    <Text>File</Text>
    <Mneumonic>F</Mneumonic>
    <Item>
      <Text>New</Text>
      <Mneumonic>N</Mneumonic>
      <Accelerator>ctrl N</Accelerator>
      <URLIcon>fw/images/new.gif</URLIcon>
      <Class>fw.action.ActionNewEditor</Class>
    </Item>
    <Separator/>
    <Item>
      <Text>Open...</Text>
      <Mneumonic>O</Mneumonic>
      <Accelerator>ctrl O</Accelerator>
      <URLIcon>fw/images/open.gif</URLIcon>
      <Class>fw.action.ActionOpenEditor</Class>
    </Item>
    …
  </Menu>
  …
  <Menu>
    <Text>Show</Text>
    <Mneumonic>S</Mneumonic>
    <Menu>
      <Text>Robot Position</Text>
      <Mneumonic>R</Mneumonic>
      <Item>
        <Text>Joint Coordinates</Text>
        <Mneumonic>J</Mneumonic>
        <Accelerator>F5</Accelerator>
            <Class>fw.action.ActionJoint</Class>
      </Item>
      …
    </Menu>
    <Menu>
      <Text>Inputs/Outputs</Text>
      <Mneumonic>I</Mneumonic>
      <Item>
        <Text>Inputs</Text>
        <Mneumonic>n</Mneumonic>
        <Accelerator>ctrl alt I</Accelerator>
```

```
            <URLIcon>fw/images/io.gif</URLIcon>
            <Class>fw.action.ActionInputs</Class>
      </Item>
      …
    </Menu>
    …
  </Menu>
  …
</MenuBar>
```

Figura 5.7: Trecho do arquivo Menu.xml

5.1.6 Criar a classe principal do simulador

O úlitmo e mais importante passo foi criar a classe *Asimov*, a classe principal do simulador. É nela que todos os ítens criados acima foram adicionados ao novo aplicativo.

Diferentemente dos passos anteriores, onde apenas se mostrava os trechos dos arquivos, neste passo será mostrada toda a classe e após isso, será explicada cada linha dela, com o intuito de demonstrar a simplicidade da implementação de um software simulador de células de trabalho utilizando o *fwWorkCell*.

```
public class Asimov
{
  public static void main(String[] args)
  {
    MainWindow formMain = new MainWindow("Asimov 1.0 fw");

    ERVII robot = new ERVII("config\\ERVII.xml");

    ACLInterpreter interpreter = new ACLInterpreter();
    robot.setInterpreter(interpreter);

    formMain.addRobotPrototype(robot);

    formMain.setXmlMenu("config\\Menu.xml");
    formMain.setXmlToolBar("config\\ToolBar.xml");
    formMain.setVisible(true);
  }
}
```

Figura 5.8: Arquivo Asimov.java

```
MainWindow formMain = new MainWindow("Asimov 1.0 fw");
```

Nesta linha é criada uma instância da classe MainWindow do framework. Esta classe é responsável pelo gerenciamento e andamento de toda aplicação.

```
ERVII robot = new ERVII("config\\ERVII.xml");
```

Instancia um objeto da classe ERVII passando o arquivo xml correspondente como parâmetro.

```
ACLInterpreter interpreter = new ACLInterpreter();
robot.setInterpreter(interpreter);
```

Instancia um objeto da classe ACLInterpreter e adiciona como sendo o interpretador do robô ERVII.

```
formMain.addRobotPrototype(robot);
```

Adiciona o robô ERVII como um dos robôs do simulador. Neste caso, ele será o único, mas se desejasse incluir mais modelos de robôs, o procedimento seria o mesmo.

Obs.: Se objetos auxiliares fizessem parte do simulador, o procedimento seria muito

semelhante a este.

```
formMain.setXmlMenu("config\\Menu.xml");
```
Informa ao fwWorkCell qual o arquivo para a configuração do Menu Principal.

```
formMain.setXmlToolBar("config\\ToolBar.xml");
```
Informa ao fwWorkCell qual o arquivo para a configuração da Barra de Ferramentas.

5.2 Interface gráfica do fwWorkCell

A interface gráfica do *fwWorkCell* foi concebida utilizando Swing do Java. A seguir serão apresentadas algumas telas do *fwAsimov* criado a partir do *fwWorkCell*.

Na figura 5.9, é apresentado a inclusão de um robô ao projeto da célula de trabalho. A quantidade de robôs presentes na células é praticamente ilimitada e o processo é o mesmo.

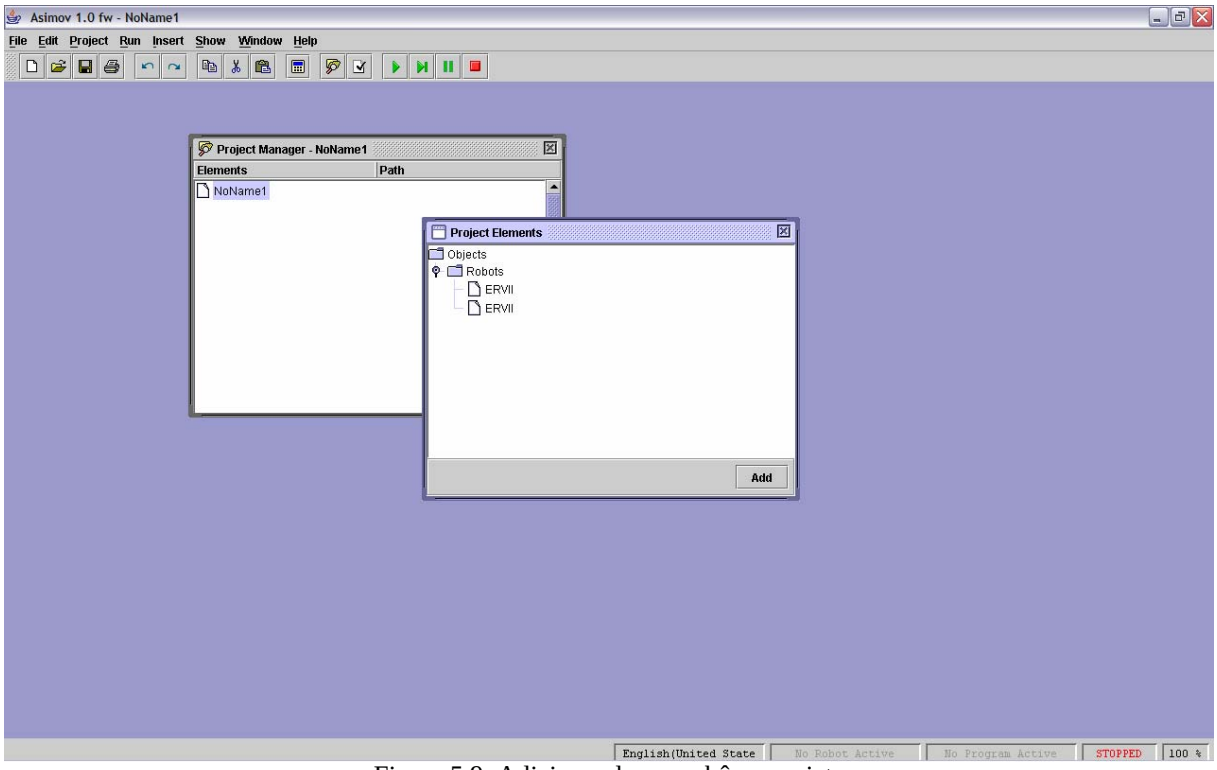

Figura 5.9: Adicionando um robô ao projeto

Com programas adicionados ao robô, pode-se verificar na figura 5.10, a escolha do item da popup, *Syntax Check*, responsável por verificar se o programa foi escrito corretamente.

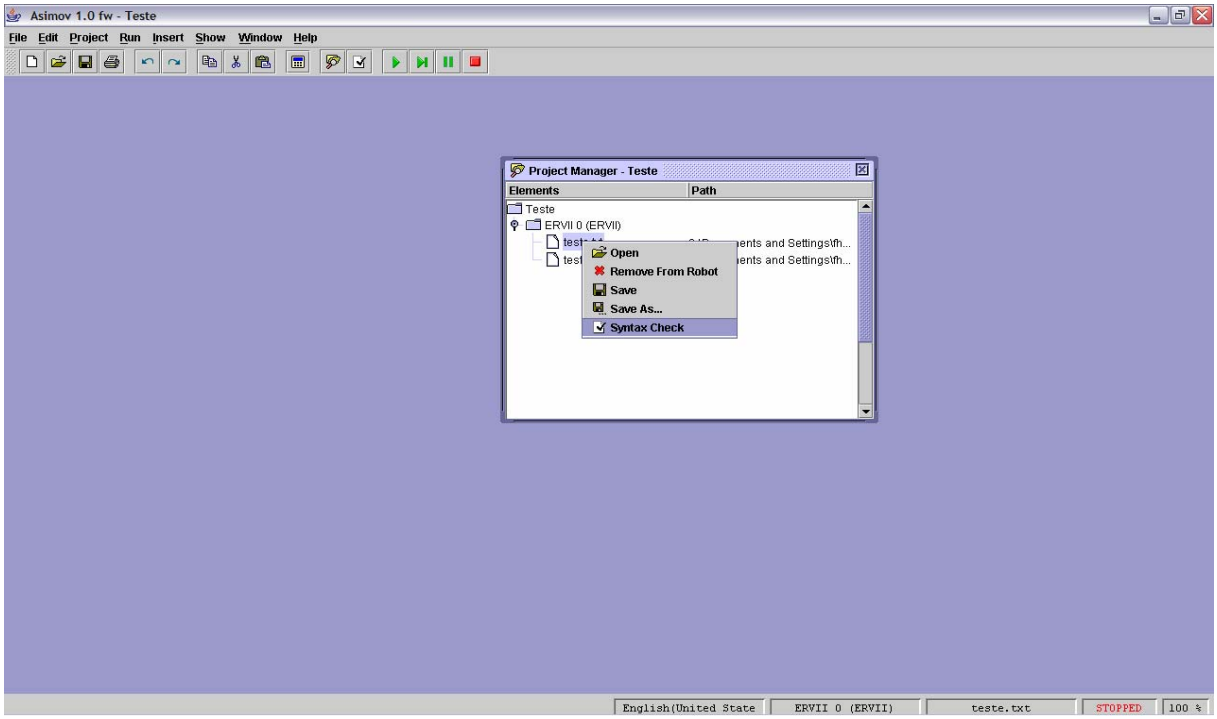

Figura 5.10: Realizando a sintaxe do programa do robô

Na figura 5.11, é apresentado a execução de um programa do robô. Cabe salientar a linha destacada no editor de texto, mostrando qual linha está sendo executada no momento e também que tal execução pode ser parada a qualquer instante com o pressionamento do botão *Pause*.

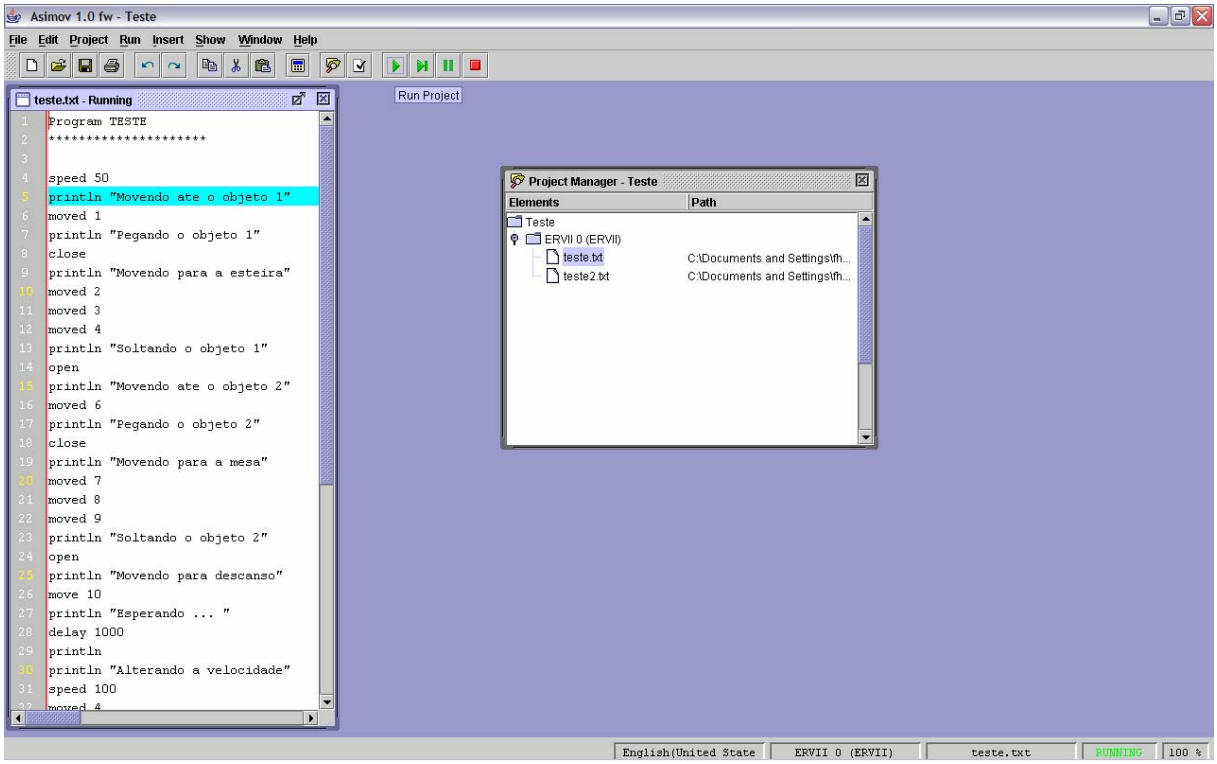

Figura 5.11: Execução de um programa do braço robótico

E por último, na figura 5.12, é mostrada a manipulação do robô através do *Teach Pendant* e as posições atuais das juntas (*Joint Coordinates*) e coordenadas(*XYZ Coordinates*), além dos estados das entradas(*Inputs*) e saídas(*Outputs*).

Cabe salientar que a cor dos valores das juntas possuem três estados: preto, amarelo e vermelho. Vermelho quando a junta atingiu o seu limite, amarelo quando está perto de atingir e o preto para os outros casos.

Outro fato relevante a citar é possibilidade de customização das janelas de informaçoes, como as visualizações dos valores das juntas e das coordenadas, por exemplo e do Teach Pendant. Isto se deve a separação das parte de controle e visualização, permitindo que se utilize uma outra janela para visualizar os mesmos dados.

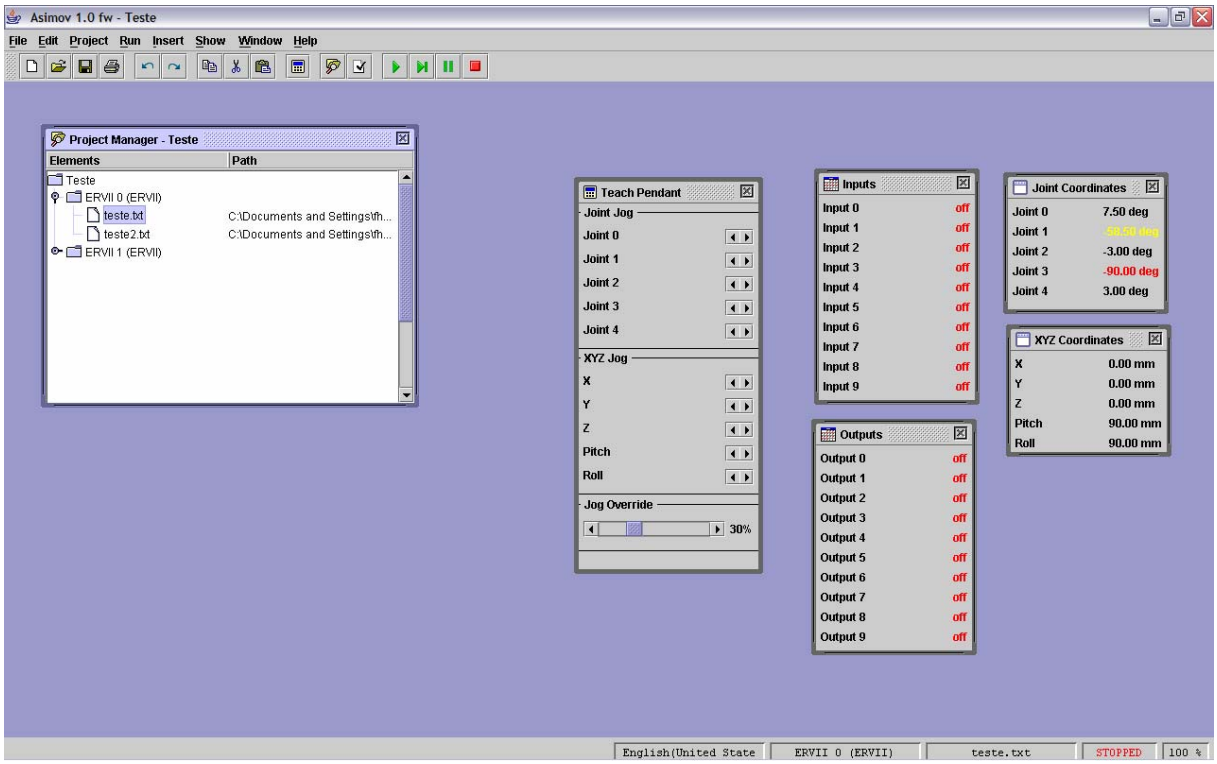

Figura 5.12: Manipulação do robô através do Teach Pendant e visualização de suas posições e estados

6 CONCLUSÃO

Este trabalho foi desenvolvido com o intuito de facilitar a criação de softwares simuladores de células de trabalho. Com a inexistência de uma ferramenta que desse um total suporte ao desenvolvimento, decidiu-se pela criação de tal ferramenta. Todavia, era necessário possuir uma estrutura orientada a objetos capaz de permitir o suporte a aplicativos a partir de uma estrutura pré-determinada, sendo então, necesário a criação de um framework orientado a objetos.

Para projetar tal framework, foi necessário um estudo mais apronfundado de UML[6], Padrões de Projeto[7] e Fundamentos de Robótica[8] e de Simuladores[9]. Com estes conceitos em mente, foi definido e construído o *fwWorkCell*.

O desenvolvimento do *fwWorkCell* inicia-se a partir do desenvolvimento de classes genéricas para a abstração de braços robóticos, seguindo pela construção de todo ambiente para suportar a manipulação de arquivos e projetos, passando pela criação de classes de controle e visualização/manipulação de informações dos objetos da célula de trabalho e finalizando com a implementação dos controles de simulação de todo o ambiente.

Para validar e comprovar sua eficiência, foi escolhida a migração de um software simulador educacional de células de trabalho já existente, desenvolvido em Borland Delphi, para a linguagem Java utilizando o *fwWorkCell* para isto. Após a migração de tal simulador, pode-se perceber a eficiência e o ganho de tempo e qualidade que a utilização do framework trouxe ao projeto como um todo, sem falar na possibilidade aberta de estender este simulador com a inclusão de outros modelos de braços robóticos ou objetos auxiliares.

Finalizando, este trabalho veio facilitar o desenvolvimento de aplicativos simuladores de células de trabalho. Antes, para a criação de tal aplicativo, era necessário se preocupar não somente com o análise/projeto/desenvolvimento do braço robótico e sim, do simulador como um todo, incluindo aí toda a parte de manipulação de arquivos e projetos, controle de simulação, edição de texto, etc. Além disto, se o simulador não fosse desenvolvido pensando na reutilização de código, a necessidade de criar um novo simulador ou adicionar um outro modelo de braço robótico a um simulador já existente, traria grandes problemas e talvez a necessidade de reescrever grande parte do que já havia sido feito. Com o *fwWorkCell*, é possível construir simuladores em pequena quantidade de tempo, com uma qualidade superior e um alto grau de reaproveitamento de código.

---

[6] [BOO 200], [BOG 2002], [FOW 2000] e [FUR 1998] foram consultados como fontes de estudo.

[7] [COO 1998], [DES 2004], [GAM 2000], [LAR 2000], [PRE 1994] serviram como fontes de estudo e pesquisa.

[8] Foram consultados [FRA 2003], [SAB 1999] e [SAN 2003].

[9] [ARS 2003] e [COS 2003] serviram como fontes de pesquisa e estudo.

6.1 Contribuições

As principais contribuições deste trabalho são:

- Diminuir consideravelmente o tempo de desenvolvimento de um software simulador de células de trabalho;
- Retirar a necessidade de programação da parte de edição e controle da simulação do aplicativo, como por exemplo, abrir e salvar programas e projetos, copiar e colar textos, iniciar e parar a simulação, etc. Focando, assim, o desenvolvimento do simulador no que realmente interessa que são os braços robóticos;
- Permitir, depois de um projeto criado, incluir mais braços robóticos, tendo apenas que adicionais tais robôs sem interferir no que já havia sido desenvolvido;
- Reaproveitamento total da programação dos braços robóticos para aplicativos simuladores futuros;
- Facilitar a criação e inserção de objetos na célula de trabalho;

6.2 Limitações do trabalho

As principais limitações do trabalho – a partir do processo de validação do *fwWorkCell* – são:

- Visualização tridimensional da célula de trabalho deve ser feita completamente pela aplicação que está utilizando o framework;
- Não foi possível manipular o posicionamento do braço robótico na célula de trabalho; e
- O aplicativo gerado exige recurso de máquina considerável, uma vez que necessita da JVM para ser executado. Isso não vem a ser um grande problema, uma vez que as máquinas padrão atuais já possuem tais recursos.

6.3 Trabalhos futuros

Pode-se citar as seguintes sugestões para trabalhos futuros:

- Criar editores para a criação dos arquivos XML de configuração dos braços robóticos, menu principal e da barra de ferramenta;
- Disponibilizar objetos auxiliares que interagem com os braços robóticos da célula, diretamente no framework, deixando a decisão de incluí-los ou não para a equipe de desenvolvimento do simulador;
- Possibilitar a separação da ferramenta de trabalho do resto do robô, podendo criar assim, um magazine de ferramentas onde ele pude-se trocá-las por meio de programação;
- Permitir a criação de objetos tridimensionais através de arquivos XML descritivos, onde os mesmos seriam criados a partir deles;
- Criar um ambiente tridimensional diretamente no framework, permitindo, através de um arquivo XML de configuração, que a equipe de desenvolvimento, decida quais objetos farão parte da célula de trabalho;
- Permitir o posicionamento de todos os elementos presentes na Célula de Trabalho, possibilitando assim, uma maior realidade na utilização do simulador;

- Construir a representação gráfica dos braços robóticos da célula de trabalho a partir de um arquivo XML descritivo e de configuração, onde este arquivo além de dar a descrição das partes formadoras do robô, também indicaria as partes de maneira que fosse possível fazer a simulação da movimentação do tal braço robótico;
- Possibilitar que um mesmo braço robótico possua vários interpretadores de diferentes linguagens como acontece no mundo real da robótica; e

- Aproveitar o recurso de internacionalização que o Java possui, para permitir uma total adaptabilidade do produto gerado a língua do país que ele fosse distribuído.

REFERÊNCIAS

[APA 2004] APACHE ANT. **Apache ANT - Welcome**. Disponível em: <http://ant.apache.org>. Acesso em: 01 out. 2004.

[ARA 2003] ARAGÃO, A.; ROMERO, R.; MARQUES, E. Computação reconfigurável aplicado à robótica. In: WORKSHOP EM COMPUTAÇÃO RECONFIGURÁVEL, CORE, 2000, Marília. **Computação Reconfigurável:** Experiências e Perspectivas. Rio de Janeiro: Editora Brasport, 2000. v. 1.p. 184-188.

[ARS 2003] ARSHAM, H. **Modeling and Simulation**. Disponível em: <http://ubmail.ubalt.edu/~harsham/simulation/sim.htm>. Acesso em: 20 jun. 2003.

[BOG 2002] BOGGS, W.; BOGSS, M. **Mastering UML with Rational Rose**. London: SYBEX, 2002.

[BOO 2000] BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. **UML – Guia do Usuário**. Rio de Janeiro: Campus, 2000.

[COO 98] COOPER, J. W. **The Design Patterns**. Boston: Addison-Wesley Professional, 1998.

[COI 2003] COSIMIR Web Presentation. Disponível em: <http://www.cosimir.com/English/COSIMIRManufacturing/COSIMIRManufacturing.htm>. Acesso em: 26 jun. 2003.

[COS 2003] COSTA, M. A. B. **Simulação de Sistemas**. 2002. Disponível em: <http://www.simucad.dep.ufscar.br/dn_sim_doc01.pdf>. Acesso em: 21 jun. 2003.

[DEC 2000] DÉCIO, O. **XML – Guia de Consulta Rápida**. [S.l.]: Novatec, 2000.

[DEI 2003] DEITEL, H.; DEITEL, P. **Java, como programar**. Porto Alegre: Bookman, 2003.

[DES 2004] ROCHA, H. da**. Curso J930:** Design Patterns**.** Disponível em: <http://www.argonavis.com.br/cursos/java/j930/J930_04.pdf>. Acesso em: 10 mar. 2004.

[FED 2004] FEDERIZZI, G. L**. APIs Java para XML.** Disponível em: <http://www.inf.ufrgs.br/procpar/disc/inf01008/trabalhos/sem01-1/t2/apis_xml_java/>. Acesso em: 20 jan. 2004.

[FEN 2004] FENDT, W. **Segunda lei de Kepler**. Disponível em: <http://planeta.terra.com.br/educacao/pifer/ph11br/ph11br/keplerlaw2_br.htm>. Acesso em: 01 ago. 2004.

[FON 99] FONTOURA, M. F. **Uma Abordagem Sistemática para o Desenvolvimento de Frameworks.** 1999. Tese (Doutorado) – PUC-RJ, Rio de Janeiro.

[FOW 2000] FOWLER, M.; SCOTT, K. **UML Essencial:** Um breve guia para a linguagem padrão de Modelagem de Objetos. Porto Alegre: Bookman, 2000.

[FRA 2003] FRANCHIN, M. N. **Elementos de Robótica**. Disponível em: <http://www.dee.bauru.unesp.br/~marcelo/robotica/conteudo.html#P1>. Acesso em: 15 jun. 2003.

[FUR 98] FURLAN, J. D. **Modelagem de Objetos através da UML – Análise e Desenho Orientado a Objetos**. São Paulo: Makron Books, 1998.

[GAM 2000] GAMMA, E. et al. **Padrões de Projeto:** Soluções reutilizáveis de Software Orientado a Objetos. Porto Alegre: Bookman, 2000.

[HAT 2003] HATCHER, E.; LOUGHRAN, S. Java Development with ANT. São Paulo: Makron Books, 2003.

[HIS 2003] HISTÓRIA da Robótica. Disponível em: <http://www.robos.com.br/arquivo/HIST%C3%93RIA.html#inicio>. Accesso em: 15 jun. 2003.

[HOR 2001] HORSTMANN, C.; CORNELL, G. **Core Java 2 – Recursos Avançados**. São Paulo : Makron Books, 2001.

[JED 2004] PESTOV, S**. jEdit – Programmer's Text Editor**. Disponível em: <http://www.mycgiserver.com/~lhbergla/SimpleJavaEditor/intro.jsp>. Acesso em: 09 jan. 2004.

[JOH 93] JOHNSON, R. E. **How to design frameworks.** [S.l.: s.n.], 1993.

[JSO 2004] PLEVRAKIS, P**. The JSource Java IDE**. Disponível em: <http://jsource.sourceforge.net/>. Acesso em: 15 jan. 2004.

[LAR 2000] LARMAN, C. **Utilizando UML e Padrões:** Uma Introdução à Análise e ao Projeto Orientado a Objetos. Porto Alegre: Bookman, 2000.

[MCL 2001] MCLAUGHLIN, B. **Java & XML**. Beijing: O'Reilly, 2001.

[MFS 2002] MICROSOFT Flight Simulator 2002. Disponível em: <http://www.microsoft.com/games/pc/fs2002.aspx>. Acesso em: 02 jul. 2004.

[NEA 2004] NÚCLEO de Educação a Distância. Disponível em: <http://nead.rs.senai.br/nead/inicio.php>. Acesso em 04 jul. 2004.

[PRE 94] PREE, W. **Design patterns for object oriented software development**. Workingham: Addison-Wesley, 1994.

[SAB 99] SABBATINI, R. M. E. **Os Robôs Chegam à Sala Cirúrgica**. 1999. Disponível em: <http://www.nib.unicamp.br/papers/checkup-07.htm>. Acesso em: 03 maio 2003.

[SAN 2003] SANTOS, N. M.; FERRARI, D. G.; ETO, R. M. **Mundo da Robótica**. Disponível em: <http://www.din.uem.br/ia/robotica/>. Acesso em: 15 jun. 2003.

[SCH 97] SCHMID, H. A. Systematic framework design. **Communications of ACM**, New York, 1997.

[SIL 2000] SILVA, R. P. e. **Suporte ao desenvolvimento e uso de frameworks e componentes**. 2000. 262 p. Tese (Doutorado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

[SIM 2004] SIMULATING Schematics Using Verilog-XL. Disponível em: <http://web.ecs.syr.edu/~anunezal/homepage/tools/verilogXL/verilogXLtutor.htm>. Acesso em: 05 jul. 2004.

[SJE 2004] SimpleJavaEditor. Disponível em: <http://www.mycgiserver.com/~lhbergla/SimpleJavaEditor/intro.jsp>. Acesso em: 10 jan. 2004.

[SUN 2001] SUN MICROSYSTEMS. **Java Look and Feel Design Guidelines:** Advanced Topics. [S.l.]: Addison-Wesley Professional, 2001.

[TIL 2002] TILLY, J.; BURKE, E. **Ant – The Definitive Guide**. [S. l.]: O'Reilly, 2002.

[WIR 90] WIRFS-BROCK, A. *et al.* Designing reusable designs: experiences designing object-oriented frameworks. In: OBJECT-ORIENTED PROGRAMMING SYSTEMS, LANGUAGES AND APPLICATIONS CONFERENCE; EUROPEAN CONFERENCE ON OBJECT-ORIENTED PROGRAMMING, 1990, Ottawa. **Proceedings…** [S.l.: s.n.], 1990.

[WIR 91] WIRFS-BROCK, A. et al. Designing reusable designs: experiences designing object-oriented frameworks. **SIGPLAN Notices**, New York, v. 26, n. 11, Oct. 1991. Trabalho apresentado na OOPSLA, 1991.