ecai

Universidade Federal de Santa Catarina
Centro Tecnológico
Curso de Engenharia de Controle e
Automação Industrial

ufsc

Aplicação de Linguagens Baseadas em Máquinas Virtuais em Comando Numérico

Monografia submetida à Universidade Federal de Santa Catarina como requisito para a aprovação da disciplina:
EEL 5901: Projeto de Fim de Curso

Hélio Souza Araújo

Florianópolis, Fevereiro de 1998

Aplicação de Linguagens Baseadas em Máquinas Virtuais em Comando Numérico

Hélio Souza Araújo

Esta monografia foi julgada no contexto da disciplina
EEL 5901: Projeto de Fim de Curso
e aprovada na sua forma final pelo
Curso de Engenharia de Controle e Automação Industrial

Banca Examinadora:

Eng. Dave Thomas
Eng. Márcio Quintaes Marchini
Orientadores Empresa

Prof. Luiz Fernando Bier Melgarejo
Orientador do Curso

Prof. Augusto Humberto Bruciapaglia
Responsável pela disciplina e Coordenador do Curso

Prof. Abelardo Alves de Queiroz, *Avaliador*

Frederico Rabello de Moraes, *Debatedor*

Fábio Pavanati, *Debatedor*

Agradecimentos

À minha família, cujo apoio através de conselhos e palavras de incentivo foi fundamental no cumprimento de mais esta etapa da minha vida.

Aos colegas, que unidos pelas noites de estudos ao longo destes cinco anos tornaram-se grandes amigos sem os quais eu não teria conseguido.

Ao pessoal do Edugraf e em especial ao amigo e professor Fernando Melgarejo, por criar um ambiente de trabalho tão fértil para o florescimento de novas idéias, onde aprendi tanto durante o meu tempo na universidade.

Aos orientadores Dave Thomas e Márcio Marchini pela oportunidade concedida de trabalhar em uma empresa tão única quanto é a OTI durante esses seis meses.

E por último, mas não menos importante a todas aquelas pessoas que algum modo contribuíram para a realização deste trabalho, em especial à empresa Audaces Automação e Informática Industrial.

Resumo

O presente trabalho tem por objetivo servir como ponto de partida em um estudo mais amplo sobre a construção de componentes de software reusáveis para aplicações relacionadas a comando numérico (CNC).

A partir da análise dos problemas mais comuns relacionados ao desenvolvimento e manutenção de softwares de uma forma geral, adota-se como ferramenta básica de programação linguagens que apresentam duas características principais:

- São orientadas a objetos;
- São baseadas em máquinas virtuais.

O projeto desenvolvido na Object Technology International (OTI), sediada em Ottawa – Canadá, com o apoio do Laboratório de Software Educacional da Universidade Federal de Santa Catarina pode ser dividido em quatro partes principais:

- Escolha e justificativa da tecnologia adotada;
- Estudo dos conceitos específicos ao domínio do problema, especificamente máquinas de comando numérico e sua programação, dando-se ênfase ao comando propriamente dito, com a sua posterior modelagem;
- Implementação dos componentes modelados que não necessitavam de hardware externo para o seu funcionamento;
- Desenvolvimento de uma aplicação completa fazendo-se uso dos componentes criados.

Abstract

This work has as its main goal to be the starting point for a more complete study about the construction of reusable software components for CNC applications.

After an analysis of the most usual problems related to software development and maintenance, it is chosen as the basic programming tool languages that present these two characteristics:

- Are object-oriented;
- Are virtual machine based.

The project, which was developed at Object Technology International (OTI), settled in Ottawa – Canada, with support of the Laboratório de Software Educacional of the Universidade Federal de Santa Catarina is made up of four main sections:

- The choice and justification of the technology to be adopted;
- Study of the basic concepts related to the problem domain, more specifically CNC machines, with emphasis on the CNC control itself, followed by its object-oriented modeling;
- Implementation of the modeled components that did not need any external hardware;
- Development of a complete application using the components just created.

Sumário

Lista de Figuras

Lista de Tabelas

1. Introdução

O primeiro passo dado pelo homem em direção à automatização de máquinas ferramenta foi a substituição da sua força física como fonte de energia por outras fontes de energia disponíveis na natureza (tração animal, energia hidráulica e energia elétrica).

O próximo passo está sendo a eliminação gradual da sua interferência no processo. Dado um programa que especifica como ferramenta e peça devem deslocar-se uma em relação à outra durante a usinagem, a máquina deve trabalhar de forma automática, necessitando o mínimo possível de supervisão.

Inicialmente as máquinas possuíam somente um acionamento central, do qual todos os demais movimentos eram obtidos através de transmissões mecânicas. Posteriormente as máquinas evoluíram no sentido em que este acionamento único foi sendo substituído por múltiplos acionamentos, cada um relativo a um grau de liberdade da máquina, aumentando com isso a sua flexibilidade (e também o seu grau de complexidade).

Neste momento surge o comando numérico (CNC) como o elemento que vai gerenciar de forma coordenada os diversos eixos da máquina ferramenta, sendo que a maneira como isso vai ser realizado será definida pelo programa armazenado na sua memória.

Historicamente nota-se que não houve uma grande preocupação em se adotar uma metodologia para o desenvolvimento dos programas que coordenam o funcionamento do comando, o que fez com que estes viessem a apresentar grandes problemas em termos de qualidade de software.

O objetivo principal deste trabalho é estudar a aplicação de linguagens baseadas em máquinas virtuais na construção de componentes de software reusáveis para aplicações CNC, tentando ao mesmo tempo minimizar os seguintes problemas que parecem ser as maiores deficiências comumente encontradas em softwares em geral:

- extensibilidade
- reusabilidade
- compatibilidade
- portabilidade

No próximo capítulo são abordados alguns conceitos básicos de qualidade de software que levaram à escolha da orientação a objetos.

No capítulo três é introduzido o conceito de máquina virtual e em seguida são estudadas quatro bibliotecas de classes correspondentes a quatro máquinas virtuais diferentes. A partir desse estudo escolhe-se uma máquina virtual alvo para todos os componentes de software a serem desenvolvidos.

No quarto capítulo é discutido o funcionamento dos comandos numéricos de uma forma geral, passando-se logo após aos conceitos básicos relacionados à sua programação, servindo de base à modelagem de um comando numérico na forma de componentes reusáveis de software.

A partir das conclusões tiradas no capítulo quatro, no capítulo cinco inicia-se o desenvolvimento dos componentes identificados anteriormente, com o projeto, implementação e testes de um protótipo do principal componente do comando numérico: o interpretador para a linguagem utilizada na programação das máquinas CNC. Após a implementação do interpretador

em Smalltalk, são realizados testes do seu funcionamento por meio de uma máquina CNC simulada. Por fim são identificados pontos que poderiam ser melhorados no intuito de se conseguir uma implementação mais otimizada.

Por determinação da empresa, uma versão em Java do interpretador deveria ser disponibilizada, tentando-se corrigir os problemas de otimização identificados anteriormente. Desta forma, no capítulo seis uma reengenharia do componente em questão é realizada, adicionando novas funcionalidades e corrigindo os problemas mencionados acima.

Após concluída a versão em Java (otimizada), o interpretador foi portado de volta ao Smalltalk e utilizado no desenvolvimento de uma aplicação completa. Trata-se de um simulador que descreve o caminho percorrido pela ferramenta durante o processo de usinagem.

E por fim, no capítulo oito são expostas as conclusões e as perspectivas para trabalhos futuros sobre o tema.

1.1. Sobre a Empresa

A OTI é uma empresa de engenharia de software líder mundial na tecnologia de orientação a objetos. Sediada em Ottawa - Canadá, onde o projeto foi realizado, a OTI tem laboratórios de pesquisa e desenvolvimento em outros dois continentes.

Entre as suas atividades principais estão o desenvolvimento de software em parceria e o licenciamento de tecnologia através de convênios com outras empresas.

Através de alianças tecnológicas de longo prazo são realizadas parcerias com os clientes, possibilitando a eles a entrega de seus produtos em tempo hábil e a preços competitivos, além de lhes dar a habilidade de influenciar a linha de produtos da OTI, direcionando a sua linha de pesquisa. Por outro lado, isto permite à OTI tomar iniciativas estratégicas em termos de pesquisa e desenvolvimento de novos produtos, mantendo com isso a sua vantagem competitiva.

A lista de parceiros da OTI inclui vários dos grandes participantes na indústria de software orientado a objetos. Entre eles estão:

- IBM, que licenciou o OTI Smalltalk e a sua tecnologia de gerenciamento de componentes, utilizando-os nos seus produtos IBM VisualAge e IBM Smalltalk;
- Siemens Nixdorf Informationssysteme AG, usuária do OTI Smalltalk e seus componentes no seu produto FIN.I.S. "Financial Institution Solution", que é um ambiente de desenvolvimento para aplicações financeiras;
- Tektronix, que licencia o OTI Embedded Smalltalk para a construção de uma família de osciloscópios de alto desempenho;
- Texas Instruments, que licencia o ENVY/Developer para o seu produto ControlWORKS, utilizado em instalações de beneficiamento de água;
- Lockheed Canada, que utiliza o OTI Multi-processor Smalltalk na construção de sistemas de radares para a marinha canadense.

Ao longo dos últimos onze anos desde a sua fundação, a OTI tem desenvolvido e distribuído sistemas para ambientes que vão desde mainframes IBM até sistemas embutidos em tempo real. Entre os seus principais produtos estão:

- *Tecnologia Smalltalk* - a OTI é um dos líderes no desenvolvimento da linguagem Smalltalk. Baseado em padrões estabelecidos pela indústria, incluindo Motif, X/Windows e POSIX, o OTI Smalltalk oferece portabilidade de aplicações entre as plataformas mais comuns como Windows, OS/2, UNIX e Macintosh.
- *Ambiente de Desenvolvimento de Componentes* - ENVY/Developer é um padrão para ambientes de Smalltalk, que permite o desenvolvimento de componentes reusáveis de software por múltiplas equipes de programadores colaborativamente. O ENVY/Developer inclui controle de versões, gerenciamento de configurações e recursos para programação em equipe, necessários para o desenvolvimento de aplicações de grande porte.
- *Desenvolvimento de Sistemas Embutidos* – O OTI Embedded Smalltalk suporta "debugging" remoto, gerenciamento de código fonte, produção de arquivos executáveis compactos para a gravação em ROM e um gerenciamento de memória eficiente. Há também uma versão especial do OTI Smalltalk, específica para o desenvolvimento para máquinas com múltiplos processadores.

Além disso a OTI realiza pesquisa e desenvolvimento em muitas outras áreas relacionadas à tecnologia de orientação a objetos. A empresa tem grande atuação nas áreas de sistemas distribuídos, ferramentas para testes, sistemas especialistas, tecnologia CASE e tecnologia de compactação de imagens, utilizada na geração de arquivos executáveis para se eliminar os componentes da linguagem utilizada que são desnecessários à uma aplicação específica.

2. Qualidade de Software

2.1. Introdução

Freqüentemente um programador após completar mais uma vez o ciclo de desenvolvimento de um novo software se depara com uma das seguintes situações:

- a especificação original do projeto sofreu alterações;
- foram detectados problemas não identificados durante os testes do projeto;
- um novo projeto apresenta a possibilidade de emprego de uma solução já implementada anteriormente.

Nas três situações a dificuldade do seu trabalho pode variar enormemente, sendo que o custo relacionado ao projeto será proporcional à dificuldade e tempo envolvidos.

2.2. As Cinco Qualidades

Freqüentemente discussões sobre software e qualidade de software consideram apenas a fase de desenvolvimento. Entretanto, esta é somente a ponta do "iceberg" já que a maior parte do custo total de um software é proveniente da sua manutenção. Com esse termo tenta-se aqui abranger não só a tarefa de remover erros de um programa, mas também a tarefa de adaptá-lo a novas especificações.

Segundo [MEYER 88], 70% do custo de um software era devido à sua manutenção. Dados de outros autores[1] do mesmo período demostram que essa era uma tendência crescente.

Visando reduzir os custos tanto de manutenção, como de desenvolvimento, torna-se evidente a necessidade de se buscar cinco qualidades que hoje em dia refletem alguns dos problemas mais sérios relacionados ao desenvolvimento de software: "corretude", robustez, extensibilidade, reusabilidade e compatibilidade.

2.2.1. "Corretude"

É a habilidade que um software possui de desempenhar exatamente as tarefas para as quais foi programado, como definido pelos requisitos e especificações do projeto.

2.2.2. Robustez

Nem todas as situações de operação de um software podem ser previstas durante o projeto, razão pela qual torna-se muito importante provê-lo com mecanismos que o impeçam de provocar algum evento catastrófico, ou na pior das hipóteses, que garantam o seu encerramento de forma elegante na ocorrência de um evento inesperado. Em outras palavras, robustez pode ser definida como a capacidade de um software de funcionar mesmo em situações anormais.

[1] Um desses autores, [PRESSMAN 87] cita como porcentagens do custo de um software dedicadas à sua manutenção nas décadas de 1970, 80 e 90 respectivamente 35, 50 e 75%.

2.2.3. Extensibilidade

É a facilidade com que softwares podem ser adaptados a mudanças de especificações. Teoricamente fazer alterações em um *software* não deveria ser um processo difícil, mas não é o que ocorre na prática à medida que os sistemas tornam-se cada vez maiores e mais complexos.

Embora alguns princípios básicos que aumentam a extensibilidade possam ser aplicados até mesmo a pequenos projetos, é nos projetos de maior porte que esses têm a sua importância evidenciada. Dois desses princípios são:

- simplicidade de projeto: quanto mais simples o projeto, mais facilmente adaptações poderão ser realizadas;
- descentralização: quanto mais autônomos os módulos de uma aplicação, menor será a probabilidade de que uma mudança em um desses módulos acarretará mudanças em todos os demais.

2.2.4. Reusabilidade

Durante o ciclo de desenvolvimento de um novo produto, muitas vezes depara-se com os mesmos problemas que já foram anteriormente solucionados em outro projeto. A reusabilidade denota exatamente essa habilidade de um componente de software de ser reaproveitado parcial ou totalmente no desenvolvimento de novas aplicações, reduzindo tempos e custos de implementação.

De uma certa forma pode-se dizer que a reutilização de soluções já existentes também favorece a confiabilidade, ou seja, a "corretude" e a robustez de um sistema, já que com o passar do tempo e com a utilização repetida de tais soluções, diminuem-se as chances de que um problema de implementação não detectado possa ocorrer.

2.2.5. Compatibilidade

Freqüentemente, na realização de uma tarefa, programas (ou em menor escala, elementos ou módulos de software) têm de interagir uns com os outros. No entanto isso torna-se tanto mais difícil quanto mais representações diferentes para a mesma entidade são utilizadas nos diversos sistemas.

Suponha o exemplo de uma biblioteca de rotinas para matemática vetorial. As funções desta biblioteca poderiam esperar que o usuário entrasse uma matriz no sistema na forma de uma seqüência de "arrays", cada um correspondendo a uma linha da matriz. Se os dados disponíveis provenientes de um cálculo anterior realizado por outra aplicação estivessem representados de outra forma, digamos na forma de um único "array" (onde os elementos de cada linha estariam simplesmente concatenados um após o outro), não seria possível a utilização direta destes dados sem antes realizar a conversão necessária.

O exemplo citado acima é extremamente simples, mas de modo análogo problemas muito mais sérios podem ser causados pela incompatibilidade entre os dados provenientes de diferentes aplicações.

De uma forma geral, soluções para este problema podem ser obtidas pela definição de protocolos[2] padronizados para o acesso às entidades importantes manipuladas por um software. Esta é a idéia existente por trás dos tipos abstratos de dados e da orientação a objetos.

2.3. O Caminho da Orientação a Objetos

As cinco qualidades discutidas na seção anterior refletem algumas das maiores dificuldades encontradas no desenvolvimento de um software, já que freqüentemente os programas:

- não desempenham exatamente a tarefa para a qual foram programados;
- não se comportam bem mediante situações anormais;
- são difíceis de serem adaptados;
- necessitam de um tempo de desenvolvimento relativamente longo por não fazerem uso de soluções já utilizadas e aprovadas anteriormente;
- são incompatíveis entre si.

No intuito de se tentar solucionar alguns dos problemas expostos acima, mais especificamente "corretude" e robustez, deve ser realizado um desenvolvimento mais sistemático, baseado em especificações e requisitos de projeto precisos.

Por outro lado reusabilidade, extensibilidade e compatibilidade podem ser conseguidas através da utilização de técnicas que produzam projetos mais flexíveis e descentralizados. Além disso é importante que estes sejam constituídos de módulos coerentes e conectados através de interfaces bem definidas [MEYER 88]. No que se refere a essas necessidades, a orientação a objetos apresenta características que a tornam uma excelente alternativa, tendo por isso sido escolhida para o desenvolvimento deste projeto. Por essa razão ela será abordada em detalhes a seguir.

2.3.1. O que é Programação Orientada a Objetos ?

Programação orientada a objetos pode ser descrita informalmente como programação por simulação. A metáfora utilizada é baseada na "personificação" de objetos físicos ou conceituais de algum domínio do mundo real nos objetos do domínio do programa. Por exemplo, objetos podem ser os clientes de uma empresa, peças em uma fábrica, apartamentos a serem administrados por uma imobiliária e assim por diante. Nessa tentativa de personificação, os objetos do programa adquirem as mesmas características e capacidades dos seus equivalentes do mundo real.

O fato de se utilizar um mapeamento direto entre objetos do mundo real e os objetos da implementação faz com que o software resultante seja muito mais fácil de entender e usar.

Uma das diferenças fundamentais entre a programação procedural tradicional e a programação orientada a objetos é que esta última descreve um sistema em termos das entidades e não das funções envolvidas.

2.3.2. Tipos Abstratos de Dados

A abordagem orientada a objetos tem muito em comum com a noção de programação utilizando tipos abstratos de dados. Na verdade, a POO não só faz uso deste estilo de

[2] Protocolo é o conjunto de funções e procedimentos que operam sobre uma estrutura de dados, ou no caso da orientação a objetos, métodos que podem ser evocados para os objetos de uma dada classe.

programação, mas também o estende incorporando duas noções a serem abordadas nas próximas seções: polimorfismo e herança.

Tipos abstratos de dados (e conseqüentemente objetos) são baseados no ocultamento de informação. A idéia por trás desse princípio é que usuários de um objeto não necessitam ter acesso nem à sua representação interna, nem à implementação das suas operações. Pode-se então pensar em objetos como tendo duas visões: uma para usuários e clientes e outra para o desenvolvedor. Enquanto os usuários podem apenas alterar o estado deste objeto de forma indireta através das operações providas[3] por ele, só o desenvolvedor tem acesso à sua representação interna.

A maior vantagem de se utilizar a abordagem acima é que ela permite ao desenvolvedor modificar a implementação de um objeto de uma maneira transparente ao usuário, o que quer dizer que a mudança passará desapercebida a este último. Esta separação entre a interface com o usuário e a implementação de um objeto é de importância fundamental para a manutenção e reusabilidade de um software.

Pode-se dizer que a utilização de um tipo de dado sem o conhecimento da sua representação interna segue o mesmo modelo das linguagens tradicionais. Todas essas linguagens fornecem suporte para um conjunto de tipos básicos como os inteiros, reais, caracteres, "arrays", etc. Cada tipo suporta um conjunto de operações bem definido como as aritméticas para os inteiros ou as lógicas para os "booleanos". Os usuários do tipo inteiro por exemplo não precisam saber se este é implementado utilizando a representação sinal-magnitude ou complemento de dois.

2.3.3. Objetos Encapsulam Estados e Operações

Um objeto é caracterizado pelo seu estado e pelas operações que podem ser realizadas nesse estado. De forma geral, objetos têm componentes e por isso o estado de um objeto é definido pelo estado dos seus componentes.

Por exemplo, em um banco tem-se clientes e contas. Os clientes são caracterizados pelos seus dados pessoais, como nome, R.G., endereço, telefone, etc. e por uma (ou mais) contas. Uma conta pode ser constituída por um número de identificação, um tipo e um saldo (Fig. 2.1).

Cada tipo de objeto suporta um conjunto de operações que podem ser aplicadas a ele para consultar ou modificar o seu estado. Por exemplo, poder-se-ia perguntar a um cliente do banco sobre os seus dados pessoais, do mesmo modo que esses dados também poderiam ser alterados em uma eventual mudança de endereço ou telefone.

Fig. 2.1 Objetos são caracterizados pelo seu estado

Conceitualmente pode-se dizer que objetos como o cliente e a conta citados como exemplos encapsulam tanto estados (ou atributos) como operações (Fig. 2.2.).

[3] No caso da linguagem Java o usuário pode acessar a representação interna de um objeto dependendo das "permissões" dadas pelo desenvolvedor, atribuídas através dos modificadores public, protected e private [MEYER 97].

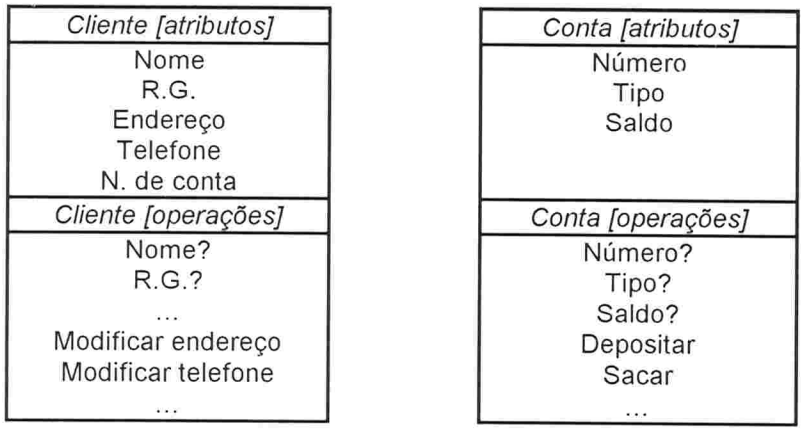

Fig. 2.2 Visão conceitual de um cliente e uma conta bancária

2.3.4. Objetos comunicam-se através de Passagem de Mensagens

Um sistema orientado a objetos pode ser descrito como um conjunto de objetos comunicando-se entre si para realizar alguma operação [LALONDE 90]. Essa comunicação dá-se através de um mecanismo chamado passagem de mensagens.

Cada vez que um objeto recebe uma mensagem, ele verifica se há alguma operação que possa ser executada como resposta à mensagem recebida. A definição de um método descreve como o objeto reagirá cada vez que receber a respectiva mensagem. Utilizando-se a terminologia adotada em orientação a objetos, refere-se ao conjunto de operações que definem o comportamento de um objeto como sendo o protocolo suportado por ele.

De forma mais geral, uma mensagem pode ser dividida em seletor, que identifica a operação requisitada do objeto e argumentos (ou parâmetros).

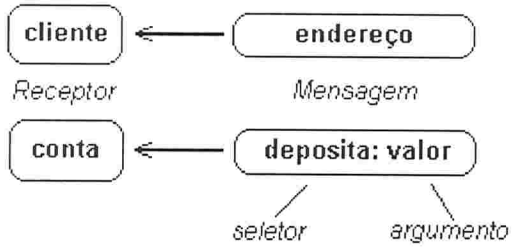

Fig. 2.3 Componentes de uma mensagem

Pode-se dizer que a passagem de mensagens em uma linguagem orientada a objetos (OO) é análoga à chamada de procedimentos e funções em uma linguagem procedural. Ou seja, quando da evocação de um método passa-se os argumentos necessários (se houver algum) e dependendo da implementação do método tem-se ou não um valor de retorno.

Em contextos como computação distribuída a passagem de mensagens freqüentemente implica concorrência. Entretanto na programação OO este geralmente não é o caso, já que a passagem de mensagens ocorre de forma síncrona, isto é, um mecanismo padrão "call/return" é usado.

Diferentes mecanismos são providos para a realização de tarefas de forma concorrente dependendo da linguagem. Por exemplo no caso de Java, as "threads"[4] e no caso de Smalltalk os

4 Java provê suporte para execução multitarefa, sendo que a cada tarefa corresponde uma "Thread".

processos[5]. Estes mecanismos não serão abordados aqui por não fazerem parte do escopo deste trabalho, mas como referências tem-se respectivamente [RITCHEY 95] e [ADELE 89].

2.3.5. POO baseia-se em Classificação

Através da classificação o ser humano é capaz de associar todos os membros de uma classe de objetos segundo as suas características. Quando se fala em POO, esta noção continua válida. Assim como alumínio, ferro e prata são classificados como metais por possuírem as propriedades comuns a estes, em POO uma *classe* é uma abstração que captura os atributos e operações comuns a um tipo de objetos.

Uma classe descreve a representação e o protocolo de mensagens "compreendidas" por cada um de seus membros, ou na terminologia POO, por cada uma de suas instâncias (Fig. 2.4.).

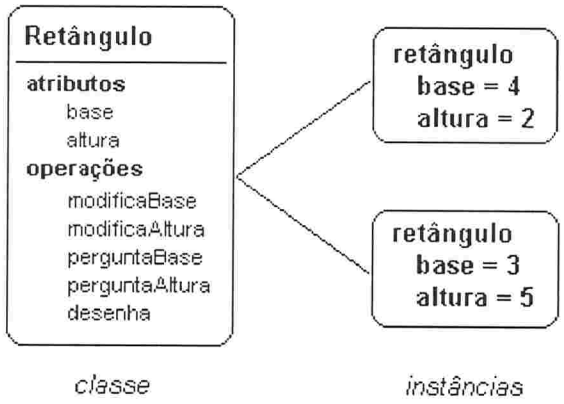

Fig. 2.4 Classe e instâncias

2.3.6. POO utiliza Polimorfismo

Uma das principais características da POO é que a interpretação de uma mensagem depende do objeto que a recebe, isto é, a mesma mensagem pode ser interpretada de diferentes formas por diferentes receptores. A esta característica dá-se o nome de polimorfismo. Como exemplo, consideremos a seguinte expressão:

umObjeto **at:** 1 **put:** 'primeiro'

Não há meios de se saber o efeito que este código terá ate que o objeto *umObjeto* seja conhecido. Se *umObjeto* for um array, na primeira posição desse array será colocada o string *primeiro*. Por outro lado, se *umObjeto* for um dicionário, a esse dicionário será adicionada uma nova associação cuja chave será *1* e cujo valor será *'primeiro'*. No caso de já existir uma associação cuja chave é *1*, o valor antigo associado a ela simplesmente será' substituído pelo valor *'primeiro'*.

Para ilustrar os benefícios trazidos pelo polimorfismo, considere como exemplo um editor gráfico onde o usuário pode desenhar figuras geométricas na tela.

Para se abstrair a entidade figura geométrica em uma linguagem procedural como o Pascal por exemplo, poder-se-ia definir um *record* com um campo destinado a identificar o tipo da

5 Em Smalltalk cada tarefa é representada por um processo, cuja criação se dá através do envio de uma mensagem específica a uma instância da classe "Block".

figura. Na implementação da operação *desenha*, para se ter um projeto modular poderia ser implementado um procedimento específico para cada tipo de figura. Neste caso, o procedimento *desenha* teria que fazer uso de uma lógica do tipo *case* (Fig. 2.5) para determinar o tipo de figura envolvida, e conseqüentemente qual o procedimento a ser chamado para desenhá-la. Como a associação entre cada procedimento especifico e o tipo de parâmetro envolvido é conhecida em tempo de compilação, diz-se que o acoplamento entre ambos é estático.

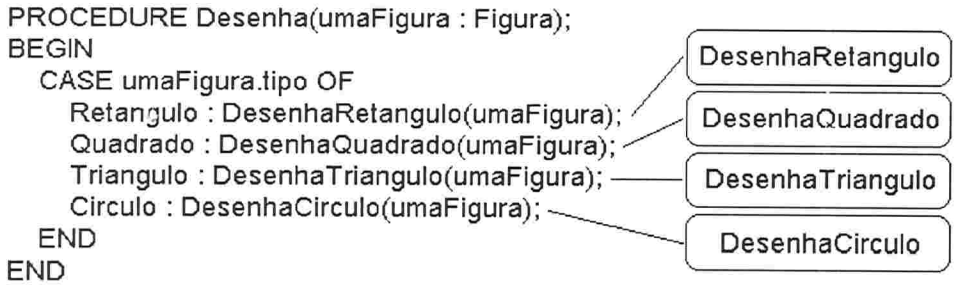

Fig. 2.5 Acoplamento estático

No caso de uma linguagem OO também se deve implementar um método *desenha* para cada figura, porém o mesmo nome é usado em todos os casos. Como conseqüência, o programador não mais precisa determinar qual o método correto a ser invocado para cada figura. Em vez disso, ele pode agora simplesmente enviar a mensagem *desenha* para qualquer figura e o método correto será localizado e executado pelo sistema. Nota-se que neste caso a relação entre a operação desenha e o objeto receptor é determinada em tempo de execução e não mais em tempo de compilação. Este acoplamento é conhecido como acoplamento dinâmico.

Esta solução polimórfica é claramente mais reusável e adaptável a mudanças. Se uma nova figura geométrica fosse criada, ela poderia em ambos os casos ser provida com todas as operações suportadas pelas demais figuras. Entretanto na solução tradicional (não OO), todas as operações suportadas (do mesmo modo que *desenha*) também sofreriam alterações para incluir os novos casos pois todas seriam baseadas em *case*. Em um sistema de maior porte este tipo de atividade é extremamente suscetível a erros pois é bem provável que ao menos uma das mudanças necessárias deixasse de ser realizada.

Em um sistema OO as alterações necessárias são localizadas. Simplesmente se adicionaria as operações necessárias à figura recém criada sem que as classes já implementadas sofressem qualquer alteração.

2.3.7. POO utiliza Herança

Com freqüência pode-se pensar em objetos como sendo especializações de outros objetos. Por exemplo, um pastor alemão, um dálmata e um vira-lata são especializações de cães, assim como estes são especializações de mamíferos e assim por diante. Aplicando esta noção à programação, pode-se ver uma classe de objetos como subclasse de outra (Fig. 2.6).

Dizer que uma classe é subclasse de uma outra significa dizer que ela apresenta (herda) todas as características da classe mais geral e além disso a estende, seja através da adição de novas operações e/ou novos atributos ou pela redefinição das operações herdadas.

Entre as vantagens proporcionadas pela presença de herança nas linguagens orientadas a objeto podem-se destacar:

- redução de tempos e custos de desenvolvimento dada a reusabilidade do código de uma classe pelas suas subclasses de forma automática;
- maior uniformidade no código produzido no que se refere aos nomes das operações, o que vem a facilitar o aprendizado de uma nova biblioteca de classes pelo usuário bem como a sua manutenção pelo programador.

Fig.2.6 Hierarquia de cães

2.4. Conclusão

Do ponto de vista de um engenheiro, toda e qualquer operação a ser realizada terá o seu custo, que em se falando de software estará associado ao seu desenvolvimento[6] ou manutenção. Na tentativa de se reduzir estes custos faz-se necessário um esforço na busca de um aumento na qualidade dos softwares produzidos.

Baseada em princípios simples e de fácil entendimento, a orientação a objetos vem se inserir neste contexto como uma ferramenta eficaz na busca desse objetivo, possibilitando a produção de soluções mais flexíveis, descentralizadas, de maior facilidade de manutenção e reutilização e por conseguinte mais econômicas.

[6] Subentende-se aqui todas as etapas envolvidas no processo, desde projeto e implementação até testes e depuração.

3. A Escolha da Linguagem

3.1. Introdução

Tendo-se definido que a tecnologia da orientação a objetos seria utilizada no projeto, restava ainda a escolha da linguagem a ser adotada na sua implementação.

As linguagens utilizadas pela empresa eram todas interpretadas. Por isso softwares implementados utilizando-se qualquer uma delas teriam que ser distribuídos com os respectivos interpretadores. Entretanto algumas aplicações dos componentes a serem desenvolvidos podem ser bastante restritivas em termos da quantidade de memória disponível, como é o caso de comandos numéricos embutidos.

Tendo-se esta preocupação em mente foi realizado um estudo comparativo das linguagens de programação disponíveis na empresa, visando levantar a quantidade de memória ocupada por cada uma delas, possibilitando assim a escolha daquela que dentre todas seria a menor.

Cabe aqui a introdução do conceito de máquina virtual, necessário para o melhor entendimento do problema.

3.2. Máquinas Virtuais

Uma máquina virtual (VM) nada mais é do que um computador implementado em software [LIU 96]. Do mesmo modo em que na programação procedural agrupa-se operações simples com o objetivo de se produzir operações mais específicas, uma máquina virtual define as suas instruções que podem ser de certa forma complexas em termos de instruções mais simples, implementadas por outras máquinas hierarquicamente inferiores (seja ela uma máquina real ou virtual).

Um bom exemplo é a máquina virtual Java. Quando se compila um arquivo fonte em Java, é gerado um arquivo composto de "bytecodes". Do mesmo modo que uma máquina como o PC executa instruções na forma de código de máquina, a máquina virtual Java "entende" e executa os "bytecodes", que nada mais são do que as suas instruções, implementadas utilizando as instruções das máquinas sob as quais a VM Java estiver rodando (Fig. 3.1).

O fato de o código compilado não fazer referência explícita aos recursos de qualquer máquina "real" em particular é imprescindível à medida que se procura a produção de softwares mais portáveis, pois um mesmo recurso existente em uma arquitetura pode não estar disponível em outra.

Ao se escrever um programa para uma máquina virtual X, fica garantido que após compilado este poderá funcionar imediatamente em tantas máquinas reais quantas implementarem a máquina virtual X. Em contrapartida, a utilização de máquinas virtuais também pode trazer alguns inconvenientes.

Pelo fato de as linguagens baseadas em VM serem interpretadas, a máquina virtual deve ser transportada juntamente com a aplicação final em tempo de execução, ocasionando um gasto extra em memória e algumas vezes diminuindo a performance do sistema.

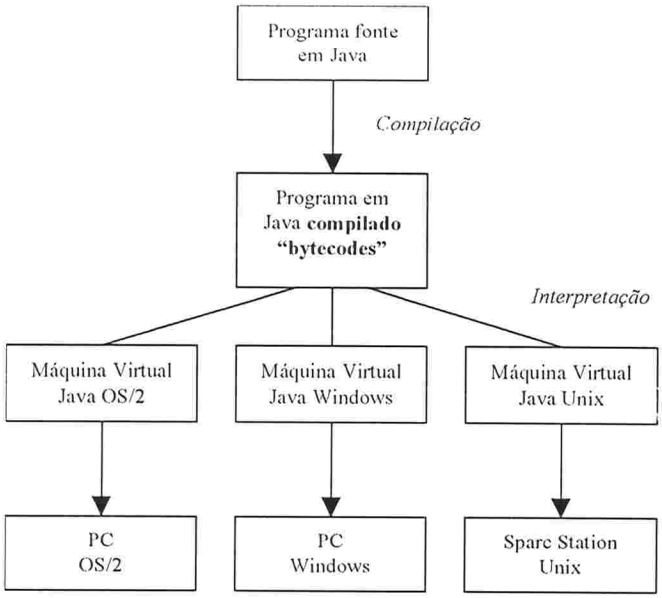

Fig. 3.1 Programa rodando sobre a máquina virtual Java em diversas plataformas

Todavia isto somente se constituirá em um problema se espaço em memória e velocidade forem restrições de projeto, ou seja, tudo depende da aplicação em questão, não sendo possível estabelecer um argumento a favor ou contra a utilização de VMs sem que se leve em consideração cada problema específico.

Como exemplos de linguagens baseadas em VMs pode-se citar Java, Smalltalk e Forth, entre outras.

3.3. As Linguagens Disponíveis

Como linguagens disponíveis na empresa tinha-se Java e três versões de Smalltalk de fabricantes diferentes (ENVY/Smalltalk, Squeak/Smalltalk [SQUEAK] e DTO/Degas [DEGAS]). Todas estas linguagens são interpretadas e foram construídas utilizando o conceito de máquinas virtuais. Isto implica que uma aplicação desenvolvida em qualquer uma delas vai necessariamente carregar consigo a respectiva máquina virtual em tempo de execução. Pelas razões já citadas anteriormente fez-se necessário realizar um estudo para se encontrar dentre estas linguagens qual teria a menor máquina virtual.

Antes de se iniciar esta análise propriamente dita, as classes de cada VM foram separadas em categorias para que se pudesse mais facilmente levantar as deficiências de cada uma. Entende-se aqui por deficiência, a falta de recursos (classes) em uma VM que estão presentes nas demais. Ou seja, de nada adianta se ter uma VM compacta mas com recursos muito limitados.

Para que se tenha uma melhor idéia dos critérios adotados na classificação, na tabela 3.1 encontram as categorias utilizadas, bem como alguns exemplos de classes pertencentes a elas.

É importante salientar que na análise de espaço ocupado não foram consideradas todas as classes das linguagens analisadas, mas apenas o menor subconjunto delas que pudesse produzir uma máquina virtual operacional.

Em ENVY/Smalltalk as classes são agrupadas em aplicações e a única aplicação realmente necessária para se ter uma VM funcional é a chamada *Kernel*.

Tipos básicos	Object, Boolean, Association, Number, Character, Date, Time, etc.
Coleções e "Streams"	Collection, Array, String, Dictionary, Set, ReadStream, WriteStream, etc
Implementação da linguagem	Block, Class, Compiler, Message, Method, Exception, etc.
Processos	Process, Semaphore, Delay, ProcessScheduler, etc.
Arquivos	FileDescriptor, FileStream, FileReader, FileWriter, etc.
Gráficos (primitivas)	Pen, Point, Rectangle
Outras	Todas as classes que não se encaixaram em nenhuma das categorias anteriores e que dependendo da linguagem utilizada, poderiam ser eliminadas da aplicação final

Tabela 3.1 *Categorias de objetos encontradas no "kernel" das linguagens analisadas*

Tanto Squeak/Smalltalk quanto DTO/Degas não têm a noção de aplicações ou pacotes de classes, por isso todas as suas classes foram analisadas. As que não encontraram equivalentes nas outras linguagens foram agrupadas na categoria *Outras*. DTO/Degas não teve nenhuma classe incluída nesta categoria por ter sido projetada visando a aplicação em sistemas embutidos, tendo por isso um número bastante reduzido de classes, todas recaindo em uma das categorias criadas.

	ENVY/ST Kernel	Squeak/ST	Degas VM	Java java.lang java.lang.reflect java.io java.util
Tipos básicos	16 classes 483 métodos 143K	19 classes 546 métodos 37K	15 classes 310 métodos 59K	13 classes 253 métodos 182K
Coleções e "Streams"	52 classes 1189 métodos 192K	32 classes 639 métodos 52K	5 classes 40 métodos 11K (não há streams)	59 classes 753 métodos 133K
Implementação da linguagem	31 classes 750 métodos 228K	44 classes 1571 métodos 338K	26 classes 722 métodos 182K	74 classes 373 métodos 52K
Processos	12 classes 307 métodos 36K	6 classes 94 métodos 7K	3 classes 30 métodos 7K	6 classes 115 métodos 18K
Arquivos	21 classes 341 métodos 48K	9 classes 151 métodos 12K	1 classe 24 métodos 5K	9 classes 126 métodos 16K
Gráficos (primitivas)	2 classes 98 métodos 11K	4 classes 185 métodos 12K	2 classes 58 métodos 9K	3 classes 46 métodos 5K
Outras	38 classes 501 métodos 78K	276 classes 4579 métodos 460K		15 classes 242 métodos 45K
Total	174 classes 3669 métodos 734K	390 classes 7779 métodos 923K	52 classes 1325 métodos 297K	176 classes 1862 métodos 445K

Tabela 3.2 *Tamanho das classes analisadas por categoria*

Por fim, Java também apresenta a noção de pacotes de classes, assim não foi necessário analisar todas as suas classes, mas somente os pacotes[7] java.lang, java.lang.reflect, java.io e java.util, que são os que possuem as classes ligadas às categorias criadas.

Na tabela 3.2 tem-se uma tabela que resume os resultados obtidos, desconsiderando a categoria *Outras* por esta conter somente as classes que podem ser excluídas sem prejuízo para a máquina virtual.

Como pode-se acompanhar pela Tabela 3.3, Degas tem o menor Kernel, sendo portanto a VM escolhida para ser utilizada no desenvolvimento do projeto.

Máquina Virtual	Espaço (*Kernel*)
DTO/Degas Smalltalk	**297K**
Java	400K
Squeak/Smalltalk	463K
ENVY/Smalltalk	656K

Tabela 3.3. *Tamanho das bibliotecas analisadas (excluindo a categoria Outras)*

O fato da VM Degas não apresentar "streams" não caracteriza um problema sério pelo fato de que uma classe "stream" simplificada poderia facilmente ser implementada, não implicando com isso um gasto de memória considerável que pudesse anular a sua vantagem sobre a segunda colocada (Java).

3.4. Degas VM

A máquina virtual Degas (DVM) possui um interpretador Smalltalk especialmente projetado para o desenvolvimento de aplicações embutidas, tendo como alvo plataformas de pequeno porte. Por essa razão esta VM foi concebida oferecendo um conjunto reduzido de classes, e por conseguinte aceitando um subconjunto do Smalltalk padrão, o que vai impor um certo número de restrições ao desenvolvedor:

Inexistência de literais ByteArray

O compilador DVM não reconhece a sintaxe ('#[' {value} ']') utilizada na declaração de literais da classe ByteArray.

Inexistência de literais LongInteger

O compilador DVM gera somente literais SmallInteger. Se um valor não pode ser armazenado como um SmallInteger, ele é automaticamente convertido para um literal Float.

Inexistência de Atoms

O compilador DVM não reconhece a sintaxe ('##' {character}) utilizada na declaração de literais da classe Atom.

Inexistência de Metaclasses

Para economizar espaço, Degas não utiliza o modelo de classes padrão Behavior, Class e Metaclass. A única destas classes que é utilizada é Class. Conseqüentemente, o compilador DVM ignora variáveis de classe, variáveis de instância de classes e métodos de classe.

[7] Do inglês *packages*.

Inexistência de variáveis temporárias dentro de blocos

O compilador não permite a utilização de variáveis temporárias no interior de blocos, como no exemplo a seguir:

```
[ :a :b |   | temp |
   temp := a size.
... ]
```

Entretanto um bloco pode utilizar as variáveis temporárias do método que o contém.

Problemas no manuseio de expressões vazias

Nem sempre o compilador manuseia de forma correta expressões vazias como no seguinte exemplo:

```
a isNil ifFalse: [ a + 1. ]
```

Em vez disso, a expressão deve ser reescrita da seguinte forma:

```
a isNil ifFalse: [ a + 1 ]
```

O ponto final dentro do bloco faz com que o compilador suponha a existência de uma outra expressão a seguir dentro do bloco, o que não é correto.

Manuseio incorreto de mensagens em cascata

O compilador não gera código para expressões em cascata, ou seja, uma seqüência de mensagens enviada para o mesmo objeto, de forma adequada. Em vez de enviar a segunda mensagem para o primeiro objeto, o compilador envia a segunda mensagem bem como todas as subsequentes para o resultado da primeira expressão em cascata. Ou seja, código como o que segue,

```
objeto X   mensagem1;
           mensagem2;
           mensagem3.
```

seria interpretado como:

```
(objeto X mensagem1)  mensagem2;
                      mensagem3.
```

3.5. Conclusão

Toda a fase de desenvolvimento e testes para a biblioteca *Degas* pode ser realizada em ENVY/Smalltalk, devendo-se somente tomar o cuidado de respeitar as restrições acima. Somente após concluído todo o desenvolvimento é que o compilador *Degas* é utilizado para a geração do produto final, onde as classes do ENVY/Smalltalk são substituídas pelas classes da VM Degas. Isto possibilita a utilização durante as fases de desenvolvimento e testes do projeto de todas as ferramentas disponíveis para o ENVY/Smalltalk, incluindo o ambiente ENVY/Developer, produzido pela própria OTI.

4. Conceitos Básicos de CNC

4.1. Introdução

O objetivo deste capítulo é fazer uma abordagem sucinta dos principais conceitos relacionados com o comando numérico, enfatizando o comando propriamente dito (CNC) e sua programação independentemente do processo sendo controlado.

Desta forma procura-se identificar as entidades envolvidas no sistema, permitindo a sua modelagem na forma de objetos, além de prover subsídios para a implementação do primeiro destes componentes: o interpretador.

4.2. Esquema Geral de um CNC

O esquema de um CNC genérico é mostrado na figura 4.1. Tem-se como entradas os parâmetros da máquina e os programas-peça. Os parâmetros da máquina informam a respeito das caraterísticas particulares da máquina ligada ao CNC. O programa-peça contém as informações geométricas e tecnológicas da peça que se deseja fabricar.

Os dados de entrada podem ser introduzidos manualmente através do painel do CNC (MDI - manual data input) ou através de alguma interface padrão como a RS232 por exemplo. A esta interface é normalmente conectada uma leitora de fita perfurada ou disco flexível, ou ainda um computador, em cujo caso se tem o DNC (Direct Numerical Control).

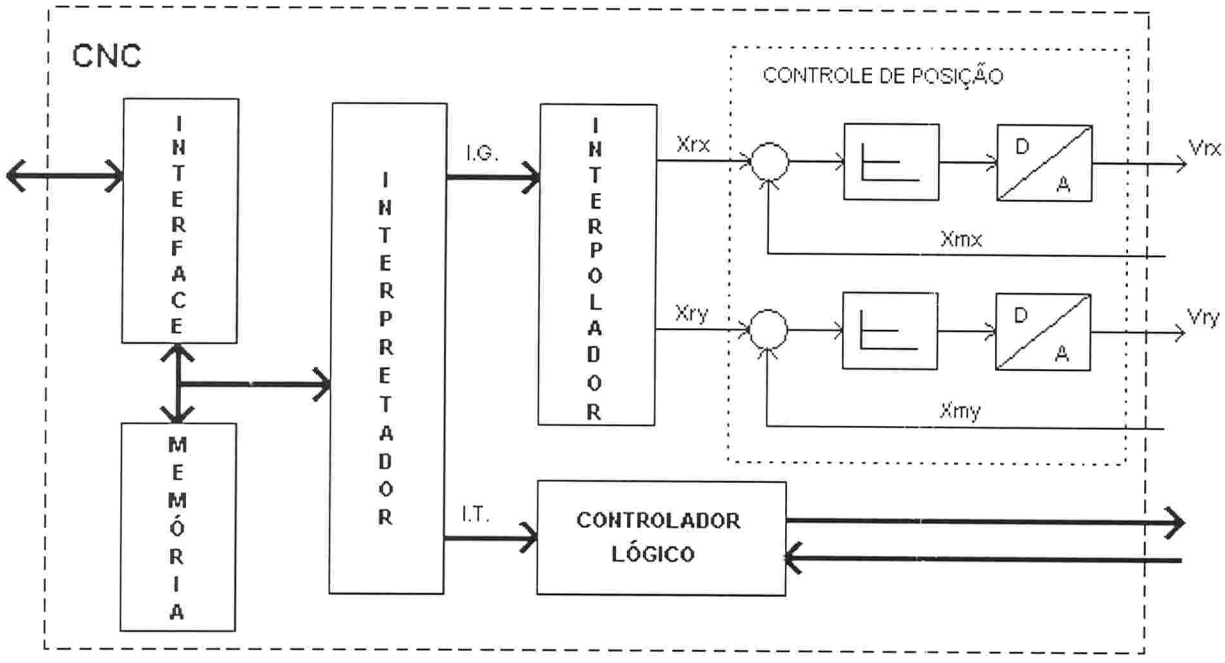

Fig. 4.1 Esquema geral de um CNC de dois eixos

Como dito anteriormente, durante a execução do programa-peça, este é decodificado em dois tipos de informação. As informações geométricas são processadas pelo interpolador, que envia aos controladores de posição o sinal de referência para cada eixo da máquina. As informações tecnológicas por sua vez são processadas pelo controlador lógico, que gera como saídas os sinais de controle para atuadores, tendo como sinais de entrada os sinais provenientes de

sensores, que indicam o estado das variáveis do processo, confirmação da execução de comandos enviados à máquina, etc.

4.2.1. Posicionadores

Na figura 4.2. tem-se o esquema de um posicionador genérico, correspondendo a um eixo. Para uma máquina com mais de um eixo, o mesmo acionamento mostrado será repetido tantas vezes quantos forem os eixos da máquina.

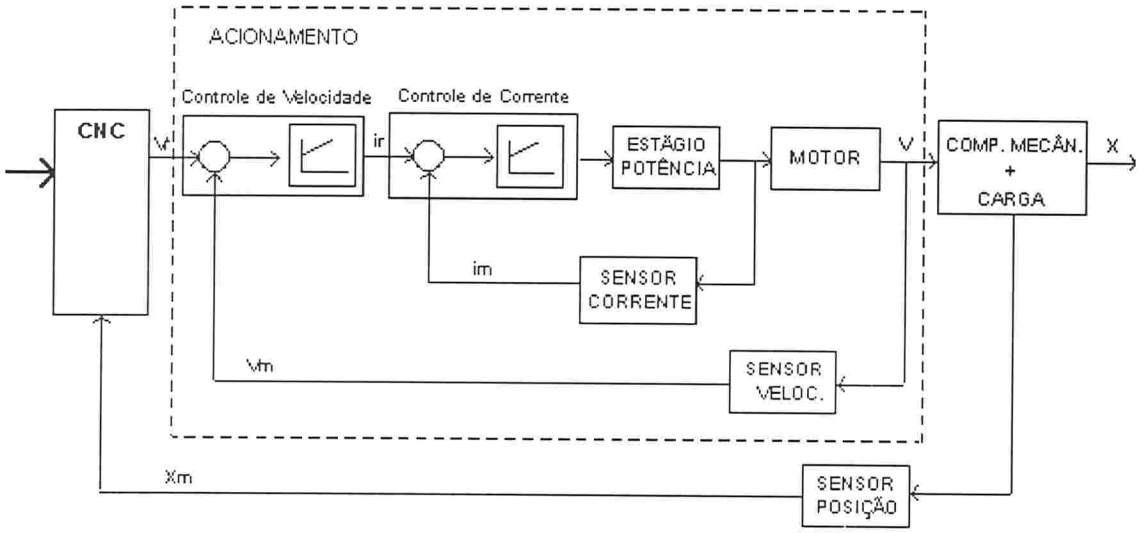

Fig. 4.2 Esquema geral de um posicionador de um eixo

4.3. Programação CNC

4.3.1. Conceitos Básicos

4.3.1.1. Direções de Movimento

Em termos de CNC, cada direção de movimento corresponde a um eixo. Dependendo da aplicação à qual se destina, as máquinas CNC têm comumente de dois a cinco eixos.

Um eixo em uma máquina CNC pode ser linear ou rotacional, sendo que em ambos os casos os principais componentes de cada eixo são um dispositivo mecânico que realiza o movimento, o motor que fornece a força para o eixo e o fuso de esferas recirculantes que transfere a força do motor para o dispositivo mecânico (Fig. 4.3).

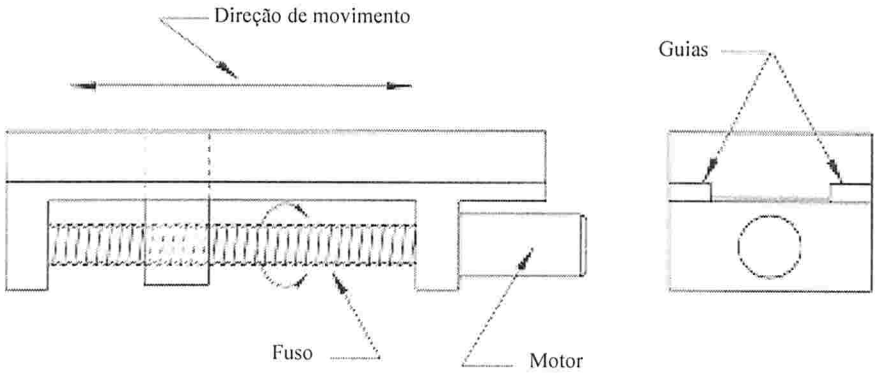

Fig. 4.3 Componentes de um eixo linear

Dependendo do tipo da máquina em questão, pode-se ter diferentes configurações de eixos, como pode-se ver nas figuras 4.4 e 4.5.

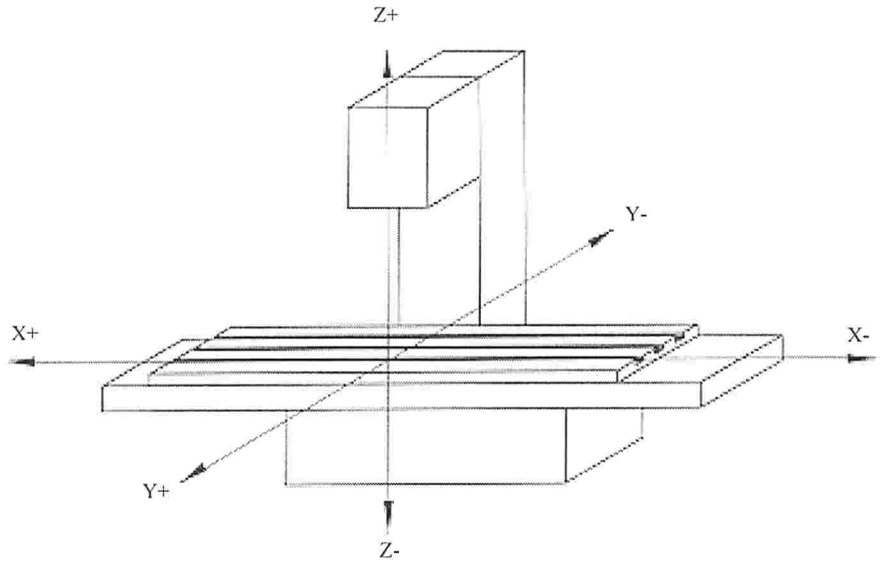

Fig. 4.4 Eixos de movimento de um centro de usinagem vertical

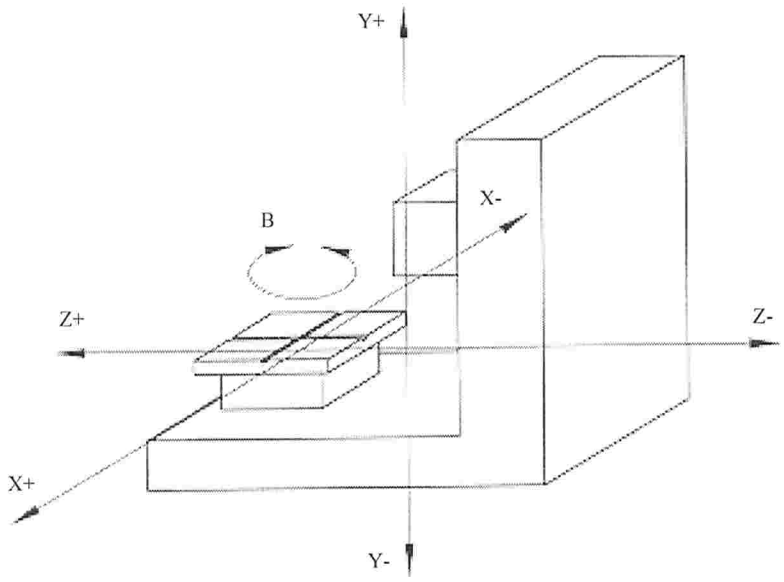

Fig. 4.5 Eixos de movimento de um centro de usinagem horizontal

4.3.1.2. O Ponto de Referência para Cada Eixo

Na maior parte dos equipamentos CNC cada eixo de movimento, seja ele linear ou rotacional, terá um ponto de referência. Este ponto de referência é uma posição muito específica ao longo do curso do eixo, cujo propósito é prover um ponto de partida conhecido para cada eixo.

Na verdade a posição exata deste ponto não necessita ser conhecida pelo programador da máquina CNC, pois este pode especificar o seu próprio ponto de referência (ou zero-programa) durante a programação. Somente o comando faz uso deste ponto para estabelecer os "offsets" entre o zero do programador e o zero real durante a usinagem.

4.3.1.3. Eixos Lineares e Rotacionais

Em um eixo linear o movimento ocorre ao longo de uma trajetória retilínea. O dispositivo mecânico realizando o movimento é normalmente muito similar àquele de uma fresadora manual. O movimento linear ocorre ao longo das guias do eixo à medida que o fuso é rotacionado pelo motor (Fig. 4.3).

O mesmo princípio básico se aplica a eixos rotacionais. A maior diferença está no modo como o dispositivo mecânico responsável pelo movimento realiza a sua tarefa. Em vez de ser linear, o movimento ocorre de forma circular. Um eixo rotacional pode ser empregado de duas formas. Algumas máquinas CNC requerem que a peça seja rotacionada durante a usinagem. Este tipo de eixo rotacional é utilizado para simplesmente mudar a orientação da peça entre duas operações ou para rotacionar a peça durante a usinagem.

Outras máquinas requerem que a ferramenta percorra uma trajetória circular durante a usinagem. Por exemplo, é muito comum na indústria aeronáutica o fresamento de superfícies não planas onde se deseja manter a fresa tão perpendicular quanto possível da superfície.

4.3.1.4. Os Modos Absoluto e Incremental

Há duas maneiras de se especificar como um movimento deve ser produzido. Um deles, o modo absoluto, é relativo à posição zero-programa. O outro, relativo à posição atual da máquina, é conhecido como incremental. A figura 4.6 ilustra a diferença entre ambos os modos, bem como a possibilidade de se realizar uma série de movimentos em qualquer um deles de forma alternada.

No.	Modo absoluto	Modo incremental
1	X1.0 Y1.0	X0.0 Y0.0
2	X2.0 Y1.0	X1.0 Y0.0
3	X2.5 Y0.5	X0.5 Y-0.5
4	X3.5 Y0.5	X1.0 Y0.0
5	X3.5 Y2.5	X0.0 Y2.0
6	X5.0 Y2.5	X1.5 Y0.0
7	X5.0 Y2.0	X0.0 Y-0.5
8	X6.0 Y2.0	X1.0 Y0.0
9	X7.0 Y3.5	X1.0 Y1.5

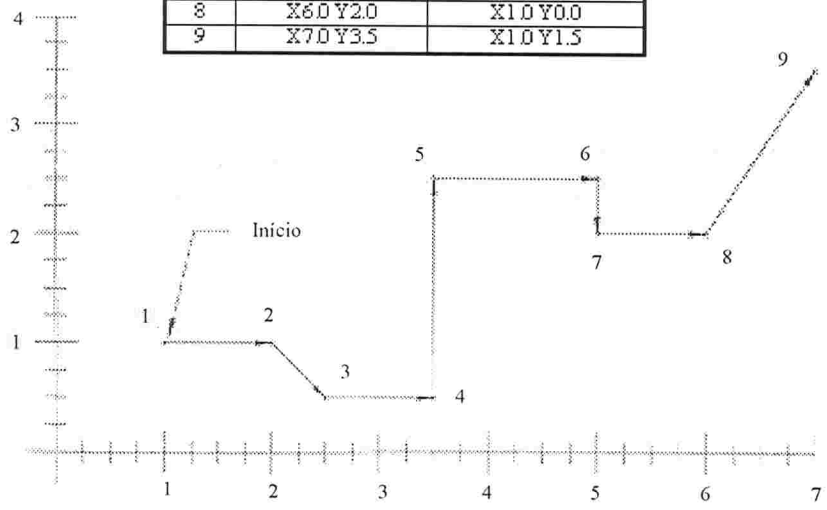

Fig. 4.6 Diferença entre os modos absoluto e incremental

4.3.1.5. Interpolação

Considerando-se que a maioria das máquinas CNC tem mais de um eixo de movimento, com freqüência o programador terá de especificar o movimento simultâneo de dois ou mais eixos. Por exemplo, suponha um centro de usinagem realizando um fresamento na periferia da peça. Este contorno pode envolver a usinagem de superfícies planas e arredondadas. Enquanto que alguns dos movimentos deste exemplo podem envolver somente um eixo, os movimentos circulares envolverão no mínimo dois eixos.

Neste contexto assim como na Matemática, interpolar significa estimar o valor de uma função num determinado ponto, tomando uma média ponderada de valores conhecidos da função na vizinhança desse ponto. Quando o comando realiza a interpolação em um movimento, ele está na verdade estimando de forma precisa o caminho programado baseando-se em uma pequena quantidade de informação.

Por exemplo, quando o comando realiza um movimento em linha reta em dois eixos (interpolação linear), toda a informação necessária do programador CNC são somente os pontos inicial e final do movimento. O comando gera todos os pontos intermediários de forma automática.

O que acontece na realidade é que o comando comanda uma série de movimentos infinitesimais alternadamente nos dois eixos, de forma que a trajetória produzida se suficientemente ampliada tem forma de escada (Fig. 4.7). A interpolação circular ocorre segundo o mesmo princípio (Fig. 4.8).

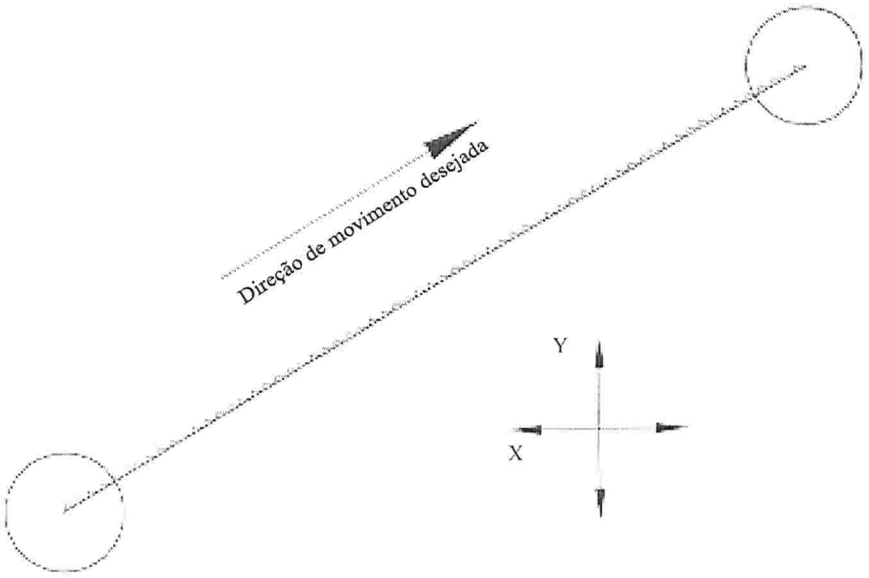

Fig. 4.7 Trajetória produzida durante a interpolação linear

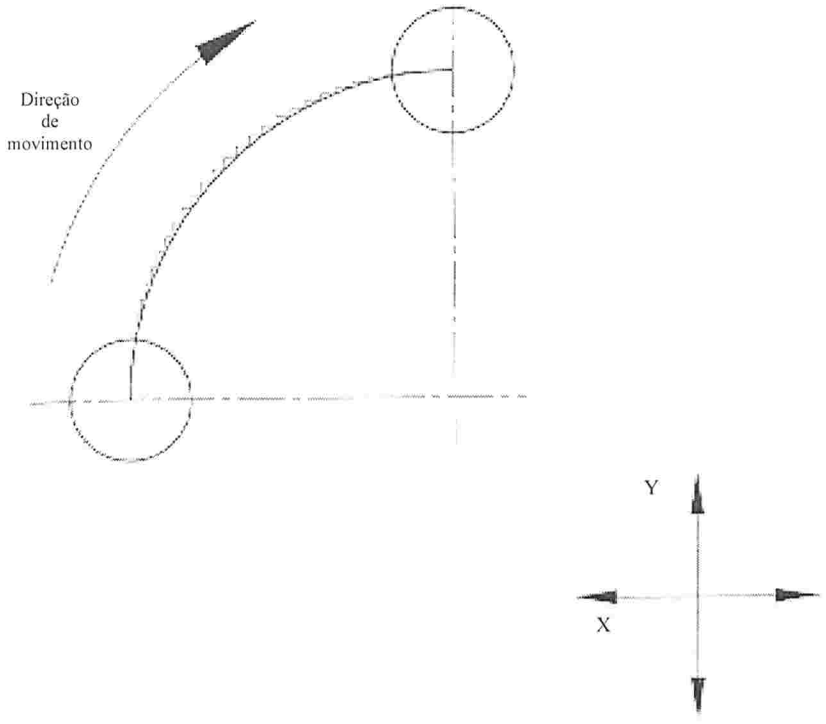

Fig. 4.8 Trajetória produzida durante a interpolação circular

4.3.1.6. Compensação

Genericamente falando, compensação é um mecanismo utilizado na programação de máquinas CNC para tornar o programa independente das ferramentas a serem utilizadas na usinagem, pelo menos no que se refere às suas dimensões quando da especificação dos movimentos a serem realizados.

Existem várias formas de compensação, mas todas elas em todos os tipos de comandos CNC trabalham com "offsets", que nada mais são do que posições de uma tabela localizada na memória do comando que serve para armazenar os valores numéricos a serem compensados.

Atualmente os comandos CNC permitem múltiplos "offsets". A maioria dos quais tem um número padrão deles que pode ir de 16 até 999, dependendo da aplicação da máquina. O operador tem controle total da tabela de "offsets" e pode realizar quaisquer alterações nela sempre que o desejar.

Alguns comandos podem ter um único "offset" referenciando vários valores. Por exemplo, cada "offset" de um torno CNC normalmente inclui pelo menos dois valores correspondentes respectivamente a um valor em X e outro em Z. Ambos são referenciados em um programa pelo mesmo número (tabela 4.1).

N. do "Offset"	X	Z
1		
2		
3		
...		
999		

Tabela 4.1 *Tabela de "offsets" utilizada pela maioria dos tornos CNC*

A seguir tem-se uma lista com os tipos de compensação utilizados pelas máquinas CNC:

Tipo de compensação	Máquina relacionada
Comprimento da ferramenta	Centro de usinagem
Diferença do raio da ferramenta	Centro de usinagem
"Offsets" de fixação	Centro de usinagem
Posicionamento da ferramenta	Centro de torneamento
Tool nose radius compensation	Centro de torneamento
Wire radius compensation	*Wire EDM machines*
Wire taper compensation	*Wire EDM machines*

Tabela 4.2 *Tipos de compensação existentes*

Um centro de usinagem pode realizar diversas operações, entre elas, fresamento, furação, rosqueamento com macho, alargamento e mandrilamento. Cada uma destas operações é efetuada por um tipo diferente de ferramenta, que terá um tamanho e uma configuração diferentes das demais. Alguns tipos de fresas são curtas para assegurar rigidez. Por outro lado, uma broca ou um alargador podem ser bem mais longos para poderem alcançar a superfície da peça a ser usinada.

Desde que o comprimento das ferramentas a serem utilizadas em um centro de usinagem pode ser tão variável, a distância que deve ser percorrida ao longo do eixo Z para que cada ferramenta alcance a mesma superfície da peça também será diferente. A figura 4.9 ilustra isto sobrepondo duas ferramentas no cabeçote de uma máquina CNC.

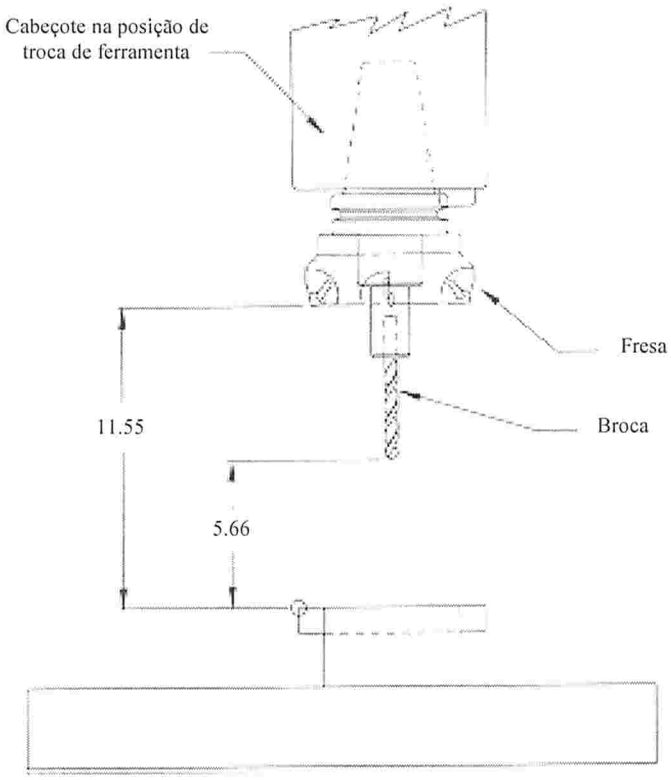

Fig. 4.9 Duas ferramentas diferentes utilizadas em um centro de usinagem

Não serão abordados aqui todos os tipos de compensação existentes. Para este propósito há uma descrição detalhada em [LYNCH 92]. O objetivo aqui é somente fornecer uma idéia geral dos conceitos envolvidos através de dois exemplos dos tipos de compensação mais utilizados: o de comprimento da ferramenta e o de raio da ferramenta.

Um programa utilizando compensação de comprimento da ferramenta comportar-se-á corretamente, não importando para isso quão longas sejam as ferramentas sendo utilizadas, desde que sejam respeitados os limites do eixo Z. Para cada ferramenta, o programador entra um comando informando que a compensação do comprimento da ferramenta deve ser utilizada. Este comando levará em consideração o comprimento desta (armazenado na tabela de "offsets"), subtraindo este valor de comprimento do valor programado para o movimento em Z. O comprimento do movimento resultante ficará reduzido então deste valor. Como os "offsets" das ferramentas são armazenados separadamente do programa, este torna-se independente dos comprimentos de cada uma delas.

A compensação do raio da ferramenta, por outro lado, permite que o programador "esqueça" sobre o raio (ou diâmetro) da ferramenta durante a programação. Este tipo de compensação não se aplica a todos os tipos de ferramentas, mas somente àquelas que têm a habilidade de usinar com a sua periferia e somente nesta situação. Alguns tipos de fresas apresentam esta habilidade. Brocas, alargadores e mandris, entre outras, não utilizam este tipo de compensação. A figura 4.10 mostra um exemplo onde o programador deseja usinar a periferia da peça.

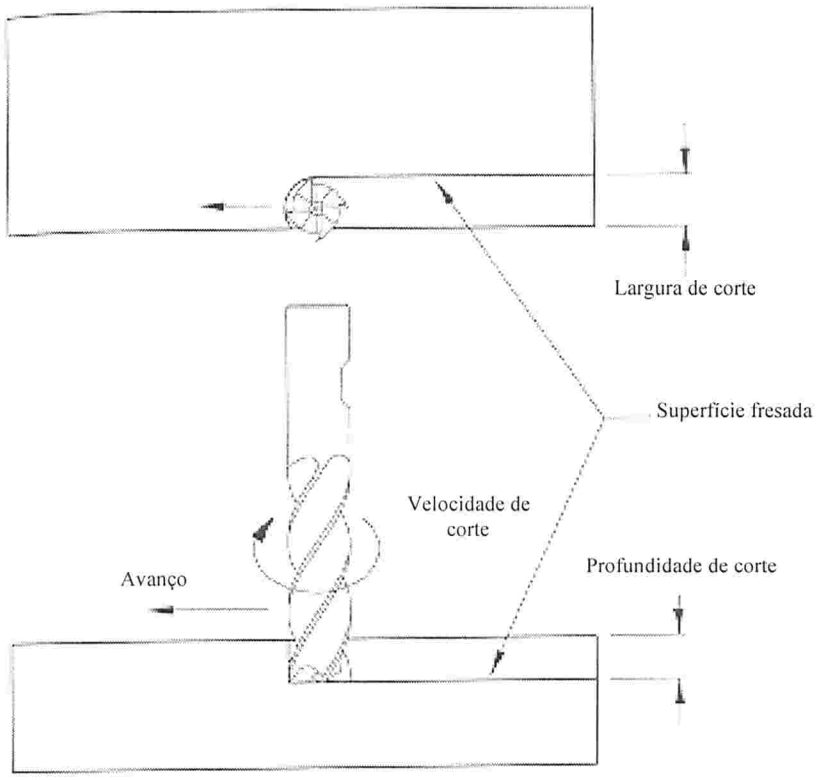

Fig. 4.10 Fresamento

Suponha que a fresa utilizada na usinagem acima tem uma polegada de diâmetro. Se o programador não utilizar a compensação do raio da ferramenta, ou seja, escrever o programa baseando-se no diâmetro uma polegada para manter o centro da ferramenta na trajetória adequada, isto pode vir a trazer alguns problemas. Se mais tarde, durante o "setup" da máquina se descobrir que a empresa não dispõe de brocas de 1", mas somente de brocas de 0,875 e de 1,25 polegadas de diâmetro, por exemplo, o programa fatalmente terá de sofrer alterações. No caso da utilização da compensação, a única coisa a ser feita para resolver o problema seria medir a ferramenta disponível e armazenar no "offset" correspondente ao número da ferramenta utilizada no programa o valor obtido.

4.3.2. A Linguagem de Programação Manual ("Código 'G'")

Cada máquina CNC tem funções programáveis próprias específicas. Por exemplo, um centro de usinagem CNC tem um dispositivo para troca automática de ferramentas (ATC - "automatic tool changer"). Um torno CNC deve permitir a programação da sua rotação de forma precisa. Uma prensa CNC deve possuir uma maneira que permita ao programador especificar que um furo seja feito. Estes são apenas alguns exemplos dos muitos recursos que podem ser controlados pelo programador nos vários tipos de máquinas CNC.

Através do programa-peça, o programador especifica ao comando o que deve ser feito e como deve ser feito. Quando um programa é executado, o CNC procura pelo primeiro bloco do programa e executa-o, passando em seguida para o próximo bloco e assim por diante. Apesar de a maioria dos CNCs permitirem a inclusão de um número de identificação em cada linha do programa, este na verdade não é utilizado já que as linhas de um programa são normalmente executadas na ordem em que aparecem.

Um bloco (ou linha) de programa é composto por palavras, que por sua vez são compostas por uma letra (ou endereço) e um valor numérico. A seguir tem-se como exemplo um bloco de programa composto de 4 palavras.

N0010 G90 X6.4 Y2.3 <EOB>

O caracter <EOB> é um caracter especial que deve ser incluído no final de cada comando. A sigla EOB significa final de bloco "end-of-block".

Alguns CNCs atuais permitem que as palavras de um bloco apareçam em qualquer ordem, sendo que nestes casos o CNC "ordena" as palavras e as executa na ordem adequada. Assim, o bloco

N0010 G01 X4.5 Y2.3 M03 <EOB>

seria interpretado da mesma forma que

N0010 M03 Y2.3 X4.5 G01 <EOB>

Entretanto, nos CNCs mais antigos, as palavras em um bloco devem obedecer a uma ordem pré-estabelecida pelo detalhe do formato (vide seção 4.3.2.1). As funções simbolizadas por cada letra serão abordadas em detalhe na seção 4.2.2.2.

4.3.2.1. O Detalhe do Formato

O *detalhe do formato* utilizado por uma máquina CNC é uma seqüência de caracteres que informa quais funções são suportadas, juntamente com o comprimento dos parâmetros numéricos requeridos por elas.

Cada endereço que aparece no detalhe do formato deve ser seguido pelo número de dígitos à esquerda do ponto decimal requerido pela respectiva função. Se necessário, um segundo dígito indicará o número de dígitos à direita do ponto decimal. Por exemplo, suponha uma máquina com o seguinte detalhe do formato:

N3.G2.X34.Y34.Z34.I34.J34.K34.F32.S4.T4.M2 <EOB>

Esta máquina suporta somente palavras que iniciam por: N, G, X, Y, Z, I, J, K, F, S, T ou M. Além disso, os parâmetros numéricos devem respeitar a seguinte convenção:

- Palavras iniciando por N podem conter até três dígitos numéricos, sem ponto decimal (N3);
- Palavras iniciando por G podem conter até dois dígitos numéricos, sem ponto decimal (G2);
- Palavras iniciando por X podem conter três dígitos numérico, seguido pelo ponto decimal e mais quatro dígitos numéricos à direita deste.

E assim por diante para as demais palavras.

Alguns CNCs fazem uso do detalhe do formato para converter parâmetros dependendo do modo que eles foram entrados.

Por exemplo, seja um CNC que realiza a conversão automática do formato fixo sem ponto decimal para o formato variável com ponto decimal. Em tal CNC o bloco **N0010 X3.453**, que especifica uma dimensão ao longo do eixo X (formato variável com ponto decimal) poderia ser entrado de uma outra forma: N0010 X3453 (formato fixo sem ponto decimal) e interpretado exatamente do mesmo modo. Como o CNC espera que o endereço X seja seguido por um número à esquerda e três à direita do ponto decimal, ele interpreta automaticamente 3453 como sendo

3.453. Esta conversão pode ou não ocorrer de forma automática, dependendo para isso do fabricante do CNC.

4.3.2.2. O Padrão EIA-274-D[8]

A modo como cada palavra em um programa é interpretada não é única. Comandos construídos por diferentes fabricantes podem interpretar a mesma palavra, por exemplo G13, como algo totalmente diferente. Neste exemplo, se o fabricante adotar a EIA-274-D a máquina selecionará um determinado eixo. Entretanto, de acordo com a ISO6983 a mesma palavra é utilizada para interpolação utilizando coordenadas polares no sentido anti-horário.

Por uma restrição da empresa o padrão EIA-274-D deveria ser adotado no projeto e por essa razão ele será aqui abordado.

A tabela 4.3 mostra uma tabela com os endereços utilizados pela EIA-274-D, juntamente com as suas respectivas funções. Deve-se notar que muitos dos recursos normalmente disponíveis nas máquinas CNC de hoje em dia não estão listados abaixo já que esta norma data de 1979.

Endereço	Função
A	Dimensão angular relativa ao eixo X
B	Dimensão angular relativa ao eixo Y
C	Dimensão angular relativa ao eixo Z
D	Dimensão angular relativa a um eixo especial ou Terceira velocidade de avanço ou Seleção de compensação de ferramenta[9]
E	Dimensão angular relativa a um eixo especial ou Segunda velocidade de avanço[9]
F	Velocidade de avanço
G	Função preparatória
H	Não utilizado permanentemente
I	Parâmetro de interpolação ou Passo de rosca paralelo a X em operações de rosqueamento
J	Parâmetro de interpolação ou Passo de rosca paralelo a Y em operações de rosqueamento
K	Parâmetro de interpolação ou Passo de rosca paralelo a Z em operações de rosqueamento
L	Não utilizado permanentemente
M	Função "miscelânea"
N	Número de seqüência
O	Número de seqüência (somente para cabeçote secundário)
P	Terceira dimensão de movimento em rápido ou Dimensão de movimento secundária paralela a X[9]
Q	Segunda dimensão de movimento em rápido ou Dimensão de movimento secundária paralela a Y[9]
R	Primeira dimensão de movimento em rápido ou Dimensão de movimento secundária paralela a Z[9] ou Raio para cálculo de velocidade superficial constante
S	Velocidade de corte
T	Função escolha de ferramenta
U	Dimensão de movimento secundária paralela a X[9]
V	Dimensão de movimento secundária paralela a Y[9]

[8] EIA-274-D ("Interchangeable Variable Block Data Format for Positioning, Contouring, and Contouring/Positioning Numerically Controlled Machines").
[9] Onde D, E, P, Q, R, U, V, e W não forem utilizados como indicado, eles podem ser utilizados de outra forma

W	Dimensão de movimento secundária paralela a Z[9]
X	Dimensão de movimento primária em X
Y	Dimensão de movimento primária em Y
Z	Dimensão de movimento primária em Z
%	Início de programa
+	Mais
-	Menos
/	Elimina bloco (deve ser o primeiro caracter no bloco)
(	Início de comentário
)	Fim de comentário
.	Ponto decimal
,	Vírgula
DEL	(*Delete*) Ignorado pelo comando
BS	(*Backspace*) Ignorado pelo comando
HT	(*Horizontal tab*) Ignorado pelo comando
NL	(*New line*) Utilizado como caracter de fim de bloco (EOB)
RT	(*Return*) Ignorado pelo comando
Espaço	Ignorado pelo comando

Tabela 4.3 *Endereços e suas funções de acordo com a EIA 274-D*

Alguns dos endereços acima podem assumir outros significados (não compatível com a EIA-274-D):

Endereço	Função
E	Passo de rosca em operação de rosqueamento
H	Seleção de compensação de ferramenta
L	Número de repetições em ciclos fixos
O	Identificador de programa
P	Período em milissegundos para o comando pausa (sem ponto decimal)
R	Raio em interpolação circular

Tabela 4.4 *Outros usos para alguns dos endereços definidos acima (não EIA-274-D)*

As tabelas 4.5 e 4.6 mostram uma listagem das funções preparatórias adotadas pela EIA-274-D. Estas funções têm como característica básica justamente "preparar" a máquina para o desempenho de um tipo de operação.

As funções preparatórias de um grupo são modais, isto é, continuam em efeito até que outra do mesmo grupo seja executada.

Palavra	Grupo	Função
G00	A	Posicionamento ponto a ponto (rápido)
G01		Interpolação linear
G02		Interpolação circular sentido horário (2D)
G03		Interpolação circular sentido anti-horário (2D)
G06		Interpolação parabólica
G33		Corte de rosca, passo constante
G34		Corte de rosca, passo crescente
G35		Corte de rosca, passo decrescente
G72		Interpolação circular sentido horário (3D)
G73		Interpolação circular sentido anti-horário (3D)
G13-G16	B	Seleção de eixo

G17	C	Seleção do plano XY
G18		Seleção do plano ZX
G19		Seleção do plano YZ
G40	D	Compensação/offset de ferramenta, Desliga
G41		Compensação/offset de ferramenta - À esquerda
G42		Compensação/offset de ferramenta - À direita
G43		Compensação do diâmetro da ferramenta – Superfícies côncavas
G44		Compensação do diâmetro da ferramenta – Superfícies convexas
G70	E	Programação em polegadas
G71		Programação em unidades métricas
G74	F	Interpolação circular multiquadrantes - Desliga
G75		Interpolação circular multiquadrantes - Liga
G80	G	Ciclo fixo - Desliga
G81		Ciclo fixo n. 1
G82		Ciclo fixo n. 2
G83		Ciclo fixo n. 3
G84		Ciclo fixo n. 4
G85		Ciclo fixo n. 5
G86		Ciclo fixo n. 6
G87		Ciclo fixo n. 7
G88		Ciclo fixo n. 8
G89		Ciclo fixo n. 9
G90	H	Entrada de dimensões em modo absoluto
G91		Entrada de dimensões em modo incremental
G93	I	*Inverse time feedrate (V/D)*
G94		Avanço em polegadas (milímetros) por minuto
G95		Avanço em polegadas (milímetros) por revolução
G96		Velocidade superficial constante em pés(metros) por minuto
G97		Revoluções por minuto

Tabela 4.5 *Funções preparatórias suportadas pela EIA-274-D*

As seguintes funções não possuem grupos e algumas delas somente afetam o bloco em que aparecem.

Palavra	Função afeta somente o bloco em que aparece	Função
G04	•	Pausa
G05		Não utilizada
G07		Não utilizada
G08	•	Aceleração
G09	•	Desaceleração
G10-G12		Não utilizada
G20-G32		Não utilizada
G36-G39		Não utilizada
G45-G49		Não utilizada
G50-G59		Reservado para controle adaptativo
G60-G69		Não utilizada
G76-G79		Não utilizada
G92	•	Definição da origem do sistema de coordenadas (zero-programa)
G98-G99		Não utilizada

Tabela 4.6 *Funções preparatórias suportadas pela EIA-274-D (sem grupos)*

A tabela 4.7 traz as funções "miscelânea" definidas pela EIA-274-D. Estas funções de programação auxiliar são designadas pela letra M e como o próprio nome indica têm múltiplas aplicações dentro de um sistema CNC, sendo geralmente do tipo liga/desliga, inverte, permuta, estabelece, etc.

Palavra	Função é executada juntamente com o início do bloco em que aparece	Função é executada após o bloco em que aparece	Função permanece em efeito até que cancelada ou anulada por um comando antagônico	Função afeta somente o bloco em que aparece	Função
M00		•		•	Parada
M01		•		•	Parada opcional
M02		•		•	Encerramento de programa
M03	•		•		Rotação da árvore (hor.)
M04	•		•		Rotação da árvore (anti-hor.)
M05		•	•		Desliga rotação da árvore
M06				•	Mudança de ferramenta
M07		•			Refrigerante n.2 - Liga
M08		•			Refrigerante n.1 - Liga
M09		•			Refrigerante - Desliga
M10			•		Blocagem dos carros
M11			•		Solta blocagem dos carros
M12		•		•	Código de sincronização
M13	•		•		Rotação da árvore horário e liga fluido refrigerante
M14	•		•		Rotação da árvore anti-horário e liga refrigerante
M15	•			•	Movimento +
M16	•			•	Movimento -
M17-M18					Não utilizado
M19			•		Desliga eixo da árvore com parada orientada
M20-M29					Não utilizado
M30		•		•	Final do programa
M31	•			•	*Interlock bypass*
M32-M35					Não utilizado
M36-M39					Não utilizado
M40-M46		•		•	Mudança de engrenamento se usado; senão não especif.
M47					Retorna ao início do programa
M48		•		•	Anula M49
M49		•		•	*Bypass override*
M50-M57					Não utilizado
M58		•		•	Anula M59
M59		•		•	*Bypass CSS updating*
M60-M89					Não utilizado
M90-M99					Reservado para o usuário

Tabela 4.7 *Funções "miscelânea" suportadas pela EIA-274-D*

4.4. Conclusão

Após realizados os estudos específicos sobre CNC, é proposto um modelo para o sistema que fica constituído por quatro classes principais:

- **Interpretador**, responsável pela análise do programa-peça e envio de ordens para que as funções de cada bloco sejam executadas na ordem correta;
- **Implementação da máquina**, responsável pela especificação de "o quê" e "como" deve ser executado, de acordo com cada ordem recebida do interpretador. Em outras palavras, assume as funções do interpolador e do controlador lógico ilustrados na figura 4.1.
- **Controlador de posição**, que é uma classe responsável por implementar uma estratégia para o controle de posição dos eixos da máquina.
- **Módulo de entrada e saída**, que é a classe responsável pelo controle da interface AD/DA que interconecta o comando aos acionamentos dos motores e outros atuadores da máquina CNC.

Na figura 4.11 tem-se representados os fluxos de informação entre as classes do modelo proposto, que começará a ser implementado no próximo capítulo.

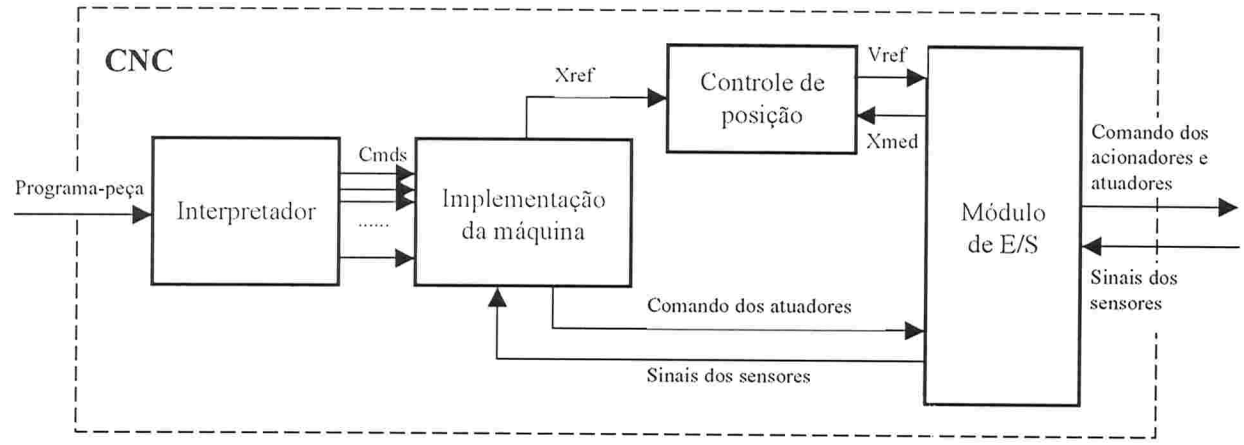

Fig. 4.11 Modelo adotado para a implementação dos componentes

5. Um Interpretador CNC em Smalltalk

5.1. Introdução

Tendo-se modelado o sistema como um todo, passa-se aqui à implementação do primeiro dos seus componentes, o interpretador.

O interpretador é responsável por quebrar os blocos do programa-peça em palavras, enviando à implementação da máquina as ordens referentes a cada palavra.

5.2. O Modelo

Por questões tanto de simplicidade como de flexibilidade, decidiu-se que o interpretador deveria aceitar as palavras componentes de um bloco de programa não importando a ordem em que aparecessem. Assim, do ponto de vista do usuário, o interpretador funcionaria para qualquer programa-peça, obedecendo ou não alguma convenção específica de formatação. Do ponto de vista do implementador, não seria necessário pelo menos no momento se preocupar com o detalhe de formato e a sua influência no comportamento do interpretador.

De forma a utilizar um mecanismo padrão no tratamento todas as funções de um programa-peça, definiu-se que durante a execução de um bloco, o interpretador simplesmente enviaria à implementação uma mensagem relativa a cada função do programa dizendo qual operação deveria ser executada.

Para que esse modelo simples funcionasse teria que ser definida uma ação para função CNC. Para algumas funções a mensagem a ser enviada seria óbvia. É o caso de funções como F,S e T, onde as mensagens seriam respectivamente alterar a velocidade de avanço, alterar a velocidade de corte e "setar" a próxima ferramenta a ser utilizada.

Em outros casos como as funções dos grupos A, D e G da tabela 4.5 (funções preparatórias para movimento e compensação) seriam necessários parâmetros adicionais. Por exemplo, o bloco **X1.000 Y1.500 G01 <EOB>** descreve um movimento em velocidade de corte da posição atual da máquina para a posição onde X = 1.000 e Y = 1.500 (supondo que a máquina está no modo de coordenadas absoluto). Ou seja, pode-se dizer que a função G01 necessita como parâmetro a posição final do movimento, representada neste caso por **X1.000 Y1.500**.

No intuito de respeitar o modelo proposto, as funções que fornecem parâmetros para movimentos e compensações de ferramentas tais como D, X, Y, Z, etc. deveriam também somente enviar uma mensagem à implementação, mensagem esta que simplesmente "setaria" alguma espécie de registrador da máquina com o valor passado. Posteriormente quando a função G descrevendo o tipo de movimento (ou compensação) fosse executada, a máquina pesquisaria nos seus registradores internos os parâmetros passados anteriormente e realizaria o movimento (ativaria a compensação).

Mas para que isso ocorresse ter-se-ia que garantir que independentemente da ordem das palavras em um bloco, a função preparatória seria sempre executada após todas as funções que carregam os parâmetros.

Assim, sobrepondo-se estas informações às informações da tabela 4.7, que especificam o momento em que as funções "miscelânea" devem ser executadas em relação ao bloco em que

aparecem, se estabelece a seguir a ordem em que as funções de um bloco de programa deverão ser executadas:

1. N.....
2. F.....
3. S.....
4. T.....
5. As funções M..... que devem ser executadas juntamente com o início do bloco. Por exemplo, algumas que definem modos de operação ou ligam a árvore ou o fluido refrigerante (tabela 4.7).
6. As funções G.....que não necessitam de parâmetros (tabela 4.5 grupos B, C, E, F, H e I).
7. Todas as outras funções exceto aquelas descritas em 8 e 9, ou seja, (A, B, C, ... X, Y, Z).
8. As funções G..... que necessitam de parâmetros (tabela 4.5 grupos A, D e G).
9. As funções M..... que devem ser executadas após o movimento descrito no bloco. Por exemplo, aquelas que desligam a rotação da árvore ou o fluido refrigerante (tabela 4.7).

5.3. Estrutura Interna

5.3.1. CNCCommandParser

A única função do interpretador (CNCCommandParser) é a leitura de um novo bloco do "stream"[10] de entrada, tendo como valor de retorno uma coleção ordenada[11] composta pelas palavras que compunham do bloco. As operações suportadas pelo CNCCommandParser estão representadas na figura 5.1, tendo-se sombreado as operações que são internas à classe e portanto inacessíveis ao usuário[12].

CNCCommandParser	
inputStream	Retorna o "stream" de entrada.
inputStream: *aStream*	Seta o valor "stream" de entrada em *aStream*.
programAtEnd	Retorna true ou false dependendo se foi encontrado o fim do programa.
readNextBlock	Lê o próximo bloco do programa.
breakUpBlock	Separa as funções contidas no último bloco lido.
nextSortedBlock	Ocasiona a leitura do próximo bloco do programa, decompondo-o nas suas funções componentes. Por fim retorna uma coleção ordenada com estas funções.

Fig. 5.1 CNCParser

Pelo fato de ser ter feito algumas suposições a respeito da implementação da máquina como a existência de algum mecanismo similar aos registradores de um computador e a

[10] "Stream" é uma estrutura genérica de entrada e/ou saída de dados utilizada na comunicação de dados quando se deseja um acesso seqüencial às informações. No caso é realizada a leitura seqüencial dos caracteres que compõem o programa-peça.
[11] Coleção ordenada é uma classe que representa uma estrutura de dados caracterizada por armazenar uma seqüência de objetos segundo um critério de ordenação definido pelo usuário.
[12] Esta notação será utilizada extensivamente daqui por diante.

possibilidade de se alterar algumas de suas variáveis, faz-se necessário realizar a definição destas características básicas em alguma classe, que será conhecida daqui por diante por CNCMachine.

5.3.2. CNCMachine

CNCMachine é uma classe abstrata que será a superclasse de todas as implementações de máquinas CNC. Isto é, ela define o comportamento básico e as variáveis presentes em todas estas máquinas. Quando necessário ela pode ser estendida, criando-se uma nova subclasse e adicionando-se as características desejadas, como novas variáveis e/ou operações.

Operações básicas como definição das velocidades de corte, avanço, número da próxima ferramenta a ser utilizada e modo para a entrada de coordenadas (absoluto/incremental) estão implementadas nesta classe e são herdadas pelas subclasses. Todas as outras operações devem ser definidas nas subclasses. A figura 5.2 mostra as operações suportadas.

CNCMachine	
executeProgram: *aStream*	Executa o programa contido no "stream" *aStream*.
askParserNextBlock	Requisita do interpretador o próximo bloco de funções na forma de uma coleção ordenada.
executeNextFunction	Procura e executa a próxima função contida na coleção recebida.
parser	Retorna o interpretador sendo utilizado.
spindleSpeed	Retorna a velocidade de corte da máquina.
feedRate	Retorna a velocidade de avanço da máquina.
toolStation	Retorna a próxima ferramenta a ser utilizada.
coordinateMode	Retorna o modo de entrada de coordenadas: absoluto ou incremental.
currentMotionFunction	Retorna o número da última função G de movimento executada. Esta é a função "default" executada quando o interpretador encontra um bloco com coordenadas para movimento sem especificação da interpolação a ser utilizada, ou seja, sem funções G para movimento.
currentCoordinates	Retorna as coordenadas atuais absolutas da máquina
cncCommandA: *argument*	Executa o código correspondente à função A, passando como parâmetro *argument*.
.....	
.....	
cncCommandZ: *argument*	Executa o código correspondente à função Z, passando como parâmetro *argument*.

Fig. 5.2 *CNCMachine*

5.4. O Algoritmo

O algoritmo para a execução de um programa é:

1. Enquanto há blocos disponíveis no "stream" de entrada, a máquina:
   1.1. Requisita do interpretador uma coleção ordenada com as palavras que compõem o próximo bloco do programa (segundo o critério definido no início do capítulo);

1.2. Para cada elemento da coleção executa um método correspondente, que deve implementar a respectiva ação. Exemplos:

cncCommandM: 03
" Liga a rotação da árvore no sentido horário "

cncCommandX: coordenada
" Associa ao parâmetro X o valor coordenada "

cncCommandG: 01
" Recupera os parâmetros passados anteriormente e executa o movimento linear em velocidade de corte."

1.3. Depois que todos os comandos do bloco atual foram executados, descarta os parâmetros referentes ao bloco atual antes da leitura do próximo bloco;

2. Quando não houver mais nenhum bloco, fecha o "stream" de entrada.

5.5. Uma Aplicação Exemplo

Um exemplo de utilização do interpretador é a CNCTextMachine. Em vez de ser uma máquina que gera trajetórias para um controlador de posição, a sua função durante a execução de um programa-peça é mostrar mensagens ao usuário, informando-o das operações sendo realizadas. As suas funções suportadas são:

- F, N, S, T, X, Y, Z e %
- G00, G01, G28, G90, G91 e G92
- M03 e M30

Exatamente do modo descrito na seção 5.3, a CNCTextMachine foi criada como uma subclasse de CNCMachine, herdando assim o seu comportamento básico. Não foi necessária a criação de nenhuma nova variável. Todo o trabalho envolvido consistiu em se definir qual mensagem deveria ser mostrada por cada função.

A figura 5.3 mostra uma programa executado pela CNCTextMachine, juntamente com a sua saída.

```
%
N0010 G91 S600.
N0020 G01 X-100. Y-100.  M03
N0030 Z-0.75 F5.
N0040 Z1.
N0050 X100. G00
N0060 Y-100.
N0070 G90 X10. Y10. Z10.
N0080 X12. Y12.
N0090 X14.
N0100 Y20.
N0110 G91 X5.
N0120 y5. g01
N0130 Z6.
N0140 M30

Line 0010:   Set spindle speed to: 600
             Set incremental mode.
```

```
Line 0020:  Turn spindle on (clockwise) .
            Point to point positioning (cutting speed) to: [ X = -100.0   Y = -100.0   Z = 0 ]
Line 0030:  Feed rate set to: 5.0
            Point to point positioning (cutting speed) to: [ X = -100.0   Y = -100.0   Z =-0.75]
Line 0040:  Point to point positioning (cutting speed) to: [ X = -100.0   Y = -100.0   Z =0.25]
Line 0050:  Point to point positioning (rapid) to: [ X = 0.0   Y = -100.0   Z =0.25]
Line 0060:  Point to point positioning (rapid) to: [ X = 0.0   Y = -200.0   Z =0.25]
Line 0070:  Set absolute mode.
            Point to point positioning (rapid) to: [ X = 10.0   Y = 10.0   Z =10.0]
Line 0080:  Point to point positioning (rapid) to: [ X = 12.0   Y = 12.0   Z =10.0]
Line 0090:  Point to point positioning (rapid) to: [ X = 14.0   Y = 12.0   Z =10.0]
Line 0100:  Point to point positioning (rapid) to: [ X = 14.0   Y = 20.0   Z =10.0]
Line 0110:  Set incremental mode.
            Point to point positioning (rapid) to: [ X = 19.0   Y = 20.0   Z =10.0]
Line 0120:  Point to point positioning (cutting speed) to: [ X = 19.0   Y = 25.0   Z =10.0]
Line 0130:  Point to point positioning (cutting speed) to: [ X = 19.0   Y = 25.0   Z =16.0]
Line 0140:  End of program.
```

Fig. 5.3 Programa exemplo executado em uma CNCTextMachine seguido pelo seu resultado

5.6. Problemas Encontrados

Uma análise detalhada do código do interpretador levantou alguns pontos chave que deveriam ser melhorados antes que se fosse adiante no projeto. Muitos dos recursos utilizados na implementação têm um alto custo em termos de velocidade e consumo de memória.

5.6.1. Dicionários

Em vários momentos foram utilizados dicionários quando poderiam ter sido utilizados "arrays", que são simplesmente uma seqüência indexada de valores. Assim, em vez de um dicionário contendo os parâmetros para os códigos G responsáveis por movimentos, ter-se-ia um "array" (Fig. 5.3), o que seria mais rápido (não há pesquisa) e mais econômico em termos de memória, pois não seriam armazenadas as chaves para pesquisa.

Dicionário	
Chave	Valor
A	
...	
...	
X	
Y	
Z	

Array	
Índice	Valor
1	
...	
...	
24	
25	
26	

Fig. 5.3 Dicionários x "Arrays"

5.6.2. Coleções Ordenadas

Coleções ordenadas são uma especialização das coleções e possuem um critério de ordenação definido pelo usuário, critério esse que será utilizado para posicionar automaticamente cada novo objeto que for adicionado à coleção.

Por causa dos cálculos envolvidos na reordenação da coleção depois que cada objeto é inserido, o processo torna-se relativamente lento. Se houver alguma alternativa à utilização de coleções ordenadas, esta deve ser seriamente considerada.

5.6.3. Criação de objetos no interior de laços de repetição

Sempre que possível deve-se evitar ao máximo a instanciação de objetos no interior de laços, tentando-se em vez disso ter uma única instância que é reaproveitada. Do contrário isto pode ocasionar a ativação excessiva do coletor de lixo[13] do sistema, denegrindo performance deste.

[13] O coletor de lixo é um componente da máquina virtual Smalltalk (e Java) responsável pela desalocação automática do espaço ocupado por objetos, o que ocorre somente a partir do momento que eles não são mais necessários. A sua ativação resulta no consumo de alguns ciclos de processamento da CPU.

6. O Interpretador em Java

6.1. Introdução

A pedido da empresa, uma versão em Java do interpretador deveria ser disponibilizada, já contando com alterações necessárias para se superar os problemas discutidos no capítulo anterior.

Esta aparente mudança nos rumos do projeto não influenciou nas suas linhas gerais, já que o desenvolvimento do interpretador seguiu seu curso, em um primeiro momento com a correção dos problemas já descritos e posteriormente com a adição de novas funcionalidades.

6.2. Mudanças Realizadas

As principais mudanças efetuadas foram:

- substituição de dicionários por "arrays" sempre que possível;
- coleções ordenadas são descartadas;
- sempre que possível evita-se instanciar um objeto no interior de um laço;

Basicamente os resultados esperados da versão Java do interpretador são idênticos aos da versão Smalltalk, sendo as diferenças entre as duas versões somente internas.

As classes que compõem o interpretador em Java são:

- CNCCommandParser
- CNCMachine
- CNCException
- CNCMachineProperties

As primeiras duas classes correspondem respectivamente às classes de mesmo nome da versão Smalltalk. Nesta versão foi introduzido um mecanismo tratador de exceções que é utilizadas no tratamento de condições de funcionamento anormais, como por exemplo um programa utilizando uma função não suportada ou um parâmetro fora da faixa permitida.

Outra classe que foi criada nesta versão foi CNCMachineProperties[14], que tem como finalidade descrever as propriedades de uma dada máquina.

6.3. O Novo Modelo

No intuito de se realizar as mudanças propostas começou-se por alterar o modo como as palavras de um bloco lido pelo interpretador são ordenadas. Na verdade durante a interpretação de um bloco as palavras não mais são ordenadas, sendo o bloco simplesmente armazenado em um "buffer" de trabalho intermediário.

O que ocorre então é que este "buffer" é "varrido" várias vezes, procurando-se a cada vez por um tipo de função específica, na seqüência definida na seção 6.4.1. Encontrada a função, realiza-se a conversão do seu valor numérico do formato fixo para o formato variável se

[14] Vide seção 6.4.2

necessário, enviando-se em seguida o respectivo comando à implementação da máquina para execução.

6.4. A Estrutura Interna

6.4.1. CNCCommandParser

Para que o novo modelo funcionasse, as funções do programa-peça foram reagrupadas da seguinte forma:

1. Funções comuns: todas exceto as funções G e M.
2. Funções M executadas antes do movimento
3. Funções G sem parâmetros
4. Funções G com parâmetros
5. Funções M executadas após o movimento

Foram criadas algumas variáveis estáticas no interpretador com o propósito de guardar uma lista com as funções de cada grupo. Por questões de economia definiu-se que as funções M que não pertencem ao grupo 2 pertencem automaticamente ao grupo 5, o mesmo ocorrendo para as funções G dos grupos 3 e 4. Desse modo são criadas somente três novas variáveis e não cinco, como seria inicialmente.

A figura 6.1 mostra as operações suportadas pela classe.

CNCCommandParser	
inputStream	Retorna o "stream" de entrada.
inputStream: *aStream*	Seta o valor "stream" de entrada em *aStream*.
handleNextBlock	Realiza o envio de ordens para que a máquina execute as funções contidas no próximo bloco do programa.
readNextBlock	Lê o próximo bloco do programa e o armazena no "buffer" de trabalho.
nextCommandInBuffer	Retorna o próximo comando contido no "buffer" de trabalho.
nextArgumentInBuffer	Retorna o próximo parâmetro (número) contido no "buffer" de trabalho.
convertArgument: *anArgument*	Retorna o valor convertido correspondente ao parâmetro *anArgument*. Conversão do formato fixo sem ponto decimal para o formato variável com ponto decimal.
handleOrdinaryFunctions	Envia à máquina ordens para a execução das funções definidas como sendo comuns.
handleMOffFunctions	Envia à máquina ordens para a execução das funções M que devem ser executadas após o movimento.
handleMOnFunctions	Envia à máquina ordens para a execução das funções M que devem ser executadas juntamente o movimento.
handleGMotionFunctions	Envia à máquina ordens para a execução das funções G que necessitam de parâmetros.
handleGPreparatoryFunctions	Envia à máquina ordens para a execução das funções G que não necessitam de parâmetros.

currentMotionFunction	Retorna o número da última função G de movimento executada. Esta é a função "default" executada quando o interpretador encontra um bloco com coordenadas para movimento sem especificação do tipo de interpolação a ser utilizada.
machine	Retorna a implementação da máquina que utiliza o interpretador (referência cruzada).

Fig. 6.1 O novo CNCCommandParser

6.4.2. CNCMachineProperties

Esta classe representa um conjunto de informações sobre a máquina composto por:

- valores mínimo e máximo permitidos para a velocidade de corte
- valores mínimo e máximo permitidos para a velocidade de avanço
- valores mínimo e máximo permitidos para ferramentas
- endereços suportados
- códigos G suportados
- códigos M suportados
- detalhe do formato das palavras suportadas

Estas informações são utilizadas pelo interpretador para:

- Gerar exceções quando uma palavra não suportada é utilizada ou quando os limites para algum parâmetro não são respeitados;
- Realizar a conversão automática de parâmetros entrados no formato fixo, sem ponto decimal para o formato variável com ponto decimal.

As operações suportadas pela classe CNCMachineProperties são mostradas abaixo, na figura 6.2.

CNCMachineProperties	
supportsFunction: *aFunction*	Retorna *true* ou *false* dependendo se a função é suportada ou não.
addFunction: *aFunction* withFormat: *aFormat*	Adiciona a função *aFunction* com o formato *aFormat* às funções suportadas.
rightDigitsForFunction: *aFunction*	Retorna o número de dígitos à direita do ponto decimal suportados pela função *aFunction*.
leftDigitsForFunction: *aFunction*	Retorna o número de dígitos à esquerda do ponto decimal suportados pela função *aFunction*.
feedRateRange	Retorna os valores máximo e mínimo permitidos para a velocidade de avanço.
feedRateRange: *aRange*	Seta os valores máximo e mínimo permitidos para a velocidade de avanço de acordo com a faixa *aRange*.
spindleSpeedRange	Retorna os valores máximo e mínimo permitidos para a velocidade de corte.

spindleSpeedRange: *aRange*	Seta os valores máximo e mínimo permitidos para a velocidade de corte de acordo com a faixa *aRange*.
toolIdRange	Retorna os valores máximo e mínimo permitidos como idenficadores de ferramenta.
toolIdRange: aRange	Seta os valores máximo e mínimo permitidos como idenficadores de ferramenta de acordo com a faixa *aRange*.

Fig. 6.2 CNCMachineProperties

6.4.3. CNCMachine

Do mesmo modo que na implementação anterior, a classe CNCMachine foi criada como sendo uma classe abstrata provendo somente o comportamento básico comum a todas as implementações de máquinas CNC. Isto corresponde à implementação das funções: F, S, T, G90 and G91. Detalhes da classe podem ser visualizados na figura 6.3.

CNCMachine	
executeProgram: *aStream*	Executa o programa contido no "stream" *aStream*.
parser	Retorna o interpretador sendo utilizado (CNCCommandParser).
properties	Retorna as propriedades da máquina (CNCMachineProperties).
spindleSpeed	Retorna a velocidade de corte da máquina.
feedRate	Retorna a velocidade de avanço da máquina.
toolStation	Retorna a próxima ferramenta a ser utilizada.
coordinateMode	Retorna o modo de entrada de coordenadas: absoluto ou incremental.
currentMotionFunction	Retorna o número da última função G de movimento executada. Esta é a função "default" executada quando o interpretador encontra um bloco com coordenadas para movimento sem especificação da interpolação a ser utilizada, ou seja, sem funções G para movimento.
currentCoordinates	Retorna as coordenadas atuais absolutas da máquina
cncCommandA: *argument*	Executa o código correspondente à função A, passando como parâmetro *argument*.
.....	
.....	
cncCommandZ: *argument*	Executa o código correspondente à função Z, passando como parâmetro *argument*.

Fig. 6.3 A nova CNCMachine

6.5. O Novo Algoritmo

O algoritmo para a execução de um programa-peça utilizado pelo novo interpretador é:

1. Checa o programa completamente à procura de funções não suportadas. Se alguma função não suportada é encontrada, dispara-se uma exceção, que deve ser tratada pela implementação da máquina. Do contrário apenas reposiciona o "stream" de entrada na sua posição inicial.
2. Enquanto houver blocos disponíveis no "stream" de entrada repete:
   2.1. Leitura do próximo bloco do programa-peça no "buffer" temporário;
   2.2. Varredura do buffer várias vezes à procura das seguintes funções (nesta ordem), executando cada função encontrada após a conversão do seu valor numérico, se necessário:
      2.2.1. A primeira "função comum" de cada tipo (vide seção 6.4.1);
      2.2.2. Todas as funções M que devem ser executadas juntamente com o início do bloco;
      2.2.3. Todas as funções G que não precisam de parâmetros;
      2.2.4. Todas as funções G que precisam de parâmetros (movimentos ou compensações);
      2.2.5. Todas as funções M que devem ser executadas após o movimento.
3. Quando não houver mais blocos disponíveis fecha o "stream" de entrada.

6.6. Um Exemplo

Para se testar o novo interpretador foi portada a CNCTextMachine para o Java, incluindo-se uma política de tratamento das exceções geradas por comandos não suportados. O detalhe de formato adotado para esta máquina foi: N4.F33.G2.M2.S33.T4.X33.Y33.Z33

As suas funções preparatórias e "miscelâneas" suportadas são:

- G00, G01, G28, G90, G91 e G92
- M03 e M30

A figura 6.4 mostra uma programa exemplo executado pela CNCTextMachine, juntamente com a sua saída (Fig. 6.5).

```
%
N0010 G90
N0020 G01 X-10000 Y-20. Z1000  M03
N0030 Z-0.75 S50000
N0040 Y10.
N0050 X100. G00
N0060 Y-100.
N0070 G91
N0080 X2300 Y4600
N0090 X14.
N0100 Y20.
N0110 Z16.
N0120 M30
```

Fig. 6.4 Programa exemplo executado em uma CNCTextMachine

```
Line 0010:  Set absolute mode.
Line 0020:  Turn spindle on (clockwise) .
            Point to point positioning (cutting speed) to: [ X = -10.0   Y = -20.0   Z = 1.0 ]
Line 0030:  Spindle speed set to: 50.0
            Point to point positioning (cutting speed) to: [ X = -10.0   Y = -20.0   Z =-0.75]
Line 0040:  Point to point positioning (cutting speed) to: [ X = -10.0   Y = -10.0   Z =-0.75]
Line 0050:  Point to point positioning (rapid) to: [ X = 100.0   Y = -10.0   Z =-0.75]
Line 0060:  Point to point positioning (rapid) to: [ X = 100.0   Y = -100.0   Z =-0.75]
Line 0070:  Set incremental mode.
Line 0080:  Point to point positioning (rapid) to: [ X = 14.3   Y = 16.6   Z =-0.75]
Line 0090:  Point to point positioning (rapid) to: [ X = 28.3   Y = 16.6   Z =-0.75]
Line 0100:  Point to point positioning (rapid) to: [ X = 28.3   Y = 36.6   Z =-0.75]
Line 0110:  Point to point positioning (rapid) to: [ X = 28.3   Y = 36.6   Z = 15.25]
Line 0120:  End of program.
```

Fig. 6.5 *Saída obtida a partir do programa exemplo*

6.7. Conclusão

A experiência de se mudar a linguagem adotada durante o desenvolvimento do projeto trouxe benefícios.

Do ponto de vista do implementador, a mudança obrigou o desenvolvimento de novas e mais econômicas soluções pela adoção de uma linguagem mais simples e com menos recursos. Isto provou ser extremamente válido em se considerando que o objetivo principal do projeto era a construção de um conjunto de componentes de software para CNC que pudesse ser utilizado em diferentes plataformas, algumas das quais podendo impor restrições quanto ao espaço em memória disponível.

7. De Volta ao Smalltalk

7.1. Introdução

Tendo-se resolvido os problemas mencionados no final do capítulo 5, o mesmo modelo (incluindo a hierarquia de classes) adotado na implementação em Java foi portado de volta ao Smalltalk seguindo a linha original do projeto. Através da manutenção da estrutura de classes, suas variáveis e do algoritmo de interpretação procurou-se produzir em Smalltalk uma cópia fiel do modelo em Java.

A reimplementação do modelo em Smalltalk não será abordada aqui por ser basicamente uma repetição do que foi realizado no capítulo 6. Assim sendo, passa-se diretamente para a próxima etapa que foi a implementação de uma aplicação completa utilizando os componentes desenvolvidos: um simulador gráfico do caminho percorrido pela ferramenta durante a usinagem.

7.2. Um Exemplo de Aplicação: Um Simulador Gráfico da Trajetória da Ferramenta durante a Usinagem

7.2.1. Introdução

Esta é uma aplicação real construída utilizando-se os componentes de software criados. Trata-se de um simulador gráfico cuja função é mostrar ao usuário o gráfico produzido pelo deslocamento da ferramenta durante a execução de um programa-peça.

Através da utilização dos componentes já desenvolvidos há uma grande economia em tempo de desenvolvimento, já que o comportamento básico de uma máquina CNC já é implementado por eles. Ou seja, só é necessário se preocupar com a parte gráfica específica desta aplicação.

A seguir estão listadas as principais características oferecidas pelo simulador:

- execução passo-a-passo;
- operação "desfazer", ou seja, passo-a-passo de trás para diante;
- "zoom" (escalonamento) do resultado produzido através de interface gráfica.
- manipulação (rotação) do resultado produzido através de interface gráfica.

Há uma limitação quanto às funções CNC suportadas pelo simulador. Estas são: I, J, K, X, Y, Z, G00, G01, G02, G03, G90 e G91. Não são suportadas funções M por estas não apresentarem uma representação visual do seu efeito.

No caso de o programa-peça conter funções não discriminadas acima, a sua execução será interrompida no bloco anterior à ocorrência da referida função, continuando somente após a correção do programa.

7.2.2. Desenvolvimento

O primeiro requisito para se fazer o simulador possível foi a existência de um conjunto de classes dedicado à matemática vetorial. Sem elas não haveria um modo simples de se representar e manipular um conjunto de elementos gráficos no espaço.

Por isso foi criada uma pequena biblioteca de classes em Smalltalk chamada VectorMath, cujas componentes são as classes Matrix, Vector e LUDecomposedMatrix[15]. É suportada a maioria das operações matemáticas envolvendo estas entidades, como inversa, determinante e resolução de um sistema de equações lineares[16]. Tendo sido desenvolvida separadamente das classes para comando numérico, esta biblioteca pode ser livremente utilizada na solução de quaisquer problemas que venham a precisar deste tipo de matemática.

Com a base matemática criada, realizou-se uma revisão dos conceitos de álgebra linear utilizados em computação gráfica na representação de transformações geométricas de pontos no espaço [FOLEY 90].

O próximo passo foi encontrar uma representação conveniente para as entidades geométricas envolvidas. Foi criada uma pequena árvore de primitivas gráficas, cuja raiz passou a ser CNCGraphicPrimitive. Sendo esta uma classe abstrata, ela definiu as variáveis de instância básicas presentes em todas as primitivas gráficas. As suas operações disponíveis ao usuário são mostradas na figura 7.1.

CNCGraphicPrimitive	
startingPoint	Retorna um vetor 3D representando a origem da primitiva no espaço.
startingPoint: *aPoint*	Seta a origem da primitiva no espaço de acordo com *aPoint*.
endingPoint	Retorna um vetor 3D representando o final da primitiva no espaço.
endingPoint: *aPoint*	Seta o final da primitiva no espaço de acordo com *aPoint*.
color	Retorna a cor a ser utilizada na apresentação da primitiva.
color: *aColor*	Seta a cor a ser utilizada na apresentação da primitiva de acordo com *aColor*.

Fig. 7.1 *CNCGraphicPrimitive*

Como primitivas gráficas concretas tem-se CNCLine, que como o nome faria supor representa uma linha reta, e CNCArc que representa um arco de circunferência no espaço.

Como cada primitiva tem um modo diferente de ser desenhada, cada uma delas terá que prover o seu próprio método "draw", responsável pelo seu desenho em uma janela. Por essa razão esta é a única operação definida por uma CNCLine, que por ser extremamente simples não necessita de maiores cálculos para poder ser desenhada, já que somente é necessário projetar ambas suas extremidades no plano da tela e utilizar uma rotina provida pelo Smalltalk para ligar os dois pontos. O protocolo suportado pela CNCLine é mostrado na figura 7.2.

[15] Uma matriz A pode ser decomposta em um produto de duas matrizes, uma sendo diagonal inferior (*lower*) e outra diagonal superior (*upper*). Instâncias da classe LUDecomposedMatrix representam justamente o resultado da decomposição LU aplicada sobre uma matriz A.
[16] Os algoritmos para as três operações mencionadas foram obtidos em [PRESS 88].

CNCLine[17]	
drawOn: *aWindow* using: *aMatrix*	Desenha a linha na janela *aWindow* utilizando a matriz de transformação *aMatrix*.

Fig. 7.2 CNCLine

No caso de arcos o problema é muito mais complicado pois um arco circular em 3D pode se tornar um arco elíptico em qualquer orientação no espaço, dependendo para isso de onde está localizado o observador (ou de como o desenho foi rotacionado pelo usuário). Infelizmente o Smalltalk não provê rotinas para o desenho de tais arcos, requerendo a criação de tais rotinas.

Considerando-se um arco circular qualquer no espaço, o algoritmo utilizado para a geração dos seus pontos realiza primeiramente o cálculo de dois ângulos. Seja γ o plano do arco no espaço. O primeiro ângulo calculado (α rad.) é o ângulo entre a intersecção de γ com o plano xy e o eixo x. O segundo ângulo (β rad.) é o ângulo entre a intersecção de γ com o plano yz e o eixo z.

Tendo-se os dois ângulos, são efetuadas duas rotações no arco 3D original: uma em torno do eixo X (de $-\alpha$ rad.) e outra em torno do eixo Z (de $-\beta$ rad.). Estas duas rotações fazem com que os pontos de início, fim e centro do arco sejam transportados para um plano onde estes pontos têm a mesma coordenada Z. Desta forma o arco pode ser tratado como 2D (chamemos este plano de plano de trabalho). A matriz de transformação resultante (acumulando as duas rotações) é mantida para uso futuro.

A geração dos pontos de um arco 2D utilizando-se coordenadas polares é trivial. Seja θ a variável de controle em um "loop" assumindo os valores entre os ângulos inicial e final do arco (r é o raio do arco).

$x = r * sen(\theta)$
$y = r * cos(\theta)$
$z = constante$

Avaliando-se as funções acima para todos os valores de θ obtêm-se os pontos que compõem o arco no plano de trabalho.

Como a matriz de transformação utilizada foi previamente armazenada, tudo que se deve fazer para se conseguir o arco em 3D é utilizar a inversa dessa matriz para transportar os pontos do plano de trabalho para o espaço. A seguir cada ponto no espaço é projetado no plano da tela e assim se obtém a projeção do arco na tela não importando a sua orientação no espaço. A figura 7.3 mostra o protocolo suportado pelos arcos circulares.

CNCArc[17]	
center	Retorna um vetor 3D representando o centro do arco no espaço.
sense	Retorna o sentido de desenho do arco: *horário* ou *anti-horário*.
drawOn: *aWindow* using: *aMatrix*	Gera os pontos para o desenho do arco na janela *aWindow* utilizando a matriz de transformação *aMatrix*.
calculatePlaneCoefficients	Retorna os coeficientes *a*, *b*, *c* e *d* do plano γ de forma se ter a representação: a.x + b.y + c.z + d = 0, utilizada no cálculo de α e β.

[17] As operações herdadas de CNCGraphicPrimitive não foram reincluídas.

calculateAlpha	Retorna o valor de α como definido acima.
calculateBeta	Retorna o valor de β como definido acima.
alignArc	Realiza as duas rotações descritas acima.
alignedCenter	Retorna um vetor 3D representando o centro do arco no plano de trabalho.
startingAngle	Retorna o ângulo inicial do arco em radianos.
openingAngle	Retorna o ângulo de abertura do arco em radianos.
threeTo2Dmatrix	Retorna a matriz que transporta os pontos do espaço para o plano de trabalho.

Fig. 7.3 CNCArc

Tendo-se os elementos gráficos já implementados, faltava ainda uma estrutura de dados que pudesse armazená-los. Mais ainda, esta estrutura deveria guardar as transformações acumuladas sobre os seus elementos, permitindo que se soubesse a orientação destes no espaço a qualquer momento. Com esta finalidade foi criado o CNCWorkspace, cujo protocolo é mostrado na figura 7.4.

O CNCWorkspace armazena as primitivas gráficas em uma coleção do tipo fila, ou seja, pode-se adicionar (ou remover) primitivas gráficas apenas no final dela.

Os elementos contidos em um CNCWorkspace não são automaticamente redesenhados após cada transformação. Isto faz com que ele possa acumular várias transformações e atualizar o desenho uma única vez mediante um comando explícito, trazendo com isso um aumento da velocidade de execução do programa, que se tornaria excessivamente lento se o desenho fosse atualizado a cada transformação.

CNCWorkspace	
contents	Retorna as primitivas gráficas contidas pelo "workspace".
addPrimitive: *aGraphicPrimitive*	Adiciona a primitiva *aGraphicPrimitive* ao "wokspace".
removeLastPrimitive	Remove a última primitiva adicionada ao "workspace".
rotateX: *degrees*	Rotaciona todos os elementos contidos no "workspace" de *degrees* em torno do eixo X.
rotateY: *degrees*	Rotaciona todos os elementos contidos no "workspace" de *degrees* em torno do eixo Y.
rotateZ: *degrees*	Rotaciona todos os elementos contidos no "workspace" de *degrees* em torno do eixo Z.
translateX: *deltaX* y: *deltaY* z: *deltaZ*	Efetua a translação dos elementos contidos no "workspace" utilizando os deslocamentos relativos dados por *deltaX*, *deltaY* e *deltaZ*.
scaleX: *scaX* y: *scaY* z: *scaZ*	Efetua o escalonamento dos elementos contidos no "workspace" utilizando os fatores de escala *scaX*, *scaY* e *scaZ*.
redrawAllElements: *aWindow*	Redesenha todos os seus elementos na janela *aWindow*.
transformationMatrix	Retorna a matrix que acumula todas as transformações sofridas pelo "workspace".

Fig. 7.4 CNCWorkspace

A operação "desfazer" foi implementada através da classe CNCSimulatorUndoCommand utilizando-se "Design Patterns"[18], mais especificamente o "Command Pattern", que é utilizado para encapsular um pedido na forma de um objeto, permitindo assim que se implemente filas de pedidos ou ainda que se suporte operações "desfazer" (*undo*) [GAMMA 94].

"Desfazer" aqui significa não somente remover a última primitiva gráfica adicionada ao "workspace", mas também restaurar valores de variáveis da máquina. Neste caso específico somente o modo de coordenadas, já que outras variáveis como velocidades de corte/avanço não têm significado nesta máquina.

Uma possível melhoria da operação "desfazer" poderia ser conseguida através da adoção do "Memento pattern"[19] [GAMMA 94], que cria uma "fotografia" do estado de um objeto em um dado momento, permitindo que o objeto retorne a esse estado mais tarde.

Depois que a base para o simulador estava pronta, o último passo foi a criação da interface com o usuário, que teria a tarefa gerenciar os eventos provenientes do teclado e do mouse, devolvendo como resposta a atualização do desenho da trajetória produzida. A interface do simulador com o usuário pode ser vista na figura 7.5.

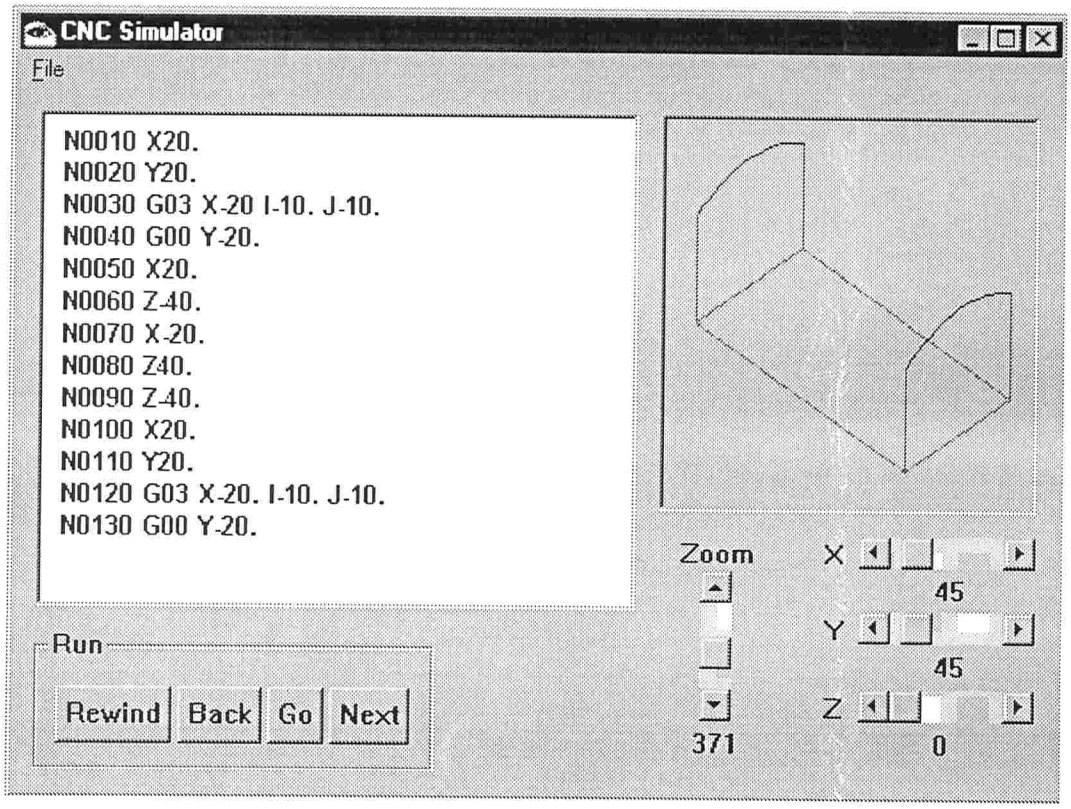

Fig. 7.5 Interface com o usuário do simulador da trajetória da ferramenta

[18] "Design Patterns" são soluções genéricas documentadas para problemas comumente encontrados na programação orientada a objetos. Vários destes "patterns" são formalizados em [GAMMA 94].

[19] "Memento pattern": "Sem violar o encapsulamento, externaliza o estado interno de um objeto de modo que ele possa retornar a este estado mais tarde [GAMMA 94]".

8. Conclusões e Perspectivas

A construção de um comando numérico sob a forma de um sistema embutido encontra-se entre as principais e mais completas aplicações às quais este trabalho pode ser aplicado.

Assim sendo, o trabalho aqui apresentado tenta abordar as fases iniciais desse projeto, modelando o CNC e posteriormente implementando parte desse modelo na forma de uma biblioteca de componentes de software reusáveis.

Inicialmente foram estudados vários conceitos importantes relacionados à engenharia de software, o que culminou com a escolha da orientação a objetos e da tecnologia de máquinas virtuais como meios de se atenuar alguns dos problemas mais comuns relacionados ao desenvolvimento e manutenção de softwares.

Após a escolha da tecnologia a ser usada, foi estudado o domínio do problema de forma a se produzir um modelo do CNC do ponto de vista da orientação a objetos, modelo este que deu origem aos componentes de software implementados na próxima etapa do trabalho.

Como uma aplicação destes componentes em um problema real, foi desenvolvido no final do trabalho um simulador da trajetória da ferramenta durante o processo de usinagem, o que inclusive resultou em outros subprodutos como as bibliotecas de classes para matemática vetorial e manipulação de primitivas gráficas no espaço, que podem ser utilizadas de forma totalmente independente em problemas não relacionados a CNC.

O uso conjunto da abordagem orientada a objetos e de uma linguagem baseada em máquinas virtuais trouxe consigo os seguintes benefícios:

- O tempo (e conseqüentemente os custos) de desenvolvimento de um produto (no caso, o simulador) foram reduzidos devido à reusabilidade automática do código já produzido anteriormente na forma de componentes;
- As adaptações que os componentes deverão sofrer no caso de sua aplicação em outros problemas são incrementais e localizadas, isto é, estas adaptações se dão na forma de criação de novas subclasses, não incorrendo em alterações (e problemas em potencial) ao código já implementado;
- O aprendizado da biblioteca de componentes pelo usuário foi facilitado pela uniformidade no código produzido quanto aos nomes das operações (uma classe herda as operações e os seus nomes de sua superclasse);
- A programação orientada a objetos, por ser baseada na simulação de objetos do mundo real, produziu um modelo de fácil entendimento, facilitando a sua futura manutenção mesmo por pessoas que não tiveram contato com o projeto desde o seu princípio;
- O código produzido ficou portável por este não ser específico a nenhuma plataforma, mas sim a uma máquina construída em software e que portanto pode ser implementada em diversas plataformas diferentes.

Do ponto de vista da experiência profissional adquirida, este projeto se constituiu em uma oportunidade única de aprendizado, pois foi realizado em um ambiente extremamente motivador e na companhia de pessoas altamente capacitadas. O fato do projeto ter uma natureza multidisciplinar só veio a contribuir nesse processo, permitindo o envolvimento com diversos ramos da engenharia:

- tecnologia CNC
- processos de fabricação
- programação orientada a objetos
- computação gráfica

Na perspectiva de se dar uma continuação ao projeto, outras áreas seriam ainda abordadas, entre elas:

- controle de movimento
- metrologia
- servoacionamentos

Nesse sentido a próxima etapa seria a utilização dos componentes desenvolvidos no controle de um dispositivo mecânico real, o que inclusive estava previsto no projeto original, mas devido a problemas relacionados à aquisição do equipamento não pôde ser realizado.

Assim, a primeira coisa a ser feita seria um gerador de trajetórias, que poderia ser implementado como uma extensão das primitivas gráficas criadas. Cada primitiva conta com um método que define como ela será desenhada na tela. De modo análogo, poderia ser criado um método em cada primitiva que definisse o mecanismo de geração dos seus pontos, que juntos definiriam a respectiva trajetória.

Para que o computador pudesse se comunicar com o dispositivo mecânico faltaria apenas o desenvolvimento do módulo de entrada e saída. A partir daí já poderiam ser realizados testes com um dispositivo mecânico em malha aberta, como uma mesa xy acionada por motores de passo, por exemplo.

O módulo de controle, requerido no caso do uso de servoacionamentos, juntamente com o módulo de E/S são as partes que requerem uma maior velocidade de execução. Isto quer dizer que se o código destes componentes for muito lento, corre-se o risco de que os algoritmos de controle não consigam controlar o dispositivo mecânico de forma satisfatória.

Em situações como esta, para resolver o problema poderia ser realizada a implementação dos trechos de código mais críticos em uma linguagem mais rápida como C, ou mesmo Assembly, o que traria como efeito colateral o sacrifício da portabilidade dos componentes envolvidos em nome da velocidade. Entretanto somente experimentos práticos envolvendo o sistema completo poderão dizer se isso realmente será necessário.

9. Bibliografia

[DEGAS]
Dev Team One. ***DEGAS 16 Bit Virtual Machine Guide***. Mountain View, California (USA).

[EIA274]
Electronic Industries Association. EIA Standard: ***EIA-274-D ("Interchangeable Variable Block Data Format for Positioning, Contouring and Countouring / Positioning Numerically Controlled Machines")***. (USA) 1979.

[FOLEY 90]
FOLEY, James T. et al. ***Computer Graphics: principles and practice***. Addison-Wesley Publishing Co., (USA) 1990.

[GAMMA 94]
GAMMA, Erich et al. ***Design Patterns: elements of reusable object-oriented software***. Addison-Wesley Publishing Co., (USA) 1994.

[GOLDBERG 89]
GOLDBERG, Adele ; ROBSON, David. ***Smalltalk-80 The Language***. Addison-Wesley Publishing Co. 1989.

[LIU 96]
LIU, Chamond. ***Smalltalk, Objects, and Design***. Prentice Hall. (USA) 1996.

[LYNCH 92]
LYNCH, Michael I. ***Computer Numerical Control for Machining***. McGraw Hill. (USA) 1992.

[MEYER 88]
MEYER, Bertrand. ***Object-oriented software construction***. Prentice Hall. (USA) 1988.

[MEYER 97]
MEYER, Jon ; DOWNING, Troy. ***Java Virtual Machine***. O'Reilly & Associates. (USA) 1997.

[ODETTE 91]
ODETTE, Louis L. ***Intelligent Embedded Systems***. Addison-Wesley Publishing Co., Massachusetts (USA) 1991.

[PRESS 88]
PRESS, William H et al. ***Numerical Recipes in C - The Art of Scientific Computing***. Cambridge University Press. Cambridge MA (USA) 1988.

[SQUEAK]
Squeak Smalltalk Home Page
URL: http://www.create.ucsb.edu/squeak/