Framework para Programação Offline de robôs

Vitor Santos Bottazzi

Dissertação submetida à Universidade do Minho para obtenção do grau de Mestre em Electrônica Industrial, elaborada sob a orientação de Professor Jaime Fonseca

Departamento de Electrônica Industrial
Escola de Engenharia
Universidade do Minho
Braga, 2006

Abstract

This master thesis addresses the on-line programming problems which deals with reduce time integration to manufacture cells thought off-line robots programming using exchange file projects. The industrial robot programming is a work for specialist in robotics. Today, this work is very hard because there are many robot manufacturers with different languages and different programming environments. Although, off-line programming is an way that can reduce drastically the machines stop time to maintenance. In this work is proposed a off-line programming environment, capable of extract the boundary information of a piece from neutral files. Handle this information to program some specific application like a piece placement, welding, or painting of a single piece placed at a stable position. This tool is based in an abstract model to program robots, encapsulate in java classes. The main advantage of object oriented paradigm is best source code utilization. Grouping the business classes in modules by functionalities, we can reduce complexity between low matching. Recognized patterns like Facade and Template Method constructs the base to develop this programming framework. The programming robot languages used in this work was Rapid, Karel and Melfa Basic IV, respectively languages used by ABB, Fanuc and Mitsubishi constructors. First experiments demonstrate the feasibility and the efficiency of the approach.

Keywords: Framework, off-line programming, exchange files, robot code generator.

Resumo

Esta tese de mestrado demonstra alguns dos problemas comuns à programação on-line de robôs industriais e como o tempo gasto com a integração de uma célula de manufactura pode ser reduzido utilizando programação off-line de robôs associada aos arquivos de troca/neutros utilizados pelas ferramentas CAD. A programação de robôs industriais é usualmente um trabalho para especialistas. Actualmente, este trabalho é muito difícil devido à existência de um grande número de fabricantes de robôs, cada um com a sua linguagem e ambiente de programação proprietários. A programação off-line destes equipamentos pode reduzir drasticamente o tempo de paragem da linha de produção para a troca ou manutenção de um determinado programa. Esta tese propõe um ambiente de programação off-line de robôs industriais capaz de extrair a informação geométrica de uma peça através da leitura de um arquivo de troca, e utilizar esta informação para programar uma tarefa específica como por exemplo posicionamento, soldadura, ou pintura de uma peça que se encontre numa posição estável. Esta ferramenta é baseada num um modelo abstracto para a programação de robôs industriais, representado por classes Java. A principal vantagem da escolha do paradigma orientado a objecto é a melhor reutilização do código fonte. Agrupando as classes de negócio em módulos divididos por suas respectivas funcionalidades podemos reduzir a complexidade e o acoplamento.

Padrões de projecto reconhecidos como Facade e Template Method construíram a base para desenvolver este motor direccionado para a programação de robôs. As linguagens utilizadas neste projecto foram Rapid, Karel e Melfa Basic IV, linguagens exploradas respectivamente pelos fabricantes ABB, Fanuc e Mitsubishi. Os primeiros testes práticos efectuados num ABB demonstraram que esta alternativa é viável e pode ser bastante eficiente.

Palavras-chave: Framework, programação off-line, arquivos de troca, gerador de código para robôs.

Conteúdo

Anexos

Anexo1: Sintaxe Melfa Basic IV para movimento por Junta

Anexo2: Sintaxe Melfa Basic IV para movimento Linear

Anexo3: Sintaxe Melfa Basic IV para movimento Circular

Anexo4: Sintaxe Rapid ABB

Anexo5: Javadoc – Documentação do software

Capítulo 1

Introdução

Para que um robô industrial possa realizar trabalho útil, é necessário programar previamente a sequência de movimentos pretendidos. Os actuais robôs industriais necessitam de uma pesada infra-estrutura de programação para torná-los funcionais. Os seus controladores estão cada vez mais sofisticados usam sistemas operativos proprietários, acessíveis apenas através dos seus ambientes de programação. Apesar da grande evolução que vem modernizando os processos de produção industriais robotizados, a programação dos robôs, ainda é, usualmente efectuada de duas formas:

- Programação Manual (On-line);

- Programação Virtual (Off-line);

A programação manual, ou on-line, refere-se a ensinar uma trajectória ao robô através da sua movimentação utilizando o joystick ou algum outro dispositivo semelhante [Lee, 1990]. Este tipo de programação apresenta algumas desvantagens: é muito lenta, necessita de equipamento disponível para efectuar a programação, dificuldade de manuseamento deste equipamento, o processo requer algum conhecimento sobre a linguagem proprietária do fabricante do robô, e um mínimo de conhecimento técnico para compreender o funcionamento do equipamento. Estas desvantagens podem ser criticas na indústria devido a necessidade de parar o processo produtivo por um período considerável de tempo. Uma possibilidade para minimizar algumas das desvantagens referidas nomeadamente a redução do tempo de paragem da linha de produção é a utilização de ambientes de programação off-line. Estes ambientes são plataformas que permitem a simulação de uma célula de trabalho robotizada. Neste contexto o programador do robô precisaria de conhecer apenas a linguagem genérica de simulação, e não a linguagem específica

desenvolvida para cada um dos robôs que podem fazer parte de uma determinada linha de produção. Outras vantagens proporcionadas pelos ambientes de programação off-line são as seguintes : disponibilizam bibliotecas com comandos de alto nível que facilitam o desenvolvimento de aplicações para processos complexos, como por exemplo a pintura e a soldadura; possibilidade de analisar a cinemática inversa do robô para o planeamento de trajectórias livres de colisões; determinar o tempo de ciclo do processo; gerar automaticamente o código para diferentes controladores. Apesar destas vantagens, este tipo de aplicações baseada na representação 3D do ambiente em que o robô se move, tem grandes dificuldades em se impor sempre que esse mesmo ambiente se altera com frequência. Este tipo de situação, pode ocorrer por exemplo, em processos de soldadura em que o tamanho e a forma das peças a soldar mudam frequentemente ou em processos similares, dificultando desta forma a construção 3D do ambiente. Actualmente, as ferramentas CAD estão sendo utilizadas largamente no desenvolvimento e documentação de produtos, assim como, no acompanhamento do seu ciclo de vida. Existem muitas ferramentas CAD comerciais no mercado, como por exemplo: AutoCAD, SolidWorks, Ideas e Cimatron, e cada uma delas com o seu formato proprietário de armazenamento da informação geométrica do produto. No entanto, é possível trocar informação entre as diferentes ferramentas CAD através da utilização de arquivos de formato neutro ou de troca como por exemplo o STL, IGES, STEP e SET. Este trabalho apresenta uma solução genérica para a programação de diferentes robôs industriais baseando-se na informação geométrica relevante, extraída dos arquivos neutros provenientes das ferramentas CAD. Esta solução foi testada em duas plataformas distintas de robôs industriais: Mitsubishi (Mitsubishi Move Master Industrial Robot) e ABB (modelo IRB 140 com controlador IRC5). A tese foi organizada da seguinte forma: no capitulo 2 é apresentado o estado da arte da programação Off-line de robôs; no capitulo 3 são apresentados os principais formatos neutros utilizados pelas ferramentas CAD do mercado; no capitulo 4 é apresentada a ferramenta de geração de código para a programação de robôs industriais e apresentados os algoritmo utilizados na extracção da informação geométrica das peças a partir dos arquivos neutros; no Capitulo 5 e 6 são exibidos os resultados e a conclusão; capitulo 7 para trabalhos futuros.

Capítulo 2

Estado da Arte da programação Off-line de robôs

Existem várias ferramentas disponíveis actualmente no mercado para a programação off-line de robôs industriais. Com estas ferramentas é possível programar os movimentos do robô para executar as mais diferentes tarefas. No entanto, a maioria das ferramentas disponíveis no mercado são proprietárias encontrando-se direccionadas para a programação de uma grupo restrito de robôs, interpretando uma linguagem compilada desenvolvida pelo fabricante do grupo de robôs em questão. A programação de robôs industriais é convencionalmente feita através do ensinamento de pontos de passagem pelo programador ao robô ficando a cargo do controlador do mesmo calcular a cinemática inversa referente ao deslocamento dos vários motores ligados a cada junta do manipulador. O programa do robô tem como objectivo representar uma sequência de somas de deslocamentos de juntas, que resultará no movimento descrito pela ferramenta acoplada à extremidade (flange) do braço mecânico, na execução de uma determinada tarefa. Este processo de ensinar denominado teaching, possui algumas características particulares descritas por [Fuller, 1999]: Primeiro é definido o problema. Depois disso planeados os passos necessários à solução. Em seguida exprimi-se estes passos numa linguagem que o computador do robô entenda, depois estes comandos são inseridos como instruções no controlador do robô, e finalmente é feito o teste para comprovar o resultado da solução desenvolvida. Todo este processo requer um gasto considerável de tempo devido a alguns factores como por exemplo : a reserva de equipamento disponível para a programar o robô, a dificuldade de manuseio do equipamento, experiência relativa à linguagem de programação utilizada e principalmente ao conhecimento tecnológico necessário para com-

preender o funcionamento do equipamento. Por outras palavras, é necessário parar a linha de produção por um tempo considerável, para a programação de uma nova tarefa ou para a manutenção de uma rotina qualquer, além de que a mão-de-obra necessária para esta operação precisa ser altamente especializada. Uma das grandes vantagens da programação off-line é a possibilidade de simular o teaching num ambiente gráfico 3D através de um PC convencional, flexibilizando a programação e reduzindo drasticamente o tempo de paragem da célula de trabalho. A criação de novas tarefas, melhoramento ou manutenção de um determinado programa pode ser feito muitas vezes em ambiente virtual, reduzindo o tempo de programação de uma tarefa em até 60 por cento.

Actualmente, a programação totalmente baseada em teaching é muito utilizada, quando comparada com a programação off-line auxiliada por computador. Para tarefas onde as condições de produção são constantes, ainda é uma solução viável. Mas quando estas condições possuem uma grande probabilidade de mudança, as peças a serem trabalhadas mudam constantemente de tamanho e forma por exemplo, a programação por teaching deixa de ser interessante devido ao grande tempo gasto na reprogramação de cada tarefa para cada novo caso.

Outra grande desvantagem da programação online é a dificuldade em programar para diferentes plataformas:

- O joystick (Teach Pendant) possui diversos tamanhos, formas e sequência de comando que variam de fabricante para fabricante.

- Existem várias linguagens de programação que apesar de possuir os mesmos princípios, possuem facilidades diferentes, implementadas de acordo com a área de aplicação foco do fabricante.

- É necessário algum conhecimento sobre os diferentes componentes electro-mecânicos que compõem o robô para se praticar uma programação efectiva de uma determinada tarefa.

Em contrapartida, a programação off-line está sujeita à recriação de todo o ambiente de célula antes do inicio da primeira programação para uma determinada situação, apesar de acelerar bastante o tempo de integração da célula de trabalho. Já a algum tempo, a utilização de ficheiros CAD para a criação e documentação de produtos se tornou bastante comum na industria. Devido a grande diversidade de softwares utilizados para este fim como por exemplo o AutoCAD, o SolidWorks, o Ideas e o Cimatron, criou-se também a necessidade de utilizar arquivos neutros para trocar informação entre as diversas plataformas. A tendência natural é para que se utilize a informação

dos ficheiros neutros ou de troca (STL, IGS, STEP, SAT) para acompanhar o ciclo de vida de um determinado produto de forma a poder manipular os seus componentes para fins destinados à produção e comercialização. Com a crescente evolução do processos de produção aumenta também a procura por uma ferramenta que flexibilize a programação dos robôs utilizando a informação extraída de ficheiros neutros como referência para essa programação. Caso a linha de produção possua mais do que um tipo de robô, esta pesquisa poderá também acumular informação acerca de diferentes linguagens de programação para robôs optimizando também o uso do equipamento disponível para a produção.

Capítulo 3

Visão geral sobre Formatos Neutros

Devido à globalização, emerge cada vez mais a necessidade de representar produtos em diferentes plataformas ou na mesma plataforma mas com a possibilidade de intercâmbio de informação entre diferentes empresas, nomeadamente produtos destinados à indústria de manufactura e construção. Esta tendência aumenta uma vez que o ciclo de vida de um produto vai desde o design, através da manufactura passando pela sua manutenção a longo prazo até que expire o seu "tempo de vida". Se for possível reunir a história de um produto num único arquivo capaz de representar as suas características e possibilitando em qualquer momento acrescentar novas funcionalidades ou corrigir o seu design, representando desta forma uma economia na documentação deste produto, e no tempo gasto para analisá-la. A utilização de sistemas CAD popularizou-se muito rapidamente no mercado industrial, principalmente pela garantia de aumento na produtividade, qualidade em design, redução da utilização de papel e tempo gasto trabalhando sobre pranchetas de desenho. Reforçando a ideia de que o gargalo no desenvolvimento de produtos, para as mais diferentes áreas da produção, comércio e serviços, é o "design". A utilização de ficheiros no formato CAD optimizou de forma espantosa a produção e manutenção dos mais variados produtos, nas mais diversas áreas. Nomeadamente na área da electrónica avançada, que é um sector em constante e rápido desenvolvimento e onde cada vez mais a documentação de produtos e acesso imediato à informação são o diferencial competitivo entre as diferentes empresas que disputam mercado na área tecnológica.

O estado da arte dos projectos CAD apareceu em virtude da importân-

cia atribuída ao desenvolvimento evolutivo do design de produtos e transformou-se numa grande descoberta para diversas áreas da ciência. A história dos ficheiros CAD começa por volta dos anos 80 quando se percebeu que o crescimento em número e complexidade de ferramentas para troca de dados entre diferentes plataformas tornava-se muito dispendiosa. Até aquele momento as ferramentas do mercado usavam um código proprietário e dependente dos códigos ASCII ou binário, para representar os dados de design de produtos ou de suas peças componentes. A sua utilização popularizou-se pioneiramente no sector da construção civil, com o aparecimento do AutoCAD, ferramenta pioneira desenvolvida pela AutoDesk, empresa conhecida mundialmente no ramo de produção de soluções para o auxilio ao design e engenharia. Estes são alguns dos motivos que explicam esta tendência que vem influenciando tão rapidamente a indústria do design de produtos a utilizar. Devido a este facto, nota-se também uma aposta em ferramentas de design auxiliadas por computador(CAD) para supervisão, montagem, pintura, testes de resistência e mesmo apenas para a documentação de produtos. Esta tendência, proporcionou o aparecimento de um grande número de ferramentas CAD ao longo dos anos, e com eles nasceram também diversos formatos de ficheiros para a representação de produtos e projectos, seguindo a linha pioneira do AutoCAD.

3.1 Exemplos de ferramentas CAD utilizadas pelo mercado

Tal como o AutoCAD, outras ferramentas desenvolvidas para os mais diversos objectivos trouxeram novas formas de representar e manipular projectos de peças, cada uma utilizando simbologias e sintaxes mais apropriadas ao seu nicho de actuação.

Alguns exemplos destas ferramentas são:

- [SolidWorks] é uma ferramenta da Dassault Systemes, possui um conjunto robusto de módulos essenciais para aumentar a produtividade e que inclui o software de projeto mecânico em 3D da SolidWorks, uma linha completa de módulos de comunicação do projecto e que apostam no aumento da produtividade CAD.

- [CATIAV5R16] é uma ferramenta também da Dassault Systemes, lider mundial em soluções 3D e PLM (Product Lifecycle Management). Acelera o processo de desenvolvimento de produtos, e com recurso optimizados já para processadores de 64 bits.

- [CIMATRON] software dirigido principalmente para a industria, engloba as mais diversas áreas da manufactura como projecto de moldes, Eléctrodos EDM, Controladores Numéricos (NC), Die Making, Micro Miling, até a área da engenharia inversa, onde as suas soluções cobrem todo o processo transformando modelos digitais em geometria pronta para o processamento e produção. A Cimatron possui ferramentas para a criação, manipulação, e edição de nuvens de pontos, curvas/superficies NURBS incluindo Stereolitography(STL) data. Importa facilmente o modelo de dados capturado por qualquer digitalizador robótico, mecânico ou óptico.

- [AutoDesk] é o sucessor do AutoCAD, producto consagrado pela AutoDesk, esta nova ferramenta de design 3D tem entre outras funcionalidades Routed System, para o projecto de cablagens, tubagens, canalizações, simulação com análise de esforços e simulação dinâmica. Devido à existência desta grande diversidade de ferramentas a procura por ficheiros em formato neutro que tornassem possível a interoperabilidade entre todos os softwares de desenvolvimento existentes aumentou.

3.2 Formato dos ficheiros neutros/troca

Existe uma grande variedades de especificações para os ficheiros de formato neutro. Estes ficheiros são utilizados para ligar as mais diferentes plataformas de software. De entre todos os ficheiros neutros disponíveis foi decidido concentrar os esforços de pesquisa nos mais utilizados pelo mercado de design de produtos. Estes deveriam possuir características que facilitem a extracção das coordenadas dos objectos para posterior utilização na programação de robôs industriais. Os formatos que estão mais popularizados entre as ferramentas descritas anteriormente são o IGES, SET, STEP(ISO 10303) e STL. Estes formatos possuem características próprias adquiridas devido à sua própria evolução e que acontecerem naturalmente direccionadas pela área de actuação para o qual foram idealizadas ou preferidas.

3.2.1 IGES

Nos anos 70 diversas plataformas CAD já estavam a ser utilizadas significativamente na indústria metalomecânica. Devido a este facto emergiu fortemente a necessidade de troca de informação entre estas diferentes plataformas, forçando a comunidade CAD a criar em 1979 o primeiro padrão internacional - IGES (Initial Garphics Exchange Specification)[Smith, 1988]. Sua especifi-

cação foi publicada pela primeira vez em 1980 nos USA pela NBS(National Bureau of Standards) [NBS, 1980]. Mas a primeira versão foi aceita e disponibilizada apenas em 1982 pela ANSI standards. O padrão neutro IGES é actualmente suportado por todos os grandes desenvolvedores de ferramentas CAD e um dos formatos neutros de troca mais conhecidos no mercado. O formato IGES foi desenvolvido originalmente para a troca de dados entre modelos 2D/3D, texto, dimensionamento de dados e um limitado conjunto de superfícies. Devido a algumas limitações iniciais que este padrão trazia ele tem sido gradualmente expandido e desenvolvido para suportar novas entidades, possuir uma sintaxe clara e consistente. As ultimas versões já apresentam entre outras características: geometrias, representações de rascunhos para desenho técnico, elementos dependentes de aplicação e Modelagem por elementos finitos(FEM). A unidade fundamental nos arquivos IGES é a entidade, que são categorizadas como geométricas e não-geométricas. A entidade geométrica representa a definição virtual de uma forma física e inclui pontos, curvas, superfícies, sólidos e relações entre colecções de entidades estruturadas de forma similar. Entidades não-geométricas são utilizadas para enriquecer o modelo fornecendo perspectivas de visualização de um desenho planar, com suas respectivas escalas e observações pertinentes. Este tipo de entidades servem também para especificar atributos e características, para uma só entidade ou para um grupos de entidades. As entidades não-geométricas típicas para definição, anotação e dimensionamento dos desenhos são: view, drawing, general note, witness line e leader. As entidades não-geométricas típicas para atributos e agrupamentos são as entidades de propriedade e associação. O arquivo IGES é composto por 6 secções diferentes: Flag, Start, Global, Directory Entry, Parameter Data e Terminate. Cada ocorrência de uma entidade consiste em dois registros, um no Directory Entry e um no Parameter Data. O Directory Entry necessita de um índice e de atributos a incluir na descrição da informação gráfica. O Parameter Data define a entidade específica. Todos os Parameter Data são definidos por registros de comprimento fixo, de acordo com as suas entradas correspondentes. Cada instância de uma entidade possui ponteiros bi-direcionais entre o Directory entry e a secção Parameter Data. Um dos grandes problemas práticos em trabalhar com ficheiros IGES é o seu tamanho e consequente tempo de processamento. A necessidade de ponteiros bi-direcionais nas secções Directory Entry e Parameter Data, em conjunto com a restrição de possuir formatos de registros fixos, têm causado problemas durante a sua utilização por pré e pós-processadores. O padrão IGES está sob o controle da NCGA(National Computer Graphics Association) e faz parte do U.S. Product Data Association(USPRO) e do IGES/PDES Organisation(IGO).

3.2.2 SET

É um padrão francês para o armazenamento, troca de dados CAE e é suportado por grande parte das plataformas CAD do mercado. Seu acrônimo deriva de "Standard d'Echange et de Transfert". Foi desenvolvido pela industria aeroespacial em 1983 para a troca de informação entre diferentes plataformas CAD. O objectivo era desenvolver uma alternativa mais autentica e fiável comparada ao IGES. Em 1985 o padrão SET tornou-se o padrão oficial francês, Afnor Z68-300. Este padrão suporta wireframe, surface e solid models, incluindo CSG e B-Rep. Foram incluídas também entidades para drafting e conectividade entre aplicações, como scientific data e FEM(Finite Element Method). Este formato neutro tem a grande vantagem de não possuir um formato ambíguo, ter um tamanho bastante compacto e ser flexível o suficiente para ser modificado segundo as necessidades da indústria CAD/CAM [Vuoskoski, 1996].

A GOSET é a organização que foi fundada por alguns dos principais utilizadores do SET e é responsável pela promoção, manutenção do padrão e eventuais desenvolvimentos. GOSET é o órgão que possui o serviço de validação para interfaces SET em desenvolvimento. A arquitectura SET é baseada em 3 níveis de hierarquia, data Assemblies, data Blocks e data Sub--blocks. As informações que são comuns a muitos blocks ou assemblies são armazenados numa sessão denominada Dictionary. Assembly é uma colecção de registros que definem informação acerca da peça, como por exemplo a informação sobre as partes mecânicas. Um ficheiro SET pode conter mais de um data assembly. Já o Block consiste num identificador seguido de um número do tipo bloco.

3.2.3 STEP

A sigla STEP deriva de "the STandard for the Exchange of Product model data"e baseia-se na norma ISO 10303 para representação e troca de informação geométrica de produtos[Owen, 1994] , ou seja, é uma série de documentos que facilitam o interface e partilha de informação, utilizada no desenvolvimento de produtos.

O desenvolvimento do padrão STEP iniciou-se em 1984 num projecto de colaboração mundial. O objectivo principal era definir um padrão que abrangesse todos os aspectos de um producto (geometria, topologia, tolerância, material, etc.), durante todo seu tempo de vida. Algo que nunca tinha sido feito anteriormente. O STEP apareceu como uma colecção de padrões

internacionais reunidos para representar e trocar informações entre produtos. O desenvolvimento do padrão STEP vem sendo controlado pela International Standards Organisation (ISO), Technical Committee 184(TC184, Industrial Automation Systems) e Subcommitte 4 (SC4, Industrial Data and Global Manufacturing Programming Languages).

Este formato é utilizado para partilhar informação CAD, modelos, estruturas complexas e desenhos técnicos de produtos durante o seu tempo de vida, através de um mecanismo independente da plataforma de desenvolvimento em que se trabalha. Isto separa a representação da informação do produto dos seus métodos de implementação. A utilização dos métodos voltados para a troca de informação entre diferentes plataformas, com uma única representação, permite a representação da informação unificada de um mesmo produto para diferentes aplicações.

Este formato é muito usual na industria aeroespacial, automóvel, naval e da construção civil. A informação geométrica dos produtos foi catalogada pela ISO como padrão nº10303, na forma de arquivos de troca, interfaces de programação para aplicações industriais e implementações para bases de dados, recebendo a denominação STEP.

O padrão STEP está dividido em diferentes partes[referenciar]:

- Princípios Fundamentais: Define os princípios fundamentais do padrão.

- Descrição de métodos: A linguagem de modelagem de dados EXPRESS [Schenck, 1994] que representa a informação sobre os produtos.

- Implementação dos Métodos: Contem a definição da representação física da informação do produto.

- Teste de Conformidade: Framework e metodologia de teste de conformidade.

- Recursos Integrados: Contem a representação da informação do produto comum a diferentes protocolos de aplicações.

- Protocolos para Aplicação: Contem a representação da informação do produto específica para uma área particular de aplicação.

- Testes Abstractos: Conjunto de testes abstractos para um protocolo de aplicação suportar os requisitos de conformidade.

O código STEP, devido à grande aplicação em projectos de conformação, nomeadamente tornos, fresas e na produção de moldes sofreu uma especialização para a norma ISO 14649 denominada STEP-NC, direccionada para

controladores numéricos. Os programas STEP-NC são descritos no mesmo formato físico dos arquivos ISO 10303, Part 21, com a vantagem de descrever "o que fazer"para produzir aquelas estruturas geométricas especificadas.

O formato STEP-NC possui o seguinte esqueleto: A primeira sessão, o cabeçalho do programa é marcado pela palavra-chave "HEADER". No cabeçalho são apresentadas algumas informações e comentários gerais a respeito do programa, como: nome do arquivo, autor, data e organização. A segunda sessão, a mais importante do arquivo, vem marcada pela palavra-chave "DATA". Como o próprio nome sugere, esta sessão possui a informação sobre a geometria e pode conter também a informação necessária para a construção da peça.

Um dos grandes problemas da utilização de ficheiros neutros na passagem da informação CAD até então, era o fluxo da informação unidireccional do design para a manufactura. As mudanças que apareciam no processo de manufactura não podiam voltar para o design do produto devido a incapacidade de trabalhar directamente sobre as maquinas CNC sem passar obrigatoriamente por um pós-processador. A norma ISO 14649 apareceu como uma extensão da norma ISO 10303, incrementando funcionalidades que possibilitaram esta ligação bidirecional entre a manufactura e o design de produtos.

Actualmente a pesquisa na área do código STEP-NC tem sido coordenado por um projecto chamado Sistema de manufactura inteligente(IMS) composto por um esforço de acções internacionais em prol da evolução das interfaces CAD para controlo de ferramentas de conformação e maquinaria.

3.2.4 STL

É um formato nativo criado por uma empresa de Valencia, CA, USA chamada 3D Systems para seu Stereolithography CAD software. Este padrão neutro é largamente utilizado sendo o preferido pelos sistemas de prototipagem rápida. Este formato de troca pode ter duas codificações de dados, ASCII ou binária, sendo a codificação binária a mais utilizada devido ao seu reduzido tamanho quando comparado com a codificação ASCII.

ASCII format

A primeira linha é a descrição do arquivo e começa com o identificador "solid"; normalmente contém o nome do objecto, o autor, a data de criação, e outras observações para a identificação do produto. A ultima linha deve conter a

palavra-chave "endsolid". As outras linhas entre o cabeçalho e o fecho do arquivo contém a descrição dos 3 vértices que formam a face de um triângulo, mais o seu vector normal. A ordem dos vértices segue a regra da mão direita, ou seja, sentido anti-horário.

A sintaxe do arquivo ASCII é a seguinte:

solid name

facet normal x y z

outer loop

vertex x y z

vertex x y z

vertex x y z

endloop

endfacet

facet normal x y z

outer loop

vertex x y z

vertex x y z

vertex x y z

endloop

endfacet

...

endsolid

O vector normal e as coordenadas dos vértices são escritas em notação científica (+- d.dddddddE+-ee). Muitas vezes os vectores normais não precisam ser disponibilizados pois serão gerados por um programa parsing. A principal restrição aplicada sobre a construção dos facets nos arquivos STL é que todos os facets adjacentes devem ter 2 vértices comuns, ver Fig 3.1.

Como exemplo considere o seguinte excerto de código STL ASCII:

```
solid X
facet normal 0.000000e+000 9.971213e-001 -7.582376e-002
outer loop
vertex 7.293332e+002 2.200000e+002 1.183396e+003
vertex 7.295713e+002 2.200000e+002 1.183396e+003
vertex 7.295713e+002 2.190000e+002 1.170246e+003
endloop
endfacet
facet normal 5.202612e-003 9.971148e-001 -7.572907e-002
outer loop
vertex 7.295713e+002 2.190000e+002 1.170246e+003
vertex 7.293332e+002 2.190000e+002 1.170229e+003
vertex 7.293332e+002 2.200000e+002 1.183396e+003
endloop
endfacet
facet normal 0.000000e+000 9.971076e-001 -7.600333e-002
outer loop
vertex 7.295713e+002 2.200000e+002 1.183396e+003
vertex 7.298094e+002 2.200000e+002 1.183396e+003
vertex 7.298094e+002 2.190000e+002 1.170277e+003
endloop
endfacet
...
endsolid
```

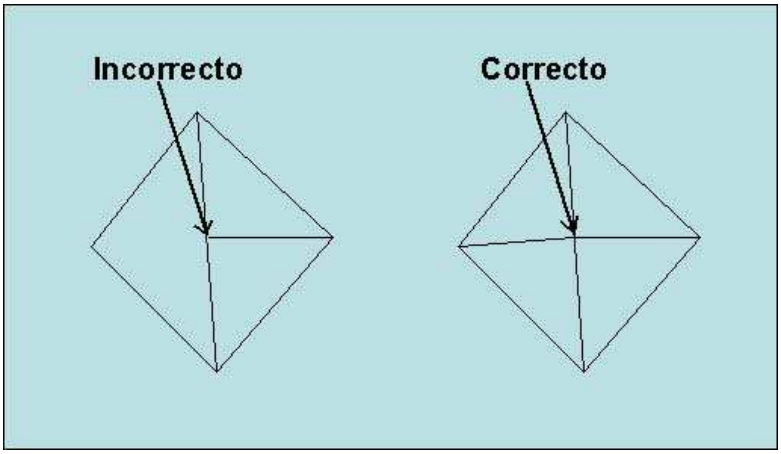


Figura 3.1: Restrição imposta pelo formato STL na construção de triangulos.

BIN format

No início do arquivo binário, os primeiros 80 bytes são reservados para o nome do arquivo e algumas pequenas observações. Em seguida os 4 próximos bytes (long int) são usados para armazenar o número de facets do arquivo STL. A partir deste ponto os facets possuem um comprimento de 18bytes cada. Dentro de cada facet, as coordenadas 3D são armazenados em vírgula flutuante, cada uma também com o comprimento de 32bits(4bytes). Então são: 4bytes para o vector normal, mais 3x 4bytes(para cada vértice do triângulo), mais 2 bytes para o "fill-bytes"que são os bytes de fecho do facet, totalizando os 18 bytes referidos anteriormente.

A sintaxe do arquivo BIN é a seguinte:

<STL file> := <name><facet number><facet 1><facet 2>...<facet n>

<name> := 80 bytes file name, filled with blank

<facet number> := 4 bytes long int integer

<facet> := <normal> <vertex 1> <vertex 2> <vertex 3> <fill-bytes>

<normal> := Nx Ny Nz

<vertex> := X Y Z

<fill-bytes> := 2 fill bytes

Capítulo 4

Gerador de código para diferentes plataformas de robôs

A abstracção é uma característica importantíssima na programação para máquinas industriais. A partir deste pressuposto é possível generalizar código de maneira a alcançar a maior quantidade possível de equipamentos industriais. Muitos destes equipamentos são baseados em sistemas compostos de eixos e elos, como por exemplo as CNCs, máquinas de corte a lazer, e principalmente a mais versátil de todas, os robôs articulados de vários eixos.

A depender da modelagem que for efectuada para as estruturas de dados básicas que utilizam mecanismos como: elos, juntas, coordenadas ou programas, por exemplo. Podem-se desenvolver rotinas que comandem máquinas com n eixos a partir de comandos básicos de movimentação proprietários implementados via software.

Neste capítulo vamos abordar técnicas de programação que viabilizam a representação deste conhecimento focando a criação de comandos reconhecidos pelos robôs através de uma linguagem proprietária disponibilizada pelo respectivo fabricante.

4.1 Padrões de Projeto

A experiência é uma característica muito valorizada em qualquer ramo de negócio. Como não poderia deixar de ser, na programação orientado a objecto (OO), utilizam-se também experiências de programadores seniores das suas descrições conhecidas como Padrões de Projeto ou "Design Pattern".

4.2 O que são "Design Pattern"?

Um dos maiores benefícios trazidos pelo paradigma de programação orientado a objectos é a reutilização de código e consequentemente das funcionalidades do software. Porém, projectar um software orientado a objectos que seja reutilizável não é uma tarefa fácil.

Durante a fase de análise, é preciso identificar modelos (ou moldes) de classes e comunicação entre objectos que podem ser reutilizados no sistema. Esses modelos resolvem problemas de projectos específicos e tornam os projectos OO mais flexíveis, elegantes e reutilizáveis. Após a familiarização com o uso de moldes, pode-se aplicá-los no desenvolvimento de projectos OO sem necessariamente ter que redescobrí-los. O projecto da aplicação deve ser capaz de dividir o problema em módulos gerais o suficiente, para serem reaproveitados futuramente em situações diversas. Os "Padrões de Projecto", são técnicas desenvolvidas para potencializar a reutilização através de modelos, organização de classes em hierarquias e da distribuição de responsabilidades entre os diferentes objectos. É um mecanismo de alto nível que reúne as melhores práticas (experiências de sucesso) de programadores, para resolver problemas comuns a diversos projectos de software.

4.3 Padrões utilizados pelo projeto

Para abstrair o modelo genérico do ambiente de programação de robôs proposto nesta pesquisa foi necessário estudar algumas das experiências de programação OO que deram origem aos "Design Patterns". Por este motivo foi necessário referenciar algumas das técnicas de programação OO utilizadas no desenvolvimento do gerador de código. Os nomes dos padrões, como utilizá-los, problemas comuns resolvidos por estas práticas, e as consequências da implantação destas micro arquitecturas [Gama, 1995] serão explicadas a seguir.

4.3.1 Factory

Este padrão centraliza a criação dos objectos que têm uma grande possibilidade de sofrer alterações no projeto em uma única classe. O termo Factory significa "Fábrica de objectos". A utilização do padrão "fábrica"deve-se à necessidade de existência de uma classe que centralize a instância de objectos, de maneira que nenhuma classe na camada de negócio instancie diretamente

objectos.

Motivação: No caso de mudanças na assinatura de um método construtor, seria necessário alterar todas as instâncias directas feitas a esta classe, tornando-se um trabalho extremamente cansativo considerando a existência de um número muito grande de classe que instanciam este objecto.

A solução proposta foi a criação de uma classe, responsável por criar e retornar a instância de todos os objectos do framework de forma centralizada. As utilização da fábrica de objectos para ser efectiva, restringe a obtenção de instâncias de objectos sempre através dos métodos da fábrica, proibindo a instância directa dos mesmos. Desta forma, se o método construtor de um desses objectos for alterado, as mudanças serão centralizadas no respectivo método getInstanciaX() da classe Factory, onde "X"é o nome da classe que contém os atributos(comportamento) do objecto em questão, ver Fig 4.1.

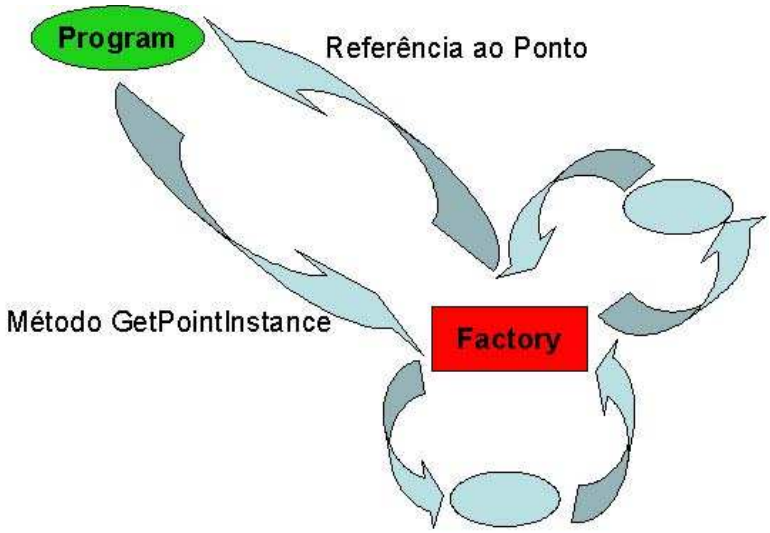


Figura 4.1: Requisição de um ponto à fabrica.

A estratégia foi listar as classes que têm uma grande propensão à mudanças e implementar os métodos getInstancia() para estas classes em uma única classe Factory.

4.3.2 Singleton

Padrão que impede a duplicação de objectos instanciados e residentes em memória. O seu significado é "instância única". Durante a execução de

um programa, inúmeros objectos são recriados sem necessidade, causando desperdício de recursos da máquina nomeadamente na alocação indevida de memória e tempo de processamento.

Motivação: O desperdício de memória causado pela subutilização de objectos já criados. E aumento no overhead da máquina devido ao coletor de lixo(garbage collector) precisar recolher constantemente os inúmeros objectos que não estão referenciados por terem sido sub-utilizados.

A solução encontrada foi verificar no momento da requisição de uma instância, se aquele objecto já foi instanciado pela fábrica, ver Fig 4.2.

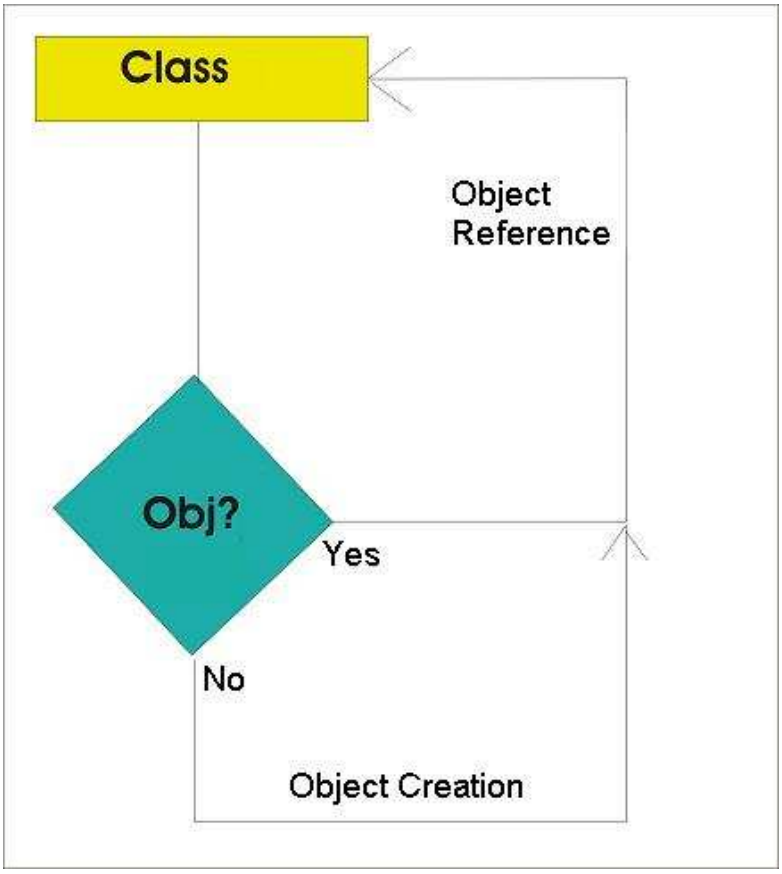


Figura 4.2: Teste feito antes da criação de um novo objecto.

As consequências directas da utilização do singleton são a optimização dos recursos de memória alocados, consequentemente menos overhead devido à baixa utilização do garbage collector, e aumento na vida útil dos objectos.

Isto reflectirá directamente no desempenho da aplicação.

A seguinte estratégia foi utilizada: Se o objecto já existe é devolvido uma referência para este, caso contrário cria-se um novo objecto e em seguida é retornado um apontador para o novo objecto.

4.3.3 Facade

O padrão Facade separa a interface do utilizador das classe de negócio do sistema. Através de um ponto único de entrada padrão liga-se a(s) interface(s) à camada de negócio.

O significado de Facade pode ser entendido como "consenso". Ele foi implantado devido à necessidade da existência de um ponto de entrada de dados padrão para as camadas de negócio da aplicação, de maneira a separar a interface do motor da aplicação.

Motivação: No caso de mudança na interface de entrada do sistema seria necessário alterar também as classes de negócio para compatibilizações com a nova interface.

Esta solução sugere criar uma classe que esconda a complexidade do sistema das camadas de interface e que seja suficientemente abstracta para interagir com qualquer input. Seja ele um prompt de comando, uma query de consulta a banco de dados, ou uma GUI(Graphical User Interface).

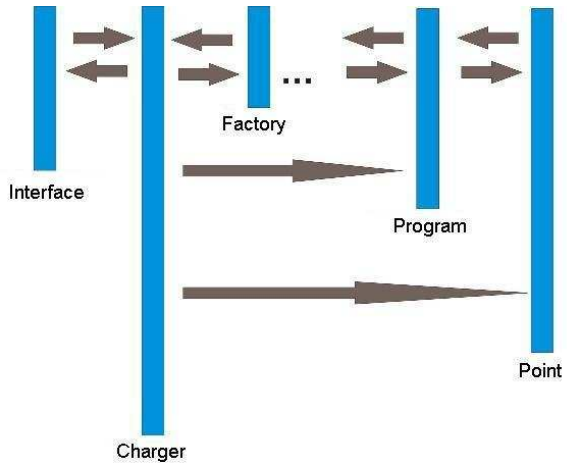


Figura 4.3: Restrição imposta pelo formato STL na construção de triangulos.

As consequências disso são a separação entre camadas de apresentação e

negócio, uma vez que todas as dependências entre as entidades envolvidas na realização no caso de uso ficam transparentes ao cliente.

A principal vantagem é o ganho na manutenção do software, devido a facilitar as alterações no input actual ou até a criação de novos inputs para a aplicação.

A estratégia utilizada foi implementar uma classe Carregador, que é conhecida pela interface, e que possui a informação necessária para ascender às classes do negócio, ver Fig 4.3. O uso desta classe Carregador permitirá a troca, ou manutenção da camada de apresentação, sem a necessidade de grandes mudanças no conjunto de classes do projecto.

4.3.4 MVC

A tríade MVC, acrônimo para Model-View-Controller é utilizada basicamente para construir interfaces orientadas a objecto. Este padão consiste em três tipos de objectos basicamente:

- O Modelo, que é o objecto aplicação.

- A View que é representação gráfica.

- O Controller que define a forma como o interface reage à entrada de dados.

Antes da criação do pattern MVC agrupavam-se todas as funcionalidades destes três objectos em um único. MVC faz a separação entre views e modelos estabelecendo um protocolo subscrição/notificação, ver Fig 4.4.

Uma view deve assegurar que a sua aparência reflete o estado do modelo e quando o estado do modelo se altera, todas as views que estão dependentes são notificadas. Esta abordagem permite que se associe um número variável de views a um modelo para se conseguir diferentes representações gráficas. É possível também associar novas views a um modelo sem grandes alterações.

O principal objectivo deste padrão como foi referido anteriormente, é, separar a lógica do negócio(Model) da interface com o usuário (View) e do fluxo da aplicação (Controller).

A grande vantagem da utilização deste padrão é permitir que uma mesma lógica de negócios possa ser acedida e visualizada através de várias interfaces, como podemos notar na Fig 4.4.

Na arquitectura MVC, o Model não sabe quantas nem quais Views estarão

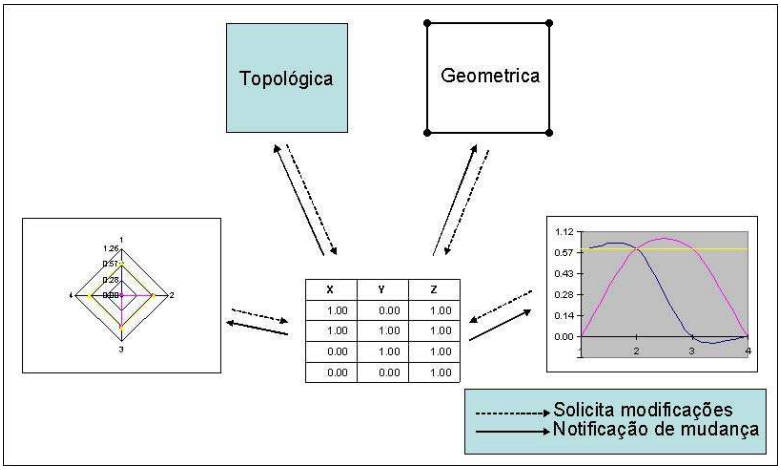


Figura 4.4: Iteração entre modelo e visões através de notificações.

exibindo o seu estado. A grande motivação para a utilização de MVC é que aplicações interactivas requerem interfaces dinâmicas. Reduzir o acoplamento da interface às classes de negócio da aplicação, reflectirá no futuro um menor esforço de desenvolvimento e uma fácil manutenção das novas versões de interfaces que agreguem novas funcionalidades/paradigmas.

4.3.5 Command

Este padrão gera a separação de solicitações geradas pelo utilizador, através da(s) interface(s) e acções desencadeadas por estas solicitações. Pode-se entender a denominação como "comando"ou "acção". O padrão Command é muito simples e prático pois o utilizador deve executar as funcionalidades do sistema, através de objectos que compõem a interface de entrada de dados (Menus, Buttons, JRadioButton,etc).

Motivação: As ferramentas para design de interfaces com o utilizador incluem botões, menus e outros objectos que realizam uma acção em resposta a uma entrada específica vinda do utilizador. Objectos de input não podem por exemplo implementar estes pedidos explicitamente num botão, porque somente as classes que possuem os métodos envolvidos naquela acção saberiam o que fazer com aquele input específico.

A solução encontrada foi separar a implementação da interface. Todos os objectos utilizados na(s) interface(s) conhece(m) apenas uma referência ao método da classe que vai assumir a responsabilidade pela execução do

comando seleccionado pelo utilizador. Como consequência temos mais esta alternativa de separar objectos da interface da camada de negócio, ver Fig 4.5.

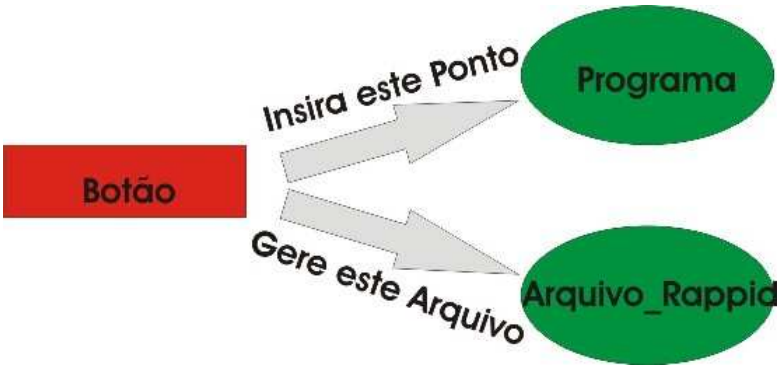


Figura 4.5: Requisição de um ponto à fabrica.

A estratégias utilizada na interface foi restringir o acesso dos objectos que compõem a interface ao ponto padrão de entrada do sistema, secção 4.3.3. Através deste ponto(única classe que sabe como aceder às classes do negócio) os objectos da interface referenciam as funcionalidades que o utilizador deseja invocar.

4.3.6 Template Method

É um padrão que define o esqueleto de um algoritmo para ser utilizado posteriormente pelas classes que herdarão aquele modelo abstracto.

Permite que as subclasses redefinam comportamentos de sua classe pai, de acordo com a sua realidade. Sua tradução para o português pode ser entendida como "método do molde".

Motivação: Ela é utilizada para criar blocos comuns de código a diferentes classes uma única vez, fazendo todas estas classes herdarem este comportamento comum da classe template abstracta. As classes filhas vão concretizar o seu comportamento sobrecarregando os métodos da classe pai através de polimorfismo.

As consequências da utilização desta técnica são o aumento da reutilização de código, mas em contrapartida aumenta o acoplamento gerado pela herança,

implementada entre a classe abstracta e as classes filhas. Esta prática é a base para a construção de frameworks, ver Fig 4.6.

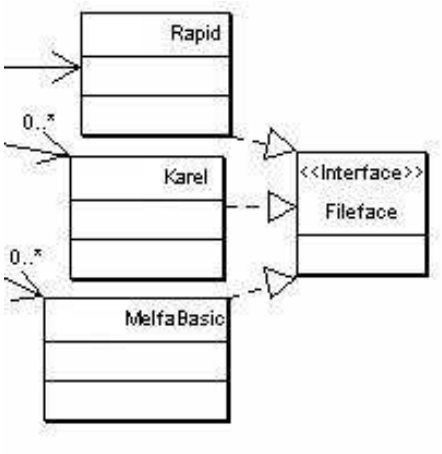


Figura 4.6: Utilização de moldes.

4.4 Framework

O Framework é uma camada de trabalho que esconde detalhes específicos de implementação acerca de um limitado universo de actuação, reduzindo a complexidade de iteração do utilizador sobre este universo.

O conceito de framework tornou-se indispensável para o desenvolvimento deste projecto devido à necessidade de trabalhar com inúmeras linguagens de programação cada uma para um robô específico.

Algumas das facilidades marcantes trazidas pelo paradigma de programação orientação a objecto como: ligação dinâmica, polimorfismo e herança, tornam possível a criação deste framework. Estas técnicas foram utilizadas muitas vezes como instrumentos de reutilização, abstracção e especialização dos objectos.

A grande dificuldade na criação de um framework é a disposição das classes de maneira que elas comuniquem dinamicamente. Caso se decida migrar para este paradigma, partindo de trabalhos onde não foi gasto o tempo devido na análise de requisitos e projecto, torna-se necessário redesenhar totalmente a aplicação para que ela suporte esta intensa interacção. Reforçando a prática que padrões de projecto fazem parte da documentação indispensável de um

framework, e através deles é que se remodela um sistema.

A metodologia de construção de framework proporcionou enormes ganhos na reutilização de código através dos padrões de projecto: Template method, Factory e Sigleton , além do encapsulamento da informação trabalhada com os padrões Command e Facade, citados neste capitulo.

4.5 Extração da informação relevante dos arquivos STL

Após um estudo feito sobre os diferentes tipos de arquivos neutros foi escolhido o formato "Standard Transform Language"(STL) pelos seguintes motivos:

- A informação geométrica é facilmente acessível;

- Através desta geometria pode-se extrair as dimensões da peça para a programação off-line de robôs;

- A perda de informação relaciona as camadas, cores e outros atributos no processo de exportação da peça para o formato STL não traz perdas significativas para esta aplicação específica;

- E principalmente porque a maioria dos softwares CAD comerciais exportam informação no formato STL;

4.5.1 Algoritmos de leitura

Como foi comentado no final do capítulo anterior a leitura da informação contida nos arquivos STL é muito simples. Isto deve-se principalmente à forma explicita como são declarados os dados dentro do arquivo ASCII.

No formato binário a informação encontra-se encriptada, mas se forem tomados os devidos cuidados, com a leitura correcta dos bytes referentes a cada entidade, sub-secção 3.2.4, não existirão maiores dificuldades em ascender à informação encriptada.

Alguns dos algoritmos utilizados na extracção da informação contidas nos arquivos STL são apresentados a seguir.

Leitura do arquivo STL Binário

Open File

To reserve 80 bytes array to header reading

To reserve 4 bytes array to facets number reading

To read header

To read facets number

While not EOF

Read normal vector float xyz coordinates

While not end of Triangle

Read triangle vertexes float xyz coordinates

EndW

EndW

Close File

Leitura do arquivo STL ASCII

Open File

To read line

While read line is not "endsolid"

If read line is equal "vertex"

While not End of Triangle

Read triangle vertexes

EndW

EndIf

To read line

EndW

Close File

4.6 RoBott

A interface de programação off-line de robôs Fig 4.7 descrita neste trabalho foi projectada utilizando conceitos de programação orientada a objecto aliados

ao padrãoes de projecto descritos neste capítulo. A metodologia utilizada foi o processo de desenvolvimento iteractivo incremental [Jacobson, 1998]. Este processo resume-se à criação de uma plataforma base onde são acrescidas novas funcionalidades. Estas novas funcinalidades vão sendo definidas pelas demandas encontradas durante o processo de pesquisa.

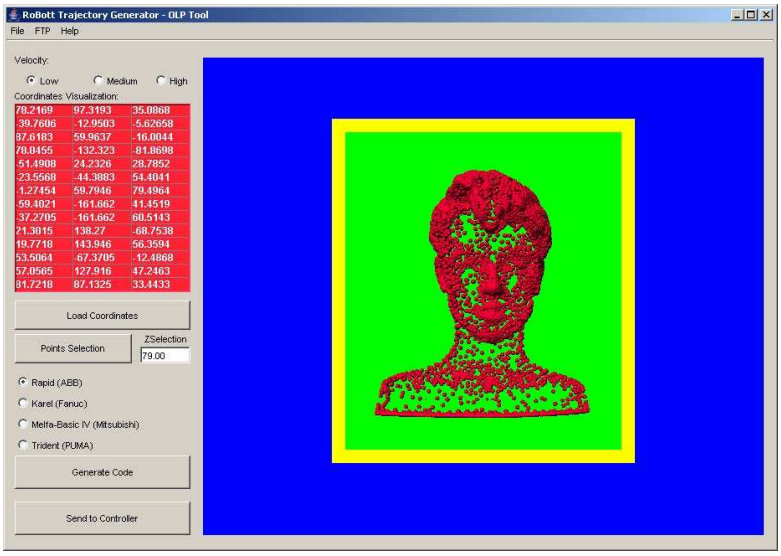


Figura 4.7: RoBott OLP Tool.

O gerador de código foi idealizado com o objectivo de tornar a programação de robôs uma tarefa simples. Por este motivo foi escolhida uma interface que pudesse ser compreendida rapidamente por qualquer profissional que saiba os conceitos minimos ralacionados a programação do movimento de um robô.

No topo da tela de comandos, ver Fig 4.8 temos três velocidades de programação "Low"(10 por cento), "Medium"(50 por cento) e "High"(100 por cento).

Logo a baixo dos selectores de velocidade podemos ver a tabela de coordenadas da peça no mundo real, posicionada no centro da área de trabalho do robô. Nesta tabela são apresentadas as coordenadas da peça retiradas do arquivo neutro, transformadas com os respectivos offsets e rotações para posicionamento no centro da área de trabalho do robô.

A seguir encontramos o botão "Load Coordinates"que dispara a selecção e leitura dos arquivos stl ASCII e BIN. Logo a baixo deste temos o botão "Select Points"para seleção de pontos no plano Z. A selecção é feita sobre

o conjunto de pontos apresentados na tela e tomando como base a caixa de texto ZSelection. Ao ser carregado um novo arquivo esta caixa assume automaticamente o valor do plano superior da peça, podendo ser mudado este valor a qualquer momento para selecção de um outro plano.

Em seguida temos o conjunto de linguagens proprietárias e seus respectivos fabricantes para os quais pode ser gerado código, para que se escolha a plataforma alvo da programação Off-line.

O botão "Generate Code"é utilizado para a escolha do sitio onde será salvo o programa, e dispara a geração do código de acordo com o conjunto de coordenadas escolhido para a programação do movimento.

Finalmente o botão "Send to Controler"é utilizado para a tranferência de programas via FTP do PC para o controlador do robô.

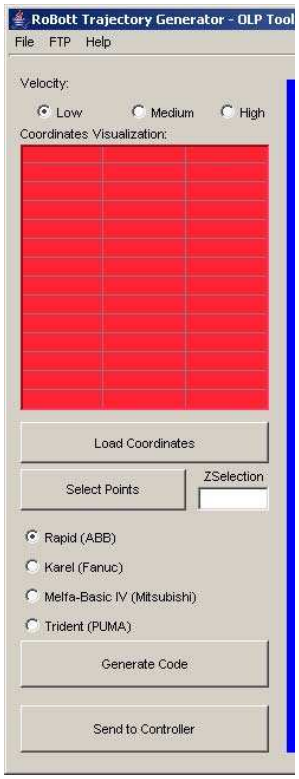


Figura 4.8: Comandos disponíveis na ferramente OLP.

A linguagem escolhida para realizar o projecto do motor de programação off-line de robôs foi o Java Enterprise Edition(JEE) versão 1.4.2.03 e o am-

biente de desenvolvimento JDeveloper versão 9.0.5.1(Build 1605) da Oracle. As grandes motivações na escolha da linguagem Java foram a portabilidade que esta linguagem proporciona, podendo ser utilizada facilmente em ambiente "web"ou executar independente em qualquer sistema operativo onde esteja funcionando a maquina virtual Java, e suporte total ao paradigma de programação Orientado a Objecto.

4.6.1 Programando através da interface Robott

Para programar um robô através do ambiente Robott são necessários 5 passos basicamente:

1 - Clique em "Load Coordinates"e escolha o caminho para o arquivo STL do qual deseja extrair as referências geométricas da peça.

2 - Digite a referência ao plano Z para a selação na caixa "ZSelection"e clique em "Select points".

3 - Selecione a linguagem para a qual desejas gerar o programa.

4 - Selecione a velocidade de movimentação do robô.

Obs: É extremamente recomendável que todo programa seja inicialmente testado a "baixa velocidade"por questões de segurança. Vale lembrar que a ferramenta propósta nesta trabalho visa facilitar a programação de robôs mas mão exclui a necessidade de treinamento e conhecimento das normas de segurança para a operação destes tipos de equipamentos. Os robôs industriais que possuem uma capacidade de aceleração surpreendente podendo causar sérios danos à saúde caso não sejam tomados os devidos cuidados respeitando a área de trabalho total do robô, enquanto o equipamento está em operação.

5 - Clique no botão "Generate Code"e selecione o nome do programa que desejas criar.

Caso o controlador com o qual estejas trabalhando possua esta facilidade:

6 - Clique em "Send to Controler"para enviar o programa para o controlador via FTP.

O transporte do programa para o robô pode ser feito na maioria dos casos via disquete. Os controladores mais modernos que já possuem sistema operacional windows permitem também o acesso via TCP/IP utilizando um navegador como por exemplo o Windows Explorer.

Capítulo 5

Resultados obtidos

Os teste foram feitos utilizando duas plataformas diferentes de robôs industriais, ABB e Mitsubishi respectivamente. A sequência de programação off-line completa desde a captura dos pontos de referência através dos arquivos .stl, transformação dos mesmo para coordenadas do mundo real, até a geração do código e execução no controlador foi feita usando o robô ABB IRB140.

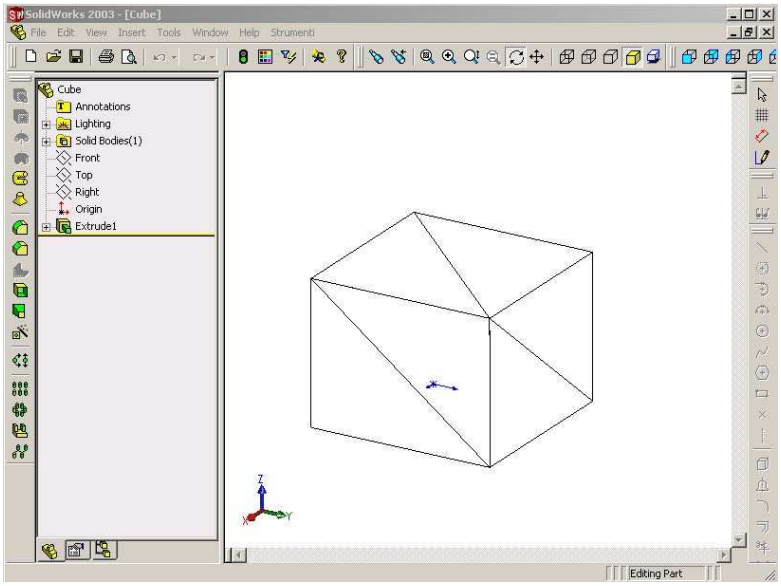


Figura 5.1: Peça no formato CAD após ser transformada para o formato stl.

Primeiro foi necessário converter o modelo CAD da peça para um for-

mato neutro que possa ser lido pela ferramenta OLP Robott. Neste caso utilizou-se o [SolidWorks] para converter o modelo CAD no formato STL Fig 5.1. Podemos observar a seguir o trecho de código referente ao arquivo stl:

```
solid Cube
facet normal -1.000000e+000 0.000000e+000 0.000000e+000
outer loop
vertex 0.000000e+000 0.000000e+000 5.000000e+001
vertex 0.000000e+000 5.000000e+001 5.000000e+001
vertex 0.000000e+000 0.000000e+000 0.000000e+000
endloop
endfacet
facet normal -1.000000e+000 0.000000e+000 0.000000e+000
outer loop
vertex 0.000000e+000 0.000000e+000 0.000000e+000
vertex 0.000000e+000 5.000000e+001 5.000000e+001
vertex 0.000000e+000 5.000000e+001 0.000000e+000
endloop
endfacet
facet normal 0.000000e+000 -1.000000e+000 0.000000e+000
outer loop
vertex 5.000000e+001 0.000000e+000 5.000000e+001
vertex 0.000000e+000 0.000000e+000 5.000000e+001
vertex 5.000000e+001 0.000000e+000 0.000000e+000
endloop
endfacet
...
endsolid
```

O segundo passo foi inserir as coordenadas de referência da área de trabalho do robô, representado pelo rectângulo amarelo na Fig 5.2.

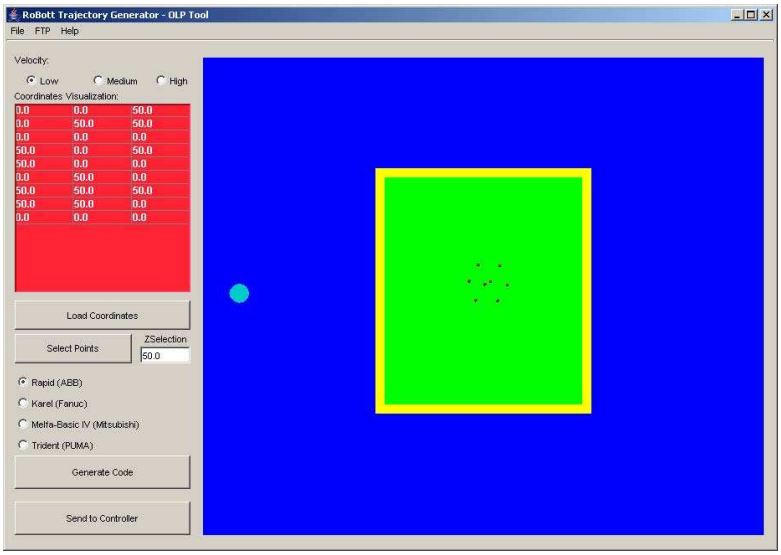


Figura 5.2: Informação geométrica retirada do ficheiro.

As referências dos limites da área de trabalho real do robô serão necessárias para o posicionamento da peça a ser trabalhada. O ponto azul representa o centro do eixo 1, a base do robô. Em seguida será necessário ler a informação geométrica da peça através do arquivo neutro escolhido, no nosso caso o formato STL Binário ou ASCII.

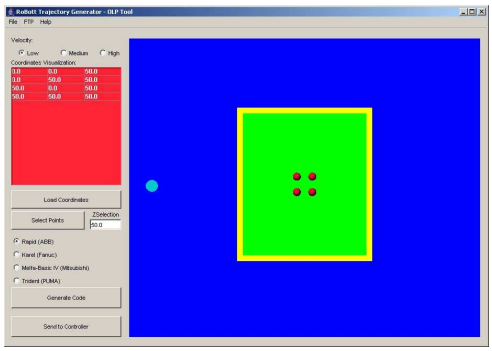


Figura 5.3: Seleção de pontos do plano superior de um cubo.

A partir desta informação a peça será posicionada no centro da área de trabalho, ver Fig 5.2 podendo ainda ser deslocada ou rodada caso seja conveniente. O algoritmo de selecção de pontos utiliza uma referência ao plano z inserida pelo utilizador para seleccionar a nuvem de pontos utilizada na

geração do programa que coordenará o movimento do robô, ver Fig 5.3.

Após inserir as referências da área de trabalho, extracção da geometria do cubo e selecção dos nós de interesse é possível gerar o programa descritor do movimento na linguagem rapid [ABB, 2000] como podemos ver no exemplo a seguir:

```
VERSION:1
LANGUAGE:ENGLISH
MODULE cubo
PERS robtarget Point0:=[[485.5,-25.0,50.0],[0.003789,0.027131,-0.999609,-0.005718],...];
PERS robtarget Point1:=[[535.5,-25.0,50.0],[0.003789,0.027131,-0.999609,-0.005718],...];
PERS robtarget Point2:=[[535.5,25.0,50.0],[0.003789,0.027131,-0.999609,-0.005718],...];
PERS robtarget Point3:=[[485.5,25.0,50.0],[0.003789,0.027131,-0.999609,-0.005718],...];
PROC main()
MoveL Point0,v10,fine,Tool0;
MoveL Point1,v10,fine,Tool0;
MoveL Point2,v10,fine,Tool0;
MoveL Point3,v10,fine,Tool0;
MoveL Point0,v10,fine,Tool0;
ENDPROC
ENDMODULE
```

As instruções geradas para a programação de robôs são basicamente strings montadas sequencialmente de acordo com um modelo estruturado de linguagem utilizada pelo fabricante, como se pode observar no extracto acima do código Rapid. A transferência do programa gerado, do PC para o controlador, foi feita através de FTP devido a esta facilidade estar incorporada ao novo controlador ABB IRC5.

Todas as linguagens de programação para robôs abordadas nesta pesquisa possuem uma característica comum que foi explorada no desenvolvimento desta aplicação. Utilizam um conjunto limitado de elementos para a representação do deslocamento do robô, através de pontos. Estes elementos são: Tipo do deslocamento, nome do ponto associado a suas coordenadas, velocidade do deslocamento, precisão e ferramenta a ser manipulada. Através de uma

simbologia abstracta e conhecimentos acerca das linguagens utilizadas pelos fabricantes de robôs é possível programar diferentes plataformas a partir de referências a pontos posicionados dentro da área de trabalho do robô. Alguns testes foram feitos também com o robô Industrial Mitsubishi Move Master, mas restritos apenas à geração do código Melfa Basic IV [Mitsubishi, 2000]. A inserção das coordenadas de posicionamento que compõem o movimento foi feita através de teaching.

Capítulo 6

Conclusão

A programação de robôs industriais continua a ser um trabalho difícil [Wörn, 1998]. Algumas das principais causas são: a dificuldade de equipamento disponível para aprender a programar, o manuseio complexo do equipamento e os conhecimentos tecnológicos necessários para entender a iteração entre os diferentes componentes do robô. Esta tese de mestrado demonstra que a integração de uma célula de manufactura pode ser acelerada usando simulação, a geração de código para robôs industriais pode ser generalizada para atingir diferentes plataformas proprietárias e o custo com a mão de obra especializada pode ser reduzido. Os métodos de programação off-line foram criados para diminuir o tempo de integração da célula de manufactura. Mas os métodos de programação off-line de robôs não tem trazido ganhos significativos na integração da célula de manufactura, tão pouco para a redução das horas de trabalho dos programadores de robôs. As ferramentas de programação off-line de robôs utilizadas pelos principais fabricantes de robôs foram construídas sem a abstracção necessária para generalizar a programação de robôs industriais. As actuais ferramentas disponíveis no mercado, limitam-se a interagir com as suas respectivas linguagens, arquivos e bibliotecas proprietárias. Esta restrição do software em interagir com diferentes linguagens é naturalmente uma questão comercial relativa à protecção da área de actuação para a qual foi desenvolvido o robô. Esta restrição por outro lado limita toda a potência de cálculo de um microcomputador à programação de um único modelo de equipamento, quando poderia estar programando diferentes robôs de diferentes fabricantes em diferentes tarefas. Neste contexto cresce a procura por uma ferramenta única que interaja com diferentes soluções e fabricantes facilitando a programação dos robôs. O foco da aplicação criada nesta pesquisa foi criar um software portável, capaz de gerar programas para diferentes fabricantes,

utilizando a informação geométrica extraída de arquivos CAD. Portável pois a linguagem utilizada para a pesquisa foi o Java Enterprise Edition (J2EE) que corre sobre a máquina virtual Java possibilitando assim a sua usual execução em plataforma Windows, Linux, Machintosh, ou qualquer outra onde possa correr a máquina virtual Java. O software desenvolvido é capaz de gerar código para diferentes plataformas, pois o modelo genérico utilizado para representar um programa de robô, aliado às classes criadas para cada fabricante possibilitam a fácil geração de código para qualquer uma das plataformas idealizadas na fase de projecto. A referência geométrica é extraída dos arquivos neutros em 2 formatos basicamente: Arquivos ASCII tabulados no formato [x y z] e arquivos STL. Resumindo o processo: A nuvem de pontos é extraída de um dos arquivos neutros citados, estas coordenadas de referência são transformadas em escala, translações e rotações, os pontos de interesse são extraídos de acordo com a tarefa planeada, o programa proprietário é gerado e enviado para o controlador do robô. Neste caso em específico, a aplicação escolhida foi a soldagem da tampa de uma peça regular simples como podemos ver na Fig 5.3.

Capítulo 7

Trabalhos futuros

Durante o desenvolvimento da interface de programação de robôs, a abstracção utilizada em suas classes de negócio permite a sua futura aplicação na programação de diferentes tipos de maquinas industriais compostas de eixos e elos, como robôs, CNCs ou máquinas 5 eixos de corte a lazer por exemplo. De acordo com a selecção de pontos adequada pode-se programar tarefas para soldadura, polimento, pintura, montagem, inspecção ou qualquer outra área da manufactura ou serviços, possibilitando abraçar novas plataformas e/ou linguagens. Nas áreas em que devido características de repetitividade e/ou alto risco, teve seu contingente de mão-de-obra substituído por trabalho robotizado, a utilização da programação off-line pode ser usada sem qualquer restrição para reduzir o tempo de integração de uma célula de manufactura.

Para trabalhos futuros sugiro a implementação de alguns algoritmos que especializem as funcionalidades disponibilizadas neste framework. Funcionalidades que podem ser acrescidas ao projecto:

- Utilização da teoria dos grafos para o refinamento da selecção de pontos, e planeamento de tarefas através de Path Planning direcionado para:

1 - Pintura e polimento usando Smoothing;

2 - Soldadura usando Waving;

3 - Ou montagem e inspeção usando Template Matching, por exemplo.

- Estudos na área de Elementos Finitos(FEM) podem também ser incorporados para o cálculo da força que deve ser aplicada pela ferramenta do robô no trabalho sobre uma determinada peça, composta de algum material conhecido, como no processo de escultura em pedra, por exemplo.

- Desenvolvimento de um módulo Real Time que recebe feedback de uma rede de sensores(extensômetros) ligados entre a flange e a ferramenta do robô corrigindo o ângulo de ataque e a força utilizada pelos sistema de servo motores.

- Implementação da conexão via RS232 para retrocompatibilidade com plataformas que ainda não possuem interface ethernet.

Nota: Esta pesquisa foi desenvolvida no contexto do Mestrado em Engenharia Electrônica Industrial, com o apoio do Programa ALBAN, Programa de bolsas de alto nível da União Européia para a America Latina, bolsa nº E04M033540BR. Agradecimentos especiais ao Departamento de Engenharia Electrônica(DEI), Universidade do Minho, Portugal e ao SENAI-BA/CIMATEC, Brasil.

Bibliografia

[Fuller, 1999] Fuller, J. L. (1999). *Robotics: introduction, programming, and projects.* Prentice-Hall Inc., London, 2nd edition.

[Gama, 1995] Gama, E., Helm, R., Johnson, R., Vlissides, J.(1995). *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison Wesley Longman, ISBN 0-201-63361-2, 1st edition.

[Jacobson, 1998] Jacobson, I., Booch, G., Rumbaugh, J.(1998). *The Unified Software Development Process.* Addison Wesley Longman.

[Lee, 1990] Lee, D. M. A. ElMaraghy, W. H.(1990). *ROBOSIM: a CAD-based off-line programming and analysis system for robotic manipulators.* Computer-Aided Engineering Journal, Vol. 7, Nº 5, ISSN 0263-9327.

[Owen, 1994] Owen, R. M. (1994). *STEP : An Introduction.* Information Geometers Ltd

[Schenck, 1994] Schenck, D. A., Wilson, P. R.(1994). *Information Modeling: The EXPRESS way.* Oxford University Press.

[Vuoskoski, 1996] Vuoskoski, J. (1996). *Exchange of Product Data between CAD systems and a Physics Simulation Program.* Tampere University of Technology, Pori Unit, page numbers (19-23).

[Wörn, 1998] Wörn, H. (1998). Automatic off-line programming and motion planning for industrial robots. In *ISR'98, 29th International Symposium on Robotics 1998.* ISR Press.

[ABB, 2000] ABB (2000). ABB Robotics AB, RAPID Reference Manual Västeras, 172p (3HAC 5780-1).

[NBS, 1980] NBS (1980). IGES 1 - Specification for CAD data exchange. Department of Commerce, Washington DC, US, 20234.

[Mitsubishi, 2000] Mitsubishi (2000). Mitsubishi Electronics Corporation. Mitsubishi Industrial Robot - Instruction Manual, Detailed explanations of functions and operations. Nagoya. 152p (BFP-A5992-C).

[Smith, 1988] Smith, B. M., Rinaudot, G. R., Reed, K. A., Wright, T.(1988). *Initial Graphics Exchange Specification (IGES). Version 4.0. IGES/PDES Organization.* Gaithersburg, MD.

[AutoDesk] www.autodesk.pt
[CATIAV5R16] www.3ds.com/products-solutions/plm-solutions/catia/overview/
[CIMATRON] www.cimatron.com
[SolidWorks] www.solidworks.com

Anexos

ANEXO (S)

Anexo1: [Mitsubishi, 2000]

2.4.2 Robot operation control

(1) Joint interpolation movement

The robot moves with joint axis unit interpolation to the designated position. (The robot interpolates with a joint axis unit, so the end path is irrelevant.)

■ Command word

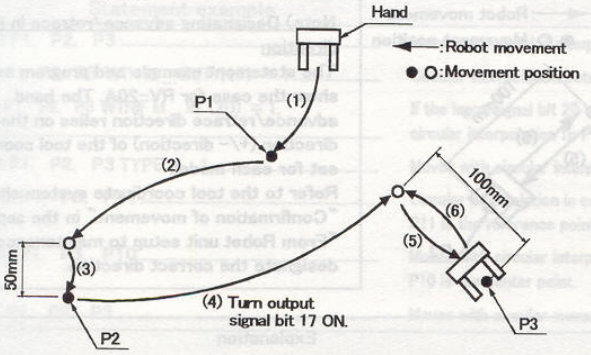
Command word	Explanation
MOV	The robot moves to the designated position with joint interpolation. An appended statement WTH or WTHIF can be designated.

■ Statement example

Statement example	Explanation
MOV P1	Moves to P1.
MOV P1 + P2	Moves to the position obtained by adding the P1 and P2 coordinate elements.
MOV P1 * P2	Moves to the position relatively converted from P1 to P2.
MOV P1, −50 (Note)	Moves from P1 to a position retracted 50mm in the hand direction.
MOV P1 WTH M_OUT (17) = 1	Starts movement toward P1, and simultaneously turns output signal bit 17 ON.
MOV P1 WTHIF M_IN (20) = 1, SKIP	If the input signal bit 20 turns ON during movement to P1, the movement to P1 is stopped, and the program proceeds to the next stop.

■ Program example

· Robot movement



⚠CAUTION

Note) Designating advance/retrace in hand direction

The statement example and program example show the case for RV-20A. The hand advance/retrace direction relies on the Z axis direction (+/− direction) of the tool coordinate set for each model. Refer to the tool coordinate system shown in "Confirmation of movement" in the separate "From Robot unit setup to maintenance", and designate the correct direction.

· Program example

Program	Explanation
10 MOV P1	;Moves to P1.
20 MOV P2, −50 Note)	;Moves from P2 to a position retracted 50mm in the hand direction.
30 MOV P2	;Moves to P2.
40 MOV P3, −100 WTH M_OUT(17)=1 Note)	;Starts movement from P3 to a position retracted 100mm in the hand direction, and turns ON output signal bit 17.
50 MOV P3	;Moves to P3.
60 MOV P3, −100 Note)	;Returns from P3 to a position retracted 100mm in the hand direction.
70 END	;Ends the program.

■ Related functions

Function	Explanation page
Designate the movement speed.	Page 16, "(5) Acceleration/deceleration time and speed control"
Designate the acceleration/deceleration time.	Page 16, "(5) Acceleration/deceleration time and speed control"
Confirm that the target position is reached.	Page 17, "(6) Confirming that the target position is reached"
Continuously move to next position without stopping at target position.	Page 15, "(4) Continuous movement"
Move linearly.	Page 12, "(2) Linear interpolation movement"
Move while drawing a circle or arc.	Page 13, "(3) Circular interpolation movement"
Add a movement command to the process.	Page 28, "(1) Appended statement"

Anexo2: [Mitsubishi, 2000]

(2) Linear interpolation movement

The end of the hand is moved with linear interpolation to the designated position.

■ Command word

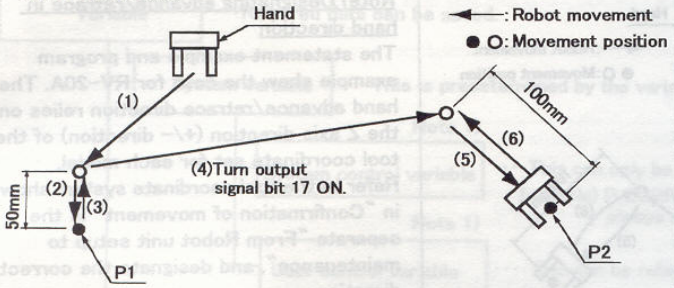
Command word	Explanation
MVS	The robot moves to the designated position with linear interpolation. An appended statement WTH or WTHIF can be designated.

■ Statement example

Statement example	Explanation
MVS P1	Moves to P1.
MVS P1 + P2	Moves to the position obtained by adding the P1 and P2 coordinate elements.
MVS P1 * P2	Moves to the position relatively converted from P1 to P2.
MVS P1, −50 Note)	Moves from P1 to a position retracted 50mm in the hand direction.
MVS , −50 Note)	Moves from the current position to a position retracted 50mm in the hand direction.
MVS P1 WTH M_OUT (17) = 1	Starts movement toward P1, and simultaneously turns output signal bit 17 ON.
MVS P1 WTHIF M_IN (20) = 1, SKIP	If the input signal bit 20 turns ON during movement to P1, the movement to P1 is stopped, and the program proceeds to the next stop.
MVS P1, TYPE 0, 2	Moves to P1 with ABC interpolation.
MVS P1, TYPE 0, 1	Moves to P1 with 3-axis orthogonal interpolation.

■ Program example

· Robot movement



⚠CAUTION

Note) Designating advance/retrace in hand direction

The statement example and program example show the case for RV-20A. The hand advance/retrace direction relies on the Z axis direction (+/− direction) of the tool coordinate set for each model.

Refer to the tool coordinate system shown in "Confirmation of movement" in the separate "From Robot unit setup to maintenance", and designate the correct direction.

· Program example

Program	Explanation
10 MVS P1, −50 Note)	;Moves with linear interpolation from P1 to a position retracted 50mm in the hand direction.
20 MVS P1	;Moves to P1 with linear interpolation.
30 MVS , −50 Note)	;Moves with linear interpolation from the current position (P1) to a position retracted 50mm in the hand direction.
40 MVS P2, −100 WTH M_OUT(17)=1 Note)	;Starts movement from P2 to a position retracted 100mm in the hand direction, and turns ON output signal bit 17.
50 MVS P2	;Moves with linear interpolation to P2.
60 MVS , −50 Note)	;Moves with linear interpolation from the current position (P2) to a position retracted 50mm in the hand direction.
70 END	;Ends the program.

■ Related functions

Function	Explanation page
Designate the movement speed	Page 16, "(5) Acceleration/deceleration time and speed control"
Designate the acceleration/deceleration time	Page 16, "(5) Acceleration/deceleration time and speed control"
Confirm that the target position is reached	Page 17, "(6) Confirming that the target position is reached"
Continuously move to next position without stopping at target position	Page 15, "(4) Continuous movement"
Move with joint interpolation	Page 11, "(1) Joint interpolation movement"
Move while drawing a circle or arc	Page 13, "(3) Circular interpolation movement"
Add a movement command to the process	Page 28, "(1) Appended statement"

Anexo3: [Mitsubishi, 2000]

(3) Circular interpolation movement

The robot moves along an arc designated with three points using three-dimensional circular interpolation.
If the current position is separated from the start point when starting circular movement, the robot will move to the start point with linear operation and then begin circular interpolation.

■ Command word

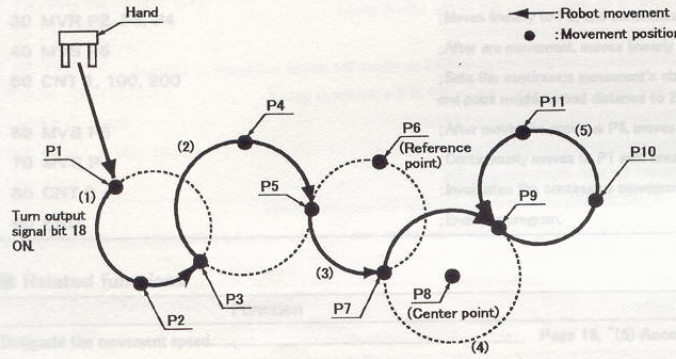
Command word	Explanation
MVR	Designates the start point, transit point and end point, and moves the robot with circular interpolation in order of the start point → transit point → end point. An appended statement WTH or WTHIF can be designated.
MVR 2	Designates the start point, end point and reference point, and moves the robot with circular interpolation from the start point → end point without passing through the reference point. An appended statement WTH or WTHIF can be designated.
MVR 3	Designates the start point, end point and center point, and moves the robot with circular interpolation from the start point to the end point. The fan angle from the start point to the end point is 0 deg. < fan angle < 180 deg. An appended statement WTH or WTHIF can be designated.
MVC	Designates the start point (end point), transit point 1 and transit point 2, and moves the robot with circular interpolation in order of the start point → transit point 1 → transit point 2 → end point. An appended statement WTH or WTHIF can be designated.

■ Statement example

Statement example	Explanation
MVR P1, P2, P3	Moves with circular interpolation between P1 → P2 → P3.
MVR P1, P2, P3 WTH M_OUT (17) = 1	Circular interpolation between P1 → P2 → P3 starts, and the output signal bit 17 turns ON.
MVR P1, P2, P3 WTHIF M_IN (20) = 1, SKIP	If the input signal bit 20 turns ON during circular interpolation between P1 → P2 → P3, circular interpolation to P1 is stopped, and the program proceeds to the next step.
MVR P1, P2, P3 TYPE 0, 1	Moves with circular interpolation between P1 → P2 → P3.
MVR2 P1, P3, P11	Circular interpolation is carried out from P1 to P3 in the direction that P11 is not passed. P11 is the reference point.
MVR3 P1, P3, P10	Moves with circular interpolation from P1 to P3 in the direction with the smallest fan angle. P10 is the center point.
MVC P1, P2, P3	Moves with circular movement from P1 → P2 → P3 → P1.

■ Program example

· Robot movement



Anexo4: [ABB, 2000]

5.2 Instruções de posicionamento

Instrução	Tipo de movimento:
MoveC	O TCP move-se através de uma trajetória circular
MoveJ	Movimento eixo a eixo (junta)
MoveL	O TCP move-se através de uma trajetória linear
MoveAbsJ	Movimento eixo a eixo (junta) absoluto
MoveCDO	Move o robô circularmente e fixa uma saída digital no canto
MoveJDO	Move o robô pelo movimento de junta e fixa uma saída digital no canto
MoveLDO	Move o robô linearmente e fixa uma saída digital no canto
MoveCSync	Move o robô circularmente e executa um procedimento RAPID
MoveJSync	Move o robô pelo movimento de junta e executa um procedimento RAPID
MoveLSync	Move o robô linearmente e executa um procedimento RAPID

1.Somente se o robô estiver equipado com a opção "Advanced Functions" (Funções Avançadas)

5.3 Procura

Durante o movimento, o robô pode procurar por uma posição de um objeto de trabalho. A posição procurada (indicada por um sinal de um sensor) é armazenada e pode ser usada posteriormente para posicionar o robô ou para calcular um deslocamento de programa.

Instrução	Tipo de movimento:
SearchC	TCP em uma trajetória circular
SearchL	TCP em uma trajetória linear

5.4 Ativação de saídas ou de interrupções em posições específicas

Normalmente, as instruções lógicas são executadas na transição de uma instrução de posicionamento para outra. Se, entretanto, instruções de movimentação especiais são usadas, estas podem ser executadas ao invés de quando o robô está em uma posição específica.

Anexo 5: Javadoc

servico

Class Ponto

```
java.lang.Object
  |
  +--servico.Ponto
```

public class **Ponto**
extends java.lang.Object

Field Summary

int	**movimento**
java.lang.String	**nome**
int	**precisao**
int	**velocidade**

Constructor Summary

protected	**Ponto**() Método construtor de um ponto genérico.
	Ponto(int newMovimento, java.lang.String newNome, int newVelocidade, int newPrecisao) Sobrecarrega o método construtor de um ponto genérico.

Method Summary

java.lang.String	**toString**() Retorna uma String com a linha de comando do ponto genérico.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Field Detail

movimento
`public int movimento`

---

nome
`public java.lang.String nome`

---

velocidade
`public int velocidade`

---

precisao
`public int precisao`

Constructor Detail

Ponto
`protected Ponto()`

Método construtor de um ponto genérico.

Ponto
```
public Ponto(int newMovimento,
             java.lang.String newNome,
             int newVelocidade,
             int newPrecisao)
```
Sobrecarrega o método construtor de um ponto genérico.

Instância um comando genérico para que no futuro, apartir do protocolo escolhido seja gerado o código de programação.

Parameters:

`newMovimento` - Inteiro que representa o tipo do movimento.
`newNome` - String que representa o ponto de passsagem do movimento.
`newVelocidade` - Valor Inteiro da velocidade.
`newPrecisao` - Valor Inteiro da precisao.

See Also:

`Ponto`

Method Detail

toString
`public java.lang.String toString()`

Retorna uma String com a linha de comando do ponto genérico.

Overrides:

`toString` in class `java.lang.Object`

Returns:

Retorna a representaçao genérica de um comando.

See Also:

`Ponto(int newMovimento,String newNome,int newVelocidade,int newPrecisao)`

servico

Class Programa

```
java.lang.Object
  |
  +--servico.Programa
```

---

public class **Programa**
extends java.lang.Object

Field Summary

java.lang.String	**id_prog**
java.util.List	**listaPontos**
java.util.List	**listaPontosVarCoord**

Constructor Summary

Programa()
Programa(java.lang.String newId) Construtor do Programa, que carrega duas listas, uma para os comandos e outra para separar as variáveis.

Method Summary

java.lang.String	**getId**() Retorna o nome identificador do programa.
java.util.Iterator	**getItLinhasComando**() Retorna um iterator para a Lista de Comandos do Programa.
java.util.Iterator	**getItVarCoord**() Retorna um iterator para a lista de variáveis dos comandos do programa.
void	**InsereComando**(Ponto ponto) Insere um ponto genérico na lista de comandos, testando se o mesmo já existe.
void	**InserePontosMovC**(java.lang.String p1) Insere as variveis das coordenadas circulares à lista de pontos das variáveis das coordenadas.
protected java.lang.String	**MostraArray**() Percorre a lista de comandos genéricos, mostrando o seu conteúdo.
protected void	**setId**(java.lang.String newId)

	Seta o nome identificador do programa.
java.lang.String	**toString**() Retorna uma string com o nome do programa e seu conteudo.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Field Detail

id_prog

public java.lang.String **id_prog**

---

listaPontos

public java.util.List **listaPontos**

---

listaPontosVarCoord

public java.util.List **listaPontosVarCoord**

Constructor Detail

Programa

public **Programa**()

Programa

public **Programa**(java.lang.String newId)

Construtor do Programa, que carrega duas listas, uma para os comandos e outra para separar as variáveis.

Parameters:

newId - Recebe o nome identificador do programa.

See Also:

ArrayList

Method Detail

setId

protected void **setId**(java.lang.String newId)

Seta o nome identificador do programa.

Parameters:

newId - Recebe o nome identificador do programa.

Returns:

void

See Also:

Programa

---

getId

`public java.lang.String **getId**()`

Retorna o nome identificador do programa.

Returns:

O nome identificador do programa.

See Also:

`Programa`

---

getItLinhasComando

`public java.util.Iterator **getItLinhasComando**()`

Retorna um iterator para a Lista de Comandos do Programa.

Returns:

Retorna um apontador para cada objeto da lista de comandos do programa.

See Also:

`Iterator`

---

getItVarCoord

`public java.util.Iterator **getItVarCoord**()`

Retorna um iterator para a lista de variáveis dos comandos do programa.

Returns:

Retorna um apontador para cada objeto da lista de variáveis dos comandos do programa.

See Also:

`Iterator`

InsereComando

```
public void InsereComando(Ponto ponto)
                    throws java.lang.Exception
```

Insere um ponto genérico na lista de comandos, testando se o mesmo já existe.

Parameters:

`ponto` - Recebe um objeto Ponto.

Returns:

void

Throws:

`Se` - já existe o ponto na lista, lança uma exceção.

`java.lang.Exception`

See Also:

`Ponto`

InserePontosMovC

```
public void InserePontosMovC(java.lang.String p1)
                      throws java.lang.Exception
```

Insere as variveis das coordenadas circulares à lista de pontos das variáveis das coordenadas.

Parameters:

`p1` - Recebe string com as variáveis, separadas por vírgulas.

Returns:

void

`java.lang.Exception`

See Also:

`java.text.SubString`

MostraArray

`protected java.lang.String **MostraArray**()`

Percorre a lista de comandos genéricos, mostrando o seu conteúdo.

Returns:

Retorna a lista de comandos genéricos.
See Also:
`ArrayList`

---

toString

`public java.lang.String toString()`

Retorna uma string com o nome do programa e seu conteudo.
Overrides:
`toString` in class `java.lang.Object`
Returns:
Retorna uma string com o nome do programa e seu conteudo.
See Also:
`Programa`

protocolo

Interface Protopon

All Known Implementing Classes:
Ponto_Abb, Ponto_Fanuc, Ponto_Mitsubishi

---

public interface **Protopon**

Method Summary

`java.lang.String`	**EscolhePrecisao**`(int pre)` Método que define a precisão.
`java.lang.String`	**EscolheVelocidade**`(int vel)` Método que define a velocidade.
`java.lang.String`	**tipoMov**`(int op)` Método que define comando de movimento.
`java.lang.String`	**toString**`()` Retorna uma String com a linha de comando do ponto.

Method Detail

tipoMov

`public java.lang.String tipoMov(int op)`

Método que define comando de movimento.
Parameters:
`op` - Receber um inteiro que vai representar o tipo do movimento.
Returns:
Retorna uma string através das variáveis static definidas na classe.
See Also:
`Ponto_Abb`

EscolheVelocidade

`public java.lang.String EscolheVelocidade(int vel)`

Método que define a velocidade.
Parameters:
`vel` - Recebe um inteiro como valor da velocidade do movimento.

Returns:
Retorna a velocidade.
See Also:
`Ponto`

EscolhePrecisao

`public java.lang.String` **`EscolhePrecisao`**`(int pre)`

Método que define a precisão.
Parameters:
`pre` - Recebe um inteiro como valor da precisão do movimento.
Returns:
Retorna a precisão.
See Also:
`Ponto`

---

toString

`public java.lang.String` **`toString`**`()`

Retorna uma String com a linha de comando do ponto.
Overrides:
`toString` in class `java.lang.Object`
Returns:
Retorna o comando de movimento.
See Also:
`Ponto`

protocolo

Class Ponto_Abb

```
java.lang.Object
  |
  +--protocolo.Ponto_Abb
```

All Implemented Interfaces:
Protopon

---

public class **Ponto_Abb**
extends java.lang.Object
implements Protopon

Field Summary	
protected java.lang.String	**ferramenta**
protected java.lang.String	**movimento**
protected java.lang.String	**nomePts**
protected java.lang.String	**precisao**
protected java.lang.String	**velocidade**

Constructor Summary

Ponto_Abb()
Ponto_Abb(servico.Ponto ponto) Construtor do comando da linguagem Rapid.

Method Summary

java.lang.String	**AbreGarra**() Abre a garra da solda a Ponto.
java.lang.String	**EscolhePrecisao**(int pre) Retorna a precisão no formato da linguagem Rapid.
java.lang.String	**EscolheVelocidade**(int vel) Retorna a velocidade no formato da linguagem Rapid.
java.lang.String	**FechaGarra**() Fecha a garra da solda a Ponto.
java.lang.String	**TempoEspera**(float tempo) Espera por um determinado tempo.
java.lang.String	**tipoMov**(int op) Preenche comando de movimento da linguagem Rapid.
java.lang.String	**toString**() Retorna uma String com a linha de comando do ponto do tipo "MoveL P1,v1000,z100,tool0".

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Field Detail

movimento

protected java.lang.String **movimento**

---

nomePts

protected java.lang.String **nomePts**

---

velocidade

protected java.lang.String **velocidade**

---

precisao
`protected java.lang.String` **`precisao`**

---

ferramenta
`protected java.lang.String` **`ferramenta`**

Constructor Detail

Ponto_Abb
`public` **`Ponto_Abb`**`()`

Ponto_Abb
`public` **`Ponto_Abb`**`(servico.Ponto ponto)`

Construtor do comando da linguagem Rapid.

Parameters:

`ponto` - Recebe um ponto genérico.

See Also:

`Ponto_Abb`

Method Detail

tipoMov
`public java.lang.String` **`tipoMov`**`(int op)`

Preenche comando de movimento da linguagem Rapid.

Specified by:

tipoMov in interface Protopon

Parameters:

`op` - Receber um inteiro que vai representar o tipo do movimento.

Returns:

Retorna uma string com o movimento na linguagem Rapid.

See Also:

`Ponto_Abb`

EscolheVelocidade
`public java.lang.String` **`EscolheVelocidade`**`(int vel)`

Retorna a velocidade no formato da linguagem Rapid.

Specified by:

EscolheVelocidade in interface Protopon

Parameters:

`vel` - Recebe um inteiro como valor da velocidade do movimento.

Returns:

Retorna a velocidade no padrão da linguagem Rapid.

See Also:

`toString()`

EscolhePrecisao
`public java.lang.String` **`EscolhePrecisao`**`(int pre)`

Retorna a precisão no formato da linguagem Rapid.

Specified by:

EscolhePrecisao in interface Protopon

Parameters:

`pre` - Recebe um inteiro como valor da precisão do movimento.

Returns:

Retorna a precisão no padrão da linguagem Rapid.

See Also:
`toString()`

---

toString

`public java.lang.String toString()`

Retorna uma String com a linha de comando do ponto do tipo "MoveL P1,v1000,z100,tool0".

Specified by:
toString in interface Protopon

Overrides:
`toString` in class `java.lang.Object`

Returns:
Retorna o comando de movimento da linguagem Rapid.

See Also:
`toString()`

FechaGarra

`public java.lang.String FechaGarra()`

Fecha a garra da solda a Ponto.

Returns:
Retorna o comando de Fechar Garra de Solda a Ponto na linguagem Rapid.

See Also:
`Ponto_Abb`

AbreGarra

`public java.lang.String AbreGarra()`

Abre a garra da solda a Ponto.

Returns:
Retorna o comando de Abrir Garra sde Solda a Ponto na linguagem Rapid.

See Also:
`Ponto_Abb`

TempoEspera

`public java.lang.String TempoEspera(float tempo)`

Espera por um determinado tempo.

Parameters:
`tempo` - Tempo de espera para executar a próxima instrução.

Returns:
Retorna o comando de espera na linguagem Rapid.

See Also:
`Ponto_Abb`

protocolo

Class Ponto_Fanuc

```
java.lang.Object
  |
  +--protocolo.Ponto_Fanuc
```

All Implemented Interfaces:

Protopon

---

public class **Ponto_Fanuc**
extends java.lang.Object
implements Protopon

Field Summary

protected java.lang.String	**ferramenta**
protected java.lang.String	**movimento**
protected java.lang.String	**nomePts**
protected java.lang.String	**precisao**
protected java.lang.String	**velocidade**

Constructor Summary

Ponto_Fanuc()
Ponto_Fanuc(servico.Ponto ponto) Construtor do comando da linguagem Karel.

Method Summary

java.lang.String	**EscolhePrecisao**(int pre) Retorna a precisão no formato da linguagem Karel.
java.lang.String	**EscolheVelocidade**(int vel) Retorna a velocidade no formato da linguagem Karel.
java.lang.String	**TempoEspera**(float tempo) Espera por um determinado tempo.
java.lang.String	**tipoMov**(int op) Preenche comando de movimento na linguagem Karel.
java.lang.String	**toString**() Retorna uma String com a linha de comando do ponto do tipo "L P[1] 100mm/s FINE"

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Field Detail

movimento
protected java.lang.String **movimento**

---

nomePts
protected java.lang.String **nomePts**

---

velocidade
protected java.lang.String **velocidade**

---

precisao
protected java.lang.String **precisao**

---

ferramenta
protected java.lang.String **ferramenta**

Constructor Detail

Ponto_Fanuc
public **Ponto_Fanuc**()

Ponto_Fanuc
public **Ponto_Fanuc**(servico.Ponto ponto)

Construtor do comando da linguagem Karel.

See Also:

Ponto_Fanuc

Method Detail

tipoMov
public java.lang.String **tipoMov**(int op)

Preenche comando de movimento na linguagem Karel.

Specified by:

tipoMov in interface Protopon

Parameters:

op - Receber um inteiro que vai representar o tipo do movimento.

Returns:

Retorna uma string com o movimento na linguagem Karel.

See Also:

Ponto_Fanuc

EscolheVelocidade

`public java.lang.String` **`EscolheVelocidade`**`(int vel)`

Retorna a velocidade no formato da linguagem Karel.

Specified by:

EscolheVelocidade in interface Protopon

Parameters:

`vel` - Recebe um inteiro como valor da velocidade do movimento.

Returns:

Retorna a velocidade no padrão da linguagem Karel.

See Also:

`toString()`

EscolhePrecisao

`public java.lang.String` **`EscolhePrecisao`**`(int pre)`

Retorna a precisão no formato da linguagem Karel.

Specified by:

EscolhePrecisao in interface Protopon

Parameters:

`pre` - Recebe um inteiro como valor da precisão do movimento.

Returns:

Retorna a precisão no padrão da linguagem Karel.

See Also:

`toString()`

---

toString

`public java.lang.String` **`toString`**`()`

Retorna uma String com a linha de comando do ponto do tipo "L P[1] 100mm/s FINE"

Specified by:

toString in interface Protopon

Overrides:

`toString` in class `java.lang.Object`

Returns:

Retorna o comando de movimento da linguagem Karel.

See Also:

`Ponto_Fanuc`

TempoEspera

`public java.lang.String` **`TempoEspera`**`(float tempo)`

Espera por um determinado tempo.

Parameters:

`tempo` - Tempo de espera para executar a próxima instrução.

Returns:

Retorna o comando de espera na linguagem Karel.

See Also:

`Ponto_Fanuc`

protocolo

Class Ponto_Mitsubishi

```
java.lang.Object
  |
  +--protocolo.Ponto_Mitsubishi
```

All Implemented Interfaces:

Protopon

---

public class **Ponto_Mitsubishi**
extends java.lang.Object
implements Protopon

Field Summary

protected java.lang.String	**ferramenta**
protected java.lang.String	**movimento**
protected java.lang.String	**nomePts**
protected java.lang.String	**precisao**
protected java.lang.String	**velocidade**

Constructor Summary

Ponto_Mitsubishi()
Ponto_Mitsubishi(servico.Ponto ponto) Construtor do comando da linguagem MelfaBasic.

Method Summary

java.lang.String	**AbreGarra**() Abre a garra da solda a Ponto.
java.lang.String	**EscolhePrecisao**(int pre) Retorna a precisão no formato da linguagem MelfaBasic.
java.lang.String	**EscolheVelocidade**(int vel) Retorna a velocidade no formato da linguagem MelfaBasic.
java.lang.String	**FechaGarra**() Fecha a garra da solda a Ponto.
java.lang.String	**TempoEspera**(float tempo) Espera por um determinado tempo.
java.lang.String	**tipoMov**(int op) Preenche comando de movimento da linguagem MelfaBasic.
java.lang.String	**toString**()

	Retorna uma String com a linha de comando do ponto do tipo "MoveL P1,v1000,z100,tool0".

Methods inherited from class java.lang.Object
`clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait`

Field Detail

movimento
`protected java.lang.String` **`movimento`**

---

nomePts
`protected java.lang.String` **`nomePts`**

---

velocidade
`protected java.lang.String` **`velocidade`**

---

precisao
`protected java.lang.String` **`precisao`**

---

ferramenta
`protected java.lang.String` **`ferramenta`**

Constructor Detail

Ponto_Mitsubishi
`public` **`Ponto_Mitsubishi`**`()`

Ponto_Mitsubishi
`public` **`Ponto_Mitsubishi`**`(servico.Ponto ponto)`

Construtor do comando da linguagem MelfaBasic.

Parameters:

`ponto` - Recebe um ponto genérico.

See Also:

`Ponto_Mitsubishi`

Method Detail

tipoMov
`public java.lang.String` **`tipoMov`**`(int op)`

Preenche comando de movimento da linguagem MelfaBasic.

Specified by:

tipoMov in interface Protopon

Parameters:
`op` - Receber um inteiro que vai representar o tipo do movimento.
Returns:
Retorna uma string com o movimento na linguagem MelfaBasic.
See Also:
`Ponto_Mitsubishi`

EscolheVelocidade

`public java.lang.String` **`EscolheVelocidade`**`(int vel)`

Retorna a velocidade no formato da linguagem MelfaBasic.
Specified by:
EscolheVelocidade in interface Protopon
Parameters:
`vel` - Recebe um inteiro como valor da velocidade do movimento.
Returns:
Retorna a velocidade no padrão da linguagem MelfaBasic.
See Also:
`toString()`

EscolhePrecisao

`public java.lang.String` **`EscolhePrecisao`**`(int pre)`

Retorna a precisão no formato da linguagem MelfaBasic.
Specified by:
EscolhePrecisao in interface Protopon
Parameters:
`pre` - Recebe um inteiro como valor da precisão do movimento.
Returns:
Retorna a precisão no padrão da linguagem MelfaBasic.
See Also:
`toString()`

---

toString

`public java.lang.String` **`toString`**`()`

Retorna uma String com a linha de comando do ponto do tipo "MoveL P1,v1000,z100,tool0".
Specified by:
toString in interface Protopon
Overrides:
`toString` in class `java.lang.Object`
Returns:
Retorna o comando de movimento da linguagem MelfaBasic.
See Also:
`toString()`

FechaGarra

`public java.lang.String` **`FechaGarra`**`()`

Fecha a garra da solda a Ponto.
Returns:
Retorna o comando de Fechar Garra na linguagem MelfaBasic.
See Also:
`Ponto_Mitsubishi`

AbreGarra

`public java.lang.String` **`AbreGarra`**`()`

Abre a garra da solda a Ponto.
Returns:

Retorna o comando de Abrir Garra na linguagem MelfaBasic.
See Also:
Ponto_Mitsubishi

TempoEspera

`public java.lang.String` **`TempoEspera`**`(float tempo)`

Espera por um determinado tempo.
Parameters:
`tempo` - Tempo de espera para executar a próxima instrução.
Returns:
Retorna o comando de espera na linguagem MelfaBasic.
See Also:
Ponto_Mitsubishi

otimiza

Class Factory

```
java.lang.Object
  |
  +--otimiza.Factory
```

---

public class **Factory**
extends java.lang.Object

Constructor Summary

`Factory``()`	

Method Summary

`persist.Arquivo_Karel`	**`getInstanciaArquivo_Karel`**`()` Instancia um objeto de Armazenamento para a linguagem Karel.
`persist.Arquivo_MelfaBasic`	**`getInstanciaArquivo_MelfaBasic`**`()` Instancia um objeto de Armazenamento para a linguagem MelfaBasicIV.
`persist.Arquivo_Rapid`	**`getInstanciaArquivo_Rapid`**`()` Instancia um objeto de Armazenamento para a linguagem Rapid.
`protocolo.CarregaProtocolo`	**`getInstanciaCarregaProtocolo`**`(int protocolo)` Instancia um objeto de carga de protocolo de trabalho.
`servico.Ponto`	**`getInstanciaPonto`**`(int newMovimento, java.lang.String newNome, int newVelocidade, int newPrecisao)` Instancia um ponto genérico.
`protocolo.Ponto_Abb`	**`getInstanciaPontoAbb`**`()` Instancia um objeto que conhece os comandos da linguagem Rapid.
`protocolo.Ponto_Abb`	**`getInstanciaPontoAbb`**`(servico.Ponto p)` Instancia um objeto que conhece os comandos da linguagem Rapid.

protocolo.Ponto_Fanuc	**getInstanciaPontoFanuc**() Instancia um objeto que conhece os comandos da linguagem Karel.
protocolo.Ponto_Fanuc	**getInstanciaPontoFanuc**(servico.Ponto p) Instancia um objeto que conhece os comandos da linguagem Karel.
protocolo.Ponto_Mitsubishi	**getInstanciaPontoMitsubishi**() Instancia um objeto que conhece os comandos da linguagem MelfaBasic.
protocolo.Ponto_Mitsubishi	**getInstanciaPontoMitsubishi**(servico.Ponto p) Instancia um objeto que conhece os comandos da linguagem MelfaBasic.
servico.Programa	**getInstanciaPrograma**() Instancia um objeto programa.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Factory

```
public Factory()
```

Method Detail

getInstanciaPonto

```
public servico.Ponto getInstanciaPonto(int newMovimento,
                                       java.lang.String newNome,
                                       int newVelocidade,
                                       int newPrecisao)
```

Instancia um ponto genérico.

Parameters:

newMovimento - Inteiro que representa o tipo do movimento newNome String que representa o ponto de passsagem do movimento newVelocidade Valor Inteiro da velocidade newPrecisao Valor Inteiro da precisao

Returns:

void

See Also:

Class#Ponto

---

getInstanciaPontoAbb

```
public protocolo.Ponto_Abb getInstanciaPontoAbb()
```

Instancia um objeto que conhece os comandos da linguagem Rapid.

Returns:

Retorna um objeto que conhece os comandos da linguagem Rapid.

See Also:

Class#Ponto_Abb

---

getInstanciaPontoFanuc

`public protocolo.Ponto_Fanuc` **`getInstanciaPontoFanuc`**`()`

Instancia um objeto que conhece os comandos da linguagem Karel.

Returns:

Retorna um objeto que conhece os comandos da linguagem Karel.

See Also:

`Class#Ponto_Fanuc`

---

getInstanciaPontoMitsubishi

`public protocolo.Ponto_Mitsubishi` **`getInstanciaPontoMitsubishi`**`()`

Instancia um objeto que conhece os comandos da linguagem MelfaBasic.

Returns:

Retorna um objeto que conhece os comandos da linguagem MelfaBasic.

See Also:

`Class#Ponto_Fanuc`

---

getInstanciaPontoAbb

`public protocolo.Ponto_Abb` **`getInstanciaPontoAbb`**`(servico.Ponto p)`

Instancia um objeto que conhece os comandos da linguagem Rapid.

Returns:

Retorna um objeto que conhece os comandos da linguagem Rapid.

See Also:

`Class#Ponto_Abb`

---

getInstanciaPontoFanuc

`public protocolo.Ponto_Fanuc` **`getInstanciaPontoFanuc`**`(servico.Ponto p)`

Instancia um objeto que conhece os comandos da linguagem Karel.

Returns:

Retorna um objeto que conhece os comandos da linguagem Karel.

See Also:

`Class#Ponto_Fanuc`

---

getInstanciaPontoMitsubishi

`public protocolo.Ponto_Mitsubishi`
`getInstanciaPontoMitsubishi``(servico.Ponto p)`

Instancia um objeto que conhece os comandos da linguagem MelfaBasic.

Returns:

Retorna um objeto que conhece os comandos da linguagem MelfaBasic.

See Also:

`Class#Ponto_Fanuc`

---

getInstanciaPrograma

`public servico.Programa` **`getInstanciaPrograma`**`()`

Instancia um objeto programa.

Returns:

Retorna um objeto programa.

See Also:

`Class#Programa`

---

getInstanciaArquivo_Rapid

`public persist.Arquivo_Rapid` **`getInstanciaArquivo_Rapid`**`()`

Instancia um objeto de Armazenamento para a linguagem Rapid.

Returns:

Retorna um objeto Armazenamento.
See Also:
`Arquivo_Rapid`

---

getInstanciaArquivo_Karel

`public persist.Arquivo_Karel` **`getInstanciaArquivo_Karel`**`()`

Instancia um objeto de Armazenamento para a linguagem Karel.
Returns:
Retorna um objeto Armazenamento.
See Also:
`Arquivo_Karel`

---

getInstanciaArquivo_MelfaBasic

`public persist.Arquivo_MelfaBasic` **`getInstanciaArquivo_MelfaBasic`**`()`

Instancia um objeto de Armazenamento para a linguagem MelfaBasicIV.
Returns:
Retorna um objeto Armazenamento.
See Also:
`Arquivo_MelfaBasic`

---

getInstanciaCarregaProtocolo

`public protocolo.CarregaProtocolo`
`getInstanciaCarregaProtocolo``(int protocolo)`

Instancia um objeto de carga de protocolo de trabalho.
Parameters:
`protocolo` - Recebe o protocolo corrente de trabalho.
Returns:
Retorna um protocolo para robôs.
See Also:
`CarregaProtocolo`

otimiza

Class Carregador

```
java.lang.Object
  |
  +--otimiza.Carregador
```

---

public class **Carregador**
extends java.lang.Object

Constructor Summary

`Carregador``()`	

Method Summary

`void`	**`Carregador`**`()` Construtor da classe Carregador.
`void`	**`CarregadorArquivo`**`()`

	Instancia a classe Armazenamento e chama seu método para a criação de um arquivo.
void	**CarregadorPonto**(int movimento, java.lang.String nomePonto, int velocidade, int precisao) Instancia novo ponto e insere na coleção.
void	**CarregadorTipoNomePrograma**(int prot, java.lang.String nome) Instancia novo programa pelo nome e seta fabricante.
java.lang.String	**toString**() Chama o método toString da classe Programa.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail

Carregador

```
public Carregador()
```

Method Detail

Carregador

```
public void Carregador()
```

Construtor da classe Carregador.

Returns:

void

See Also:

Class#Carregador

CarregadorTipoNomePrograma

```
public void CarregadorTipoNomePrograma(int prot,
                                       java.lang.String nome)
```

Instancia novo programa pelo nome e seta fabricante.

Parameters:

prot - Recebe o inteiro que representa o protocolo que vai ser utilizado para criar o progama.
nome Recebe o nome do programa.

Returns:

void

See Also:

Class#Carregador

CarregadorPonto

```
public void CarregadorPonto(int movimento,
                            java.lang.String nomePonto,
                            int velocidade,
                            int precisao)
```

```
                    throws java.lang.Exception
```

Instancia novo ponto e insere na coleção.

Parameters:

`movimento` - Inteiro que representa o tipo do movimento. nomePonto String que representa o ponto de passsagem do movimento. velocidade Valor Inteiro da velocidade. precisao Valor Inteiro da precisao.

Returns:

void

`java.lang.Exception`

See Also:

`Class#Programa`

CarregadorArquivo

```
public void CarregadorArquivo()
                        throws java.lang.Exception
```

Instancia a classe Armazenamento e chama seu método para a criação de um arquivo.

Returns:

void

Throws:

`@Exception` - Lança uma exceçao caso ocorra algum erro na criação do arquivo.

`java.lang.Exception`

See Also:

`Class#Armazenamento`

---

toString

```
public java.lang.String toString()
```

Chama o método toString da classe Programa.

Overrides:

`toString` in class `java.lang.Object`

Returns:

Retorna através do toString do programa genérico, as linhas de comando carregadas.

See Also:

`Class#Programa`