UNIVERSIDADE ESTADUAL DE CAMPINAS

FACULDADE DE ENGENHARIA ELÉTRICA

DEPARTAMENTO DE ENGENHARIA DE COMPUTAÇÃO E AUTOMAÇÃO INDUSTRIAL

SISTEMA INTERATIVO DE MODELAGEM E SIMULAÇÃO DE SISTEMAS FLEXÍVEIS DE MANUFATURA – SISMA

Autor : Ricardo Guimarães Borba

Orientador : Prof. Dr. Fernando Antônio Campos Gomide

Coorientador : Prof. Dr. Maurício F. Magalhães

Tese de Mestrado apresentada à Faculdade de Engenharia Elétrica da UNICAMP como Parte dos Requisitos para obtenção do Grau de Mestre em Engenharia Elétrica.

1992

Este trabalho é dedicado a meus pais

RUGUEM e IRENICE,

e a GISELDA,

a quem devo um agradecimento muito especial.

AGRADECIMENTOS

à minha família pela dedicação e incentivo durante todos estes anos.

a Giselda Alves Pontes da Silva pela grande ajuda na execução deste trabalho, sugestões e incentivos.

ao Prof. Dr. Fernando Antônio Campos Gomide por todos os ensinamentos e orientação, indispensáveis durante a execução deste trabalho.

ao Prof. Dr. Maurício Ferreira Magalhães pelas sugestões e revisões do texto, além do grande incentivo e amizade.

aos demais professores do Departamento de Computação e Automação Industrial que contribuíram para minha formação nestes últimos três anos.

aos engenheiros Paulo Tadamitsu Hosoe e Victor Valenzuela Diaz por todo o apoio e colaboração.

a José Augusto Dantas pelas idéias utilizadas no desenvolvimento da interface com usuário.

aos demais amigos, não citados nominalmente, mas que me apoiaram e me incentivaram durante a realização deste trabalho.

ao CPqD-Telebrás pela disponibilidade dos seus recursos físicos e computacionais após o expediente e finais de semanas.

RESUMO

A atual complexidade dos sistemas de manufatura controlados por computador tornou ineficiente a maioria dos métodos analíticos de análise na prevenção de problemas. Isto tem levado muitas organizações ao caminho da simulação para análise dinâmica destes sistemas antes mesmo de qualquer implementação.

A atividade de pesquisa em simulação é motivada principalmente pelo fato de que os estudos para modelagem de um sistema tendem a ser uma atividade de baixa produtividade, consumindo bastante tempo e trabalho. Na maioria dos casos, o computador é utilizado apenas para execução do modelo, ficando a construção, projeto dos experimentos e análise dos resultados a cargo do usuário.

Neste presente trabalho, apresenta-se um sistema de modelagem e simulação de sistemas flexíveis de manufatura destinado a usuários com pouca experiência em termos de programação. Suas principais características incluem: sistema integrado de modelagem, simulação e análise; interface gráfica com utilização de menus hierárquicos, ícones e suporte a *mouse*; sistema de captura de esquemático para definição de *layout* da célula de manufatura; sistema de parametrização com direção própria; animação gráfica durante a simulação; e ajuda sensível ao contexto.

O sistema foi projetado e implementado seguindo-se a filosofia de Programação Orientada por Objetos (também chamada programação por atores). Os elementos constituintes de uma célula de manufatura foram implementados como objetos autônomos que se comunicam através de mensagens. O controle de sequenciamento das operações destes objetos é efetuado por um núcleo de simulação especialmente projetado para esta aplicação. O sistema foi desenvolvido em Turbo Pascal Orientado por Objetos e se destina a microcomputadores compatíveis com IBM-PC sob ambiente DOS. Foram realizados testes de validação com uma célula de montagem de placas de circuito impresso, caracterizada pela concorrência nas operações de dois robôs. Os resultados demonstraram que o protótipo implementado viabiliza a continuidade do projeto para a obtenção de um sistema com grande potencial em aplicações industriais.

ABSTRACT

Inside the current complexity of the highly automated, computer-controlled manufacturing systems, even the most careful analytical planning sometimes fails to prevent major blunders. This fact has caused organizations to increasingly turn to simulation for dynamic analysis of these systems prior to any implementation.

However, simulation modelling studies tend to be a low productivity work, consuming a great deal of time and effort. Most of the times, the computer is used mainly for execution of the model, while construction, design of experiments and analysis of results are shouldered by the user. These facts are ones of the main motivations of research activities on simulation.

In this work, a modelling and simulation software applied to flexible manufacturing systems and specially designed to non-programmer users is described. The software is an integrated modelling, simulation and analysis system with a graphic user-friendly interface. It has a schematic capture editor for cell layout description, animation of simulation performance, statistical analysis, mouse support and context sensitive help.

The design and implementation of the system were based on the object-oriented paradigm. The elements available for building a manufacturing cell were implemented as autonomous objects which communicate through message passing. Operations sequencing control is performed by a dedicated simulation kernel. The system was written in Object-Oriented Turbo Pascal and runs under DOS on PC-compatible microcomputers. Tests were performed with a printed circuit board assembly application. Results have shown that the implemented prototype can be improved in order to be used as a powerful tool in industrial applications.

Índice

Figuras

Tabelas

Capítulo 1

INTRODUÇÃO

Dentro da complexidade atual dos sistemas automatizados de manufatura controlados por computador, o mais cuidadoso planejamento analítico usualmente falha na prevenção de problemas (na maioria das vezes relacionados a altos prejuízos), tais como robôs que não conseguem manipular determinadas cargas solicitadas, congestionamentos em sistemas de veículos auto-guiados (AGVs), e problemas de desbalanceamento de capacidade entre diferentes partes de uma planta (dimensionamento de buffers, etc.). Tal complexidade tem levado muitas organizações ao caminho da simulação para análise dinâmica destes sistemas antes de qualquer implementação [Sha88].

A atividade de pesquisa em simulação é motivada, principalmente, pelo fato de que estudos para modelagem de um sistema tendem a ser uma atividade de baixa produtividade [Eng85]. O ciclo de vida de uma modelagem tradicional consome bastante tempo e trabalho. O computador é utilizado principalmente para execução do modelo, enquanto que a construção, projeto de experimentos e análise de resultados são realizados pelo usuário. O objetivo do desenvolvimento de novos processos de modelagem é fazer com que esta seja menos caracterizada como "trabalho braçal", além de tornar possível que engenheiros de produção efetuem estes estudos de maneira simples e correta, sem um treinamento prévio muito elaborado.

O propósito de um modelo de simulação é o de integrar conjuntos de objetos desconectados em um todo coerente, a fim de mostrar os efeitos do intercâmbio de seus relacionamentos. A base de conhecimento sugerida é a extensão lógica deste conceito. Pesquisas atuais indicam que as técnicas de programação orientada por objetos podem ser perfeitamente utilizadas para facilitar o armazenamento e recuperação de modelos e módulos, para integração em diferentes configurações apropriadas a determinados problemas. Um dos propósitos de tal sistema seria o de assistir ao usuário na síntese e integração de instâncias para criação de um modelo adequado.

O uso de gráficos para entrada de informações pelo usuário vem ganhando cada vez mais popularidade ([Cox87], [Goo87]). Escolhendo e posicionando ícones pré-definidos e respondendo a perguntas feitas pelo computador, o usuário pode definir um sistema de manufatura de uma maneira relativamente rápida. Sistemas de simulação como o SIMFACTORY [Sha88] têm demonstrado que é possível definir sistemas complexos de manufatura sem a necessidade de qualquer programação no sentido estrito da palavra.

A animação gráfica do desempenho da simulação é parte integrante da execução do programa. O usuário deve ter a opção de parar a execução em qualquer momento e observar os resultados naquele ponto, e assim terminar a simulação, modificá-la ou continuá-la. Ao final do

experimento, o sistema deve executar determinadas análises estatísticas baseado em resultados requisitados pelo usuário, que será então capaz de gerar representações gráficas de alta resolução dos resultados, ou até de gerar um *playback* da simulação animada.

Há basicamente duas maneiras nas quais os gráficos podem ajudar na simulação: (a) facilitando a definição de modelos e a sua depuração; e (b) mostrando e ajudando a entender os resultados da simulação. Quanto à animação, sua maior vantagem consiste na depuração preliminar e na apresentação do modelo ao corpo gerencial da indústria. A animação torna clara e imediata o que de outra maneira seria uma seca e, às vezes, obscura apresentação de tabelas e figuras, aumentando a confiabilidade do modelo desenvolvido.

1.1 Objetivos do Trabalho

Pretende-se com este trabalho obter uma especificação de requisitos que permita a implementação de um simulador de sistemas flexíveis de manufatura (SFM) altamente interativo, visando usuários com pouca experiência em termos de programação. Este trabalho inclui a implementação de um protótipo deste simulador, compreendendo os módulos básicos para interface, modelagem e simulação.

1.2 Contexto do Simulador a ser Desenvolvido

Pode-se caracterizar os diversos tipos de simuladores de eventos discretos existentes na literatura e aplicáveis a sistemas de manufatura em três classes, descritas a seguir.

Classe 1

Nesta classe estão agrupadas as linguagens de uso geral para simulação de sistemas de eventos discretos como SIMULA [Dah66], SIMSCRIPT [Fis78], GPSS [Gor75] e SIMAN [Peg85], entre outras. A modelagem de sistemas de manufatura só é conseguida com profundos conhecimentos da linguagem e com extensivo trabalho de programação.

Classe 2

Os simuladores pertencentes a classe 2 procuram integrar ferramentas de interface homem-máquina comercialmente disponíveis com linguagens de simulação da classe 1. Exemplos de sistemas de simulação classe 2 são o GDSS [Goo87] e o sistema para simulação de SFM em ambiente PC apresentado por Zimmers e Valenzuela em [Zim87]. A estrutura de tais sistemas é ilustrada na Fig. 1.1.

O bloco Interface com Usuário manipula os dados de entrada. O Modelador de Sistemas traduz os dados gerados pelo bloco anterior em um padrão reconhecido pelo simulador. O bloco Simulador, que pode ser qualquer uma das linguagens comerciais da classe 1, simula o modelo e gera relatórios que podem ser analisados pelo usuário. O GDSS, por exemplo, utiliza um ambiente CAD[1] de uso genérico para executar as funções de interface com usuário, um módulo

1. *Computer Aided Design*

escrito em FORTRAN como modelador e a linguagem MAP/1 para simulação do modelo. Já em [Zim87], é apresentada uma interface com usuário escrita em LOTUS 123, um modelador também escrito em FORTRAN e a linguagem SIMAN para simulação.

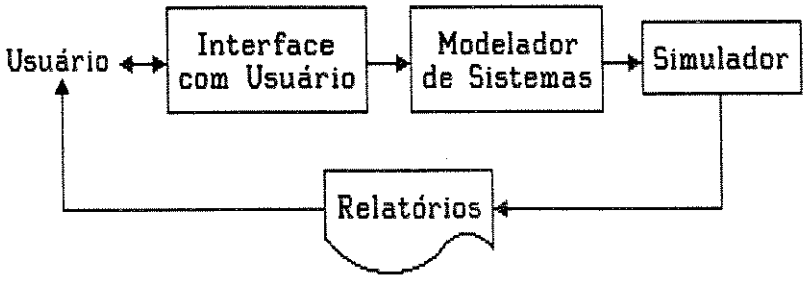

Figura 1.1: Simuladores Classe 2

Classe 3

A classe 3 de simuladores introduz novos conceitos ao ambiente de simulação de sistemas de manufatura. Com todas as funções integradas em um só pacote, estes sistemas fazem parte de uma nova geração de sistemas de modelagem e simulação [Sha88]. Tais sistemas tendem a utilizar interfaces modernas e interativas utilizando gráficos, diálogos via linguagem natural e menus hierárquicos, além de se estruturarem sobre bases de conhecimento e aplicarem técnicas de inteligência artificial. Um dos principais esforços desta classe de simuladores é direcionado para poupar o usuário da tarefa de modelagem, deixando-o mais livre para o trabalho de análise sistêmica.

As características do sistema de modelagem e simulação apresentado neste trabalho permitem situá-lo nesta última classe de simuladores.

1.3 Organização do Trabalho

Esta dissertação está organizada em 5 capítulos. O capítulo 1 (Introdução) enfoca o escopo, o objetivo e a estrutura da tese. No capítulo 2 (Fundamentos da Simulação), é abordada a simulação de sistemas, englobando seus conceitos fundamentais e sua evolução a partir da aplicação de inteligência artificial em simuladores. No capítulo 3 (Projeto do Simulador), é descrito o SISMA — Sistema Interativo de Modelagem e Simulação de Sistemas Flexíveis de Manufatura — implementado neste trabalho. No capítulo 4 (Resultados de Simulação), é apresentada a metodologia de testes e os resultados obtidos. Por fim, no capítulo 5 (Discussão e Conclusões), são discutidos os resultados apresentados no capítulo 4 e as aplicações do sistema desenvolvido, além da apresentação de sugestões para trabalhos futuros.

Capítulo 2

FUNDAMENTOS DA SIMULAÇÃO

2.1 Introdução

Apresentam-se neste capítulo os principais conceitos e terminologias sobre simulação de sistemas seguindo-se a abordagem mais específica de duas classes de sistemas: sistemas de tempo contínuo e sistemas de eventos discretos. Como decorrência, conclui-se que, para sistemas de manufatura, os modelos de sistemas de eventos discretos são os mais adequados à simulação. Segue a descrição detalhada deste tipo de sistema, enfatizando-se o desenvolvimento de sistemas especialistas e a aplicação de inteligência artificial em simuladores. Como tópicos mais específicos, será dada uma visão geral da programação orientada por objetos e sua aplicação em simulação de sistemas de manufatura. Por fim, serão analisados os benefícios que a animação gráfica traz para a simulação de sistemas de manufatura.

2.2 Modelos e Simulação de Sistemas

2.2.1 Conceitos

A utilização do termo sistema em uma grande variedade de contextos dificulta a escolha de uma definição geral o bastante para cobrir seus vários usos e, ao mesmo tempo, concisa o suficiente para fazê-la servir a um propósito útil ([Cho65] e [Gor78]). Começar-se-á portanto com uma definição simples, expandindo-a a seguir com a introdução de alguns termos comumente usados na discussão de sistemas.

Um **sistema** é definido como uma coleção de objetos, unidos por interações ou interdependências geralmente regulares. Esta definição é abrangente o bastante para incluir sistemas estáticos, mas o principal interesse neste trabalho serão os sistemas dinâmicos, onde as interações causam mudanças no decorrer do tempo.

O termo **entidade** será usado para denotar um objeto de interesse em um sistema. O termo **atributo** denotará uma propriedade da entidade. Pode haver, claramente, muitos atributos para uma só entidade. Qualquer processo que causa mudanças no sistema será chamado **atividade**. O termo **estado do sistema** será usado para representar a descrição de todas as entidades, atributos e atividades em qualquer instante de tempo. O progresso do sistema é estudado seguindo-se as mudanças no estado do sistema.

Para melhor fixar estes conceitos, serão utilizados dois exemplos clássicos de sistema. Como exemplo de um sistema conceitualmente simples, será considerado o sistema de controle de nível de líquido [Oga70] da Fig. 2.1(a). Um controlador automático mantém o nível do líquido no tanque comparando o nível real com o desejado, corrigindo qualquer erro através do ajuste da abertura da válvula pneumática[2]. A Fig. 2.1(b) apresenta um diagrama de blocos do sistema de controle.

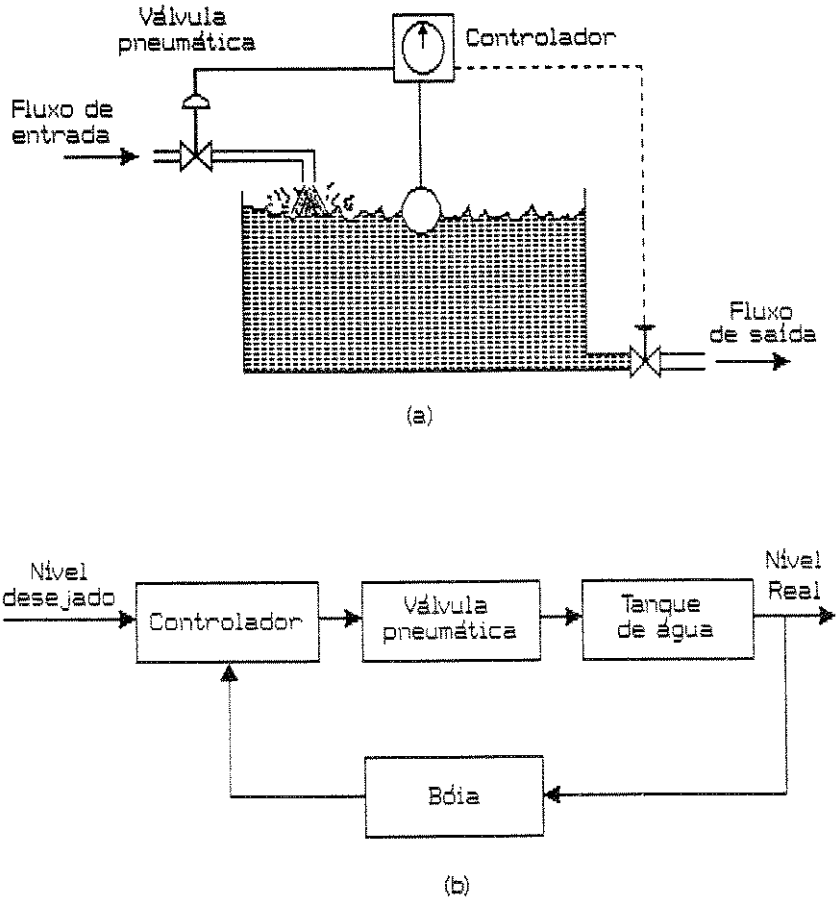

Figura 2.1: (a) Sistema de controle de nível de líquido; (b) Diagrama de blocos.

Como um segundo exemplo, considera-se uma fábrica que produz peças para montagem de um determinado produto (Fig. 2.2). Os dois maiores componentes do sistema são o departamento de fabricação, que produz as peças, e o departamento de montagem, que monta os produtos. Um departamento de compras mantém o suprimento de acordo com uma lista de material e o

2. Para prevenir transbordamento em caso de falha desta válvula, pode ser permitido ao controlador, em casos de emergência, agir diretamente sobre uma válvula de saída de modo a forçar uma vazão adicional do líquido armazenado.

departamento de expedição libera os produtos acabados. Um departamento de controle da produção recebe pedidos e designa tarefas a todos estes departamentos.

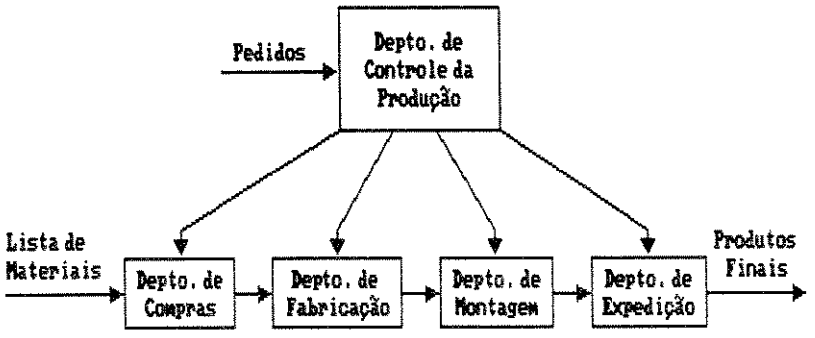

Figura 2.2: Um sistema de fabricação

Observando estes sistemas, vê-se que há distintos objetos, cada um possuindo propriedades de interesse. Há também certas interações ocorrendo entre os objetos, que provocam mudanças no sistema.

Na descrição do sistema de controle de nível de líquido, as entidades do sistema são o tanque de água, a válvula pneumática, e o controlador. Seus atributos são nível do líquido, abertura da válvula e a configuração do controlador. As atividades são a operação de abertura e fechamento da válvula pneumática e a diminuição ou aumento do fluxo de entrada. No sistema da fábrica, as entidades são os departamentos, pedidos, peças e produtos. As atividades são os processos de manufatura dos departamentos. Os atributos são fatores como quantidade referente a cada pedido, tipo de peça ou número de máquinas em um departamento.

A Fig. 2.3 apresenta exemplos adicionais do que poderiam ser considerados entidades, atributos e atividades para alguns outros sistemas. Considerando-se o movimento de carros em um sistema de tráfego, cada carro é tomado como uma entidade, tendo como atributos sua velocidade e distância percorrida. Entre as atividades, está a de dirigir. No caso de um sistema bancário, os clientes do banco são entidades com saldos e créditos como atributos. Uma atividade típica seria a ação de fazer um depósito.

EXEMPLOS DE			
SISTEMA	ENTIDADES	ATRIBUTOS	ATIVIDADES
Tráfego	Carros, Ruas,...	Velocidade, Distância,...	Estacionar, Dirigir,...
Banco	Clientes, Agências,...	Saldos, Créditos,...	Depositar, Sacar,...
Comunicações	Mensagens, Transmissores,...	Tamanho, Prioridade,...	Transmitir, Receber,...
Supermercado	Clientes, Fornecedores,...	Lista de compras,...	Pagar, Vender,...

Figura 2.3: Exemplos de sistemas e suas entidades, atributos e atividades.

A Fig. 2.3 não apresenta uma lista completa de todas as entidades, atributos e atividades para os sistemas. De fato, uma lista completa não pode ser feita sem o conhecimento do propósito da descrição do sistema. Dependendo de tal propósito, vários aspectos do sistema serão de interesse e determinarão o que precisa ser identificado.

2.2.2 Ambiente

Um sistema é frequentemente afetado por mudanças ocorridas externamente. Tais mudanças são ditas ocorrerem no **ambiente do sistema**. Um passo importante na modelagem é decidir sobre os limites entre o sistema e seu ambiente. Esta decisão deve estar associada ao propósito do estudo.

No caso do sistema de fabricação, por exemplo, os fatores que controlam a chegada de pedidos podem ser considerados como externos à influência da fábrica e portanto parte do ambiente. Entretanto, se for considerado o efeito do atendimento destas demandas, existirá uma relação entre a produção da fábrica e a chegada de pedidos. Esta relação deve ser considerada uma atividade do sistema.

O termo **endógeno** é usado para descrever atividades que ocorrem dentro do sistema e o termo **exógeno** é usado para descrever atividades no ambiente que afetam o sistema. Um sistema para o qual não há atividade exógena é dito ser um sistema **fechado**, em contraste com um sistema **aberto**, que tem atividades exógenas.

2.2.3 Atividades Estocásticas

Uma outra distinção feita entre atividades depende da maneira pela qual elas podem ser descritas. Quando o comportamento de uma atividade é descrito completamente em termos de sua entrada, a atividade é dita ser **determinística**. Se os efeitos da atividade variarem aleatoriamente sobre vários comportamentos possíveis, a atividade é denominada **estocástica**.

A aleatoriedade de uma atividade estocástica parece implicar que a atividade é parte do ambiente do sistema, visto que seu estado exato em um dado instante não é conhecido. Entretanto, a saída aleatória pode frequentemente ser medida e descrita na forma de uma distribuição de probabilidade. Neste caso, a atividade não constitui parte do ambiente. Tomando-se como exemplo o caso da fábrica, o tempo gasto por uma determinada operação de uma máquina pode ser descrito por uma distribuição de probabilidade, sendo esta operação considerada uma atividade endógena. Por outro lado, pode haver falhas na alimentação em intervalos de tempo aleatórios. Estas falhas serão resultados de uma atividade exógena.

A utilização de um conjunto de dados para representar um modelo frequentemente envolve um elemento de incerteza advindo do erro experimental ou de amostragem. Um valor para algum atributo de um modelo, conhecidamente fixo, deve ser selecionado de um conjunto de valores armazenados que contêm erros aleatórios. Decidir qual a melhor estimativa é um exercício estatístico. Na maioria das vezes, uma média aritmética é considerada acurada o suficiente.

2.2.4 Modelagem de Sistemas

A análise de desempenho de sistemas, em alguns casos, pode ser efetuada a partir de experiências com o próprio sistema em questão. Por outro lado, o objetivo de muitos estudos é o de prever desempenho antes da realização final, optando-se nestes casos pela construção de um protótipo. Entretanto, isto pode ser muito caro e consumir bastante tempo. Além disso, mesmo com um sistema já existente é às vezes impossível ou impraticável fazer experimentos reais. Por exemplo, não é possível estudar sistemas econômicos mudando arbitrariamente oferta e procura de produtos. Consequentemente, estudos são efetuados com um **modelo** do sistema, não sendo necessário (nem possível), na maioria dos casos, considerá-lo em todos os seus detalhes. Desta forma, um modelo não é apenas um substituto para o sistema. É também uma simplificação do mesmo.

Define-se um modelo como o corpo de informações sobre o sistema agrupadas com o propósito de estudá-lo. Desde que o propósito do estudo determina a natureza destas informações, não existe nenhum modelo único de sistema. Diferentes modelos de um mesmo sistema podem ser produzidos, dependendo do enfoque dado na sua análise.

A tarefa de derivar um modelo de um sistema pode ser dividida em duas subtarefas: estabelecer a estrutura do modelo e suprir os dados. O estabelecimento da estrutura determina os limites do sistema e identifica suas entidades, atributos e atividades. Os dados provêem os valores que os atributos podem apresentar e definem as relações envolvidas nas atividades.

2.2.5 Tipos de Modelos

Os modelos utilizados no estudo de sistemas são classificados de várias maneiras. As classificações utilizadas são ilustradas na Fig. 2.4. Os modelos serão primeiramente separados em **físicos** e **matemáticos**.

Os modelos físicos são baseados em alguma analogia como, por exemplo, entre os sistemas mecânico e elétrico, ou elétrico e hidráulico. Em um modelo físico, os atributos do sistema são representados por medidas como tensão ou velocidade. As atividades do sistema são refletidas nas leis físicas que regem o modelo. Por exemplo, a taxa pela qual o eixo de motor de CC gira depende da tensão aplicada no motor. Se a tensão aplicada é utilizada para representar a velocidade de um veículo, então o número de giros do eixo é a medida da distância que o veículo percorreu. Quanto maior a tensão, ou velocidade, maior é o total de giros, ou distância percorrida, em um dado instante de tempo.

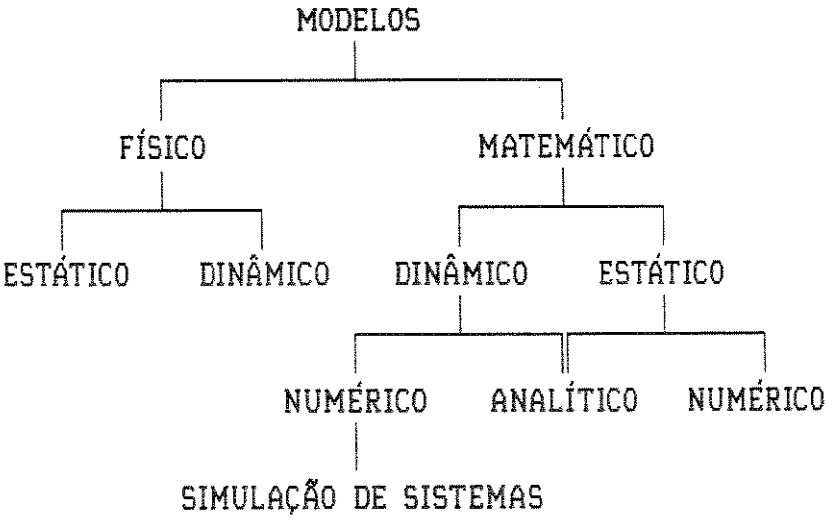

Figura 2.4: Tipos de Modelos

Os modelos matemáticos utilizam notação simbólica e equações matemáticas para representar um sistema. Os atributos são representados por variáveis e as atividades, por funções matemáticas que se interrelacionam com variáveis.

A segunda distinção é entre modelos **estáticos** e **dinâmicos**. Os modelos estáticos podem mostrar apenas os valores que os atributos do sistema possuem quando em equilíbrio. Os modelos dinâmicos, por outro lado, seguem as mudanças causadas pelas atividades do sistema.

No caso de modelos matemáticos, uma terceira distinção é feita entre métodos **analíticos** e **numéricos**. Aplicar técnicas analíticas significa usar o raciocínio dedutivo da teoria matemática para resolver o modelo. Na prática, apenas certas formas de equações podem ser resolvidas. Usar

técnicas analíticas, portanto, é uma questão de achar o modelo que possa ser solucionado e que seja mais adequado ao sistema em estudo. Por outro lado, métodos numéricos envolvem aplicacão de procedimentos computacionais para solucionar equações.

A simulação de sistemas é considerada uma técnica computacional numérica utilizada em conjunto com modelos matemáticos dinâmicos, como ilustrado na Fig. 2.4.

O sistema de controle de nível de água e a fábrica utilizados como exemplos na seção 2.2.1 respondem a mudanças ambientais de diferentes maneiras. O aumento do nível de água no tanque ocorre suavemente, enquanto que as mudanças na fábrica ocorrem descontinuamente. Pedidos de lista de material ou finalização da montagem de um produto, por exemplo, ocorrem em pontos específicos no tempo.

Sistemas como o de controle de nível, para os quais as mudanças são predominantemente suaves, são chamados **sistemas contínuos**. Sistemas como a fábrica, na qual as mudanças são predominantemente descontínuas, são chamados **sistemas discretos**. Poucos são os sistemas que são completamente contínuos ou discretos. O sistema de controle de nível, por exemplo, pode fazer ajustes discretos quando o volume de água se altera, enquanto que, no exemplo da fábrica, as máquinas trabalham continuamente, mesmo que no início e no final de um trabalho ocorram mudanças discretas. Entretanto, na maioria dos sistemas, um tipo de mudança predomina, de maneira que os sistemas podem ser geralmente classificados como sendo contínuos ou discretos.

Há também sistemas que são intrinsicamente contínuos, mas as informações sobre eles são disponíveis apenas em pontos discretos no tempo. Estes são chamados **sistemas amostrados**. O estudo de tais sistemas inclui a determinação dos efeitos da amostragem discreta, especialmente quando a intenção é a de controlar o sistema baseado na informação obtida pela amostragem.

A ambiguidade na representação de um sistema ilustra que a descrição de um sistema, e não a natureza deste, determina que tipo de modelo será usado. Uma distinção precisa ser feita porque os métodos de programação usados para simulação de modelos contínuos e discretos diferem. Entretanto, nenhuma regra específica pode ser dada de como um sistema particular deva ser representado. O propósito do modelo, juntamente a um princípio geral de simplicidade, determinará o nível de detalhes e a precisão com os quais o modelo necessita ser desenvolvido. Pesando estes fatores e planejando de acordo com a experiência de especialistas se decidirá o tipo de modelo mais adequado.

Embora seja possível suavizar as flutuações de um sistema essencialmente discreto, a ponto de um modelo contínuo poder ser aplicado, existem muitos sistemas que devem ser representados por modelos discretos, se algum grau de realismo tiver de ser mantido. As atividades de tal sistema podem ser dependentes do estado exato do mesmo, a ponto de não se poder suavizar ou abstrair médias. Modelos de sistema de manufatura a nível de chão de fábrica, sistemas de comutação e sistemas de circuitos digitais são exemplos de tais sistemas.

2.2.6 Simulação de Sistemas Contínuos

Um sistema contínuo é aquele no qual as atividades predominantes causam mudanças suaves nos atributos das entidades do sistema. Quando tal sistema é modelado matematicamente, as variáveis do modelo que representam os atributos são controladas por funções contínuas. Um modelo de sistema contínuo pode descrever as relações entre os atributos do sistema na forma de equações algébricas lineares. De maneira mais geral, em sistemas contínuos, os relacionamentos descrevem as taxas nas quais os atributos mudam, de tal forma que o modelo consiste de equações diferenciais não lineares.

Os modelos mais simples têm uma ou mais equações diferenciais lineares com coeficientes constantes. Desta forma, é frequentemente possível resolver o modelo sem o uso de simulação. Mesmo assim, o trabalho envolvido pode ser tão grande que é preferível utilizar técnicas de simulação. Entretanto, quando não-linearidades são introduzidas no modelo, frequentemente torna-se impossível ou, pelo menos, muito difícil resolvê-los. Por outro lado, os métodos de simulação para resolução de modelos não mudam fundamentalmente quando não-linearidades ocorrem. Os métodos de aplicação de simulação para modelos contínuos podem, portanto, ser desenvolvidos aplicando-os a modelos onde as equações diferenciais são lineares e têm coeficientes constantes, e então generalizando-os para equações mais complexas.

2.2.6.1 Técnica Computacional Numérica para Modelos Contínuos

Para ilustrar a técnica numérica geral de simulação baseada em modelo contínuo, considera-se o seguinte exemplo [Gor75]. Um construtor observa que a taxa na qual vende casas depende diretamente do número de famílias que ainda não possuem casas. À medida em que o número de pessoas sem casa diminui, a taxa de vendas cai. Seja **H** o número potencial de casas construídas e **y** o número de famílias com casas. A situação é representada na Fig. 2.5. A linha horizontal em **H** é o mercado potencial total de casas. A curva **y** indica como o número de casas vendidas aumenta com o tempo. A derivada da curva diminui à medida em que (**H** – **y**) se torna menor. Isto reflete a desaceleração das vendas quando o mercado se torna saturado.

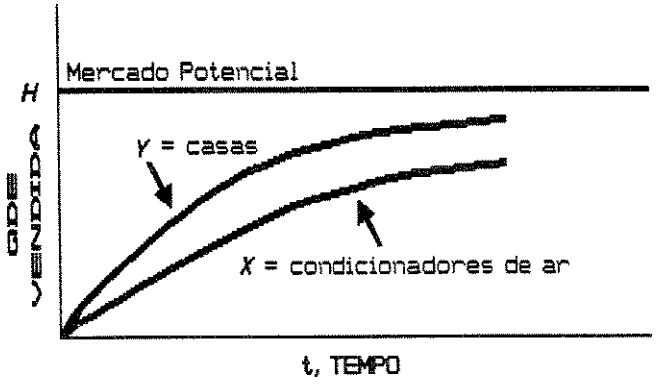

Figura 2.5: Exemplo de Modelo Contínuo: Vendas

Matematicamente, a tendência pode ser expressada pela equação

$$\frac{\delta y}{\delta t} = k_1(H - y), \qquad y = 0 \text{ em } t = 0$$

Considera-se agora um fabricante de condicionadores de ar centrais projetados para casas. Sua taxa de vendas depende do número de casas construídas (por simplicidade, assume-se que todas as casas instalarão um condicionador de ar). Assim como a venda de casas, a taxa de vendas diminui à medida em que o mercado ainda não preenchido diminui.

Seja **x** o número de condicionadores de ar instalados. O mercado não preenchido é a diferença entre o número de casas e o número de condicionadores de ar instalados. A tendência das vendas pode ser representada matematicamente pela expressão

$$\frac{\delta x}{\delta t} = k_2(y - x), \qquad x = 0 \text{ em } t = 0$$

A mudança de **x** com o tempo é também ilustrada na Fig. 2.6. As duas equações constituem um modelo do crescimento das vendas de condicionadores de ar. De fato, devido à sua simplicidade, é possível resolver o modelo analiticamente.

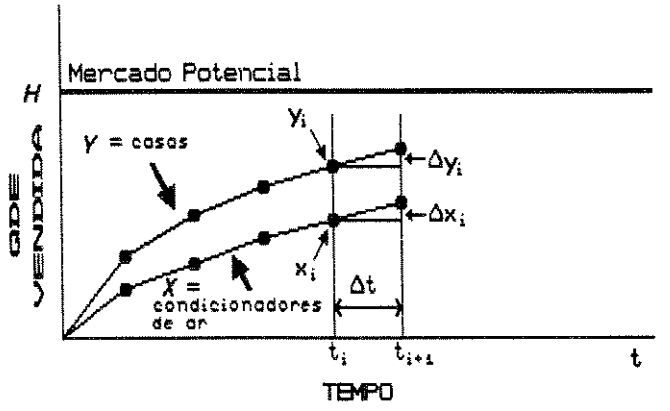

Figura 2.6: Cálculo para o Modelo de Vendas

Entretanto, ele se torna rapidamente insolúvel quando expandido para se tornar mais representativo das condições reais de mercado. O potencial de mercado de casas, por exemplo, pode não ser estável. Ele poderia crescer com o aumento da população ou flutuar de acordo com a situação econômica. Os coeficientes que determinam as taxas de crescimento poderiam também ser influenciados pela quantidade de dinheiro gasto em propaganda, e assim por diante, complicando consideravelmente o modelo.

O modelo simples, entretanto, servirá para ilustrar os métodos gerais aplicados à simulação contínua. A técnica de simulação deve computar a saída passo-a-passo. Supõe-se que o processamento é feito em intervalos uniformes de tempo e que os cálculos já tenham progredidos até o tempo t_i, quando as duas variáveis do problema tiverem os valores y_i e x_i. A Fig. 2.6 mostra o próximo passo neste cálculo.

O cálculo se adianta em intervalos δt para $t_{i+1} = t_i + \delta t$. As taxas de venda são consideradas constantes durante cada intervalo. Estas taxas podem ser interpretadas como a variação por unidade de tempo. Ou seja,

$$\text{taxa de variação de } y = \frac{\delta y_i}{\delta t}$$

$$\text{taxa de variação de } x = \frac{\delta x_i}{\delta t}$$

Das equações do modelo, isto pode ser escrito como

$$\delta y_i = k_1(H - y_i)\delta t$$

$$\delta x_i = k_2(y_i - x_i)\delta t$$

Desde que y_i e x_i são conhecidos, basta obter os valores de y e x no instante t_{i+1}. Entretanto, nota-se que a equação para δy_i deve ser resolvida primeiro para se obter o valor de y_i necessário na equação para x_i.

A repetição do cálculo utilizando os novos valores de y e x produzem a saída no final do próximo intervalo. Como ilustrado na Fig. 2.6, o cálculo é equivalente a calcular a inclinação em cada ponto e projetar um pequena reta neste ponto com esta mesma inclinação. A saída da simulação é uma série de segmentos de linha, aproximadamente igual à curva contínua que representa a verdadeira saída do modelo.

O método descrito acima é uma maneira muito simples de integrar equações diferenciais numericamente. Entretano, não é um método muito acurado, a menos que pequenos passos sejam utilizados. Há outras maneiras muito mais precisas, e frequentemente mais eficientes, de integração numérica que não se baseiam simplesmente no último valor conhecido das variáveis [Mar69]. Ao invés disso, elas utilizam vários valores anteriores para predizer a taxa na qual as variáveis estão mudando. Além disso, o intervalo computacional é frequentemente ajustado em tamanho para se adequar a esta taxa.

Existem muitos sistemas de programação disponíveis que incorporam linguagens de simulação de sistemas contínuos. Elas normalmente incluem um certo número de métodos computacionais que podem ser selecionados pelo usuário.

Devido às características deste modelo, torna-se um pouco difícil a sua utilização em sistemas, em especial o de manufatura, onde os eventos ocorrem em pontos dicretos no tempo, e não de maneira suave. Os modelos de sistemas contínuos têm, entretanto, grandes aplicações em, por exemplo, circuitos eletrônicos analógicos, controle de processos, reações químicas, testes balísticos, simuladores de vôo, etc.

2.2.7 Simulação de Sistemas de Eventos Discretos

Modelos de sistemas discretos foram caraterizados na seção 2.2.5 como modelos nos quais ocorrem mudanças descontínuas de estado. O termo evento é usado para representar tal mudança ocorrida em um instante específico de tempo. Um evento pode alterar o valor de um atributo de uma entidade, criar ou destruir uma entidade, iniciar ou terminar uma atividade.

Desde que a técnica de simulação consiste em acompanhar as mudanças em um modelo de sistema, a tarefa de simular sistemas discretos requer um programa que consiga construir a sequência de eventos. Registros de todas as atividades em andamento e das entidades envolvidas devem ser mantidos e periodicamente atualizados para refletir a sequência de eventos. Para isto, os instantes de tempo de todos os eventos devem ser registrados e procedimentos devem computar futuros eventos à medida em que a simulação é efetuada.

A passagem do tempo é registrada como um número denominado relógio. O relógio é inicialmente zero e, à medida que a simulação prossegue, é atualizado para refletir a passagem do tempo. Na seção 2.2.7.2, serão analisados com maiores detalhes a representação do tempo na simulação de sistemas discretos.

Como um exemplo, considera-se um setor de máquinas onde partes chegam para serem processadas. Cada parte é de um tipo escolhido aleatoriamente dentro de um certo número de tipos. O padrão de chegada pode ser descrito como o intervalo de tempo entre chegadas. Este intervalo também pode variar aletoriamente. O tempo de processamento de uma parte também poderia ser uma variável aleatória. Aparentemente, com temporizações aleatórias, as chegadas e as liberações das partes não necessariamente se alternam. Entretanto, se há mais do que uma máquina trabalhando, de maneira que várias partes possam ser operadas simultaneamente, pode acontecer das partes serem finalizadas fora da ordem na qual chegaram. Isto se deve à variação dos tempos de operação das máquinas. Se uma fila de espera se forma na entrada do setor de máquinas, o evento referente à finalização de uma parte implica no início do processamento de outra parte, de maneira que um evento é condicional de outro.

Uma simulação deve manter registros de todos estes tipos de eventos, organizá-los cronologicamente, e estar ciente de todas as dependências condicionais. A simulação deve acompanhar os efeitos dos eventos sucessivos, produzindo resultados que consigam ilustrar a variação do número de chegadas e partidas das partes. Os resultados são normalmente resumidos em termos das distribuições e médias dos tempos de espera e dos tamanhos da fila de espera. Na próxima seção, será demonstrado com um exemplo prático a maneira de se controlar uma simulação de sistema discreto.

2.2.7.1 Técnica Computacional Numérica para Modelos Discretos

Para ilustrar a técnica computacional geral de simulação com modelos discretos, considera-se o seguinte exemplo. Um operador começa seu dia de trabalho com uma pilha de partes a serem processadas, onde o tempo de operação de cada parte é variável. Ele inicia o processamento de uma parte tão logo termine a parte anterior. Este procedimento se interrompe por um intervalo de cinco minutos de descanso se, ao terminar uma operação, tenha-se passado uma hora ou mais desde o início do trabalho, ou desde o último intervalo. Assume-se que os tempos de operação das partes são conhecidos. Mantém-se também uma contagem de quantas partes ainda restam. O contador, denotado por **N**, deve ser inicialmente programado com o número de partes no início do dia, e, neste modelo simples, assume-se que nenhuma parte nova chega durante o dia. O contador será decrementado para cada operação concluída, e o trabalho estará terminado quando o contador chegar a zero.

O cálculo computacional envolvido pode ser organizado como mostrado na Tabela 2.1. A i-ésima linha corresponde à i-ésima parte. A primeira coluna enumera as partes. Há então quatro colunas contendo vários tempos, medidos em minutos a partir do tempo zero. A segunda coluna mostra o instante de tempo em que o operador começa a processar a parte, denotado por $\mathbf{t_b(i)}$. A terceira coluna mostra o tempo necessário para operar a parte, denotado por $\mathbf{t_w(i)}$, e a quarta coluna mostra o tempo em que cada parte foi terminada. A quinta coluna contém o tempo acumulado desde o início do trabalho ou desde o último intervalo para descanso, medido no instante em que cada operação é completada. Este tempo é denotado por $\mathbf{t_c(i)}$. Há ainda uma sexta coluna que contém um indicador, denotado por **F**, que assume o valor *um* se o operador deve descansar após a i-ésima parte, e *zero* caso contrário. O operador trabalha até que não existam mais partes a serem processadas, ou quando o tempo de trabalho tenha ultrapassado certo limite.

Tabela 2.1: Simulação do Processamento de Partes

Número da Parte	Início	Tempo de Operação	Fim	Tempo Acumulado	Indicador de Intervalos	Nr. de Partes
i	t_b	t_w	t_f	t_c	F	N
1	0	45	45	45	0	57
2	45	16	61	61	1	56
3	66	5	71	5	0	55
4	71	29	100	34	0	54
5	100	33	133	67	1	53
6	138	25	163	25	0	52
7	163	21	184	46	0	51

A simulação prossegue linha após linha, da esquerda para a direita. A primeira linha mostra que a primeira parte começa a ser processada no instante de tempo zero. O tempo de operação é de 45 minutos. Assim a operação termina em 45 com um tempo acumulado também de 45. Ainda não é hora para descanso, portanto ao indicador F é atribuído zero. O contador, que era inicialmente de 57 partes, cai para 56.

Devido ao indicador ser zero e o contador ser maior que zero, a operação sobre a segunda parte é iniciada em 45. Precisa-se de 16 minutos para processá-la, o que leva a um tempo acumulado de 61 minutos. Desta forma ao indicador é atribuído 1, o que indica um descanso de cinco minutos para o operador. Assim, a terceira parte começa a ser processada em 66. A simulação continua desta maneira até que o contador de partes a serem processadas, **N**, chegue a zero, ou o tempo de finalização, t_f, alcance algum limite que represente o fim do período de trabalho.

2.2.7.2 Representação do Tempo

A representação do tempo é registrada por uma entidade denominada **relógio**. A esta entidade normalmente é atribuído o valor zero no início da simulação, seguindo indicação de quantas unidades de tempo simulado se passaram desde o início da simulação. A menos que especificado o contrário, o termo **tempo de simulação** significa o tempo indicado pelo relógio, e não o tempo que o computador levou para executar a simulação. Não há relação direta entre estes dois tempos. O fator principal que determina o tempo de computação é o número de eventos que ocorrem. Dependendo da natureza do sistema simulado, e dos detalhes sob o qual é modelado, a razão tempo simulado sobre tempo real pode variar bastante. Por exemplo, uma simulação de um circuito eletrônico digital onde os eventos ocorrem em frações de nanossegundos, mesmo se executada por um supercomputador, poderia facilmente durar milhares de vezes mais do que a operação no sistema real. Por outro lado, para a simulação de um sistema econômico, onde os eventos estão agregados a ocorrer uma vez ao ano, centenas de anos de operação podem ser facilmente simulados em alguns segundos de CPU.

Dois métodos básicos existem para atualizar o relógio do sistema. Um método é o de avançar o relógio para o instante em que o próximo evento deverá ocorrer. O outro método é o de avançar o relógio de intervalos pequenos (normalmente uniformes) e determinar, a cada intervalo, se algum

evento deve ocorrer neste instante. O primeiro método é chamado de **orientado por evento**, e o segundo método, de **orientado por intervalo**. Simulações de sistemas discretos utilizam geralmente o método orientado por evento, enquanto que simulações de sistemas contínuos geralmente usam o método orientado por intervalo. Apesar disso, não há nada que determine a maneira pela qual o tempo deva ser representado em simulações de sistemas discretos e contínuos. Um programa orientado por intervalos detectará mudanças discretas e portanto pode simular sistemas discretos, enquanto que um programa orientado por eventos pode ser feito para seguir mudanças contínuas, introduzindo artificialmente eventos que ocorrem em intervalos de tempo regulares. E também, o método orientado por evento não é necessariamente mais rápido que o método orientado por intervalos para sistemas discretos [Gor78].

2.2.7.3 Tarefas de um Programa de Simulação

Tendo demonstrado com um exemplo simples a maneira pela qual uma simulação discreta se processa, é possível agora delinear as tarefas envolvidas em um programa para simulação. Há três tarefas principais a serem desempenhadas, como mostrado na Fig. 2.7. A primeira é gerar um modelo e iniciá-lo. Da descrição do modelo, um conjunto de números devem ser criados para representar o estado do sistema. Este conjunto de números é chamado **imagem do sistema**, desde que seu propósito é o de refletir o estado do sistema em todos os instantes. As atividades do sistema devem ser representadas como rotinas que criam os eventos discretos fazendo mudanças na imagem do sistema. A segunda tarefa é a de programar o procedimento que executa o ciclo de ações envolvidas no andamento da simulação. Este procedimento é referido como **algoritmo de simulação**. A terceira tarefa é a geração de um relatório de saída. As estatísticas acumuladas durante a simulação normalmente serão organizadas por um gerador de relatórios.

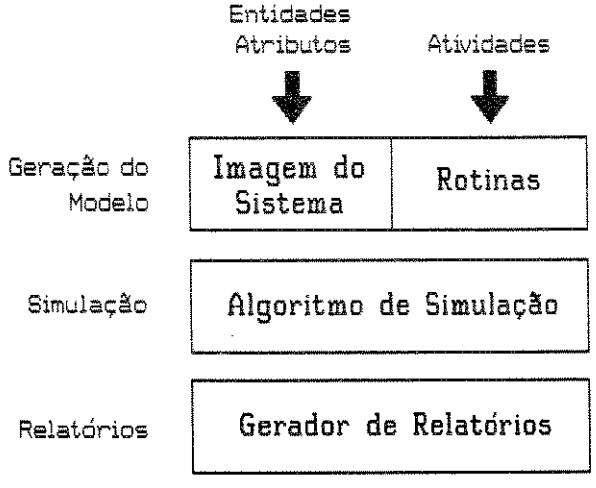

Figura 2.7: Tarefas de um Programa de Simulação

O fluxo geral de controle durante a execução de um programa de simulação é ilustrado na Fig. 2.8. No topo da figura, está a tarefa de geração do modelo, que é executada uma vez. No final está a tarefa de geração de relatórios, embora seja comum gerar resultados intermediários durante a simulação. A execução da simulação envolve executar repetidamente os seguintes passos :

1. Achar o próximo evento potencial.
2. Selecionar uma atividade.
3. Testar se o evento pode ser executado.
4. Mudar a imagem do sistema.
5. Acumular estatísticas.

Se o teste feito no passo 3 achar que o evento selecionado não pode de fato ser executado, o evento deve ser abandonado. Se entretanto, o evento deve ser executado posteriormente, quando as condições forem propícias, ele é denominado **evento condicional**. Como indicado na Fig. 2.8, alguns registros devem guardar os eventos condicionais. A existência de eventos condicionais deve ser considerada na procura do próximo evento.

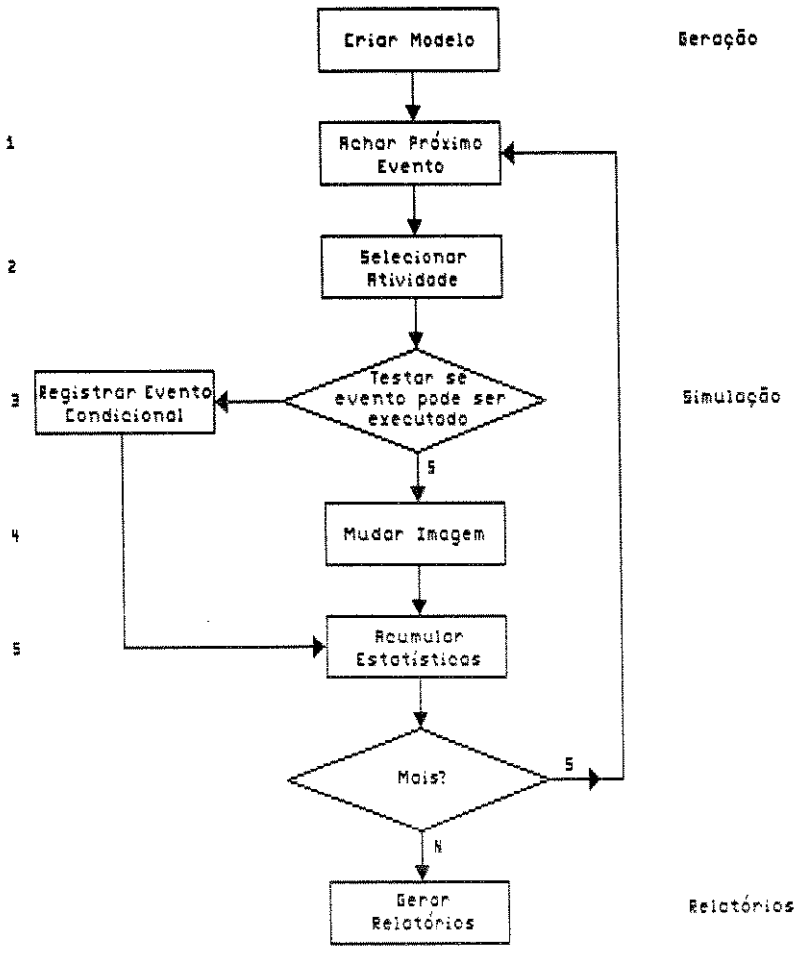

Figura 2.8: Execução de um Algoritmo de Simulação

2.2.7.4 Relatórios Estatísticos

A maioria dos sistemas de simulação inclui um gerador de relatórios para imprimir estatísticas acumuladas durante a simulação. As estatísticas exatas requeridas para um modelo dependem do estudo que esteja sendo feito. Entretanto, há certas estatísticas comuns que normalmente são incluídas no relatório. Entre as estatísticas mais comuns estão :

(a) **Contadores** que indicam o número de entidades de um tipo particular ou o número de vezes que certo evento ocorreu.

(b) **Medições resumo**, tais como valores máximos, médias e desvios padrões.

(c) **Utilização**, definida como a fração (ou percentagem) de tempo em que alguma entidade está sendo utilizada.

(d) **Ocupação**, definida como a fração (ou percentagem) de um grupo de entidades em uso na média.

(e) **Distribuições** de variáveis importantes, tais como tamanhos de fila ou tempos de espera.

(f) **Tempos de trânsito**, definida como o tempo que uma entidade leva para mover-se de um ponto do sistema para outro.

Preferencialmente, estes relatórios deverão ser gerados sob a forma de gráficos e histogramas, o que sem dúvida, torna os resultados da simulação bem mais legíveis e agradáveis ao usuário.

2.2.7.5 Linguagens de Simulação de Sistemas Discretos

Um grande número de linguagens de programação têm sido produzidas para simplificar a tarefa de escrever programas de simulação de sistemas discretos. Uma lista de 23 destas linguagens poderá ser encontrada em [Fis73].

Entretanto, nos últimos anos, novas gerações de simuladores têm sido desenvolvidos com o objetivo de livrar o usuário da tarefa de programação. Estes simuladores utilizam novas técnicas de programação como Inteligência Artificial e Sistemas Especialistas. Estes tópicos serão abordados nas seções subsequentes.

2.3 Aplicação de Inteligência Artificial em Simulação

O uso da simulação de eventos discretos para sistemas de manufatura modernos tem aumentado significativamente nos últimos anos [Sch89]. Existem várias razões para este crescente interesse. Primeiro, as indústrias estão ativamente buscando métodos para aumentar a produtividade, melhorar a qualidade e reduzir custos. Além disso, a simulação está se tornando mais aceitável pelos gerentes e diretores como uma garantia barata contra erros caros.

Uma outra razão para este aumento de interesse em simulação é que muitas das bem conhecidas linguagens de simulação foram transportadas para microcomputador. Por exemplo, o GPSS foi convertido para o PC [Cox87]. Além disso, algumas das novas linguagens de simulação foram introduzidas especificamente para modelar sistemas de manufatura, por exemplo o SIMAN [Peg85]. Um catálogo recente destas linguagens de simulação é apresentada em [SIM86] e [SIM87].

Uma terceira razão para todo este interesse em simulação tem sido a adição de capacidades de animação, com gráficos muito bem elaborados em muitas das linguagens de simulação [Lue89]. Estas capacidades têm aumentado bastante a popularidade de liguagens de simulação para PC. Por exemplo, o GPSS/PC e o SIMAN têm gráficos elaborados e animação. Juntamente a estas razões, o interesse pela Inteligência Artificial (IA) tem levado ao desenvolvimento de pesquisas que visam sua aplicação, especialmente em sistemas especialistas para simulação [Sha85].

Entretanto, existem vários fatores críticos a serem considerados ao se utilizar simulação de eventos discretos. Primeiro, tornar-se um usuário experiente requer considerável treinamento e conhecimento da linguagem de simulação com a qual se está trabalhando. Segundo, são poucos os profissionais que têm todo este talento e experiência para desenvolver modelos válidos de simulação [Sha85]. Um terceiro fator é que o desenvolvimento de modelos de simulação consome bastante tempo [Sha88]. De fato, um tempo consideravelmente maior é geralmente necessário para construir, verificar e validar o modelo do que costuma-se estimar, ou até mesmo do que costuma ser disponível. Isto é crítico especialmente na análise de várias alternativas para uma nova facilidade na manufatura.

2.3.1 Inteligência Artificial e Sistemas Especialistas

Inteligência artificial (IA) se refere ao projeto de sistemas de computadores que, para certas áreas limitadas, emulam algumas das características do pensamento humano — a habilidade de aprender, explicar, resolver problemas e entender o linguajar humano. Com o advento das interfaces homem-máquina interativas e dos bancos de dados relacionais, os computadores deixaram de ser meras máquinas processadoras de dados e passaram a serem vistas como verdadeiros sistemas de suporte na tomada de decisões. A interatividade provê uma comunicação mais natural com o computador, enquanto que os sistemas de banco de dados relacionais facilitam a procura e a utilização de relações entre os vários tipos de dados armazenados. Estas características tornaram possível o desenvolvimento de sistemas computadorizados que diretamente auxiliam na análise e no processo de tomada de decisão. A aplicação dos resultados das pesquisas em inteligência artificial está tornando possível o avanço nas ferramentas de apoio à decisão. Estes novos sistemas serão capazes de gerar (bem como de analisar) soluções alternativas para situações problemáticas.

Desde sua introdução nos anos 50, a IA se destina a estudar como os homens adquirem, organizam e usam o conhecimento. As pequisas e aplicações caem dentro de duas classes :

(1) Aquelas que tentam duplicar ou imitar as habilidades naturais humanas (visão, processamento da linguagem, etc.) da mesma maneira que um ser humano as desempenha,
(2) Aquelas que tentam duplicar os resultados do aprendizado sem se importarem se é exatamente o mesmo processo utilizado por um especialista real.

A segunda classe de aplicações inclui aquelas que comumente são denominadas **sistemas especialistas**. Estas aplicações se destinam a automação de tarefas que são normalmente desempenhadas por pessoas talentosas ou especialmente treinadas. Aplicações de sistemas especialistas (SE) devem ser diferenciadas da pesquisa pura em IA porque o objetivo principal de tais aplicações não é o de entender os mecanismos básicos utilizados pelo especialista humano para chegar a um resultado, mas pelo contrário, é o de consistentemente reproduzir os resultados de um especialista humano. Sistemas especialistas ou baseados em conhecimento, são projetados para compilar a experiência de vários especialistas em um dado campo em uma série de regras, as quais são utilizadas para propiciar inferências e sugerir ao usuário (ou executar automaticamente) um rumo ou uma ação perante um dado problema.

Sistemas especialistas são sistemas de resolução de problemas, que podem alcançar um nível de desempenho comparável ao de um especialista humano, em certos domínios especializados de problema. Sistemas especialistas diferem dos programas tradicionais no que se refere à arquitetura e ao processo de desenvolvimento pelo qual são construídos. Em contraste com programas tradicionais, onde a ênfase é na instrução procedimental para o computador, o enfoque do desenvolvimento de um SE é na aquisição e organização das bases de conhecimento. Um dos maiores problemas no desenvolvimento de SEs é como representar e usar o conhecimento que os especialistas possuem e usam. Em um SE típico, o conhecimento é organizado em três categorias [Sha85] :

(1) Uma base de dados global que provê os dados de entrada (às vezes chamados de conhecimento declarativo) definindo o problema particular e cuidando do estado ou situação da solução corrente.

(2) Uma base de conhecimento que descreve os fatos e as heurísticas associadas com o domínio geral do problema. Como este conhecimento frequentemente toma a forma de regras, ele é às vezes denominado conhecimento relacional ou procedimental.

(3) Uma estrutura de inferência ou controle que define a abordagem de resolução do problema, ou como os dados e o conhecimento podem ser manipulados para resolver o problema.

Portanto, sistemas especialistas tipicamente provêem estruturas de dados e construções de linguagem para armazenar ou representar conhecimento declarativo (dados do problema) e conhecimento específico do domínio da resolução do problema (relações), bem como estruturas de controle (de inferência ou procedimental) para a manipulação destes conhecimentos através de interfaces amigáveis com o usuário (Fig. 2.9). Assim, a maioria dos sistemas especialistas organizam o conhecimento em três níveis : dados, base de conhecimento, e estrutura de controle. Um dos segredos na construção de um SE é o de como representar e armazenar estes dados, tópicos que serão abordados a seguir.

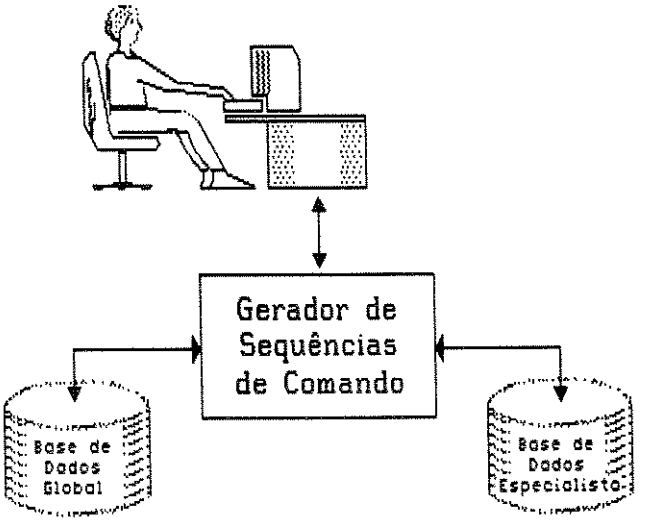

Figura 2.9: Sistemas Especialistas

I. Nível de Dados (Base de Dados Global)

A entrada da base de dados global é o conhecimento declarativo na forma de dados ou fatos, sobre o problema que deve ser resolvido. Existem várias alternativas para representar este conhecimento. A seleção da forma de representação apropriada depende da aplicação em questão. Os fatores mais importantes são armazenamento eficiente, recuperação e facilidade de modificação. Entre os métodos utilizados atualmente estão :

Lógica de Predicados — Linguagem formal na qual uma ampla variedade de definições pode ser expressa. Fatos declarativos podem ser representados como predicados. Expressões são denominadas **fórmulas bem formadas**.

Frames — Um *frame* é uma coleção de fatos e informações na qual todo o conhecimento sobre alguma coisa ou algum conceito é guardado. A maioria dos esquemas de representação do conhecimento baseados em *frames* incluem a idéia de ter diferentes tipos de *frames* para diferentes tipos de objetos, cada um com campos ou *slots* destinados a conter as informações relevantes ao tipo de *frame*.

Redes Semânticas — O conhecimento declarativo também pode ser representado por uma representação gráfica. Redes semânticas são como *frames* no sentido de que o conhecimento é organizado por objetos. Mas aqui, os objetos são representados por nós em um grafo e as relações entre eles, por arcos. Apesar da mesma informação poder ser representada na forma de lógica de predicados, certos tipos de relações podem ser representadas de maneira mais clara em uma representação gráfica.

II. Nível de Conhecimento

O nível de conhecimento descreve o problema ou o conhecimento do domínio específico para o tipo particular de problema a ser resolvido pelo sistema (por exemplo, sistemas de manufatura, sistemas de distribuição, redes de computadores, etc.). Este conhecimento é geralmente procedimental (ou relacional), no sentido de que ele informa como os dados do problema podem (ou devem) ser manipulados de maneira a resolvê-lo. Algumas formas de representação deste conhecimento são descritas a seguir.

Programa de Computador Convencional
: Um programa convencional pode ser escrito para manipular dados e obter uma solução quando o procedimento de resolução é bem entendido e pode ser programado como um algoritmo.

Programas Evocados por Padrão
: O conhecimento dependente do domínio é codificado na forma de operadores ou programas evocados por padrões, que são ativados pela estrutura de controle quando certas condições são satisfeitas nos dados.

Regras de Produção
: Um tipo de programa evocado por padrão de particular interesse é a regra de produção. Estes programas são estruturados na forma "Se <condição> Então <ação primitiva>", onde a condição é normalmente uma conjunção de predicados que testam propriedades sobre o estado corrente. A ação primitiva muda este estado.

Representação Lógica
: O conhecimento procedimental é também frequentemente representado na forma de lógica de predicados de primeira ordem (por exemplo, `A :- B1,B2.`). Este conhecimento, às vezes, é armazenado na forma de grafos E/OU. Uma das características mais atrativas da lógica de predicados é que ela permite a quantificação das regras sobre todo o domínio.

Probabilidades Condicionais
: O conhecimento também é, às vezes, representado em termos de probabilidades condicionais de ocorrência de eventos dado que outros eventos tenham ocorrido.

III. Nível de Controle

Ao nível de controle, o programa de computador decide sobre a pergunta a ser respondida e a estratégia de controle de utilização da base de dados e a base de conhecimento, na solução de um dado problema. Uma estratégia de controle pode ser classificada como **irrevogável** ou **tentativa**. Em uma estratégia de controle irrevogável, uma regra aplicada é selecionada e aplicada irrevogavelmente sem provisões para reconsiderá-la mais tarde. Em uma estratégia de controle tentativa, uma regra é selecionada e aplicada, mas uma provisão é feita para retornar a este ponto da computação posteriormente, para aplicar uma regra diferente se nenhuma solucão satisfatória

(ou mesmo nenhuma solução) for encontrada. A estratégia de controle é uma função do problema a ser resolvido e várias abordagens podem ser incluídas :

Encadeamento Para Frente
: A busca da solução se inicia a partir de um conjunto inicial de dados e condições (base de dados global existente) e se move em direção a algum objetivo ou conclusão (*goal*).

Encadeamento Para Trás
: O objetivo ou conclusão desejada já é conhecida, mas o caminho para se chegar àquela conclusão não o é. Assim se faz uma busca a partir das regras da base de regras que contenham nas suas conclusões o objetivo ao qual se quer chegar.

Redução do Problema
: Nesta técnica, o problema é decomposto em subproblemas que podem ser resolvidos separadamente.

Propagação de Restrições
: Nesta técnica de resolução de problemas, o conjunto de soluções possíveis se torna cada vez mais restrito por regras ou operadores que produzem "restrições locais", nas quais partes pequenas da solução devem assemelhar-se ([Sta77] e [Ste82]).

2.3.2 Inteligência Artificial e Simulação

Em sistemas de IA, assim como na simulação, primeiramente se cria um modelo do sistema. Em sistemas de simulação convencionais, o modelo inclui conjuntos de estruturas de dados interconectados. Um grupo de procedimentos operam nestas estruturas e assim representam a dinâmica do modelo. Quando as estruturas de dados e os procedimentos são de baixo nível, o modelo construído é difícil de se entender e de se modificar.

Os projetistas de sistemas de IA vêem a modelagem como um processo de altíssimo nível [Red87]. Para isto, eles desenvolvem um conjunto compreensivo de ferramentas coletivamente denominadas sistemas de **Representação do Conhecimento** (RC). Entre a extensa variedade de ferramentas de RC, a mais comumente utilizada é o sistema de representação orientado por objeto. Sob este esquema, todos os elementos de um modelo podem ser vistos como um objeto. Um objeto é descrito por suas propriedades. Estas propriedades podem ser descritivas (p. ex., torno1.cor = azul), estruturais (p. ex., torno1.entrada = fila1), comportamentais (p. ex., torno2.carga = procedimento_de_carga), ou taxinômicas (p. ex., torno1.instância = furadeira). Estes quatro tipos de propriedades descrevem suficientemente qualquer modelo. Desta forma, um modelo é um conjunto de instâncias de objetos interconectadas que interagem umas com as outras executando suas propriedades comportamentais (conhecidas como métodos). Além disso, estes objetos formam taxonomias para representar refinamentos sucessivos de conceitos abstratos (p. ex., torno é um refinamento do conceito abstrato de máquina).

Os conceitos de objetos e classes não são estabelecidos na inteligência artificial. Eles foram introduzidos no Simula, uma das primeiras linguagens de simulação [Dah66]. Assim, a unificação da IA com a simulação já foi feita há muito tempo atrás, mas não foi ainda completamente desenvolvida. Uma sucessão das assim chamadas linguagens orientadas por objetos, tais como o SMALLTALK [Gol83] e o C++, refinaram o conceito de classe/subclasse do Simula e introduziram novas terminologias como mensagens, herança, *demons* e mundos/contextos para descrever as várias facetas dos sistemas de representação do conhecimento.

Além da programação orientada por objetos, várias outras ferramentas de RC baseadas em regras e lógica foram desenvolvidas. Ferramentas de mais alto nível tais como máquinas de inferência e *shells* foram elaboradas para auxiliar no desenvolvimento de sistemas de resolução de problemas, como por exemplo, os sistemas especialistas. Estas técnicas de IA podem ser utilizadas para criar ambientes de simulação baseados em conhecimento, os quais abordaremos na seção seguinte.

A Tabela 2.2 apresenta o mapeamento dos conceitos de IA que podem ser aplicados efetivamente aos conceitos de simulação. Quase todos estes conceitos podem ser utilizados para alguns aspectos da simulação, o que auxilia na criação de ambientes de sistemas baseados em conhecimento.

2.3.3 Simulação Baseada em Conhecimento

Uma definição do que seria a simulação baseada em conhecimento (SBC) é encontrada em [Red87], onde as características que tal sistema deve apresentar são identificadas como :

- Aceitar uma descrição do problema e sintetizar um modelo de simulação consultando uma base de conhecimento apropriada.
- Aceitar um objetivo na forma de um conjunto de expectativas ou restrições, selecionar um modelo em um nível apropriado de abstração, determinar a medição de desempenho, gerar um conjunto de cenários plausíveis para procura, executar o modelo de simulação controlando a seleção de cenário e, finalmente, recomendar um cenário que satisfaça o objetivo definido.
- Explicar a racionalidade do porquê de apenas certos cenários terem sido explorados e um em particular ter sido recomendado.
- Apresentar o modelo resultante (isto é, o construído pelo sistema de SBC) com um alto grau de lucidez e confiabilidade.

Estas funções são apresentadas esquematicamente na Figura 2.10. Uma descrição mais completa de um ambiente de simulação ideal pode ser encontrada em [Red86].

Tabela 2.2: Mapeamento dos Conceitos da IA nos Conceitos de Simulação

IA	Simulação
Objeto	Entidade/recurso/transação/restrição/ processo/atividade/nota de evento/ relógio
Propriedade	Descrição de Entidade/ Armazenamento de dados
Método	Comportamento da entidade
Demon	Evento Condicional
Mensagem	Execução de evento
Herança	Descrição de entidade *default*
Rede de Objetos	Modelo/Estado
Mundo/Contexto	Cenário/Ponto de verificação
Regras	Restrições/Comportamentos de evento/ Métodos para gerar experimentos
Lógica	Restrições/Consistência e completude
Planejamento	Projeto dos experimentos
Diagnóstico	Análise do resultado
Aprendizado	Detecção de conexões causais no modelo

2.3.4 Linguagens de Quinta Geração Aplicadas a Simulação

Os projeto de computadores de quinta geração (*fifth-generation*) dos japoneses e do projeto MCC (*Microelectronics and Computer Coorporative*), sediado em Austin, Texas, são dois esforços direcionados no avanço do estado da arte em hardware e software aplicados a inteligência artificial. Pode não ser claro de onde vem nem o que significa o termo *Quinta Geração*. Em termos de software, ele se refere à maneira pela qual nos comunicamos e interagimos com o computador. A primeira geração foi a linguagem de máquina. A segunda foram as linguagens Assembly. As linguagens de terceira geração incluem as linguagens de alto nível e de propósitos gerais como o FORTRAN, Pascal, Ada e C. Todas estas linguagens imperativas (procedimentais) são baseadas na arquitetura de Von Neumann.

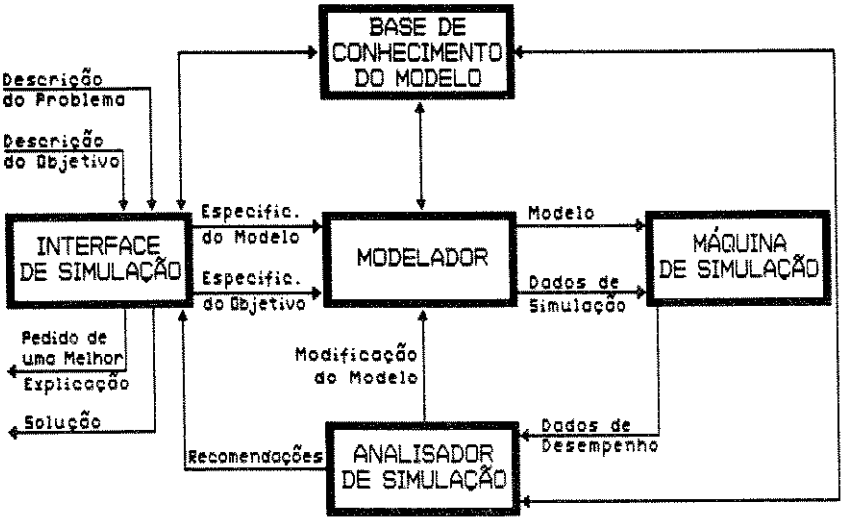

Figura 2.10: Ambiente de Simulação Baseada em Conhecimento

As linguagens de quarta geração incluem várias categorias de software. São pelo menos três as principais áreas : linguagens de apresentação (questionamentos formais, questionamentos naturais, relatórios, gráficos, etc.), linguagens especializadas (planilhas de cálculo, modelagem, análise, simulação, etc.), e geradores de aplicação que trabalham com a captura, modificação e definição de dados (gerenciadores de bancos de dados). Embora estas linguagens já sejam ferramentas para o usuário-final, o fato permanece de que elas precisam ser aprendidas. Apenas algumas delas são realmente ferramentas para não-programadores. (p. ex., planilhas de cálculo). A maioria delas requer um treinamento considerável. Na realidade, linguagens de quarta geração são excelentes ferramentas de produtividade para programadores.

As linguagens de quinta geração possuem várias características que as distinguem das de quarta geração: usam comandos de linguagem natural o que as tornam mais fáceis de se aprender; imprimem suas próprias documentações, simplificando assim atualizações e alterações; são fáceis de serem transportadas de um computador para outro. Além disso, se não-programadores forem capazes de desenvolver suas próprias aplicações, então as linguagens terão completamente automatizado a tarefa de programação. Sistemas que apresentam questionários a serem preenchidos, ícones ou perguntas interativas serão necessários. As aplicações desejadas serão construídas a partir de perguntas feitas para o usuário. Em outras palavras, os sistemas de simulação de quinta geração integrarão as ferramentas desenvolvidas na quarta geração e capturarão o conhecimento dos especialistas em programação, bem como dos especialistas em modelagem de sistemas para simulação.

Outra maneira de observar o impacto do software de quinta geração é considerar a Fig. 2.11. As linguagens de terceira geração facilitaram a codificação e a execução de modelos de simulação (em comparação à linguagem Assembly ou de máquina). Entretanto, continuariam necessários os serviços de especialistas. Com o desenvolvimento das linguagens de quarta geração atuais, a aceitação e o uso da simulação cresceram. Mas infelizmente, a simulação requer ainda hoje uma

grande experiência no que se refere à tradução das necessidades e objetivos do usuário em um modelo apropriado. O objetivo do sistema de simulação especialista de quinta geração é o de embutir a experiência no software, de maneira que o usuário possa definir o sistema de interesse em linguagem significativamente natural, possivelmente através de um sistema gráfico interativo com manipulação de ícones e animação da simulação, e deixar com que o sistema especialista faça o resto.

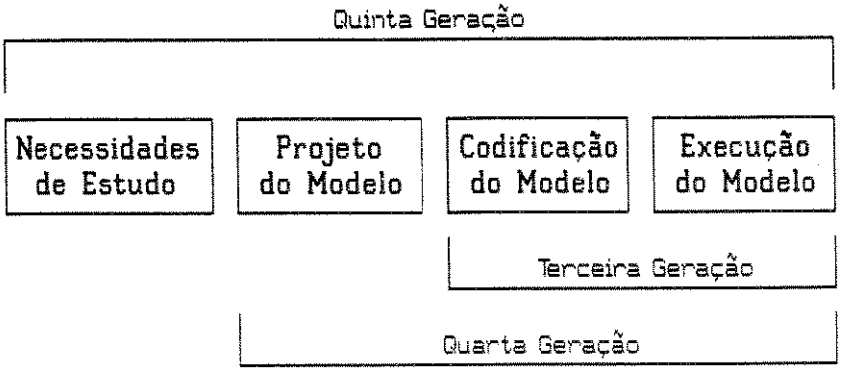

Figura 2.11: Papel das Linguagens de Simulação

Com o que até agora foi apresentado pode-se concluir que :

- Os sistemas de simulação baseados em conhecimento do futuro tendem a ser orientados por objetos com uso extensivo de conhecimento.
- A entrada de dados e a construção do modelo do sistema a ser simulado tende cada vez mais a ser através de uma interface gráfica altamente interativa. Além disso, haverá necessidade de animação gráfica e os resultados deverão ser gerados na forma de gráficos.

Estes dois temas serão focalizados nas próximas seções, apresentando uma análise mais profunda das necessidades em termo de simulação de sistemas flexíveis de manufatura.

2.4 Programação Orientada por Objetos

Existem duas maneiras principais de estruturação de dados em linguagens de programação. A primeira e a mais comum, usada por exemplo na linguagem Pascal padrão [Jen85], pode-se dizer que é derivada de conceitos matemáticos [Car90]. Os dados são organizados como produtos cartesianos (isto é, registros ou *records*), somas disjuntas (uniões ou tipos variantes) e funções (funções e procedimentos).

O segundo método pode ser dito que é derivado da biologia e taxonomia. Os dados são organizados em uma hierarquia de classes e subclasses. Os dados, em qualquer nível da hierarquia, herdam todos os atributos dos dados acima na hierarquia. O nível mais alto desta hierarquia é comumente chamada de classe de todos os objetos. Funções e procedimentos (denominados métodos) são considerados como ações locais aos objetos, em oposição às operações globais, que atuam sobre os objetos.

Estas maneiras de se estruturar os dados gerou classes distintas de linguagens de programação e induziram estilos de programação diferentes. A programação com os dados organizados taxonomicamente é chamada de **programação orientada por objetos** (POO), e tem sido recomendada como uma maneira efetiva de se estruturar programas, ambientes, bancos de dados, e sistemas de grande porte em geral.

As noções de herança e POO apareceram primeiramente no Simula [Dah66]. Nesta linguagem, objetos são agrupados em classes e classes podem ser organizadas em uma hierarquia de subclasses. Objetos são semelhantes a registros (*records*) com funções como componentes, e elementos de uma classe podem aparecer onde elementos da respectiva superclasse são esperados (polimorfismo). As subclasses herdam todos os atributos de suas superclasses. No Simula, a comunicação entre objetos é implementada através de passagem de mensagens entre processos.

O Smalltalk [Gol83] adota e explora a idéia de herança, com algumas modificações. Um objeto no Smalltalk não é normalmente um processo separado. A passagem de mensagens é realizada por chamadas de função, embora a associação de nomes de mensagens a funções não seja direta. O Smalltalk é uma linguagem de uso interativo e interpretada.

Os objetos são organizados estruturalmente como um conjunto de campos (*slots*). Cada *slot* corresponde a um pedaço de informação relacionada com o objeto (uma propriedade ou um comportamento). Um nome identifica cada *slot* e é público a todos os objetos. A informação associada a cada *slot*, entretanto, é interna ao próprio objeto e é escondida dos outros objetos (encapsulamento de dados). Estes dados são obtidos enviando-se uma mensagem pedindo a informação através do nome do *slot*.

2.4.1 Linguagens

Recentemente, linguagens de programação bastante populares têm sido adaptadas à filosofia da POO, entre elas o C++ ([Borl91c] e [Eld90]) e o Pascal [Borl89a]. A tendência de adaptação de linguagens existentes à POO se confirma a partir de trabalhos como o de Canning et al., 1989, onde são propostas interfaces de POO para as linguagens Ada e Modula-2 [Can89], e os de Dieβl et al., 1990 e Scortesse, 1990, onde são apresentadas duas extensões orientadas por objetos para a linguagem CHILL ([Die90] e [Sco90]).

Linguagens como o Pascal já se mostraram bastante adequadas para simulação de sistemas flexíveis de manufatura. Trabalhos como o apresentado por Cheng (1985) utilizam o Pascal para mini e microcomputadores [Che85]. Além disso, o Pascal é uma linguagem de programação geral bem mais compreensível para usuários gerais do que linguagens especiais de simulação

como o Q-GERT e o GPSS. E mais, o Q-GERT e o GPSS requerem compiladores especiais que têm um alto custo e são difíceis de serem adquiridos.

2.4.2 POO Aplicada a Simulação de Sistemas de Manufatura

Objetos consistem de *slots* que contêm valores. Novos objetos podem ser definidos de maneira tal que herdem os valores dos *slots* de objetos previamente definidos. O novo objeto pode ser especializado com a adição de novos *slots*. Desta forma, pode ser criado um objeto composto simples que se comporte um grupo de objetos mais primitivos. Tomemos como um exemplo um caso apresentado em [Sha88], que ilustra o uso de classes, subclasses e metaclasses em uma classificação de objetos.

Um tipo de fato pode ser um membro em uma classe. Uma classe representa o conceito familiar de conjunto. Por exemplo, a classe "torno" representa um subconjunto específico da metaclasse (ou superclasse) "máquina". Um "torno vertical" é uma subclasse da classe "torno". Assim, pode-se descrever uma hierarquia complexa de objetos (ver Fig. 2.12). As ligações indicadas por flechas são chamados ligações de subtipo. Em um diagrama de rede semântica, ligações de subtipo são desenhados entre dois dos mesmos tipos de objetos, de um objeto mais geral para outro objeto mais específico. Ligações sempre têm uma direção, isto é, uma ligação de subtipo sempre aponta para um tipo de objeto mais específico. Pode-se também desenhar ligações que apontam na direção oposta e que são chamadas supertipos. Em algumas linguagens, ligações de super tipo são chamadas ligações **é-um**. As ligações inversas são chamadas de ligações **tipo**. A maioria das linguagens permite especificar apenas o tipo de ligação e a inversa é automaticamente criada.

2.4.3 Sistemas de Banco de Dados Orientados por Objetos

A área de Bancos de Dados Orientados por Objetos (BDOO) emergiu da convergência de várias linhas de pesquisa. As áreas de linguagens de programação, IA e engenharia de software contribuiram para o uso da tecnologia orientada por objeto na área de banco de dados. O desafio do ponto de vista dos bancos de dados é o de integrar estas linhas em um só projeto de sistema que mantenha as características desejadas de cada um dos campos. O resultado deve manter as características centrais de um banco de dados moderno, incluindo persistência, controle de concorrência, recuperação, consistência e uma linguagem *query*.

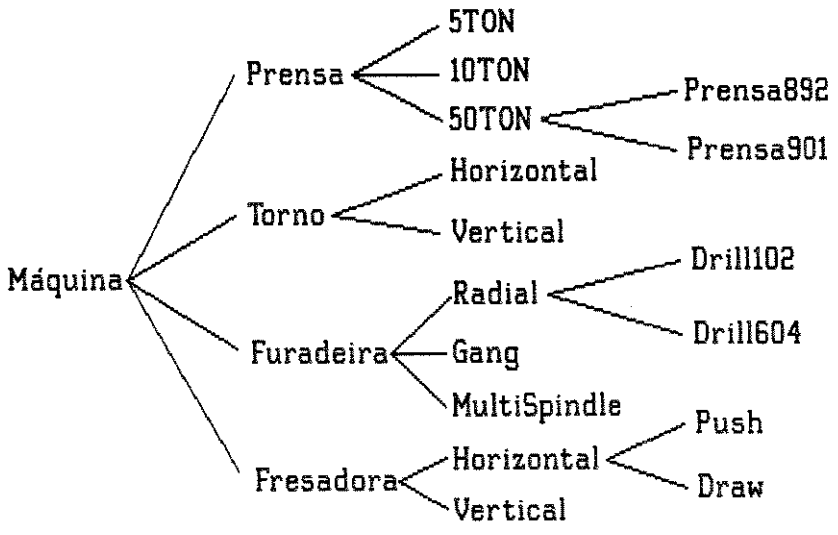

Figura 2.12: Rede Semântica Parcial de um Sistema de Manufatura

2.4.3.1 Aplicação

Embora se tente dizer que os BDOOs são bancos de dados projetados para endereçar problemas de ambiente de projeto (p. ex., Projeto Auxiliado por Computador – CAD), sua aplicabilidade se extende além desta faixa estreita. Ambientes de projeto são apenas um primeiro exemplo de categoria bem mais ampla de aplicações que poderíamos caracterizar como **Programação em Larga Escala Intensiva de Dados**.

Um programa intensivo de dados é um programa que produz e/ou requer uma grande quantidade de dados – tantos quanto não seja possível colocar ao mesmo tempo todos na memória virtual do computador. Programação em Larga Escala se refere ao processo de engenharia de software necessário a programadores múltiplos na produção de programas muito grandes e complexos. Programação em Larga Escala Intensiva de Dados, então, se refere à produção de sistemas enormes e complexos que também requerem grandes quantidades de dados. Ferramentas de CAD são exemplos de tais aplicações.

A complexidade destes sistemas aparece não apenas nos programas que manipulam dados mas também nos próprios dados. Por exemplo, dados que são usados em aplicações de projeto eletrônico contêm várias interconexões complexas com muitas restrições de como estas interconexões podem ser feitas.

Os BDOOs endereçam ambas as fontes de complexidade incluindo facilidades para gerenciar o processo de engenharia de software (p. ex., abstração de dados e herança), e características para capturar mais diretamente algumas das interconexões e restrições nos dados (p. ex., propriedades, relações e objetos complexos).

2.5 Animação Gráfica na Simulação de Sistemas de Manufatura

Existem três maneiras pelas quais os resultados de uma simulação podem ser apresentadas para o usuário [Lue89]: numericamente, graficamente e através do uso da animação. Os resultados numéricos são de longe os mais frequentemente utilizados. Os resultados gráficos fazem uso de gráficos de barras, de torta, de linha, tridimensionais e outros tipos de gráficos para comunicar ao usuário o que aconteceu durante a simulação. A animação utiliza gráficos no computador para criar um filme do sistema que está sendo simulado, mostrando mudanças que ocorrem no estado do sistema.

O uso da animação como uma ferramenta de simulação tem crescido rapidamente nos últimos anos. Embora seja a animação principalmente vista como uma ajuda a apresentação, ela aprimora todos os estágios do desenvolvimento do modelo. Os benefícios estão em três áreas [Smi87] :

(1) Benefícios para o construtor do modelo.

(2) Benefícios de comunicação entre o construtor do modelo e os usuários do mesmo modelo.

(3) Benefícios na apresentação para usuários e gerências.

Para o construtor do modelo, a animação é um elemento de produtividade que acelera o processo de localização e remoção de erros no modelo. Mas talvez a maior vantagem da animação seja comunicação que ela permite entre o construtor e o usuário do modelo. Devido a esta comunicação, o usuário pode tornar-se mais envolvido durante o ciclo de desenvolvimento do modelo do que seria sem a animação. Isto, sem dúvida, leva a melhorias significativas no modelo.

A combinação do maior envolvimento do usuário durante o desenvolvimento com as vantagens da animação para a apresentação dos resultados do modelo produz benefício criticamente importante: o aumento da credibilidade do modelo perante usuários e gerentes, e uma maior valorização dos objetivos da simulação. Isto é um simples reconhecimento do velho provérbio de que uma figura (ou filme) vale mais do que mil palavras [Sha88].

2.5.1 Desenvolvimentos Recentes

A maioria dos mais recentes desenvolvimentos em **simulação interativa visual** (SIV) tem sido em projetos de software ao invés de metodologias [Bel87]. É interessante notar que uma linguagem de animação em desenvolvimento para simulação de eventos discretos é orientada por objetos ([Sha88] e [Mag87]).

Uma lista de pelo menos 18 sistemas de simulação interativa visual (tais como o SIMFACTORY, SEE-WHY, FORESIGHT, CINEMA, etc.) pode ser encontrada em [Bel87]. Além desta lista, um catálogo de lançamentos de softwares para simulação é listado em [SIM86] e [SIM87].

2.6 Resumo

Com o estudo apresentado neste capítulo chega-se às seguintes conclusões :

(1) A transição para os sistemas de modelagem baseados em conhecimento já está em andamento [Sha88].

(2) Os sistemas de simulação baseados em conhecimento do futuro tendem a ser orientados por objetos com extensivo uso de regras embutidas. Também, as linguagens de terceira geração evoluirão para um ambiente de projeto e implementação orientados por objetos.

(3) Os banco de dados envolvidos em tais sistemas de simulação terão de ser orientados por objetos (BDOO).

(4) Na falta de grandes resultados na área de linguagem natural, a comunicação do usuário com o computador terá de ser feita através de interfaces gráficas altamente interativas, com extensiva utilização de mouse, ícones, gráficos e ajuda sensível ao contexto. Além disso, haverá necessidade de animação gráfica e os resultados deverão ser gerados na forma de gráficos.

A seguir será apresentada a descrição de projeto do SISMA, dentro do contexto que foi analisado neste capítulo. O simulador a ser desenvolvido tentará explorar todos estes conceitos com o objetivo principal de ser eficiente, confiável e fácil de usar.

Capítulo 3

PROJETO DO SIMULADOR

3.1 Introdução

Será apresentado neste capítulo o projeto de implementação do SISMA — Sistema Interativo de Modelagem e Simulação de Sistemas Flexíveis de Manufatura — compreendendo o princípio de seu funcionamento, um diagrama de blocos e a descrição de cada um de seus módulos. Informações mais detalhadas a respeito de sua estrutura de implementação e dos algoritmos utilizados poderão ser encontrados em [Borb92].

3.2 Objetivos e Contexto

O SISMA é uma ferramenta de software aplicada à modelagem e simulação de sistemas de manufatura que permite a construção, testes e experimentos interativos de modelos. O SISMA integra os componentes fundamentais de modelagem, que são a descrição do sistema, a aquisição e o gerenciamento dos dados, entradas e saídas gráficas, e simulação de desempenho. A idéia fundamental é descrever o fluxo de informação, material e recursos utilizando objetos autônomos e parametrizados. A entrada de dados é feita através de captura de esquemático em uma tela gráfica, utilizando ícones correspondentes aos elementos da célula de manufatura especificada (robôs, máquinas, buffers, etc.), e através de preenchimento de campos de dados paramétricos referentes ao modelo. A saída inclui medições de desempenho geral ou particular a cada elemento da célula, sob forma de tabelas e gráficos.

O SISMA coloca nas mãos dos menos iniciados na área de programação uma ferramenta poderosa e de fácil utilização. Suas principais características incluem :

- Programação e base de dados orientadas por objetos;
- Direção própria, oferecendo ao usuário uma sequência de procedimentos para modelagem;
- Animação gráfica;
- Programação exclusivamente declarativa ao invés de procedimental;
- Interface com o usuário gráfica e totalmente interativa, utilizando *mouse*, ícones e janelas relocáveis e redimensionáveis;
- Definição de células de manufatura via captura de esquemático; e
- Ajuda sensível ao contexto em todos os níveis.

3.3 Uma Visão Geral

O SISMA se destina à modelagem e simulação de sistemas de manufatura a nível de chão-de-fábrica. Seu projeto é plenamente dedicado a células flexíveis de manufatura, sendo capaz de simular sistemas de fabricação de produtos discretos. O sistema a ser simulado é construído a partir de objetos autônomos que se comunicam através de mensagens.

3.3.1 Características Relacionadas com o Computador

O SISMA é executado em ambiente MS-DOS e requer um mínimo de 512 Kbytes de memória RAM disponível, duas unidades de disco flexível (embora um disco rígido seja bastante recomendado), um monitor de vídeo gráfico (CGA, EGA ou VGA) e um *mouse* de dois ou três botões[3]. O SISMA tem aproximadamente 20 mil linhas de código-fonte e foi escrito em Turbo Pascal Orientado por Objetos versão 5.5 e posteriormente adaptado para a versão 6.0 desta linguagem[4] [Borl90].

3.3.2 Construção do Modelo

Os usuários do SISMA devem primeiramente construir um modelo da célula de manufatura a ser simulada. Para isto, o simulador provê um sistema de edição gráfica altamente interativo, semelhante aos utilizados por sistemas CAD para capturas de esquemático. A edição do modelo provê a utilização de *mouse*, suporte para hardware de alta definição, além de ajuda sensível ao contexto. O usuário deve fornecer apenas os dados necessários à simulação sem desenvolver nenhum modelo ou programação.

3.3.3 Organização do Modelo – Visão Conceitual

O SISMA utiliza programação orientada por objetos, também chamada de "programação por atores", tratando o sistema de manufatura como um conjunto de objetos que desempenham ações, enviando e recebendo mensagens. Desta forma, a abordagem orientada por objetos é especialmente valiosa ao prover uma correspondência entre os objetos simulados e os objetos do mundo real. As classes de uma célula de manufatura são hierarquizadas em uma árvore semântica de classes, como ilustrado na Figura 3.1. Apenas as classes grifadas em `itálico` foram implementadas nesta primeira versão do SISMA. Entretanto, todas elas têm suas especificações incluídas neste trabalho (seção 3.5.3).

Antes de dar início à simulação propriamente dita, bem como **durante** o processo de simulação, o usuário tem a oportunidade de visualizar os estados atualizados de todos os componentes do sistema, podendo, se desejar, alterar parâmetros do ambiente.

---

3. No desenvolvimento e nos testes, foi utilizado o *Genius Mouse GM-6* de 200 dpi de resolução ligado à interface RS-232C do microcomputador.

4. As cópias oficiais destes compiladores utilizadas durante o projeto são de propriedade do CPqD-Telebrás.

3.3.4 Participação do Usuário

A interface com o usuário do SISMA segue uma filosofia semelhante à utilizada pelo Smalltalk [Gol83], [Gol84]. A partir de um menu principal, o SISMA utiliza uma combinação de *pull-down menus* associados a um sistema de ajuda sensível ao contexto disponível a todo momento.

Um dos pontos-chave na modelagem do SISMA é que, uma vez especificada a configuração (*layout*) do sistema de manufatura bem como o conjunto de parâmetros de cada objeto, a simulação pode ser imediatamente processada, e no próprio ambiente de captura de esquemático.

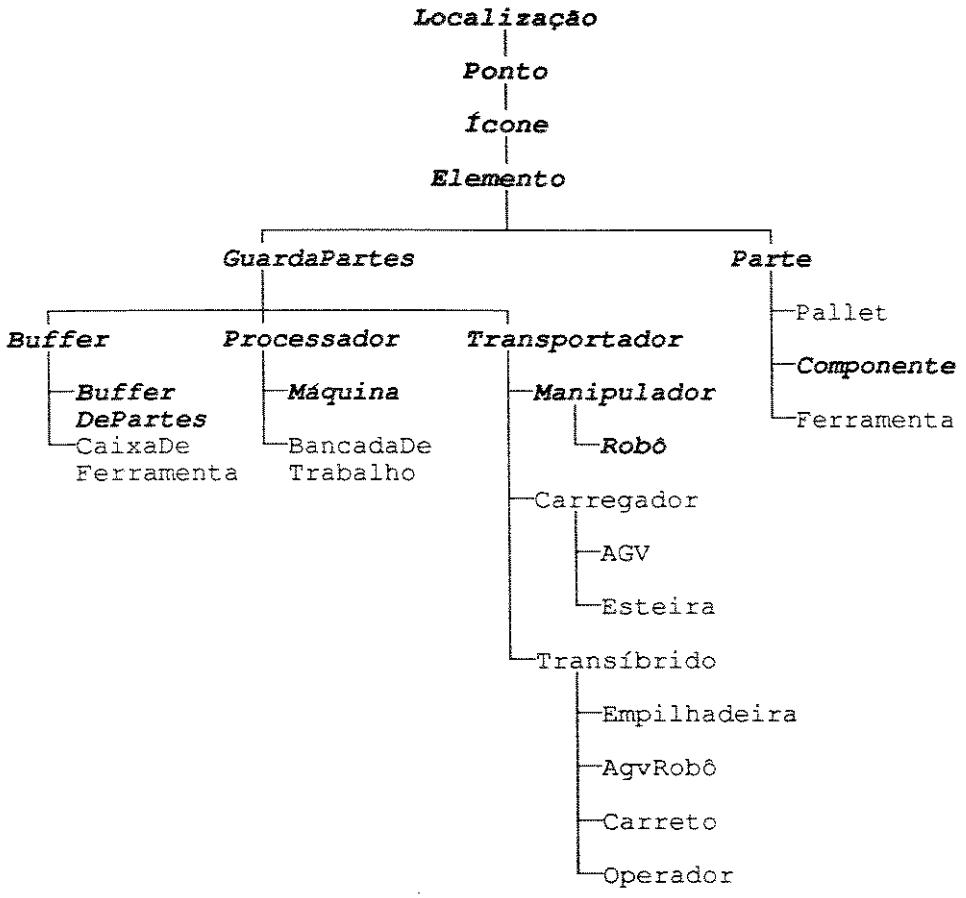

Figura 3.1: Árvore Hierárquica das Classes do Simulador

O SISMA permite ao usuário participar do gerenciamento do sistema de manufatura durante o processo de simulação, sendo possível, por exemplo, alterar o estado de uma máquina ou aumentar o número de partes em um determinado buffer.

3.4 Descrição do Sistema

O objetivo principal do SISMA é prover um ambiente interativo de auxílio à modelagem, programação, simulação e análise de projetos de células de manufatura. Neste ambiente, deverá ser possível modelar e simular explicitamente todos os elementos importantes de uma aplicação. O sistema também deverá facilitar a rápida elaboração de protótipos de novas células, bem como reprojetos de células já existentes.

A filosofia de projeto e implementação adotada para o SISMA foi a orientada por objetos. Os mecanismos de generalização/especialização e abstração de dados, que a programação por objetos provê, são particularmente úteis para modelagem e simulação de processos e entidades de células de manufatura. Além disso, programas escritos em linguagens orientadas por objetos tendem a ser de fácil entendimento, facilitando um futuro refinamento e extensão do software desenvolvido.

A linguagem de programação utilizada suporta estilos de programação tanto procedimental (através de *procedures* e funções) quanto orientada por objetos. Ela provê diversos tipos de dados, estruturas de controle, capacidades de E/S e rotinas matemáticas, bem como mecanismos de **classes** para definição de novos tipos de dados e operações associadas (**métodos**) para operação sobre estes dados. O Turbo Pascal também oferece um depurador simbólico bastante poderoso (Turbo Debugger) e um macro-assembler (Turbo Assembler) para implementação de rotinas em linguagem Assembly ligadas ao programa em alto nível. Todo o sistema de desenvolvimento funciona dentro de um ambiente integrado, compreendendo editor de texto multi-arquivos, compilador e depurador simbólico.

Os motivos principais que levaram à escolha desta linguagem foram:

(1) O compilador é muito barato (aproximadamente 150 dólares) e largamente utilizado;

(2) O Turbo Pascal é uma linguagem esteticamente agradável [Tav89], fácil de usar e entender [She90], muito poderosa e que oferece suporte à programação por objetos [Borl89a];

(3) Em comparação com o Smalltalk, o Turbo Pascal é muito mais eficiente por se tratar de uma linguagem compilada (o Smalltalk é interpretado). Por outro lado, o Smalltalk tem a vantagem de oferecer uma padronização gráfica que facilitaria a implementação do SISMA. Entretanto, o Turbo Pascal suporta um grande conjunto de funções gráficas (*Borland Graphics Interface* — BGI) que, tendo sido incorporado à interface com usuário orientada por objetos desenvolvida em [Borb90], ofereceu um resultado bastante satisfatório. Além disso, futuras versões do SISMA poderão ser escritas em Turbo Pascal for Windows [Borl91b], tornando possível sua operação em ambientes Windows e OS/2 2.0 [Lin91].

(4) Em comparação com a linguagem C++ ([Borl91c] e [Eld90]), o Turbo Pascal é potencialmente menos perigoso para programadores com pouca experiência. Apesar de ser uma linguagem muito poderosa, programas escritos em C ou C++ por programadores

sem muita experiência frequentemente geram códigos ilegíveis e difíceis de serem depurados. O C++ é, sem dúvida, uma linguagem mais eficiente do ponto de vista de desempenho. O Turbo Pascal, porém, se destaca em clareza. Esta escolha facilitará a continuidade do projeto no caso de outras pessoas serem envolvidas no seu desenvolvimento.

3.4.1 Princípio de Funcionamento do Simulador

O processo de simulação de uma célula de manufatura no SISMA se dá em quatro etapas, como ilustrado na Fig. 3.2. A etapa de **Captura de Esquemático** consiste na construção do *layout* da célula de manufatura na tela do computador. Utilizando recursos semelhantes aos encontrados em sistemas CAD (*Computer Aided Design*), o usuário do SISMA escolhe objetos de uma biblioteca pré-definida e posiciona ícones representativos destes objetos nas posições desejadas, dentro de uma janela de modelagem. Os ícones podem ser movimentados, copiados, rotacionados e removidos durante este processo. O usuário deve também atribuir a cada elemento incluído na célula um designador de referência[5] único, o qual será utilizado durante as etapas consecutivas.

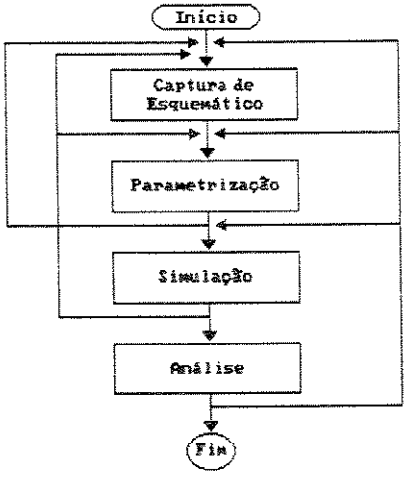

Figura 3.2: Processo de Simulação de uma Célula de Manufatura

Terminada a especificação dos componentes da célula, o usuário deve passar para a etapa de **Parametrização** dos elementos do sistema. Nesta etapa, o SISMA produz uma sequência de perguntas dirigidas ao usuário baseado nos objetos do modelo de célula especificado. O usuário é convidado a informar os tipos de partes que trafegarão na célula, suas rotas de produção, as capacidades máximas de cada buffer, os tempos de deslocamento de cada transportador, a

5. Designadores de Referências definem um código único ao objeto dentro do modelo. Exemplos: `ROBO_01`, `BUFFER07` e `TORNO02`.

quantidade inicial de partes, entre outros ítens. As perguntas são dirigidas de acordo com o que foi respondido em ítens anteriores. Todas as informações fornecidas pelo usuário permanecem guardadas no sistema nas suas formas originais de maneira que, se o usuário solicitar uma nova parametrização, os valores *default* de cada resposta serão aqueles anteriormente informados.

Efetuada a parametrização dos elementos da célula, o usuário pode requisitar a **Simulação** do modelo informando o conteúdo inicial do relógio de simulação e o tempo de simulação desejado. Entretanto, antes de iniciar este processo, o sistema promove uma compilação dos parâmetros fornecidos verificando a consistência do modelo. Se algum problema for detectado, o pedido de simulação é cancelado e o usuário é informado do erro ocorrido. Se nenhuma inconsistência for encontrada a simulação é então iniciada. As condições que resultam no término da simulação podem ser o tempo de simulação expirado, a fila de eventos vazia ou interrupção do usuário. Em todos estes casos, o controle do programa retorna ao modelador. O usuário pode então modificar parâmetros, acrescentar novos elementos à célula, recarregar o modelo, enfim, retornar a qualquer das etapas anteriores, alterar o modelo e então requisitar nova rodada de simulação.

A última etapa do processo é a de **Análise** da simulação executada. Esta análise pode ser geral (da célula como um todo) ou particular a um elemento da célula. Na análise geral, são fornecidas informações sobre o número de partes processadas, tempo total de simulação, tempos de ciclo mínimo, médio e máximo das partes, etc. As características da análise individual dependem do objeto em questão. Cada objeto tem um relatório de desempenho particular contendo informações características às suas atividades.

O diagrama de blocos do simulador é ilustrado na Fig. 3.3. A **Interface com Usuário** é responsável pelos recursos gráficos do sistema, pelo gerenciamento de janelas, menus e mensagens na tela do computador e pelo controle de teclado e *mouse*. O **Modelador** utiliza os recursos da Interface com Usuário e provê um sistema de captura de esquemático e um sistema de parametrização dos elementos do modelo, além de dispor do conjunto de classes utilizadas na contrução dos modelos de células. O **Modelo** nada mais é do que uma lista dos objetos especificados na captura de esquemático e parametrizados pelo usuário. O **Núcleo de Simulação** atua sobre este modelo gerenciando uma Fila de Eventos em memória. Cada evento retirado desta fila informa a instância que deve ser chamada a atuar. A instância ativada pode, por sua vez, inserir novos eventos na fila e enviar mensagens a outras instâncias, desencadeando assim novos eventos. O Núcleo de Simulação provê também um conjunto de funções estatísticas, que podem ser utilizadas pelas instâncias do modelo, e um sistema de análise de simulação que desempenha a última etapa do processo ilustrado na Fig. 3.2.

3.5 Modelador

Uma célula de manufatura é modelada por uma lista de instâncias de objetos pré-definidos no sistema. Cada instância representa um elemento da célula, como por exemplo, uma máquina ou um robô, que internamente contém outras estruturas de dados que armazenam as características particulares do elemento. O modelador é responsável por manter esta lista atualizada durante a modelagem e por permitir o acesso aos parâmetros internos de cada elemento desta lista.

O sistema de modelagem no SISMA é dividido em três blocos principais: o **Sistema de Captura de Esquemático**, o **Sistema de Parametrização** e o **Conjunto de Classes para Modelagem.**

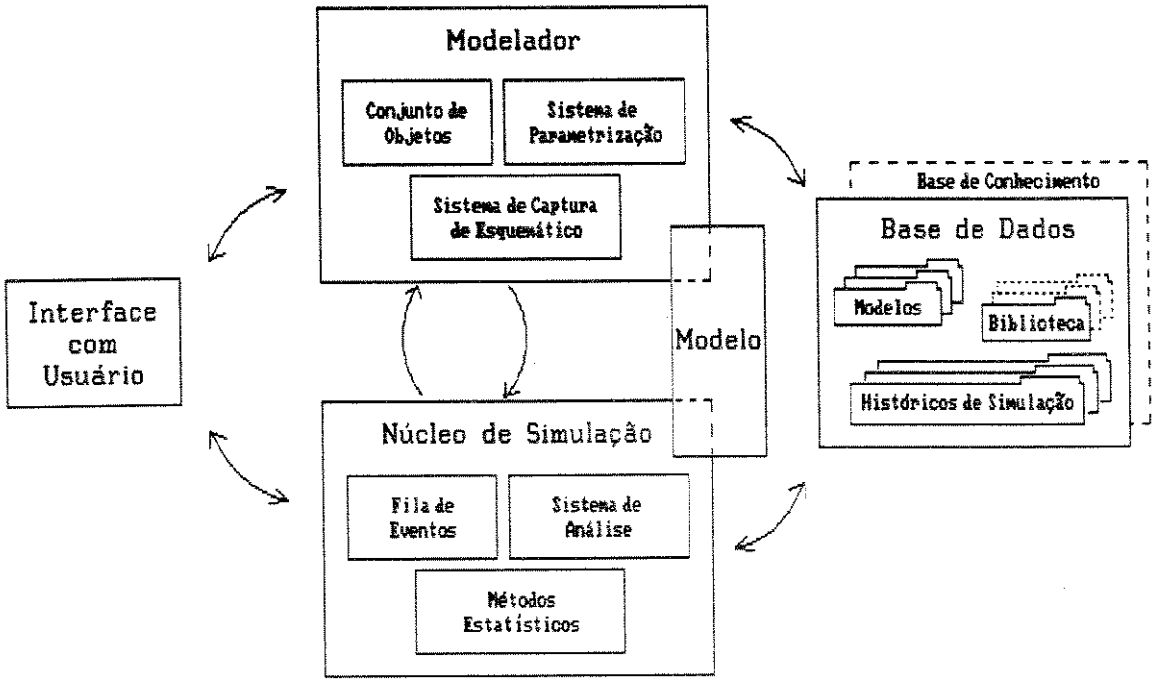

Figura 3.3: Diagrama de Blocos do Simulador

3.5.1 Sistema de Captura de Esquemático

Este bloco provê um sistema de edição gráfica que possibilita ao usuário descrever a configuração da célula de manufatura a ser simulada. Este sistema é constituído por uma janela de edição do modelo e uma estrutura de menus de comandos (Fig. 3.4).

A janela de edição do modelo segue o conceito de janela virtual apresentado em [Cox87] e ilustrado na Fig. 3.5. A área total para modelagem da célula é um quadrado de 65,536x65,536 m^2. Esta área corresponde a 64Kx64K pontos gráficos e é denominada **janela virtual**. A janela de edição do modelo que é mostrada na tela do PC corresponde apenas a uma parte da janela virtual e é denominada janela de posicionamento. Durante a modelagem, a janela de posicionamento pode ser deslocada por sobre toda a janela virtual, bastando para isto pressionar o botão esquerdo do *mouse* sobre a borda da janela de posicionamento correspondente à direção do movimento desejado.

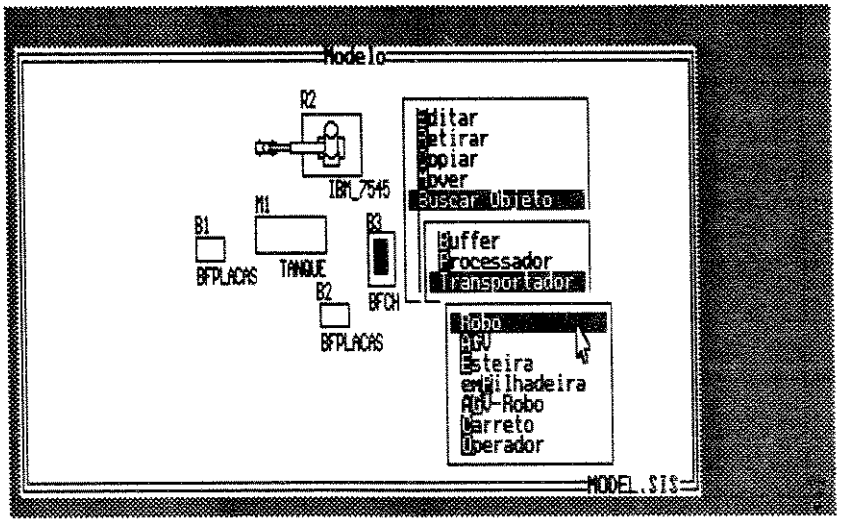

Figura 3.4: Janela e Menu Utilizado na Interface Gráfica

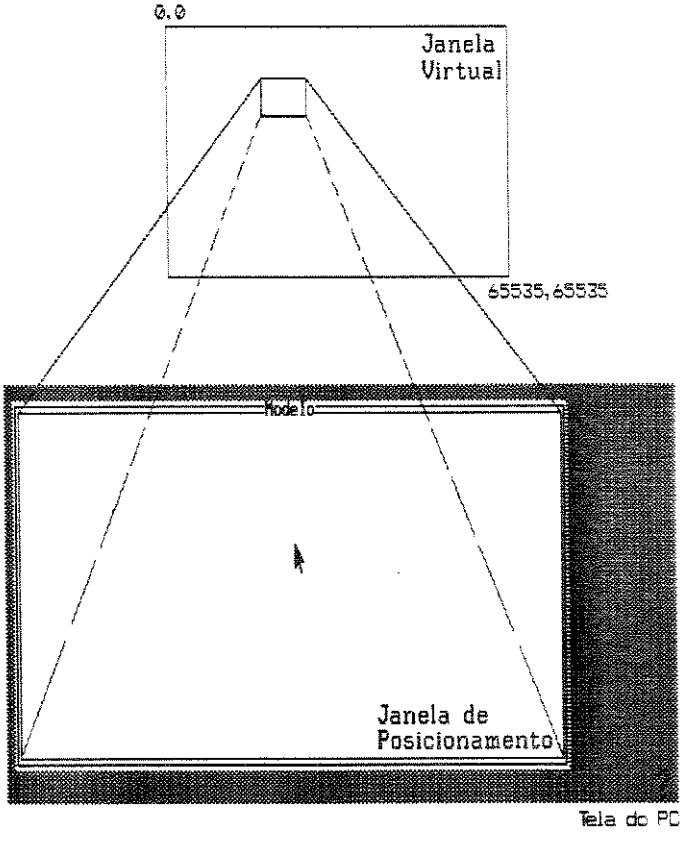

Figura 3.5: Janela Virtual e de Posicionamento

A descrição da célula consiste na escolha e no posicionamento de ícones associados a objetos dentro da janela de edição, como ilustrado na Fig. 3.6.

Comandos necessários à manipulação de objetos dentro da janela de posicionamento podem ser solicitados a partir de um menu principal de comandos (Fig. 3.7). Para chamar este menu, o usuário deve pressionar o botão esquerdo do *mouse* quando este estiver dentro da janela de posicionamento. Algumas opções de menu principal, se escolhidas, resultam em novos menus, fomando assim uma árvore de comandos. A Fig. 3.8 ilustra a estrutura completa de menus do SISMA a partir do seu menu principal.

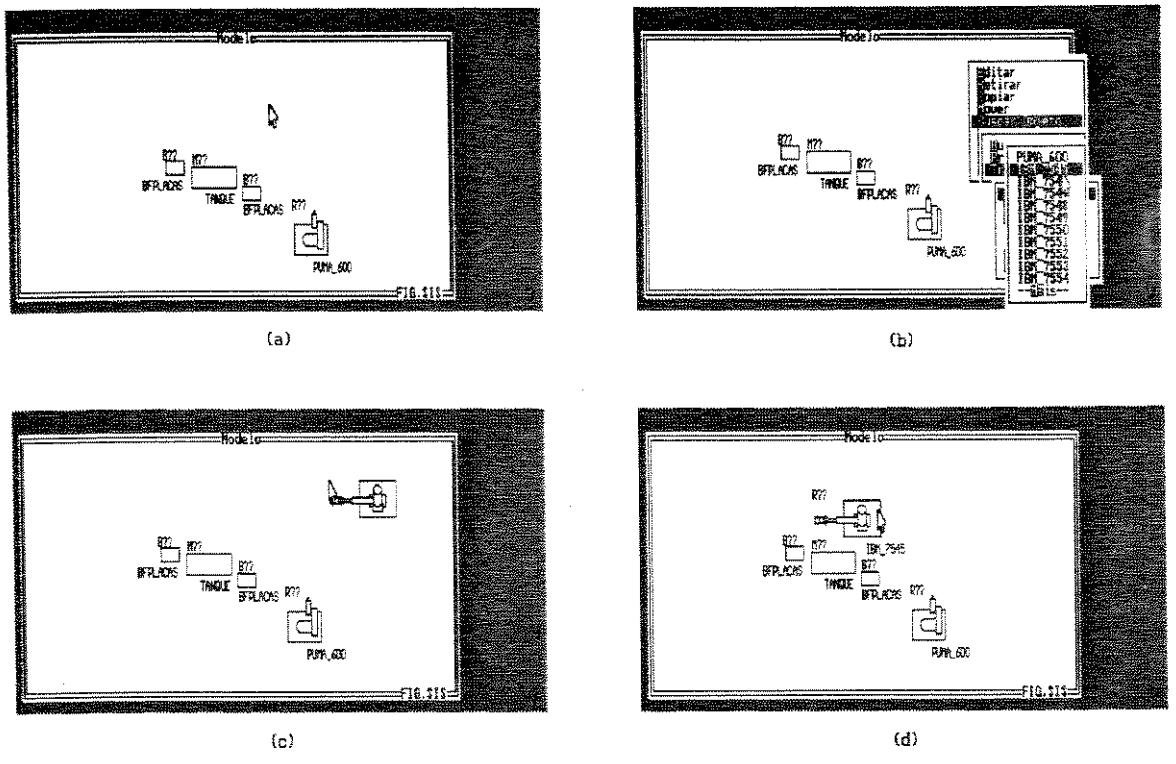

Figura 3.6: Escolha e Posicionamento de Ícones.
(a) Célula antes da inserção do novo elemento. (b) Escolha do novo elemento através de menus. (c) Posicionamento do elemento escolhido. (d) Célula logo após inserção do novo elemento.

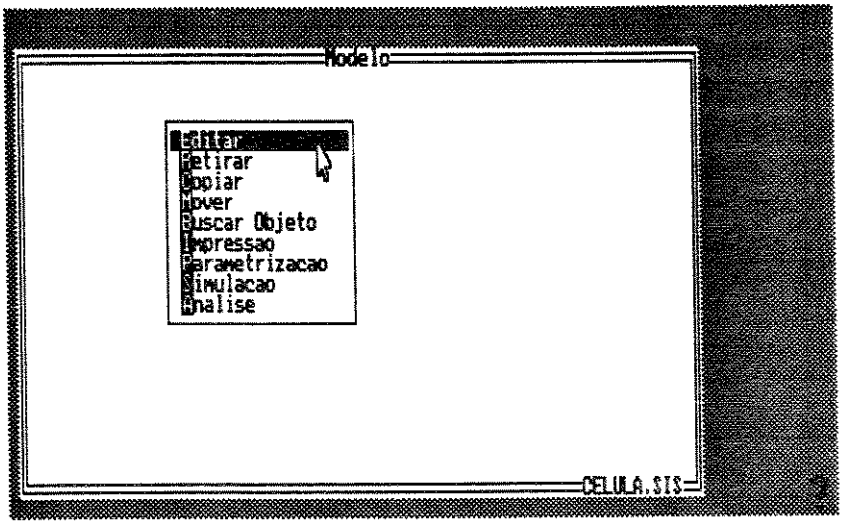

Figura 3.7: Menu Principal de Comandos

Os comandos que podem ser solicitados através do menu principal são :

Editar	Permite a edição de parâmetros do objeto apontado.
Retirar	Remove do modelo o objeto apontado.
Copiar	Cria uma cópia do objeto apontado inserindo-a no modelo.
Mover	Move o objeto apontado.
Buscar Objeto	Permite consultar a biblioteca de classes para inserção de um novo objeto no modelo.
Impressão	Imprime o modelo em uma impressora matricial.
Parametrização	Entra no modo de parametrização dos objetos.
Simulação	Inicia o processo de simulação do modelo.
Análise	Chama o sistema de análise de simulação.

A utilização do sistema de captura de esquemático, bem como do sistema de parametrização e simulação poderá ser melhor visualizada no capítulo 4 — Resultados de Simulação.

3.5.2 Sistema de Parametrização

A parametrização dos objetos do modelo é efetuada a partir de uma série de perguntas elaboradas pelo sistema. Estas perguntas são agrupadas em capítulos, de acordo com as classes utilizadas no modelo, e solicitadas ao usuário em uma sequência de janelas para entrada de dados. A Fig. 3.9 ilustra uma das janelas da sequência. Para o conjunto particular de classes implementado no protótipo, a parametrização é dividida em 5 capítulos:

(1) Rotas de Produção

(2) Processadores

(3) Transportadores

(4) Buffers

(5) Partes

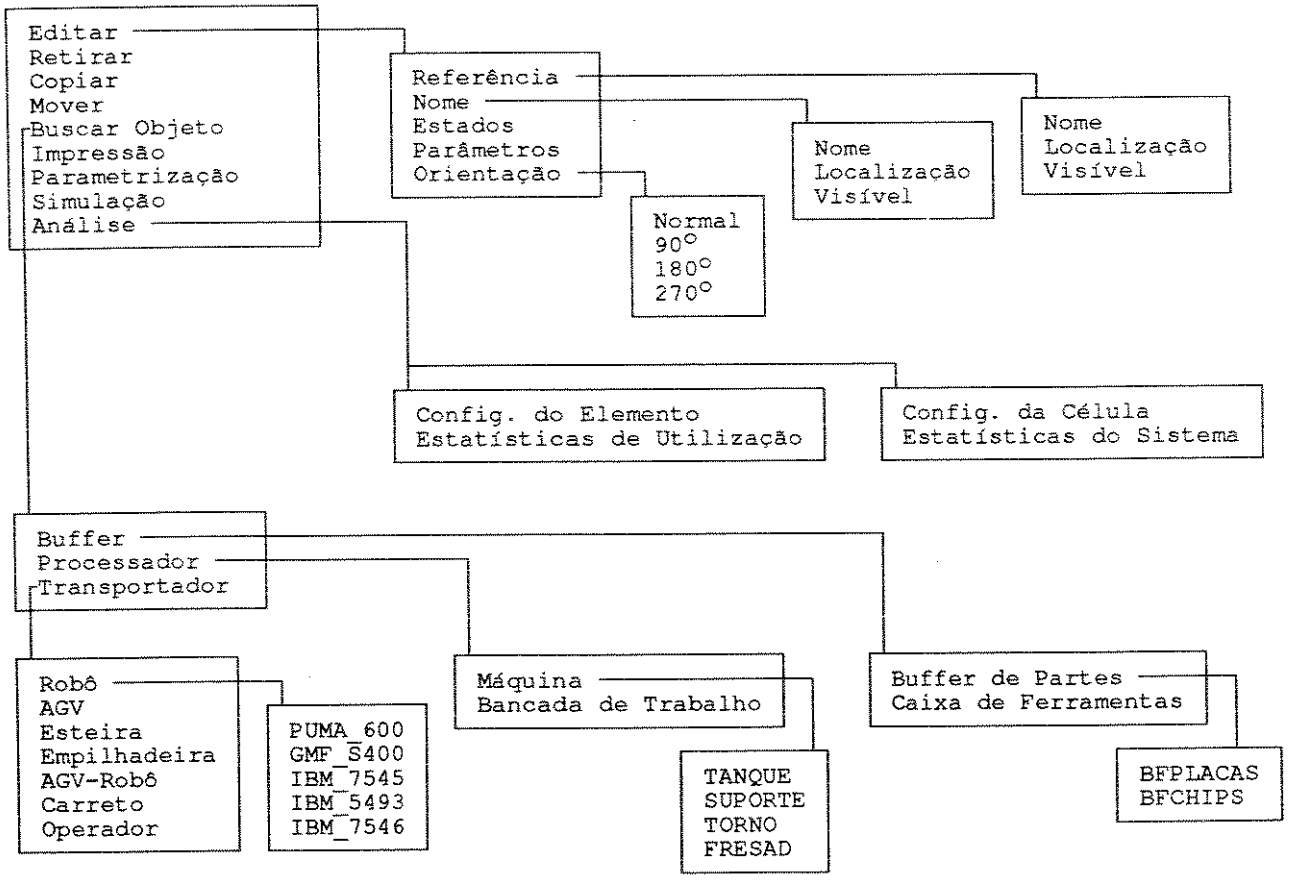

Figura 3.8: Estrutura do Menu do SISMA

As Rotas de Produção especificam o caminho produtivo na célula de manufatura. Existe pelo menos uma rota de produção para cada parte. Ela identifica o caminho percorrido pela parte, desde onde é criada até onde é transformada, além de indicar os elementos que a transportam durante o percurso.

O sistema de parametrização considera as rotas de produção como base de todas informações fornecidas pelo usuário. O formato utilizado para representar uma rota de produção seguido de alguns exemplos é ilustrado a seguir.

Formato:

```
{P} : {E} {T}→ {E} [{T}→ {E} ...]

onde,

  {P} = Designador de referência de uma Parte
  {E} = Designador de referência de um Elemento (Buffer ou Processador)
  {T} = Designador de referência de um Transportador
```

Exemplos:

```
P1 : B1 R3→ M1
P2 : B1 R2→ M2 R3→ B2
```

onde `P1` e `P2` são designadores de referência de partes; `B1` e `B2`, de *buffers*; `M1`, `M2` e `M3`, de processadores, e `R2` e `R3`, de transportadores.

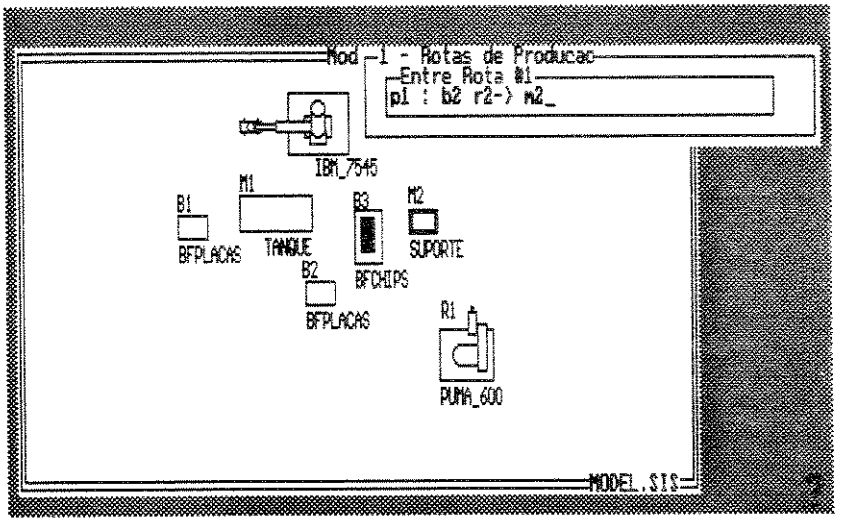

Figura 3.9: Janela do sistema de parametrização.

A partir das informações contidas nas rotas de produção, o sistema passa a solicitar dados a respeito de cada objeto mencionado. Para cada um dos Processadores, o sistema pergunta o processo de transformação ativo (no protótipo, cada processador pode exercer apenas uma transformação, chamada processo ativo), o tempo de processamento, o tempo de configuração (*setup*) e o operador requisitado para a transformação. Os processos de transformação são especificados da seguinte forma :

```
[(Nr_P)](P) [ + [(Nr_P)](P) ...]  → [(Nr_P)](P) [ + [(Nr_P)](P) ...]

onde,

  (Nr_P) = Número de partes
  (P)    = Designador de referência de uma Transportador
```

Exemplos:

```
P1 + 2P2 + P3 → P4
P1 → 4P2
P1 + P2 → 2P3 + 2P4
```

onde `P1`, `P2`, `P3` e `P4` são designadores de referência de partes.

No terceiro capítulo de perguntas, o SISMA requer informações sobre os transportadores especificados nas rotas de produção. Para cada um destes transportadores o sistema pergunta a posição de segurança associada (*safe place*), os tempos de deslocamento entre os pontos "visitados" pelo transportador, os tempos de *picking*, os tempos de *placing* e os elementos que devem ser sinalizados quando o transportador estiver na posição de segurança.

No quarto capítulo, o usuário deve informar a capacidade e a quantidade inicial de cada tipo de parte em cada um dos buffers da célula. No quinto e último capítulo, o sistema pergunta o nome de cada parte envolvida no modelo e sua prioridade no transporte dentro da célula.

3.5.3 Descrição das Classes de Modelagem

O conjunto de classes envolvidas em uma simulação de célula de manufutara pode ser classificado seguindo-se a hierarquia de características e funções ilustrada na Fig. 3.1. Estas classes são organizadas em três grandes grupos, de acordo com o nível de abstração utilizado na sua definição:

- Classes Primitivas,
- Classes Intermediárias, e
- Classes Finais.

Apesar de todos estes grupos terem sido implementados, apenas as classes finais podem ser instanciadas.

3.5.3.1 Classes Primitivas

No SISMA, existem classes que, apesar de não poderem ser instanciadas diretamente, possibilitam a definição de outras classes mais complexas. Estas classes mais simples formam a base da árvore hierárquica do sistema e são denominadas **Classes Primitivas**. Este grupo é composto pelas seguintes classes:

- Localização,
- Ponto,
- Ícone, e
- Elemento.

Localização

É a mais primitiva das classes do SISMA e define tão somente um posicionamento bidimensional de simulação. A classe **Localização** é constituída de dois campos que informam coordenadas nos planos X e Y, e de métodos de inicialização e leitura destas coordenadas.

Ponto

Pode-se a partir de **Localização** definir uma classe para representar um ponto. Um ponto é uma localização que pode ser visível ou não. Desta forma, define-se a classe **Ponto** como descendente de **Localização**. **Ponto** herda todas as características de **Localização** (campos + metódos) e adiciona as informações sobre visibilidade.

Este procedimento pelo qual um tipo herda as características de um outro tipo é chamado **herança**. O herdeiro é chamado **descendente** e o tipo do qual o descendente herda é chamado **ancestral**.

Por ter características de visibilidade, **Ponto** contém métodos que permitem movê-lo, mostrá-lo e escondê-lo. O acesso a estes métodos se estenderá por todos os seus descendentes.

Ícone

Ícone contém todas as características de um ponto, acrescentadas de informações de forma e tamanho, sendo sua representação mapeada bit-a-bit. Métodos especialmente desenvolvidos permitem sua movimentação na tela durante a simulação ou modelagem.

Elemento

A partir de **Ícone** pode-se, dentro da hierarquia do sistema, definir uma classe mais geral chamada **Elemento**, que acrescenta características que a tornam mais próxima de objetos reais.

Um **Elemento** possui um código particular de identificação comum a todas as instâncias da mesma classe, um nome destinado a conter o tipo de equipamento físico representado, um código de referência dentro do sistema que o individualiza perante os demais objetos e, por fim, um identificador de estado.

3.5.3.2 Classes Intermediárias

A partir de **Elemento**, pode-se definir classes de um novo nível hierárquico denominadas **Classes Intermediárias**. Neste grupo estão contidos objetos básicos para formação dos componentes de uma célula de manufatura :

- Partes, e
- GuardaPartes.

Partes

Para definir um tipo **Parte** descendente de **Elemento**, precisa-se acrescentar uma informação de tempo de vida no interior da célula, chamada de tempo de ciclo. Com esta informação será possível gerar uma estatística sobre a permanência de qualquer peça dentro da célula durante o processo de manufatura. O tempo de ciclo está diretamente relacionado à eficiência.

Como cada unidade de peça que trafega no sistema de manufatura será instanciada, ou seja, ocupará uma área de memória, é necessário um número reduzido de campos de dados dentro de sua estrutura de representação. Se, por exemplo, em um sistema com rotatividade média de, por exemplo, 5 mil peças, cada instância ocupar 100 bytes, chega-se a um total de 500 Kbytes de memória ocupada, o que facilmente tornaria a memória de um PC insuficiente (o MS-DOS gerencia até 640 Kbytes de memória).

Há então um conflito: instanciar todas as peças do sistema, mantendo informações como nome, identificação, etc., sem ocupar espaço excessivo em memória.

A solução mais razoável é a de criar dois tipos de classes distintas para representar as partes. O primeiro tipo, **ParteTipo**, é um descendente direto de **Elemento** e é instanciado somente uma vez para cada tipo de parte existente no sistema. Esta instância conterá as informações comuns às partes do mesmo tipo, como nome, ícone, etc.

O segundo tipo, **ParteUnid**, tem um número reduzidíssimo de campos, não descendendo de nenhuma classe. Servirá para instanciar cada uma das unidades das partes do sistema. Cada instância de **ParteUnid** tem um apontador para uma instância de **ParteTipo** e é lá que poderá buscar outras informações a seu respeito.

Guarda-Partes

Um **GuardaPartes** é aquele elemento que, de uma forma ou de outra, retém uma ou mais partes no seu interior, seja para armazenamento, processamento ou transporte. Exemplos de guarda-partes são buffers, caixas de ferramentas, AGVs, máquinas e robôs, todos eles descendentes desta classe.

Os **GuardaPartes** estão divididos em três grandes subgrupos:

- Buffers,
- Processadores, e
- Transportadores.

Buffers

Os buffers servem para armazenar temporariamente um ou mais tipos de partes na célula de manufatura. São utilizados geralmente na entrada e saída de elementos produtivos (máquinas, bancadas de montagem, etc.)

Um buffer é descendente do tipo **GuardaPartes**. Os métodos associados a um buffer incluem Receber Parte e Fornecer Parte. Aqui se estabelece uma das características da Programação Orientada por Objetos : como uma máquina também é descendente de **GuardaPartes**, se algum objeto (p. ex., um robô) quiser guardar uma parte em um buffer ou em uma máquina, ele o fará enviando a mensagem Receber Parte para o elemento destinatário, independentemente do seu tipo. Ou seja, ao enviar a mensagem o elemento se abstrai do fato de quem a está recebendo.

Na visualização do volume de partes armazenado, um buffer pode fornecer esta informação de três maneiras:

- **Consulta:** Com o *mouse* sobre o ícone do buffer, o usuário pode interromper a simulação e verificar todas as quantidades de partes nele armazenadas.
- **Gráfico Interno:** Interno ao próprio ícone do buffer, pode ser programado um pequeno gráfico de barras (ou um mostrador digital) indicando o volume de partes atual. A visualização do buffer é dinâmica e paralela à simulação.
- **Janela de Visualização:** Pode ser aberta uma Janela de Visualização contendo um gráfico de barras que indica dinamicamente o estado do buffer.

Um buffer somente pode receber ou fornecer partes de transportadores (robôs, AGVs *forklift*, operadores, etc).

Em uma célula de manufatura, podem existir dois tipos de buffers: os buffers de partes (**BufferDePartes**), que armazenam partes durante o ciclo produtivo, e as caixas de ferramentas (**CaixaDeFerramenta**), que guardam diversos tipos de ferramentas utilizadas pelas diversas máquinas que compõem o sistema. Note que as caixas de ferramentas são classificadas como buffers apenas para facilitar a definição hierárquica do sistema. No entanto, o sistema de

armazenamento e gerenciamento de uma caixa de ferramentas é idêntico ao de um buffer de partes, o que nos leva a aceitar melhor esta classificação.

Processadores

As características inerentes a um processador podem ser intuitivamente percebidas. Um processador (por exemplo, uma máquina) deve ter associadas as seguintes informações:

1. Tempos de Processamento

Para cada tipo de operação executada pelo processador, deverá haver um tempo especificado para sua execução. Este tempo pode ser, por exemplo, referente à produção de uma peça ou montagem de um subproduto ou produto.

2. Tempos de Configuração (*Setup*)

Caso exista troca de configuração (p. ex. carregamento de programas ou troca de ferramentas), haverá para cada tipo de operação um tempo de configuração. Uma configuração pode ou não requerer a presença de um operador. Caso requeira, então será necessário definir um operador associado.

3. Ferramentas Utilizadas

Um processador deverá ter o controle das ferramentas necessárias para a execução de determinado tipo de operação. O armazenamento de ferramentas poderá ser interno (*tool magazine*), externo (caixas de ferramentas) ou misto. No caso da ferramenta não estar presente na máquina no momento da preparação para a operação, esta deverá ser requisitada a um transportador associado. Um máquina com troca automática de ferramentas pode assim ser facilmente implementada com uma máquina, um robô e uma caixa de ferramenta, conforme mostrado na Fig. 3.10.

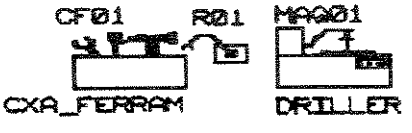

Figura 3.10: Máquina com troca automática de ferramenta composta a partir de três objetos.

4. Transformações

Uma transformação é a operação sobre uma ou mais partes resultando em uma outra ou outras novas partes. Exemplos de transformações: montagem (junção de várias peças em uma), corte (uma parte resultando em várias partes) e renomeação (uma peça passando a ter outro nome depois de operada). Cada transformação deve ser especificada no processador.

5. Múltiplas operações

Um processador pode executar simultaneamente várias operações do mesmo tipo (por exemplo, uma máquina perfuradora pode fazer um mesmo conjunto de furos em várias placas ao mesmo tempo).

6. Transportador de entrada

Elemento a quem o processador deve requisitar uma parte para ser processada. Este transportador (ou grupo de transportadores) deve estar encarregado de fazer a ligação entre o processador e o vértice fornecedor de partes (ver seção 3.5.4).

7. Transportador de saída

Elemento que deve ser notificado pelo processador quando a operação estiver terminada e a parte pronta para ser despachada. O momento da notificação pode ser programada no processador de acordo com o tempo de processamento da operação, de 0 a 100%. Se o momento for relativo a 0%, a notificação será enviada no início do processamento. Se for relativo a 50%, na metade do processamento; e assim por diante. Isto pode reduzir o tempo de espera por transporte no processador. O transportador notificado é o elemento definido pelo usuário para levar a parte até o próximo vértice.

8. Transportador para busca de ferramenta

Elemento a quem o processador deve solicitar as ferramentas necessárias para a operação a ser executada. Ao trazer cada ferramenta, o transportador pode levar para a caixa de ferramenta a ferramenta que estava sendo utilizada. O processador pode também bufferizar internamente um número limitado de ferramentas (*tool magazine*), onde cada ferramenta pode ter um peso e o buffer interno do processador, uma capacidade ponderada, de modo que ferramentas distintas ocupem espaços diferentes.

9. Escalonamento

As tarefas a serem executadas em um processador podem ser ordenadas de diversas maneiras:

• Escolha por prioridades

O processador escolhe a próxima parte a ser trabalhada a partir de prioridades definidas para cada tipo de operação. Estas prioridades podem ser definidas pelo usuário ou associadas aos tempos de processamento, às quantidades existentes em determinado buffer ou grupo de buffers, ou às prioridades das partes a serem processadas.

• Escolha aleatória

O processador sorteia a próxima operação aleatoriamente. Para este sorteio, podem ser definidos pesos para cada operação, causando uma escolha ponderada.

• Sequência de produção

Para possiblitar a simulação de escalonamentos agregados de fabricação e montagem (Fig. 3.11), como os descritos por Kusiak em [Kus90], pode ser definido para um processador uma sequência de produção do tipo P1, P2,..., Pn (ou A1, A2,..., An para o caso de submontagens). Com isto torna-se possível a simulação de atividades produtivas como a representada na Fig. 3.12, cujo diagrama de Gantt do escalonamento ótimo é mostrado na Fig. 3.13.

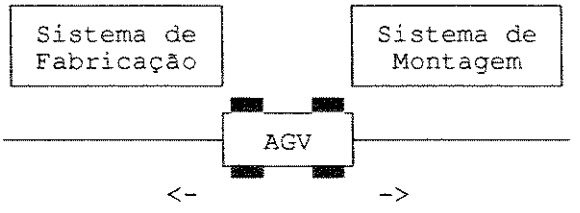

Figura 3.11: Estrutura de um Sistema de Manufatura

• Regras de produção

Para possibilitar a simulação de sistemas com escalonamento dinâmico, pode-se associar um processador a uma base de regras de produção utilizando a mesma base de conhecimento do simulador. O SISMA provê este suporte para todos os seus processadores e transportadores. No entanto, esta técnica exige profundos conhecimentos em programação e sobre as estruturas de dados utilizadas pelo SISMA, devendo assim ser utilizada com bastante cuidado e atenção. Note que esta é a única informação que pode ser entrada de maneira procedimental no SISMA, não sendo porém necessária para a maioria das aplicações.

10. Falhas

Pode ser especificado para cada processador o tempo médio entre falhas, tempo médio para reparo, tempo médio entre manutenções e tempo médio para manutenção. Também pode ser definido uma taxa de falhas de operação para cada tipo de operação suportada pela máquina. Desta forma, uma transformação pode resultar em um de dois tipos de produtos: normal e defeituoso (Fig. 3.14). Estes produtos, bons e defeituosos, são considerados distintos e, como tais, podem ter associadas a eles rotas de produção distintas.

11. Dados estatísticos

Cada processador acumula os tempos associados a cada estado para futuras gerações de relatórios estatísticos. Com isto será possível obter resumos sobre utilização, aproveitamento, quebras, etc.

12. Janelas de visualização

Da mesma forma que um buffer, um processador pode ter associado uma janela contendo informações atualizadas sobre seu estado. Pode-se acompanhar visualmente, por exemplo, o tipo de operação que está sendo processado, as requisições de transporte na entrada e na saída, e dados estatísticos sobre o desempenho do processador.

Transportadores

Assim como **Buffer** e **Processador**, o tipo **Transportador** também descende de **GuardaPartes**, herdando deste todas as suas características. Alguns dos fatores que diferem um transporte de um simples guarda-partes podem ser de imediato percebidos. Qualquer transporte tem um caminho associado e uma posição em determinado instante de tempo dentro deste caminho. Por exemplo, um AGV está relacionado a uma trilha que guia o seu deslocamento, e a uma coordenada que indica sua posição dentro desta trilha. Um robô está relacionado a uma sequência de posições que definem seu trajeto, e a um valor que indica sua posição atual.

Os transportadores podem ainda ser divididos em manipuladores (**Manipulador**) e carregadores (**Carregador**). Um tipo híbrido, que tem capacidade de manipulação e carregamento (por exemplo, empilhadeiras), pode ser definido a partir da junção destas duas classes. Este tipo híbrido é denominado **Transíbrido**.

O usuário pode definir um Ponto Preferencial de Estacionamento (PPE) para o transportador. Este ponto é o local (normalmente junto a algum elemento) para onde o transportador deve deslocar-se ao fim da execução de cada tarefa. A Fig. 3.15 ilustra um caso em que esta característica é aplicável. Neste caso, o AGV encarregado do transporte de partes entre o sistema de fabricação e o de montagem sempre leva as partes no sentido fabricação-montagem. É natural que, ao fim de cada tarefa, o AGV retorne para o ponto referente ao sistema de fabricação para

aguardar e atender prontamente a um outro pedido de transporte de partes. Vale salientar que se nenhum PPE for definido o transportador permanece no ponto de entrega das partes.

Um transportador deve ter na sua estrutura de dados uma lista de tarefas, as quais está encarregado de executar. Isto permite que, mesmo participando de um grupo de transportadores, um transportador se dedique em especial a determinado tipo de tarefa. Esta característica é chamada de associação ou amarração e pode ser melhor entendida no exemplo dado na seção 3.5.4.

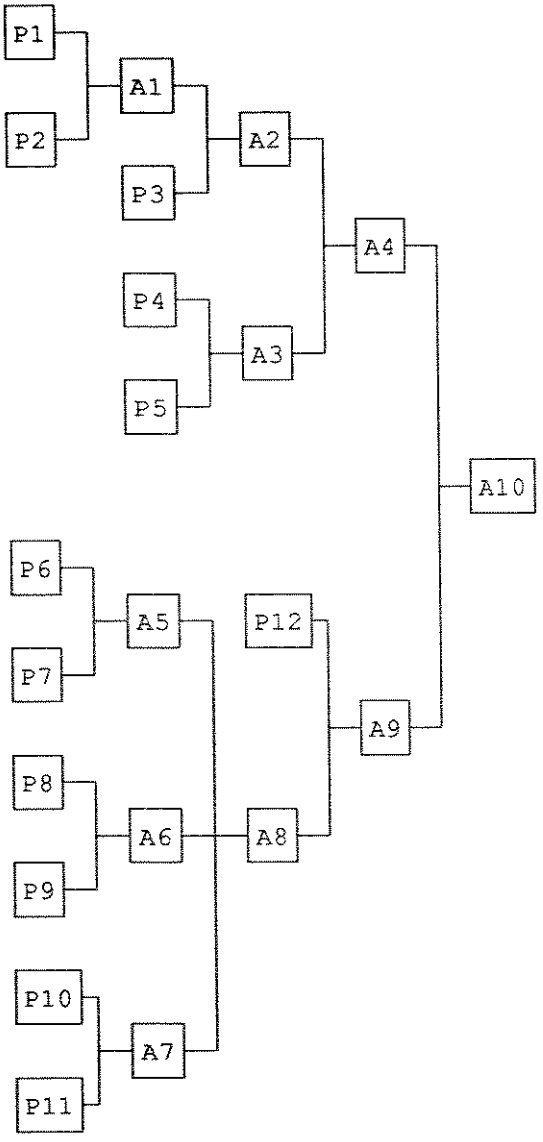

Figura 3.12: Estrutura de Produção

Os Transportadores estão divididos em três subgrupos:

- Manipuladores,
- Carregadores, e
- Transíbridos.

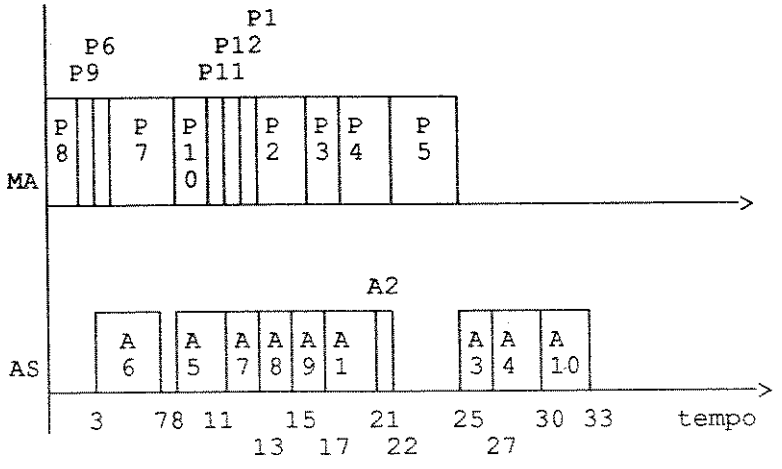

Figura 3.13: Diagrama de Gantt correspondente ao escalonamento ótimo da produção representada na Figura 3.12

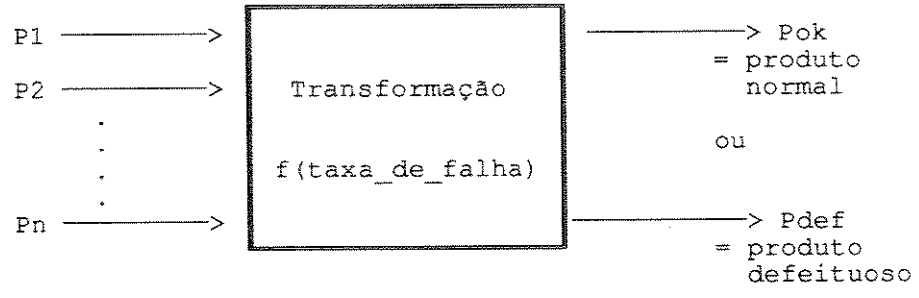

Figura 3.14: Falhas de Produção

Manipuladores

Manipuladores caracterizam-se principalmente por serem elementos com posições fixas no interior da célula, não podendo se deslocar no seu interior. Um **Manipulador** tem a característica de poder pegar uma ou mais partes de um **GuardaPartes** e colocá-las em um outro elemento. Um exemplo de manipulador típico é um robô do tipo *Pick-and-Place*.

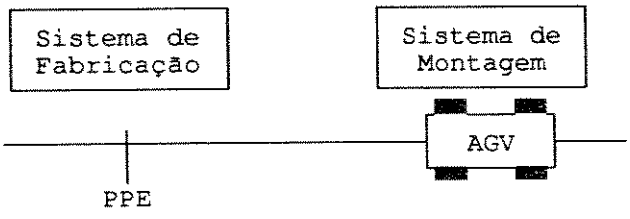

Figura 3.15: Ponto Preferencial de Estacionamento (PPE) do AGV em um sistema de manufatura

Carregadores

Um **Carregador** se caracteriza por ter um caminho ou trilha pré-determinada por onde deve obrigatoriamente se deslocar. Ao contrário de um manipulador, um carregador não pode pegar nem colocar partes. Exemplos de descendentes de carregadores são AGVs e Esteiras.

Transíbridos

Um **Transíbrido** (transportador híbrido) é um elemento que tem características de manipulador e de carregador ao mesmo tempo. Como manipulador ele pode pegar (ou colocar) partes de uma máquina ou de um buffer. Como carregador ele pode se deslocar de um ponto a outro da célula. Um transíbrido pode ou não ter uma trilha associada para poder se deslocar. Exemplos de transíbridos : empilhadeiras, AGVs-Robô, Carretos e Operadores.

3.5.3.3 Classes Finais

Classes Finais são aquelas que representam objetos reais da célula de manufatura podendo ser instanciados pelo usuário durante a modelagem do sistema de fabricação.

Pallets

Um **Pallet** pode ser modelado em uma célula no SISMA exatamente como é modelada uma parte. A diferença é que um pallet pode permanecer continuamente dentro da célula, sem desembocar em um ponto de saída. Podem ser definidas rotas de produções fechadas para os pallets, bem como buffers para armazenamento.

Componentes

Componentes são partes que se integram a outras partes em transformações executadas por processadores. Podem perfeitamente ser modeladas no SISMA como outra parte qualquer. Normalmente os componentes têm pontos de saída do tipo processador.

Em um sistema real, um componente pode por exemplo ser um conjunto de circuitos integrados necessários para a montagem de uma placa de circuito impresso.

Ferramentas

Ferramentas são vistas como recursos necessários para determinados tipos de operação. Uma mesma ferramenta pode ser usada para várias operações do mesmo tipo ou de tipos diferentes. Ferramentas podem também ser armazenadas em buffers especiais denominados caixas de ferramentas. O transporte entre estas caixas e as máquinas da célula é efetuado por transportadores como robôs ou AGVs. Desta forma, uma ou mais ferramentas podem ser compartilhadas por várias máquinas, desde que existam transportadores definidos (e disponíveis) para as trocas de ferramenta[6].

Cada ferramenta pode ter um tempo de vida médio pré-estabelecido. Quando este tempo de vida está para se esgotar, a ferramenta tem de ser substituída por outra do mesmo tipo. Nesta situação, o transportador associado à máquina que estiver utilizando a ferramenta é notificado para efetuar a sua substituição. Ferramentas também podem ser criadas em pontos de entrada, do mesmo modo que as partes utilizadas na produção.

Buffer de Partes

Os buffers de partes servem para armazenar temporariamente um ou mais tipos de partes na célula de manufatura. São utilizados geralmente na entrada e saída de elementos produtivos (máquinas, bancadas de montagem, etc.)

Um buffer de partes é descendente do tipo **Buffer**, que por sua vez descende do tipo **GuardaPartes**. Os métodos associados a um buffer de partes incluem Receber Parte e Fornecer Parte.

Caixas de Ferramentas

Caixas de ferramentas são buffers adequados para o armazenamento de ferramentas. Seu funcionamento é semelhante ao de um buffer de partes. Caso seja necessário definir um ponto de entrada de ferramentas, este pode ser do tipo buffer (ou caixa de ferramentas).

Máquinas

Uma **Máquina** é descendente do tipo processador e quase não tem características a mais que este. Tudo o que foi dito para um processador na seção 3.5.3.2 vale para a classe máquina.

---

6. Existindo apenas uma ferramenta para ser compartilhada por mais de uma máquina, deverá existir um critério de prioridade de produção associado aos tipos de parte manufaturados, de forma a determinar qual das máquina terá a preferência no uso da ferramenta. Este também pode ser aplicado no caso de existir mais de uma ferramenta.

Bancada de Trabalho

Um **BancadaDeTrabalho** pode ser visto quase que como uma máquina, mas é possuidor de algumas sutilezas. É previsto para um trabalhador, por exemplo, horários de descanso, de almoço, e de ausência. A ausência de um trabalhador do seu local de trabalho (bancada) pode ser visualizada na simulação. O tempo médio entre falhas associado a um processador pode ser visto como os casos em que o trabalhador fica enfermo ou sofre um acidente, e o tempo médio para reparo, como o tempo que ele leva para se recuperar ou para ser substituído.

Robô

Entre as características possuídas por um **Robô**, pode ser definido o seu grau de liberdade, que limita a sua movimentação. Pode também ser especificado um posicionamento de segurança (*safe place*) ao qual o robô deve-se dirigir quando estiver no estado livre, sem nehuma tarefa a efetuar. Os robôs também possuem um sistema de sinalização que permite a comunicação entre eles. O principal objetivo deste sistema é o compartilhamento de espaço físico de trabalho, de modo a evitar choques durante os deslocamentos.

AGVs

O **AGV** é um **Carregador** que deve se movimentar apenas por sobre uma trilha. Esta trilha é definida durante a captura de esquemático do sistema de manufatura.

O **AGV** é um objeto extremamente autônomo. Se durante a simulação o usuário alterar a forma da trilha, o AGV automaticamente se adapta a esta mudança, como num caso real. Se no percurso entre dois pontos houver mais de um caminho, o AGV é capaz de escolher aquele que é mais curto. Em um sistema de AGVs, onde vários carrinhos compartilham as mesmas trilhas, os AGVs se comunicam entre si de modo a evitar engarrafamentos.

Um AGV precisa sempre de um manipulador ou transíbrido para ser carregado ou descarregado.

Esteiras

Esteiras são definidas em uma célula de manufatura de maneira semelhante a um **AGV**. A uma esteira podem ser atribuídas várias tarefas de transporte, podendo serem programadas para se deslocar em um só sentido ou nos dois. Também são objetos autônomos, podendo decidir a melhor forma de transportar as peças a elas atribuídas (no caso de esteiras bidirecionais). A partir da indicação dos elementos fornecedor e recebedor de partes, a esteira sabe o momento certo de parar para ser carregada/descarregada. Uma esteira é liberada pelo operador (manipulador ou transíbrido) quando este termina o carregamento/descarregamento das partes.

Empilhadeiras

Uma **Empilhadeira** é, como no mundo real, capaz de transportar partes entre buffers. Uma empilhadeira não tem uma trilha pré-determinada. Ela escolhe o seu caminho ao se deslocar entre dois pontos.

AGVs-Robô

Um **AGV-Robô** pode ser visto exatamente como um AGV normal, exceto que não precisa de um operador para carregar e descarregar as partes que transporta.

Carretos

Um **Carreto** é um carrinho de partes sendo empurrado ou guiado por um operador humano. Funcionalmente se assemelha a um AGV-Robô. Como operador, este elemento pode também ser encarregado de alguma tarefa junto a uma máquina, caso esta precise.

Operadores

Operadores são trabalhadores humanos encarregados de tarefas de transporte de partes e ferramentas, e operação de máquinas. Assim como o trabalhador da bancada de trabalho, um operador pode se ausentar do seu local de trabalho por algum motivo (descanso, horário de almoço, etc.). A frequência de ausência e seu respectivo intervalo pode ser programado pelo usuário durante a modelagem do sistema. Quando um operador se ausenta, ele desaperece do campo visual da simulação caminhando para fora da tela. Durante sua ausência, um outro operador (ou grupo de operadores) pode ser designado para as tarefas do ausente. Este operador é chamado de operador substituto.

3.5.4 Rotas de Produção

Uma Rota de Produção determina o caminho percorrido por uma parte no interior da célula de manufatura. Cada rota é composta de vértices e arcos. Cada **vértice** representa uma entidade armazenadora, uma entidade produtora, um ponto de entrada ou um ponto de saída de partes da célula. Por sua vez, cada **arco** representa um trecho de transporte de partes, associado a um transportador ou a um grupo de transportadores. Cada vértice deve ser a convergência de no máximo dois arcos.

Uma entidade armazenadora pode ser um buffer ou um grupo de buffers. Do mesmo modo, uma entidade produtora pode ser uma máquina ou um grupo de máquinas que executam a mesma função sobre determinada parte.

Cada arco deve ter associado um ou mais transportadores (manipuladores, carregadores ou híbridos). Um exemplo de arco com mais de um transportador é uma malha de AGVs.

A definição das rotas de produção se dá com o auxílio do *mouse*, apontando-se cada vértice de acordo com o esquema de produção que se quer descrever. Da mesma forma, a indicação do(s) transportador(es) associado(s) a cada arco é feita com o *mouse*, indicando-se o(s) elemento(s) incumbido(s) da tarefa de transportar as partes de um vértice a outro, ao longo do trecho.
O vértice inicial de uma rota pode ser um ponto de entrada da célula (gerador de partes) ou uma entidade produtiva, no caso da parte ser produto de uma transformação. O vértice final pode ser um ponto de saída ou também uma entidade produtiva, no caso da parte ser insumo de uma transformação.

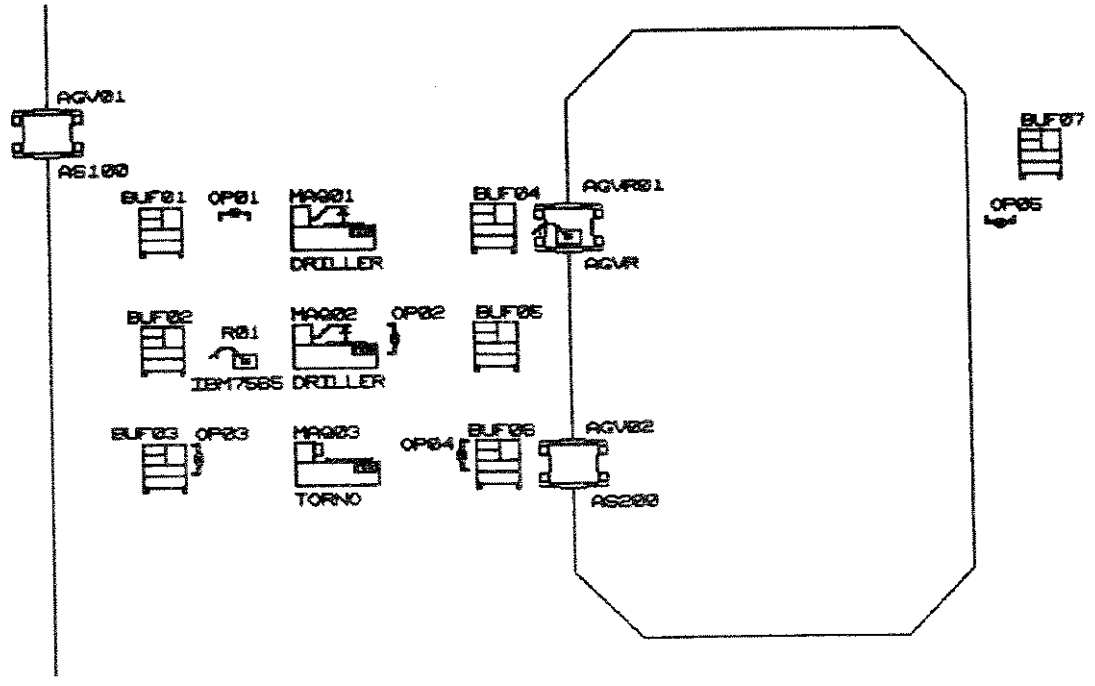

Figura 3.16: Uma Célula de Manufatura

Para ilustrar os conceitos relativos a rotas de produção, considera-se a célula de manufatura ilustrada na Fig. 3.16. Esta célula é composta de duas máquinas-furadeiras, um torno, cinco operadores, um robô, sete buffers, dois AGVs e um AGV-Robô. A célula é abastecida pelo AGV AS100 (AGV01), que deve ser descarregado pelos operadores. As partes recebidas devem ser colocadas nos buffers de entrada (BUF01 a BUF03) e posteriormente transferidas para as máquinas MAQ01 a MAQ03 pelos operadores e pelo robô. As partes podem ser processadas por mais de uma máquina, sendo que, nos intervalos entre os processamentos, são levadas de volta aos buffers de entrada. Depois de processadas, as partes são levadas até buffers intermediários (BUF04 a BUF 06) para serem transportadas pelos AGVs internos (AGV02 e AGVR01). Um dos AGVs necessita da intervenção de um operador para o seu carregamento. O outro AGV, o AGV-Robô, tem carregamento autônomo. Estes dois transportadores devem levar as partes processadas até o buffer de saída BUF07. O operador OP05 auxilia o descarregamento das partes neste ponto.

A Fig. 3.17 identifica os vértices nos quais partes de um tipo P1 devem percorrer até chegar no buffer de saída. De acordo com esta ilustração, uma parte que chega à célula trazida pelo AGV01, é descarregada e armazenada no buffer BUF03 e processada pelo torno MAQ03. Saindo deste torno, a parte deve ser levada a um dos buffers de entrada do grupo GB1-2 (definido como BUF01 e BUF02) para aguardar um novo processamento. Este processamento deve ser efetuado por uma das máquinas-furadeiras do grupo GM1-2 (MAQ01 e MAQ02). Feito isto, a parte é levada para um dos buffers intermediários do grupo GB4-5, formado pelos buffers BUF04 e BUF05, e, posteriormente, transportada para o buffer de saída.

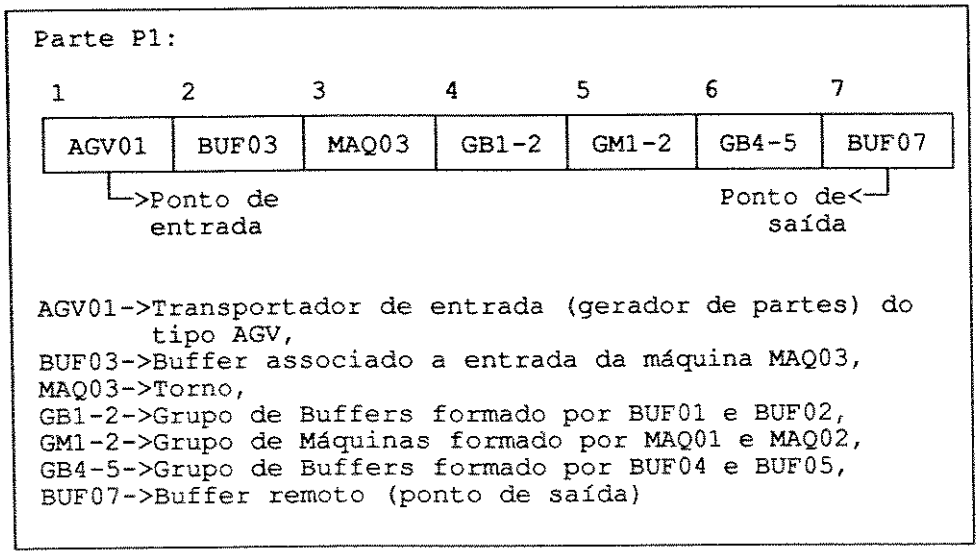

Figura 3.17: Rota de produção da parte P1 sem associação de transportadores.

A associação de transportadores a cada um dos arcos de produção é efetuada em seguida. Cada trecho de produção pode ficar sob responsabilidade de um transportador ou de um grupo de transportadores, como ilustrado na Tabela 3.1. Nota-se que um transportador pode ser associado a mais de um arco de produção e que, dentro de um grupo de transportadores, pode haver atribuições individuais a determinados trechos.

Cada objeto processador da rota de produção deve ter definidos os tempos de processamento e de configuração (*setup*), o tipo de transformação, as ferramentas utilizadas, etc.

Deve-se notar que a construção do sistema envolve uma grande quantidade de detalhes que devem ser definidos pelo usuário durante a especificação. No entanto, o simulador dará sempre que necessário um pedido para a descrição dos parâmetros, evitando que o usuário acabe perdido, a procura de um lugar correto para colocar sua informação.

3.5.5 Pontos de Entrada – Geradores de Partes

Um ponto de entrada representa o vértice inicial de uma rota de produção. Neste ponto, são geradas as partes que trafegarão no interior da célula de manufatura.

Para alguns casos, como por exemplo, para a simulação de determinada sequência de escalonamento, um ponto de entrada pode ser simulado estaticamente, utilizando-se um buffer de partes com uma determinada quantidade inicial de partes, correspondente às operações que se quer escalonar. Isto cobre boa parte das soluções de escalonamento obtidos com códigos de programa comerciais.

Tabela 3.1: Transportadores associados à rota de produção da parte P1

Arco (trecho)	
Transportador	Observação
AGV01→BUF03	
OP03	OP03 é um operador humano. Sua tarefa neste caso é a de descarregar as partes do AGV01 no BUF03.
BUF03→MAQ03	
OP03	Nota-se que um transportador pode ter mais de uma tarefa em mais de um ponto da célula.
MAQ03→GB1-2	
OP04	O operador OP04 leva as partes produzidas por MAQ03 a um dos buffers do grupo GB1-2.
GB1-2→GM1-2	
GT-1	A tarefa de levar as partes de um grupo de buffers a um grupo de máquinas é atribuída a dois transportadores (grupo GT-1). Neste caso, pode-se "amarrar" o operador OP01 entre o BUF01 e MAQ01, e o robô R1 entre BUF02 e MAQ02.
GM1-2→GB4-5	
OP02	Um mesmo operador (OP02) fará o transporte de partes entre o grupo de máquinas e o grupo de buffers. Não há amarrações neste caso.
GB3-4→BUF07	
GT-2	A tarefa de transporte de partes entre o grupo GBUF3-4 e o buffer BUF07 é dividida entre dois AGVs. AGVR01 tem carregamento e descarregamento autônomo. Já AGV02 necessita de algum manipulador ou transíbrido para executar esta tarefa. No caso, pode-se associar o operador OP04 à conexão GB4-5.AGV02, e o operador OP05 à conexão AGV02.BUF07.

No entanto, para simular uma célula de manufatura de maneira realmente dinâmica, é necessário simular a chegada das partes no sistema dinamicamente, ou seja, durante o seu funcionamento. Isto é feito com a definição de pontos de entrada, também chamados de Geradores de Partes, no interior da célula.

Podem ser definidos vários pontos de entrada dentro de uma mesma célula, cada um gerando partes independentemente uns dos outros. Os parâmetros necessários para a definição de um gerador de partes em muito se assemelha aos utilizados nos blocos GENERATE do GPSS [Gor75], com algumas variações acrescentadas.

Parâmetros Associados a um Ponto de Entrada

1. $\mathbf{A_t \pm B_t, C_t, D_t, E_t, F_t}$

Tempo de ação do ponto de entrada. O tempo de ação corresponde ao intervalo de tempo entre duas gerações consecutivas de partes. O campo A_t diz o tempo médio. O campo B_t, chamado de variação ou modificador, produz uma variação aleatória ("randômica") sobre a média. Se B_t é uma constante, menor ou igual a média, é chamado de variação, e o tempo de ação é computado com qualquer inteiro dentro do escopo da média mais ou menos a variação, sendo dada igual probabilidade a cada número do intervalo. Por exemplo, os valores A_t=6 e B_t=2 produzirão um dos números 4, 5, 6, 7 ou 8, cada um com probabilidade igual a 1/5. Se a variação é 0 ou o campo B_t não for especificado, o tempo de ação é constante igual ao tempo médio. O campo A_t também pode ser 0, fazendo com que nunca haja partes geradas. É possível introduzir funções no campo B_t. O campo B_t passa então a se chamar modificador. O tempo de ação é calculado chamando-se a função e multiplicando-se o resultado pela média no campo A_t. Se o campo A_t não for especificado, o valor 1 é assumido, de forma que o valor da função se torne o tempo de ação.

A geração de partes é iniciada automaticamente a partir do instante 0 da simulação. No entanto, é possível deslocar este instante definindo-se o início da geração no campo C_t. Se o campo C_t for especificado, seu conteúdo é interpretado como o atraso de tempo depois do qual as partes começarão a ser geradas. Por exemplo, se C_t=10 e a unidade de tempo for minutos, a geração de partes somente se iniciará depois de se passarem 10 minutos de simulação.

O campo D_t pode ser usado para controlar o número de gerações feitas pelo ponto de entrada. Se o campo D_t não for especificado, não haverá limites e o bloco continuará gerando partes o tanto quanto durar a simulação. Se contiver um inteiro positivo, o ponto de entrada encerrará suas atividades depois da D_t-ésima geração.

As partes geradas podem ter prioridades. Se o campo E_t não for especificado, a prioridade dada às partes será 0, que é a prioridade mais baixa do sistema. Caso contrário, é dada a prioridade indicada no campo E_t, que pode assumir valores entre 0 e 255, que é a prioridade mais alta do sistema. Esta prioridade é normalmente utilizada na tomada de decisão dos transportadores.

Cada parte gerada ao ser entregue ao sistema tem o seu Tempo de Ciclo zerado no momento da entrada da parte no sistema. O Tempo de Ciclo é utilizado para medir o tempo de permanência da parte no sistema.

2. $A_p[i] \pm B_p[i]$, $C_p[i]$, $D_p[i]$, $E_p[i]$

Quantidade de parte do tipo i gerada no instante determinado pelo tempo de ação $A_t \pm B_t$. A definição de $A_p[i]$ segue o mesmo padrão utilizado para o tempo de ação.

O campo $C_p[i]$ representa o instante a partir do qual as partes do tipo i poderão começar a serem geradas. Este instante é relativo ao C_t especificado para o ponto de entrada. Por exemplo, se C_t = 100, $C_p[P1]$ = 50, e a unidade de tempo utilizada é minutos, então a parte P1 só começará a ser gerada quando tiverem decorridos 150 minutos de simulação. Se $C_p[i]$ não for especificado, será considerado igual a zero.

O número de partes geradas durante a simulação pode ser limitado a uma quantidade especificada. O conteúdo do campo $D_p[i]$ indica esta quantidade. Depois de geradas $D_p[i]$ partes do tipo i, o ponto de entrada não mais gerará este tipo de parte, mesmo que outros tipos continuem a ser gerados. Note que se o campo D_t for especificado para o ponto de entrada, este poderá encerrar suas atividade sem ainda ter gerado a quantidade de partes do tipo i especificada em $D_p[i]$. Se $D_p[i]$ não for especificado, não haverá limites para a geração de partes do tipo i.

Se o ponto de entrada for um transportador (os tipos de elemento associados ao ponto de entrada serão discutidos no próximo ítem), o campo F_t indica o tempo máximo de parada do transportador na célula. O transportador espera ser descarregado para poder partir. Se no entanto o tempo máximo de parada estourar, o AGV parte mesmo que o seu descarregamento não tenha sido completamente efetuado. Se o campo F_t for deixado vazio, o AGV espera o descarregamento indefinidamente.

Cada parte gerada pelo ponto de entrada pode receber uma prioridade de acordo com o seu tipo. Se o campo $E_p[i]$ for especificado, o ponto de entrada dá a cada parte do tipo i gerada a prioridade indicada pelo conteúdo de $E_p[i]$. Se o mesmo campo for deixado vazio, é assumida a prioridade indicada por E_t do ponto de entrada.

Elementos associados a Pontos de Entrada

Um ponto de entrada da célula pode ser qualquer elemento descendente de **GuardaPartes** (exceto manipuladores). Isto possibilita a colocação de tipos variados de elementos como geradores de partes endogêneos ou exogêneos (ver item 2.2.2. Ambiente). Os tipos de pontos de entrada de uma célula são analisados a seguir :

- **Ponto de Entrada do tipo Buffer**

Um ponto de entrada representado por um buffer de partes (não faz muito sentido falar em gerador de ferramentas, embora isto seja implementável no nosso simulador) acumula todas as partes geradas no seu interior. Torna-se adequado ter o cuidado de dimensionar bem a capacidade do buffer para evitar estouros durante a geração de partes. Entretanto, qualquer estouro que ocorra no buffer será sinalizado visualmente e no relatório final da simulação.

- **Ponto de Entrada do Tipo Processador**

Uma característica adicionada ao ponto de entrada quando este é do tipo processador é que só poderá haver uma nova geração de partes quando as partes criadas na última geração já tiverem sido retiradas por algum transportador.

- **Ponto de Entrada do Tipo Transportador**

Quando um ponto de entrada é do tipo transportador, pode-se definir um tempo máximo de parada (Ft), que corresponde ao tempo disponível para efetuar o descarregamento na célula das partes trazidas pelo transportador. O descarregamento pode ser feito por algum manipulador ou transíbrido da célula. Na Fig. 3.16 tem-se um exemplo de ponto de entrada representado por um

AGV. A cada tempo de ação, o AGV, trazendo todas as partes geradas, estaciona no seu PPE (Ponto de Preferencial de Estacionamento) e aguarda o descarregamento ser efetuado pelo(s) operador(es) encarregado(s), no caso o OP03. Quando o descarregamento for completado, o AGV parte por sobre sua trilha e sai do campo visual da simulação. Se antes do descarregamento ser completado o tempo máximo de parada estourar, o AGV abandona a célula levando as partes não descarregadas. Este fato é notificado visualmente durante a simulação e no relatório gerado no final.

3.5.6 Pontos de Saída

Simetricamente a um ponto de entrada, um ponto de saída representa o vértice final de uma rota de produção. Nestes vértices serão armazenadas as partes produzidas pelo sistema de manufatura simulado.

Podem ser definidos vários pontos de saída dentro de uma mesma célula, cada um recebendo as partes produzidas independentemente uns dos outros. Um ponto de saída equivale aproximadamente a um bloco TERMINATE do GPSS. Cada ponto de saída tem um contador de terminação $CT_p[i]$ para cada parte do tipo i, que é incrementado de $A_p[i]$, frequentemente 1, cada vez que chega uma parte deste tipo. Quando este contador atinge um valor determinado por $B_p[i]$, a simulação é terminada. O valor de $B_p[i]$ pode ser determinado pelo usuário numericamente ou associado ao valor do campo $D_p[i]$ de algum dos pontos de entrada da célula.

Da mesma forma que o GPSS, se for necessário que a simulação termine em um instante de tempo específico, uma maneira de se fazer isto é incluir um ponto de entrada que gera apenas uma parte do tipo i neste instante de tempo e imediatamente a envia, por intermédio de um transportador, para um ponto de saída com os campos $A_p[i] = 1$ e $B_p[i] = 1$. A geração desta parte irá então interromper a simulação no instante de tempo desejado.

Um ponto de saída deve ser sempre representado por um buffer de partes.

3.6 Núcleo de Simulação

O núcleo de simulação gerencia o conjunto de processos concorrentes associados aos objetos do modelo. Estes processos desempenham suas operações em grupos denominados **fases ativas**. Entre quaisquer duas fases ativas de um processo, qualquer número de fases ativas de outros processos podem ocorrer. A sequência de operações no sistema como um todo torna-se assim uma sucessão de fases ativas dos processos presentes no modelo. Este modelo de operação é denominado "quasi-paralelo" [Dah66].

O núcleo de simulação do SISMA implementa um mecanismo de manipulação de conjuntos ordenados conveniente para organizar uma fila de processos. Também estão embutidos no núcleo de simulação uma biblioteca de procedimentos estatísticos para definição de funções de distribuição de probabilidades e um sistema de análise pós-simulação, além de procedimentos para acumulação de integrais de tempo e geração de histogramas.

Nesta seção, serão apresentados os conceitos envolvidos e a descrição de funcionamento do núcleo de simulação.

3.6.1 Processos

O Turbo Pascal, por não ser uma linguagem concorrente por definição, não suporta o conceito de processo. Devido a este fato, no SISMA, os processos são implementados na forma de métodos das classes do modelo. Todas as classes descendentes de **Elemento** (seção 3.5.3.1) têm um método virtual chamado `React`, que desempenha para o núcleo de simulação o papel de processo. Apenas as classes **Processador**, **Transportador** e seus respectivos descendentes implementam este método, sendo por isto denominados de **classes ativas**. As classes **Buffer**, **Parte** e seus descendentes não possuem processos associados e são denominados **classes passivas**.

Os métodos `React` de todas as classes são implementados na forma de **máquinas de estado**. Estas máquinas se baseiam na estrutura CASE do Pascal, tendo como variável de controle o estado do objeto. Os blocos de execução do CASE correspondem às fases ativas do processo. Dependendo do estado do objeto, um destes blocos é executado, podendo fazer com que o objeto permaneça no mesmo ou transicione para outro estado. A Fig. 3.18 ilustra esta estrutura de implementação.

Os processos são ativados pelo núcleo de simulação através de chamadas aos métodos virtuais `React` dos diversos objetos do modelo.

```
PROCESSO Transportador.React

INÍCIO

  CASE estado OF

    livre  : INÍCIO (* Bloco de execução *)
               .
               . (* Ações ref. à preparação p/ o deslocamento *)
               .
               Calcular_tempo_de_deslocamento_para_origem;
               MudarEstado(movendo_se_para_origem);
               CSQ.AtivarDelay(tempo_para_deslocamento)
             FIM

    movendo_se_para_origem :
             INÍCIO (* Bloco de execução *)
               .
               . (* Ações ref. à preparação p/ retirada da parte *)
               .
               Calcular_tempo_para_retirada_da_parte;
               MudarEstado(retirando_parte);
               CSQ.AtivarDelay(tempo_para_retirada)
             FIM

    retirando_parte :
             INÍCIO (* Bloco de execução *)
               .
               . (* Ações ref. à preparação p/ novo deslocam. *)
               .
               Calcular_tempo_de_deslocamento_para_destino;
               MudarEstado(movendo_se_para_destino);
               CSQ.AtivarDelay(tempo_para_deslocamento)
             FIM

    movendo_se_para_destino :
             INÍCIO (* Bloco de execução *)
               .
               . (* Ações ref. à preparação p/ entrega da parte *)
               .
               Calcular_tempo_para_entrega_da_parte;
               MudarEstado(entregando_parte);
               CSQ.AtivarDelay(tempo_para_entrega)
             FIM

    entregando_parte :
             INÍCIO (* Bloco de execução *)
               .
               . (* Ações ref. à finalização da tarefa *)
               .
               MudarEstado(livre)
             FIM

  FIM DO CASE
FIM DO PROCESSO
```

Figura 3.18: Estrutura de implementação de um processo.

3.6.2 Elementos e Conjuntos

Tanto o modelo da célula de manufatura quanto o controle da simulação utilizam estruturas de dados do tipo lista. Os conceitos de elementos e conjuntos servem para facilitar e padronizar a manipulação destas estruturas [Dah66]. Um **conjunto** é uma sequência ordenada de **elementos**. Além das características já mencionadas na seção 3.5.3.1, os elementos contêm em sua estrutura apontadores que permitem encadeamento em um conujnto (Fig. 3.19). Além disso, a classe **elemento** define o método virtual `React` e serve como base para implementação das demais classes do modelo.

Os dois primeiros componentes de um elemento são apontadores para os elementos próximo e anterior no conjunto, denominados *next* e *prev*, respectivamente. Se `X` é um elemento em um conjunto, `X.next` é seu elemento sucessor e `X.prev` é seu predecessor. Estes apontadores entre elementos são atualizados implicitamente por métodos da classe **conjunto**, tais como `S.Append(X)`, `S.Insert(X)` e `S.Remove(X)`, onde `S` é um conjunto.

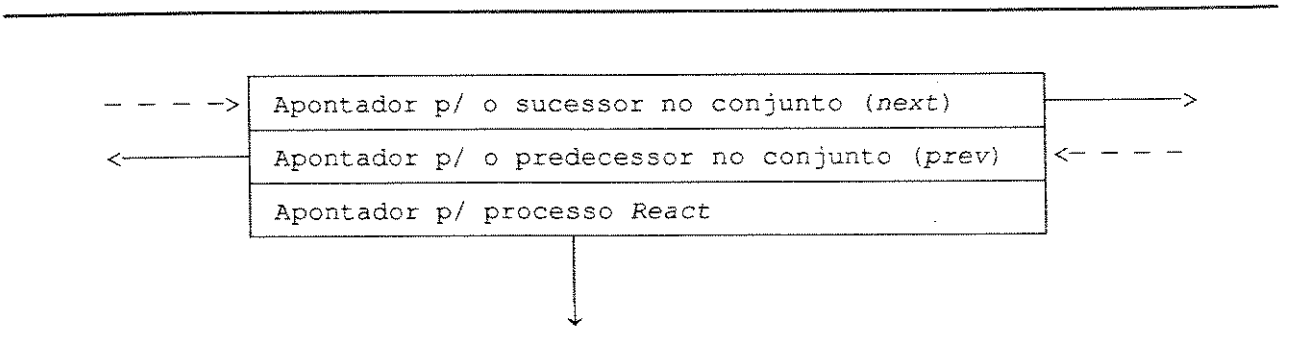

Figura 3.19: Estrutura básica da classe **Elemento**.

O terceiro componente é um apontador para o método `React` do objeto. Um mesmo processo pode ser apontado por mais de um elemento no modelo. Porém, cada elemento cria para si uma **instância** deste processo.

A classe **conjunto** tem um ponteiro chamado *Last*, que aponta para seu último elemento. Juntamente com os elementos do conjunto, este ponteiro forma uma lista circular de elementos (Fig. 3.20). Um conjunto vazio tem o *Last* igual a `nil` (nulo).

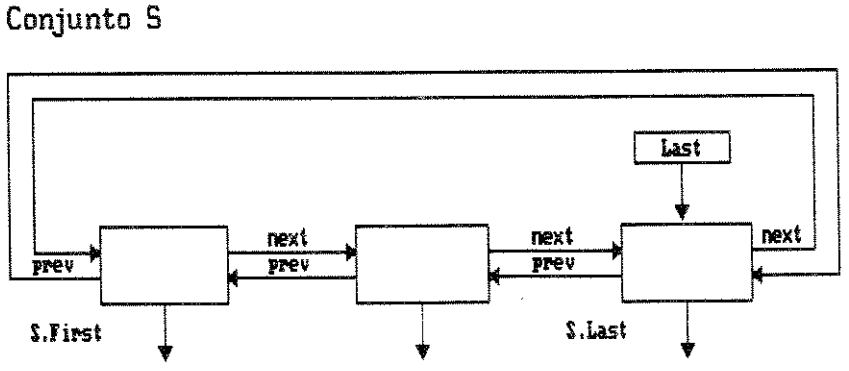

Figura 3.20: Conjunto com 3 elementos.

3.6.3 Controle de Sequência

As ações de um processo são agrupadas em fases ativas separadas por períodos de inatividade. Porém, apenas um processo está ativo executando ações em um dado instante. O período de atividade de um processo é causado pelo fim do bloco de execução associado ao seu estado atual. Tipicamente, os blocos de execução são finalizados com instruções que definem o próximo estado do objeto e que estipulam o intervalo de tempo no qual devem permanecer inativos. A primeira das instruções é efetuada pelo método `MudarEstado` (Fig. 3.18), que além de alterar o estado do objeto também atualiza sua situação (posição, representação, etc.) na tela do computador.

Para definir um período de inatividade, o núcleo de simulação oferece às classes do modelo o método `Delay(dt)`. Este método faz com que o objeto em questão seja novamente ativado depois de decorrido um tempo de simulação especificado (`dt`). As instruções `Delay` e `MudarEstado` definem um **ponto de reativação** do processo.

3.6.4 O Conjunto de Sequenciamento

Um processo é sempre tornado ativo devido à ocorrência de um **evento**. Cada evento tem associado um **tempo de sistema**, que permanece constante durante a execução da fase ativa do processo e que forma uma sequência não decrescente de números inteiros, representativos das horas, minutos e segundos de simulação.

Um evento pode ser escalonado para ocorrer imediatamente ou após decorrido um tempo determinado. O processo para o qual um evento ainda não ocorrido tenha sido escalonado tem uma **nota de evento** associada. Esta nota de evento (Fig. 3.21) consiste de um ponteiro para o objeto do processo associado e de um número inteiro denominado **referência de tempo**. As

notas de evento são ordenadas em um **conjunto de sequenciamento** (CSQ) na ordem crescente de suas referências de tempo.

O CSQ é responsável por ativar os processos `React` dos objetos do modelo na mesma sequência em que as notas de eventos estão dispostas no conjunto. Para cada processo existente no modelo, pode haver no máximo uma nota de evento.

A primeira nota de evento do CSQ é denominada **nota de evento corrente**. Seu processo associado é o atualmente ativo e sua referência de tempo é considerada como valor corrente do relógio de simulação. Quando a fase ativa deste processo é completada, o controle do programa é devolvido ao CSQ, que apaga a respectiva nota de evento, torna sua sucessora na lista a nota de evento corrente e transfere o controle do programa para o novo processo.

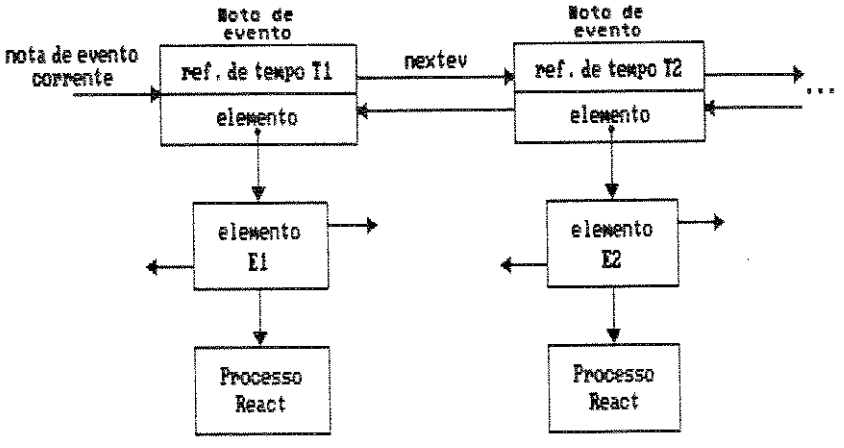

Figura 3.21: O Conjunto de Sequenciamento.
`E1.React` é o processo correntemente ativo, `E2.React` está suspenso e o relógio de simulação indica o tempo `T1`.

3.6.5 Estados dos Processos

Um processo pode estar em um de quatro **estados** possíveis: ativo, suspenso, passivo e terminado. A medida em que a simulação prossegue, os estados dos processos podem ser alterados.

Ativo — Um processo correntemente ativo pode, por **instruções de sequenciamento**, alterar os estados de outros processos e também de si próprio. Em um dado instante, existe apenas um processo ativo no modelo.

Suspenso	Um processo está suspenso quando tem uma nota de evento associada e que não é a nota de evento corrente. A menos que uma mudança de estado seja causada por um outro processo, sua próxima fase ativa se iniciará quando sua nota de evento se tornar a corrente.
Passivo	Um processo passivo não tem nota de evento associada. Ele permanecerá passivo até que uma mudança de estado seja causada por um outro processo.
Terminado	Um processo se torna terminado quando seu objeto associado é removido do modelo[7]. Vale ressaltar que o que é terminado é a instância do processo, e não sua definição (método `React`). Esta continua existindo no modelo, podendo estar associada ainda a outros objetos da célula.

3.6.6 Instruções de Sequenciamento

Instruções de sequenciamento atuam sobre o CSQ alterando os estados dos processos, como ilustrado na Fig. 3.22. Uma instrução de sequenciamento pode remover uma nota de evento, escalonar um novo evento gerando uma nota e incluindo-a no CSQ ou reescalonar um evento anteriormente definido, alterando a posição de sua nota de evento no CSQ.

O conjunto de sequenciamento é implementado como uma classe do sistema (`CSQ`), tendo como métodos as seguintes instruções:

(1) `CSQ.Cancelar(elemento)` remove do CSQ a nota de evento, se existir, associada ao processo do elemento especificado. O processo torna-se então passivo;

(2) `CSQ.Terminar(elemento)` cancela o processo do elemento especificado e encerra sua participação no modelo;

(3) Instruções de escalonamento.

Uma **instrução de escalonamento** pode criar uma nota de evento para um processo especificado e incluí-la no CSQ ou alterar a referência de tempo de uma nota de evento já existente (reativação).

7. O fato do objeto ser removido do modelo não impede o sistema de análise de consultá-lo após a simulação.

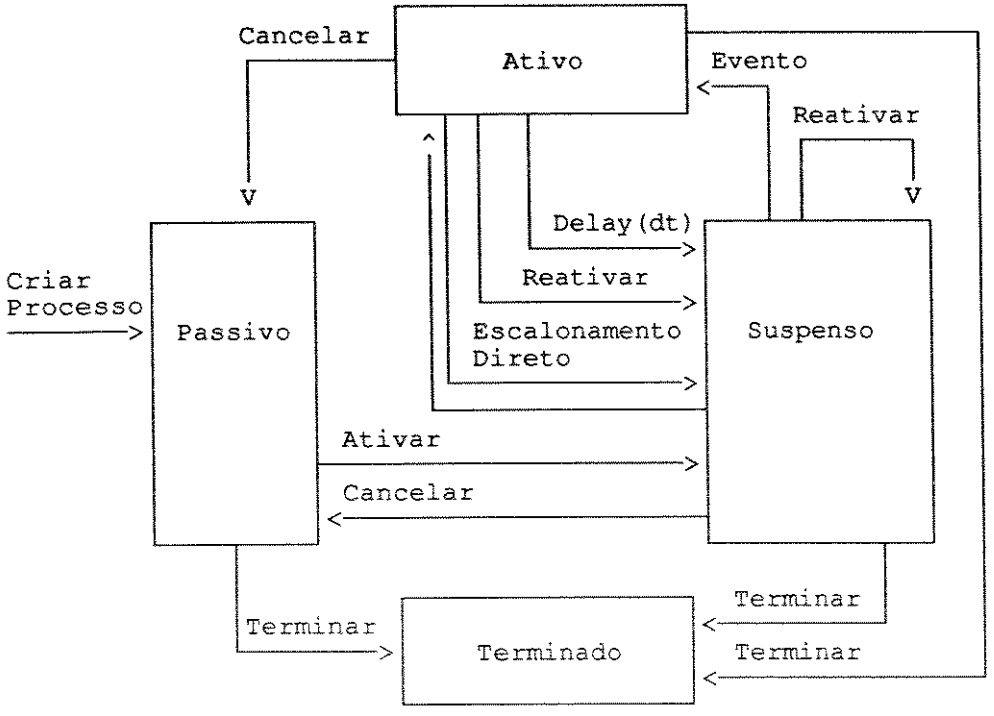

Figura 3.22: Diagrama de Estados de Processos.

As instruções de escalonamento básicas são:

```
CSQ.Ativar          (elemento,  tempo,  prioridade)
CSQ.Reativar        (elemento,  tempo,  prioridade)
CSQ.AtivarAntes     (elemento1, elemento2)
CSQ.ReativarAntes   (elemento1, elemento2)
CSQ.AtivarDepois    (elemento1, elemento2)
CSQ.ReativarDepois  (elemento1, elemento2)
CSQ.AtivarDelay     (elemento,  intervalo_de_tempo)
CSQ.ReativarDelay   (elemento,  intervalo_de_tempo)
```

A instrução `Ativar` causa a geração de uma nota de evento apenas se o processo referenciado estiver passivo, enquanto que `Reativar` remove a nota de evento associada ao processo ativo ou suspenso e reescalona o evento.

O parâmetro `tempo` especifica a referência de tempo da nota de evento gerada, o que determina sua posição no CSQ. O parâmetro `prioridade` informa se a nota de evento deve ser inserida antes ou depois de todas as notas de mesma referência de tempo já existentes. Pode-se também especificar esta referência relativamente (`Antes` ou `Depois`) à referência de tempo da nota de evento associada a um segundo processo. A nota de evento gerada recebe a mesma referência de tempo que esta última. Se o segundo processo estiver passivo ou terminado, nenhum escalonamento é efetuado.

Nas instruções `AtivarAntes(elemento,evento_corrente.elemento)` e `ReativarAntes(elemento,evento_corrente.elemento)`, a nota de evento gerada é colocada na frente da nota de evento corrente, tornando suspenso o processo até então ativo. O evento escalonado passa a ser assim o evento corrente. Isto é denominado **escalonamento direto** [Dah66]. A única restrição imposta pelo SISMA a estas duas instruções de escalonamento é que elas devem ser sempre a última instrução de um bloco de execução do processo.

A instrução `AtivarAntes(elemento,evento_corrente.elemento)` é portanto algo similar a uma chamada de procedimento , a medida em que as ações são evocadas como uma "subrotina" para o processo chamador. Nota-se, entretanto, que a nota de evento associada ao processo chamador pode ser removida ou reescalonada antes do controle retornar.

Com estas instruções de sequenciamento, podem-se promover comunicação entre dois processos concorrentes através de variáveis não-locais. Dois processos `X` e `Y`, chamando um ao outro através de, respectivamente, `ReativarAntes(Y, X)` e `ReativarAntes(X, Y)`, funcionariam como corrotinas [Tan87].

3.6.7 Controle da Célula

O gerenciamento da produção durante o processo de simulação é desempenhado pela classe denominada **controlador da célula**. Ela é responsável pela atribuição de tarefas de produção aos processadores a partir do sequenciamento de produção especificado pelo usuário. Todos os modelos criados no SISMA recebem automaticamente um objeto desta classe. Suas principais funções são:

Escalonamento	O controlador pode ativar, suspender e terminar processos associados aos objetos do modelo.
Sequenciamento	O controlador atribui tarefas aos objetos baseado nas rotas de produção do modelo e nos planos de sequenciamento de produção estabelecidos para cada um dos processadores.
Validação	O controlador é responsável pela compilação e armazenamento das rotas de produção e dos planos de sequenciamento de produção do modelo.

O processo de produção de uma célula de manufatura simulada no SISMA é dividido em três camadas, como ilustrado na Fig. 3.23. A camada 3, representada pelo controlador da célula, é responsável pelo gerenciamento da produção através de atribuições de tarefas elaboradas a partir de um plano de sequenciamento de produção especificado. A camada 2, representada pelos processadores, recebe estas tarefas e se encarrega da execução dos processos de produção através de pedidos de transporte de partes enviadas à camada 1, representada pelos transportadores, e de procedimentos internos de transformação de partes.

Todos os elementos das três camadas são autônomos, ou seja, desempenham suas tarefas de maneira independente, tomando suas próprias decisões. Desta maneira, há uma divisão bem definida de trabalho entre as camadas do modelo. O controlador não precisa se preocupar em controlar diretamente uma máquina ou um robô durante o processo de produção. Basta apenas designar para o processador uma tarefa do tipo: "Transforme uma parte P1 em duas partes P2", ou melhor, "P1 → 2P2"[8]. O processador então se encarrega de solicitar uma parte P1 ao transportador definido nas rotas de produção, transformá-la em duas partes P2 e novamente pedir a um transportador que as retire transportando-as para o destino indicado também nas rotas de produção. Terminada a operação, o processador envia uma mensagem de tarefa concluída ao controlador da célula, que por sua vez poderá atribuir uma nova tarefa ao processador.

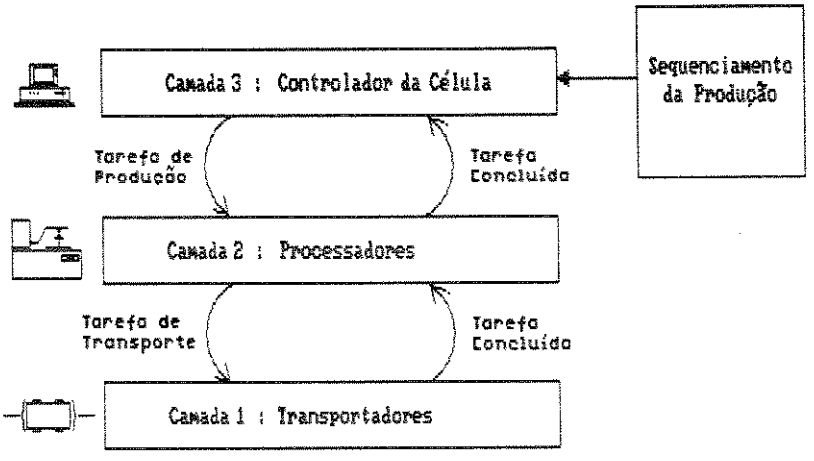

Figura 3.23: Arquitetura de controle do processo de produção.

3.6.8 Métodos Estatísticos e Sistema de Análise

Os métodos estatísticos, como os utilizados para geração de números aleatórios a partir de funções de distribuição de probabilidade, são aqueles largamente encontrados na literatura ([Fis73], [Fis78], [Gor78] e [Har75]). Na parametrização dos elementos da célula, o usuário pode em alguns casos definir a função de distribuição de probabilidade a ser utilizada na determinação de certo parâmetro (tempo de processamento, taxa de falhas, geração de eventos, etc). São 5 as funções de distribuição de probabilidade disponíveis no SISMA :

- Bernoulli,
- Binomial,
- Exponencial,
- Normal, e
- Poisson.

8. Sintaxe adotada no SISMA.

O SISMA possui também um sistema de análise para geração de relatórios com informações estatísticas acumuladas no decorrer da simulação. Estas análises podem ser feitas sobre apenas um elemento da célula, sobre um grupo de elementos ou sobre toda a célula[9]. Entre as estatísticas incluídas nos relatórios estão:

- Contadores (por exemplo, número de peças processadas por uma máquina, número de passagens de um AGV por certo ponto da célula, etc.)
- Medições Resumo (por exemplo, valores máximos, médias e desvios padrões do volume de partes em determinado buffer)
- Utilização (por exemplo, percentagem de tempo em que uma máquina esteve em operação)
- Ocupação (por exemplo, percentagem média de máquinas em uso dentro de um grupo de máquinas)
- Distribuições (por exemplo, tamanhos de filas ou tempos de espera)
- Tempos de Trânsito (por exemplo, tempo de deslocamento de um AGV ou de uma parte de um ponto a outro da célula)

A maioria dos relatórios são gerados sob a forma de gráficos e histogramas, como os ilustrados na Fig. 3.24. No protótipo desenvolvido, entretanto, esta característica não foi implementada.

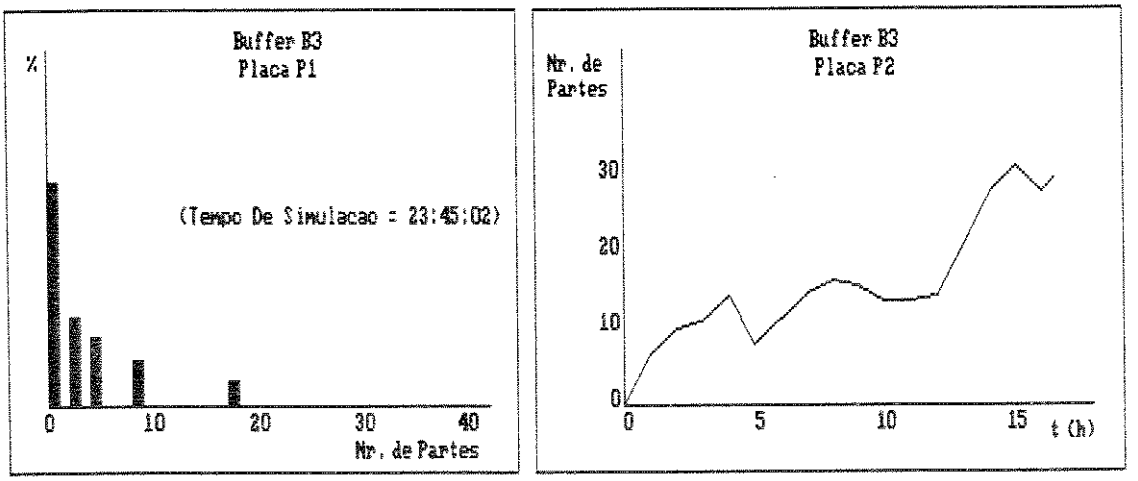

Figura 3.24: Exemplos de gráficos gerados pelo sistema de análise.
(a) Distribuição no tempo do nº de partes em um buffer.
(b) Volume de partes armazenadas em um buffer durante a simulação.

9. No protótipo, não foi implementado o conceito de grupo de elementos. Portanto, as análises podem ser feitas apenas sobre um elemento do modelo ou sobre toda a célula.

3.7 Base de Dados

A base de dados do SISMA é composta por uma **biblioteca de objetos**, um conjunto de **modelos de células de manufatura** editados pelo usuário e **históricos de simulação** gerados a partir de simulações dos modelos criados. O SISMA poderá futuramente suportar também uma *base de conhecimento*, na forma de Regras de Produção. Os arquivos da base de dados são descritos a seguir.

3.7.1 Arquivos de Modelos

Um arquivo de modelo de célula contém todas as informações referentes à estrutura e às características do sistema de manufatura. O modelo de célula é gerenciado pelo bloco modelador (seção 3.5) durante a etapa de modelagem do sistema. Na etapa de simulação, depois de submetida a uma verificação de consistência, o modelo de célula é manipulado pelo núcleo de simulação, que, ativando os processos de cada elemento do sistema, permite a troca de mensagens entre os objetos da célula. Esta interação provoca alterações no estado dos elementos, simulando assim o processo no mundo real.

Os arquivos de modelos são arquivos sequenciais de objetos. Todas as classes implementadas no SISMA têm dois métodos, `Load` e `Store`, para carregamento e armazenamento de seu conteúdo (campos) em arquivo. Isto permite que uma lista de objetos seja totalmente transferida da memória para um arquivo e vice-versa. Um arquivo de objetos pode conter um número ilimitado de instâncias, podendo estas pertencerem a classes diferentes, com tamanhos diferentes.

3.7.2 Biblioteca de Objetos

A biblioteca de objetos contém elementos comumente utilizados em células de manufatura e é mantida pelo modelador, que, a partir de comandos do usuário, pode buscar instâncias para composição do modelo, apagar instâncias existentes e adicionar novas instâncias criadas durante a modelagem. Os ícones dos objetos são concebidos utilizando-se o editor gráfico Fontasy[10], disponível comercialmente [PRO86]. Esta ferramenta provê facilidades adequadas para edição gráfica de figuras e salvamento destas em arquivos com formato accessível. O SISMA "entende" este formato a partir de um método implementado na classe **Ícone** para leitura de tais arquivos.

Uma biblioteca de objetos é um arquivo de instâncias associado a um *arquivo de índice*, cuja função é permitir um acesso não-sequencial aos elementos contidos na biblioteca. Cada componente do arquivo de índice indica a posição de um objeto no arquivo de instâncias. A chave de acesso a um objeto é seu nome de referência (por exemplo, IBM_7545, PUMA_600, etc.).

10. Utilizou-se uma cópia oficial pertencente ao CPqD-Telebrás.

O SISMA suporta várias bibliotecas de objetos, separadas em arquivos distintos. As bibliotecas podem ser organizadas de acordo com fabricantes de equipamentos, com as classes dos objetos nelas armazenados ou mesmo com as características das instâncias utilizadas por diferentes usuários.

3.7.3 Históricos de Simulação

Um arquivo de histórico armazena eventos ocorridos durante a simulação, que sejam significativos para um pós-processamento. Este histórico é mantido pelo núcleo de simulação e é constituído de uma coletânea de eventos ocorridos, que representam passo-a-passo o comportamento da célula de manufatura. A partir deste histórico, é possível gerar relatórios estatísticos referentes ao desempenho do sistema durante a simulação.

3.8 Interface com Usuário

O bloco **Interface com Usuário** (Fig. 3.3) é composto de um ambiente de trabalho gráfico, sobre o qual atuam as janelas do sistema, e de um conjunto de classes responsáveis pelos recursos de interação com o usuário.

3.8.1 Descrição do Ambiente

O ambiente de trabalho do SISMA é ilustrado na Fig. 3.25. O ícone "?" localizado no canto inferior direito da tela indica a existência de auxílio ao usuário. Para acioná-lo, basta levar o cursor do *mouse* até ele e, imediatamente, uma janela com um texto de introdução ao sistema é aberta no centro da tela. Este texto ensina ao usuário os princípios básicos de utilização do SISMA (Fig. 3.26) e indica os passos necessários para modelagem e simulação de uma célula de manufatura. O serviço de ajuda fornecido por este ícone é **sensível ao contexto**, ou seja, o assunto abordado pelo texto de auxílio se refere ao ponto do programa no qual o usuário está no momento do pedido de ajuda.

Dois dos botões do *mouse* têm significados únicos em todo o sistema. O botão esquerdo está sempre associado à função "Entrar" e pode ser substituído pela tecla Return. O botão direito tem sempre a função de "Cancelar", sendo similar à tecla ESC. O botão central, opcional em alguns *mice*, tem sua função dependente da aplicação. No interior da janela de modelagem, por exemplo, o botão central, quando pressionado, desloca a janela de posicionamento fazendo com que o ponto indicado pelo cursor passe a ser o centro desta janela. Este botão pode ser substituído pela barra de espaço.

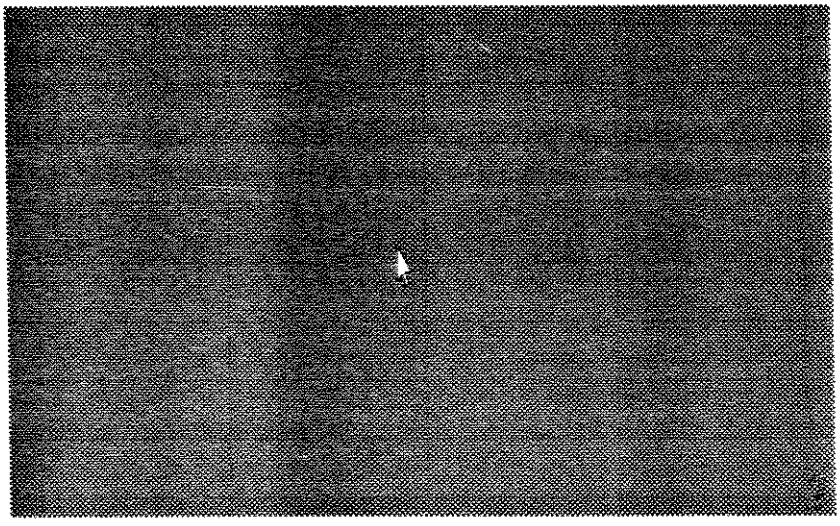

Figura 3.25: Ambiente de Trabalho do SISMA

O menu de comandos do ambiente é ativado pressionando-se o botão esquerdo do *mouse* com o cursor apontando para o "pano de fundo" do ambiente (estando sobre uma janela ou sobre um ícone, um outro menu, referente ao objeto apontado, será ativado). Sua estrutura é ilustrada na Fig. 3.27.

Arquivos

Esta opção permite carregar e salvar modelos em disco, bem como verificar e mudar de diretório.

- **Carregar** – Carrega do disco um arquivo de modelo já existente. Se modelo corrente estiver modificado, o programa pergunta se o usuário deseja salvá-lo antes de carregar o novo arquivo. Esta opção também é acessada de qualquer lugar do programa através da *hot-key* F3.
- **Novo** – Limpa a memória de modelagem e inicia a edição de um novo modelo.
- **Salvar** – Salva em disco o arquivo correntemente editado (*hot-key* F2).
- **Outro nome** – Salva o modelo corrente em outro em um novo arquivo.
- **Diretório** – Permite a visualização dos arquivos localizados no diretório corrente.
- **Mudar dir** – Muda o *drive* e o diretório correntes.

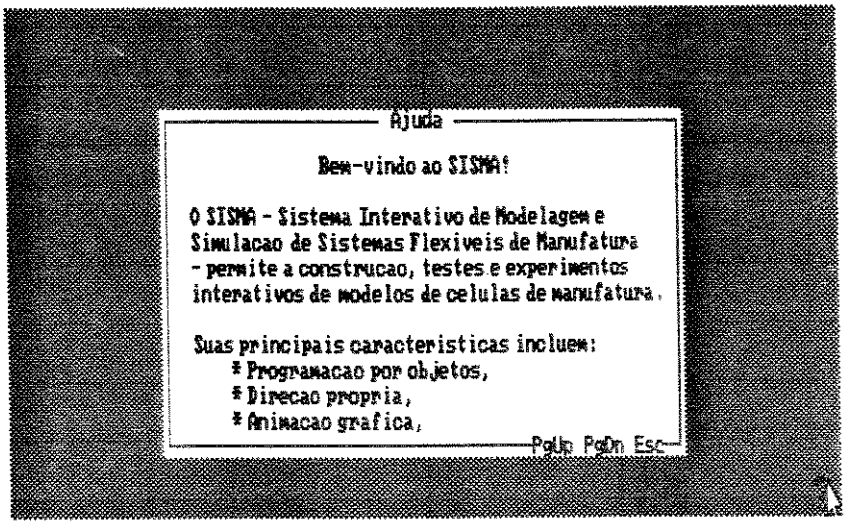

Figura 3.26: Janela de ajuda com texto de apresentação do sistema

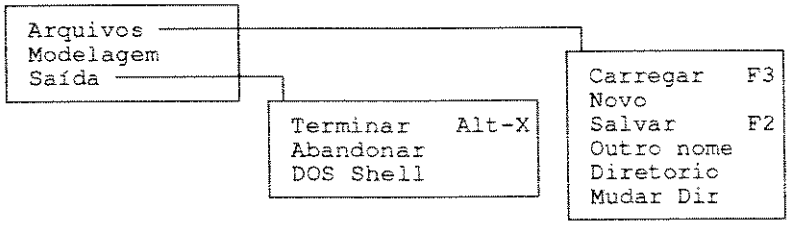

Figura 3.27: Estrutura do Menu de Ambiente do Sistema

Modelagem

Esta opção faz abrir a janela de posicionamento do sistema de modelagem (Fig. 3.5) ativando o sistema de captura de esquemático.

Saída

Esta opção oferece três alternativas de encerrar a execução do programa :

- **Terminar** – Salva o modelo e retorna ao DOS (*hot-key* Alt-X).
- **Abandonar** – Retorna ao DOS sem salvar o modelo.

- **DOS Shell** – Passa temporariamente o controle do computador ao DOS deixando o SISMA residente em memória. Para retornar ao programa, o usuário deverá entrar o comando `exit` após o *prompt* do sistema operacional ('C>').

3.8.2 Descrição das Classes da Interface

Um conjunto de classes destinadas à interação do sistema com o usuário foi implementado. Estas classes possibilitam o acesso ao teclado e ao *mouse* através de filas de eventos e a utilização de janelas gráficas, ícones e menus. A estrutura hierárquica das classes é ilustrada na Fig. 3.28.

Uma breve descrição de cada uma das classes implementadas é dada a seguir.

Janela Gráfica

A classe `Janela_Gráfica` é o tipo base de todos os menus e janelas do **Sistema de Gerenciamento de Múltiplas Janelas** [Borb90] implementado no SISMA. Neste sistema, qualquer janela deve salvar a área de tela a ser ocupada imediatamente antes de sua abertura, de modo a poder recompor a tela após ser fechada. Para este gerenciamento, o sistema um *buffer de janelas* e uma *pilha de descritores* que armazenam as áreas ocupadas pelas janelas e os apontadores para estas áreas, respectivamente. O sistema suporta até 20 janelas abertas simultaneamente.

Esta classe também especifica os campos que indicam coordenadas, dimensões, títulos, rodapés, atributos de cor e tipos de borda das janelas. As informações contidas nestes campos podem ser melhor visualizados através da Fig. 3.29. Além disso, a classe define métodos que permitem abrir, fechar, escrever texto, desenhar gráficos, limpar, mover e redimensionar janelas.

Janela Virtual

Esta classe implementa a filosofia de janelas virtual e de posicionamento descrita na Seção 3.5.1 e utilizada no sistema de captura de esquemático do modelador.

Menus Horizontais e Verticais

Os menus de opções implementados se baseiam no sistema adotado pela Borland Inc. nas interfaces com usuário de seus produtos [Borl89b]. O menu horizontal é uma barra de opções onde cada uma delas, quando escolhida, pode resultar em um menu vertical, aberto imediatamente abaixo da opção. Neste menu, cada ítem escolhido pode causar a abertura de um novo menu vertical em casacata, e assim sucessivamente como exemplificado na Fig. 3.30. Ambos os menus têm suporte a ajuda sensível ao contexto associada a cada uma de suas opções e podem ser manipulados com o *mouse*.

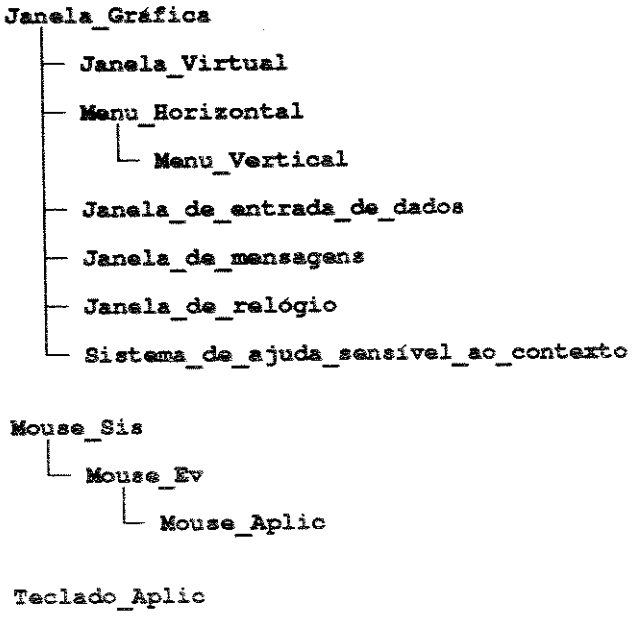

Figura 3.28: Hierarquia das classes da interface gráfica.

Janela de Entrada de Dados

Através desta classe de janela, é possível pedir ao usuário a entrada de um *string* de maneira elegante e com suporte a edição de linha e ajuda sensível ao contexto. As janelas de entrada de dados podem ser utilizadas em conjunto com os menus verticais, como ilustrado na Fig. 3.30.

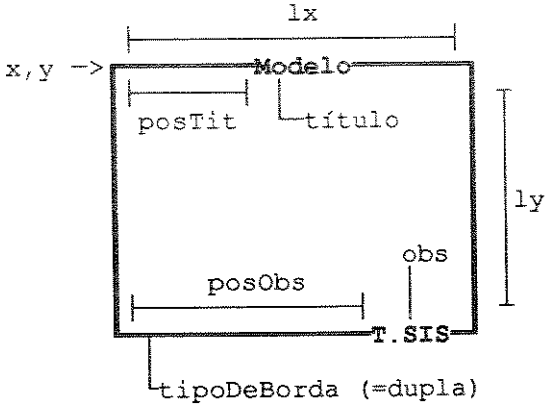

Figura 3.29: Campos do Objeto `Janela_Gráfica`

Janelas de Mensagens

As janelas de mensagens suportam dois tipos de comportamento. O primeiro é utilizado para mensagens de erro, as quais são sempre seguidas da mensagem *Press. ESC* como forma de obter uma confirmação do usuário. O segundo tipo é utilizado para mensagens de status, quando a mensagem pode permanecer na tela por alguns segundos e então desaparecer.

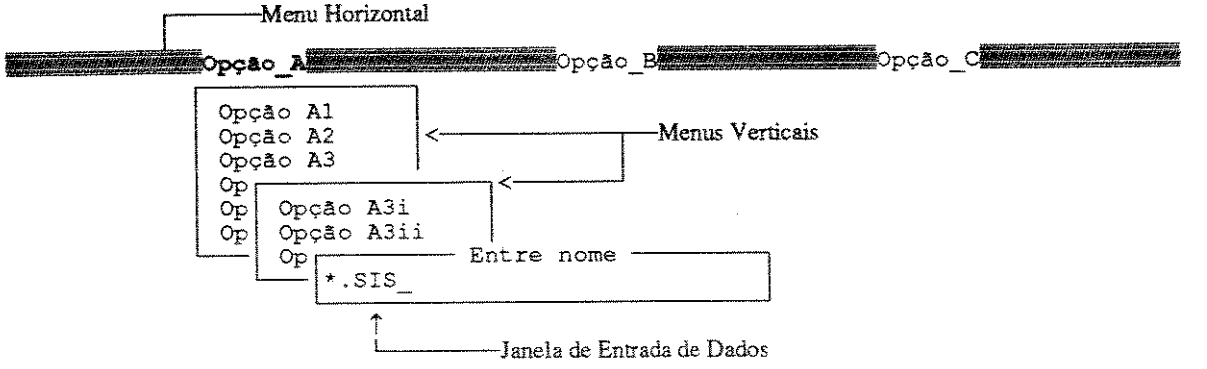

Figura 3.30: Menus Horizontais e Verticais

Janela de Relógio

Esta janela é ativada durante a simulação do modelo e mostra, no seu interior, o valor atual do relógio de simulação (Fig. 3.31).

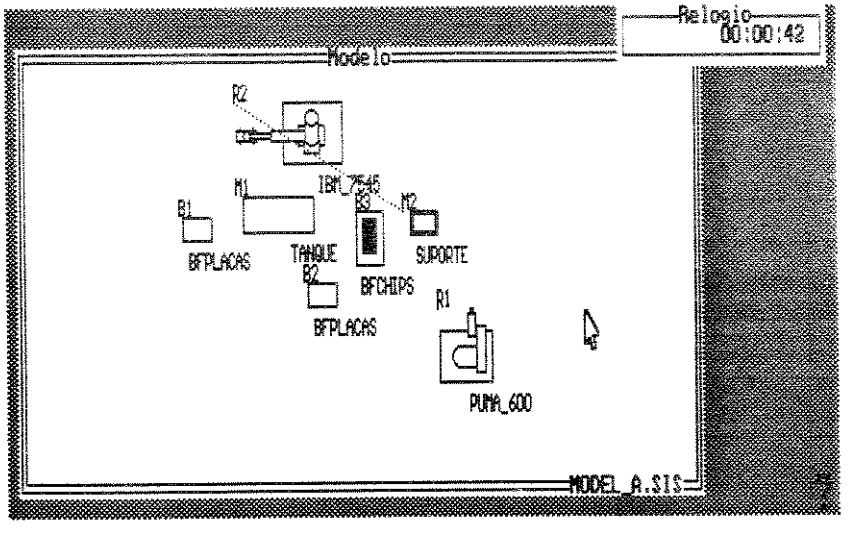

Figura 3.31: Janela do Relógio de Simulação

Sistema de Ajuda Sensível ao Contexto

Este sistema, em qualquer situação do programa, permite a abertura de uma janela de ajuda contendo textos explicativos sobre um determinado tema, referente ao estado atual do programa. Isto normalmente ocorre quando o usuário pressiona a tecla de socorro [F1] ou pressiona o botão esquerdo do *mouse* sobre o ícone de ajuda '?'.

Os textos de auxílio são armazenados em um arquivo-texto contendo também as palavras-chave de acesso a cada tema. Estas chaves são utilizadas pela aplicação para indicar ao sistema de ajuda qual texto deve ser fornecido ao usuário. O formato deste arquivo é ilustrado na Fig. 3.32.

```
Comentarios
.CHAVE1
  ...
  Texto de ajuda
  referente
  a CHAVE1
  ...
.CHAVE2
  ...
  Texto de ajuda
  referente
  a CHAVE2
  ...
...
```

Figura 3.32: Formato do arquivo de ajuda.

As palavras-chave devem sempre começar com um '.', que deve ser o primeiro caracter da linha. O delimitador do texto de ajuda é uma linha com outra palavra-chave ou o fim de arquivo. O texto entre o início do arquivo e a primeira palavra-chave não é considerado pelo sistema, podendo ser utilizado como área de comentário.
Se o texto exceder o número de linhas da janela de ajuda, o usuário pode navegar por ele através das teclas [PgDn] e [PgUp], ou dos botões direito e esquerdo do *mouse*.

Teclado

Esta classe implementa o sistema gerenciador de teclado que possibilita a utilização de *hot-keys* para permitir a escolha direta de opções localizadas em menus horizontais ou verticais, independentemente do nível de profundidade destes.

A classe `Teclado_Aplic` é composta de um buffer circular interno de teclas e quatro métodos de acesso, permitindo a leitura das teclas pressionadas pelo usuário e a simulação via *software* do pressionamento de uma ou mais teclas.

A simulação de teclas tem duas aplicações bastante práticas: a tradução de *hot-keys* e a restauração de teclas. A tradução de teclas pode ser exemplificada pela utilização no ambiente do SISMA da tecla [F3] para abreviação de um [Enter] [A] [C] (Arquivo/Carregar). A rotina que lê a

tecla [F3] simplesmente simula o pressionamento destas três teclas inserindo-as no buffer de teclas do sistema gerenciador (utilizando um dos métodos disponíveis). O segundo exemplo, a restauração de teclas, é utilizado numa sequência de menus e submenus em cascata quando, em um determinado instante, um submenu é cancelado por alguma tecla definida como *hot-key*. O submenu que leu esta tecla deve restaurá-la no buffer de teclas antes de retornar. Isto faz com que o menu ascendente (chamador) também possa considerá-la e interpretá-la convenientemente (se for o caso, restaurando-a novamente no buffer de teclas para que possa ser lida pelo próximo menu ascendente).

Os métodos de acesso ao Sistema Gerenciador de Teclado permitem:

- Verificar se alguma tecla foi pressionada ou simulada;
- Ler uma tecla do buffer;
- Inserir uma tecla no buffer;
- Restaurar a última tecla retirada do buffer;

Mouse

O controle do *mouse* é particionado entre três classes, formando a estrutura hierárquica ilustrada na Fig. 3.28. A classe `Mouse_Sis` implementa uma interface Pascal com o *driver* que acompanha o dispositivo, traduzindo os chamados de interrupção de acesso às funções do *mouse* em métodos da classe. Estes métodos incluem :

- Verificação da instalação do *driver*;
- Verificação da instalação do *mouse*;
- Verificação do tipo de *mouse* instalado;
- Leitura do estados dos botões;
- Leitura e definição de posicionamento; e
- Definições de regiões.

A classe `Mouse_Ev` extende as características da classe anteriormente descrita ao criar um buffer de eventos atualizado por interrupção. Este buffer armazena os eventos causados pelo pressionamento dos botões ou deslocamento do mouse, de forma a garantir à aplicação que nenhuma ação do usuário será perdida (Fig. 3.33). Os diversos tipos de eventos podem ser "mascarados", permitindo à aplicação a definição de um armazenamento seletivo.

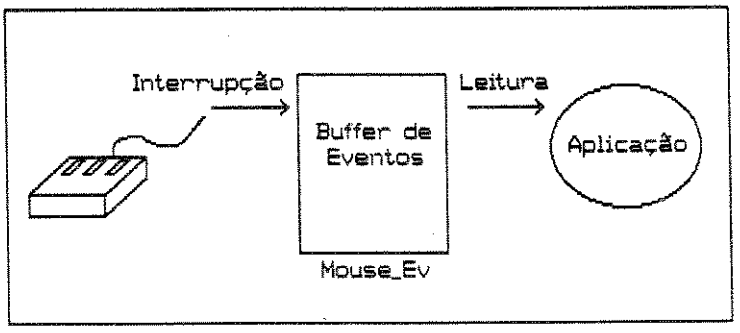

Figura 3.33: Tratamento de interrupções do *mouse* efetuado pela classe `Mouse_Ev`. Os eventos são armazenados para posterior leitura da aplicação.

A classe `Mouse_Aplic` é a interface direta da aplicação com o dispositivo. Ela herda todas as características da classe `Mouse_Ev` e define todos os ícones do *mouse* utilizados pela aplicação. Estes ícones, utilizados para identificar as funções do *mouse* em diversas situações do programa, são descritos na Fig. 3.34.

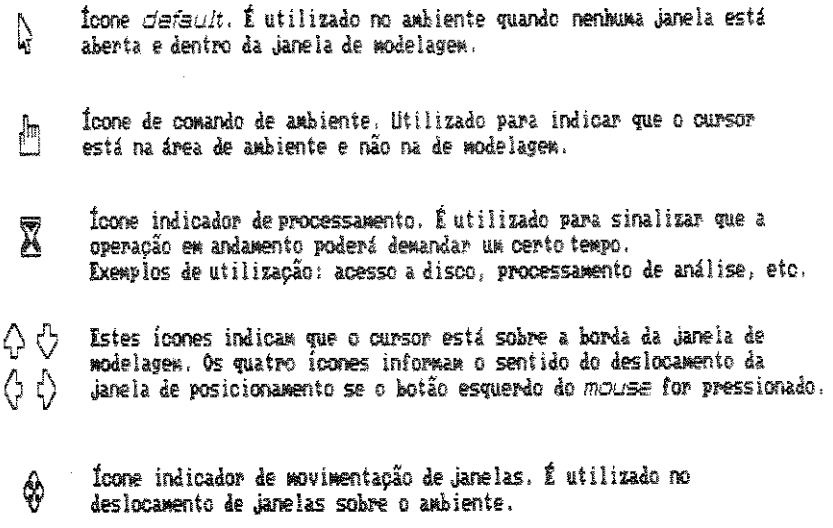

Figura 3.34: Ícones utilizados no cursor do *mouse*.

3.9 Resumo

Neste capítulo, foram apresentadas as principais características do SISMA - Sistema Interativo de Modelagem e Simulação de Sistemas Flexíveis de Manufatura. Foi dada uma visão geral da sua capacidade e do seu funcionamento. Definiu-se uma árvore semântica dos elementos que

compõem um modelo de célula de manufatura do SISMA e especificamos funcionalmente cada um destes objetos. Foram explicados os conceitos de rota de produção, pontos de entrada e pontos de saída. Por fim, foi descrito de maneira geral a estrutura de implementação do SISMA.

No próximo capítulo, serão mostrados os resultados obtidos com a implementação do primeiro protótipo do simulador, incluindo a descrição do processo de modelagem e a simulação de uma célula encontrada na literatura.

Capítulo 4

RESULTADOS DE SIMULAÇÃO

4.1 Introdução

Neste capítulo, os resultados obtidos com o SISMA serão ilustrados com um exemplo de aplicação extraído do trabalho apresentado por Levas e Jayaraman em [Lev89]. Este exemplo consiste de uma célula aplicada à montagem de cartões de circuito impresso. Detalhes adicionais deste exemplo e do SISMA em geral podem ser encontrados em [Borb92].

4.2 Descrição da Célula de Montagem

A célula utilizada tem por tarefa montar placas de circuito impresso (PCI) preenchendo-as com dois tipos de circuitos integrados (*chips*). As especificações de montagem informam a quantidade de *chips* e suas respectivas posições nas placas[11]. Além de inseridos, os componentes também devem ser soldados nas PCIs. Um plano empregando dois robôs foi proposto para obter-se concorrência nas operações e para acomodar os requisitos de manipulação de diferentes partes na aplicação. O *layout* da célula de montagem é ilustrado na Fig. 4.1.

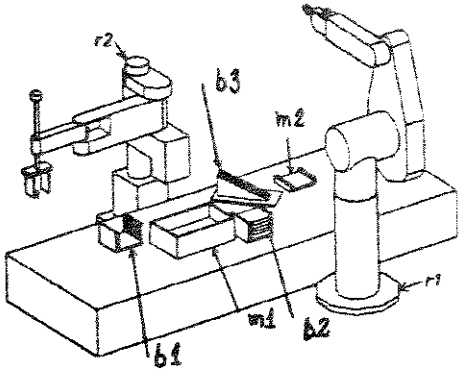

Figura 4.1: *Layout* da Célula de Montagem de PCIs.

11. Efetivamente, o modelador não necessita desta última informação, mas sim do tempo necessário para a colocação de cada componente na placa.

Um robô é utilizado para inserir os *chips* nas PCIs e outro, para transportar e soldar as placas montadas. A concorrência nas operações dos robôs torna-se evidente após a primeira placa ter sido preenchida. Os componentes são fornecidos à célula através de um alimentador do tipo "*gravity feeder*" [Lev89]. Dois buffers são utilizados: um para armazenar placas nuas e outro para armazenar placas prontas e semi-prontas. Além disso, um suporte é colocado na célula para segurar cada PCI durante a inserção dos *chips*. Um tanque de solda é utilizado para a operação final da produção. O processo é descrito passo-a-passo na Tabela 4.1. Esta tabela ilustra a comunicação necessária entre os dois robôs, dado que seus espaços de trabalho se sobrepõem.

Dois modelos de robôs são utilizados: um PUMA 600 (**r1**) para inserção de *chips* e um IBM 7545 (**r2**) para manipulação de placas. Ambos são modelados como instâncias da classe **Robô**. O alimentador de componentes (**b3**) foi modelado como uma instância de **Buffer**, já que esta classe suporta o armazenamento de diferentes tipos de partes. Nesta classe, também foram instanciados os dois buffers de placas da célula (**b1** e **b2**).

O suporte de PCI (**m2**) foi caracterizado como um processador pelo fato de sobre ele ser procedida uma transformação (placa nua + componentes →placa montada), sendo assim modelado como instância da classe **Máquina**. O tempo de operação para esta transformação foi considerado nulo[12]. O tanque de solda (**m1**) também foi modelado como instância de **Máquina**. Sua operação é transformar placas montadas em placas soldadas, requisitando um operador para isto, no caso, o robô de manipulação (**r2**).

As partes envolvidas no processo foram identificadas como sendo os dois tipos de chips (**p4** e **p5**) e as placas nuas (**p1**), montadas (**p2**) e soldadas (**p3**). Todas foram modeladas como instâncias da classe **Parte**.

Nota-se, portanto, que toda a célula foi modelada com apenas 4 classes distintas: **Robô**, **Buffer**, **Máquina** e **Parte**. Mesmo com a relativa variedade de elementos (total de 12) foi possível utilizar um número bem menor de classes devido à grande abrangência obtida na implementação.

4.3 Captura do Esquemático

Com a utilização dos recursos de captura de esquemático apresentados no capítulo 3, foi possível especificar o *layout* da célula de montagem ilustrado na Fig. 4.2. Os designadores de referência (r1, r2, b1, etc.) foram atribuídos de acordo com o *layout* original (Fig. 4.1).

12. O tempo de operação compreende o intervalo de tempo entre a colocação da última parte necessária à transformação e o término desta. No caso do suporte de PCI a placa nua passa a ser considerada uma placa montada imediatamente após a inserção do último componente.

Tabela 4.1: Processos dos robôs da célula de montagem de placas de circuito impresso.

Robô Insersor de *Chips* (r1):

1.	Espere por um sinal do robô manipulador de PCIs para iniciar o preenchimento de uma nova placa.
2.	Ao receber este sinal, preencha a placa colocada no suporte com os *chips* apropriados (**b3 -> m2**).
3.	Ao terminar o preenchimento, mova-se para o lugar seguro e sinalize o robô manipulador de PCIs para remover a placa montada.
4.	Volte para o passo 1.

Robô Manipulador de PCIs (r2):

1.	Retire uma placa nua do buffer de placas nuas e coloque-a no suporte (**b2 -> m2**).
2.	Mova-se para o lugar seguro e sinalize o robô insersor de *chips* para começar a montagem.
3.	Espere por um sinal do robô insersor de *chips* indicando o término da montagem.
4.	Ao receber o sinal, remova a placa montada do suporte e coloque-a temporariamente no buffer de placas prontas (**m2 -> b1**).
5.	Retire uma placa nua do buffer de placas nuas e coloque-a no suporte (**b2 -> m2**).
6.	Mova-se para o lugar seguro e sinalize o robô insersor de *chips* para começar a montagem.
7.	Pegue a placa montada, leve-a para o tanque de solda e, após este processo, coloque-a de volta no buffer de placas prontas (**b1 -> m1 -> b1**).
8.	Volte para o passo 3.

4.4 Parametrização do Modelo

Na etapa de parametrização, uma série de perguntas agrupadas em capítulos são elaboradas pelo sistema. As perguntas variam de modelo para modelo dependendo dos objetos utilizados. A cada capítulo, o sistema elabora questões baseado nas respostas fornecidas pelo usuário nos capítulos anteriores. Para o exemplo de célula utilizado, as perguntas elaboradas e os parâmetros especificados como respostas são descritos a seguir.

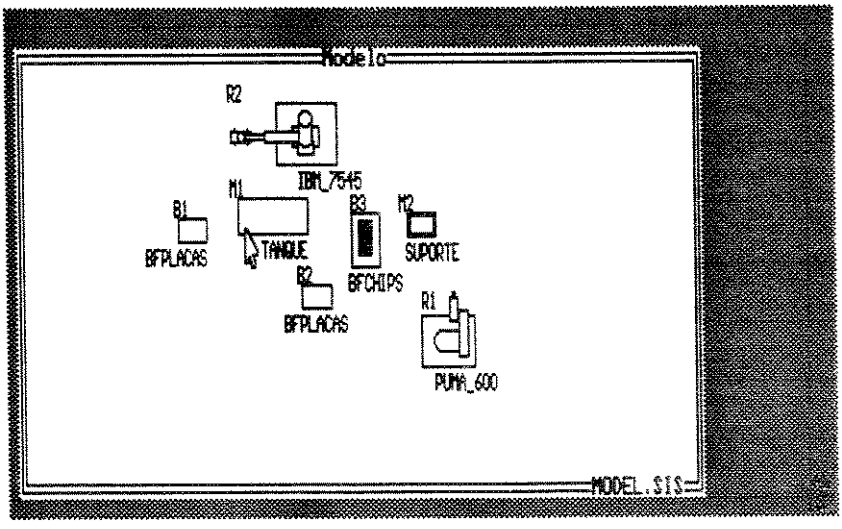

Figura 4.2: *Layout* da célula de montagem capturado com o SISMA.

Rotas de Produção

As seguintes rotas de produção foram especificadas:

Pergunta	**Resposta**	
Rota #1:	p1 : b2 r2-> m2	*(1)*
Rota #2:	p2 : m2 r2-> b1 r2-> m1	*(2)*
Rota #3:	p3 : m1 r2-> b1	*(3)*
Rota #4:	p4 : b3 r1-> m2	*(4)*
Rota #5:	p5 : b3 r1-> m2	*(5)*

A primeira coluna (**Pergunta**) corresponde às questões elaboradas pelo sistema de parametrização. A segunda coluna (**Resposta**) indica as informações fornecidas ao sistema. A terceira coluna é composta por uma numeração de referência.

A rota *(1)* informa que o robô manipulador de placas (**r2**) deve transportar placas nuas (**p1**) do buffer de placas nuas (**b2**) para o suporte de placas (**m2**).

A rota *(2)* especifica que as placas montadas (**p2**) são criadas no suporte de montagem (**m2**), levadas pelo robô manipulador de placas (**r2**) que as guarda temporariamente no buffer de placas prontas (**b1**) e, finalmente, transportadas deste buffer para o tanque de solda (**m1**) pelo mesmo robô. A descrição das demais rotas segue o mesmo raciocínio utilizado para as rotas *(1)* e *(2)*. A sexta rota de produção é especificada vazia para indicar que apenas cinco rotas são definidas.

Processadores

No protótipo implementado, apenas um processo por máquina pode ser definido. Portanto, na parametrização, somente as informações sobre os processos ativos dos processadores são solicitadas.

Pergunta	**Resposta**	
m2. Processo Ativo:	p1 + 2p4 + 2p5 -> p2	*(6)*
m2. Tempo de Processamento:	0	*(7)*
m2. Tempo de Configuração:	0	*(8)*
m2. Operador Requisitado:		*(9)*
m1. Processo Ativo:	p2 -> p3	*(10)*
m1. Tempo de Processamento:	14	*(11)*
m1. Tempo de Configuração:	0	*(12)*
m1. Operador Requisitado:	r2	*(13)*

O processo ativo do suporte de montagem (**m2**) indica que uma placa nua (**p1**) juntamente com duas unidades de cada tipo de *chip* (**p4** e **p5**) dão origem a uma placa montada (**p2**). Os tempos de processamento *(7)* e configuração *(8)* para esta operação são considerados nulos. A informação *(9)* especifica que nenhum operador é requisitado para acompanhar a operação.

Para o tanque de solda (**m1**), o processo ativo *(10)* indica que uma placa montada (**p2**) é transformada em uma placa soldada (**p3**). O tempo de processamento *(11)* é de 14 segundos, sendo necessária a presença do robô de manipulação de placas (**r2**) durante toda a operação. O tempo de configuração *(12)* foi especificado nulo.

As informações fornecidas pelo usuário até este ponto, permitem ao sistema conhecer a trajetória de todas as partes na célula e as quantidades necessárias a cada transformação. Nota-se que nenhuma informação quanto à sinalização entre robôs (envio e espera de sinais) foi especificada, pelo fato de toda esta sinalização entre transportadores ser **transparente** para o usuário. Os próprios objetos, por serem autônomos, cuidam de evitar conflitos e choques entre si.

Transportadores

Para tornar possível a simulação do modelo, o sistema precisa conhecer os tempos de deslocamento de cada transportador e os tempos necessários para colocação e retirada de partes nos diversos armazenadores do sistema (**Guarda-Partes**). Quanto aos tempos de deslocamento, apenas aqueles entre os pontos efetivamente visitados pelo transportador são pedidos. O mesmo acontece com os tempos de colocação e retirada de partes. O sistema de parametrização "filtra" as informações fornecidas nos dois capítulos anteriores (Rotas de Produção e Processadores) e pergunta somente o estritamente necessário.

Pergunta	Resposta	
r2. Local Seguro:	b2	*(14)*
r2. Tempo de Deslocamento b2-m2:	4	*(15)*
r2. Tempo de Deslocamento b2-b1:	4	*(16)*
r2. Tempo de Deslocamento b2-m1:	2	*(17)*
r2. Tempo de Deslocamento m2-b1:	8	*(18)*
r2. Tempo de Deslocamento m2-m1:	6	*(19)*
r2. Tempo de Deslocamento b1-m1:	2	*(20)*
r2. Tempo de Retirada b2, p1:	15	*(21)*
r2. Tempo de Retirada m2, p2:	10	*(22)*
r2. Tempo de Retirada b1, p2:	15	*(23)*
r2. Tempo de Retirada m1, p3:	1	*(24)*
r2. Tempo de Colocação m2, p1:	10	*(25)*
r2. Tempo de Colocação b1, p2:	15	*(26)*
r2. Tempo de Colocação m1, p2:	1	*(27)*
r2. Tempo de Colocação b1, p3:	15	*(28)*
r1. Local Seguro:	fora	*(29)*
r1. Tempo de Deslocamento b3-m2:	1	*(30)*
r1. Tempo de Deslocamento b3-fora:	2	*(31)*
r1. Tempo de Deslocamento m2-fora:	2	*(32)*
r1. Tempo de Retirada b3, p4:	20	*(33)*
r1. Tempo de Retirada b3, p5:	16	*(34)*
r1. Tempo de Colocação m2, p4:	20	*(35)*
r1. Tempo de Colocação m2, p5:	16	*(36)*

A informação *(14)* indica que a posição escolhida como local seguro do robô de manipulação de PCIs (**r2**) é a posição sobre o buffer **b2**. Os tempos de deslocamento em segundos entre os pontos visitados pelo robô (**b1**, **b2**, **m1** e **m2**) são especificados nos ítens *(15)* a *(20)*[13]. A questão *(21)* solicita o tempo que o robô leva para retirar uma placa nua (**p1**) do buffer de placas **b2**. A questão *(22)*, o tempo que o mesmo robô leva para retirar uma placa montada (**p2**) do suporte (**m2**), e assim sucessivamente. O mesmo procedimento é efetuado para os tempos de colocação, também de acordo com as rotas de produção especificadas.

As informações *(29)* a *(36)* se referem ao robô de inserção de componentes (**r1**). O local seguro "fora" é uma palavra reservada do sistema que indica um lugar externo à área de atuação da célula. Esta posição pode representar o braço recolhido ou levantado (Fig. 4.3).

13. Verifica-se que as perguntas (16), (17) e (19) para o robô r2, além da questão (31), não são indispensáveis à simulação. Apesar dos robôs visitarem todos os pontos mencionados, não há deslocamento entre os pontos indicados por estas perguntas. Um aperfeiçoamento do protótipo para fazê-lo dispensar estas questões deverá ser feito futuramente.

Buffers

Nesta etapa da parametrização, o sistema pergunta a quantidade inicial e a capacidade máxima de cada um dos buffers do modelo. Novamente, as rotas de produção são investigadas para identificação dos tipos de partes possíveis de serem armazenados em cada buffer da célula, de forma a reduzir o número de perguntas ao mínimo necessário.

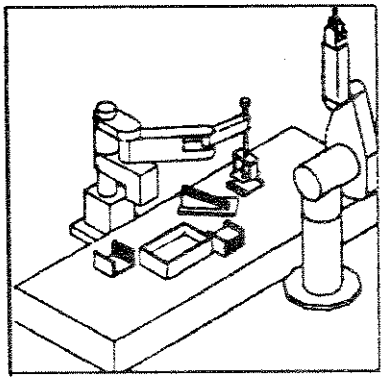

Figura 4.3: Robô insersor de *chips* (**r1**) com o braço levantado (lugar seguro).

No exemplo apresentado, o SISMA elaborou as seguintes questões:

Pergunta	Resposta	
b2. p1. Capacidade Máxima:	10	*(37)*
b2. p1. Quantidade Inicial:	10	*(38)*
b1. p2. Capacidade Máxima:	1	*(39)*
b1. p2. Quantidade Inicial:	0	*(40)*
b1. p3. Capacidade Máxima:	10	*(41)*
b1. p3. Quantidade Inicial:	0	*(42)*
b3. p4. Capacidade Máxima:	20	*(43)*
b3. p4. Quantidade Inicial:	20	*(44)*
b3. p5. Capacidade Máxima:	20	*(45)*
b3. p5. Quantidade Inicial:	20	*(46)*

Partes

Na última etapa do questionário, devem ser informados os nomes de cada um dos tipos de placas e suas respectivas prioridades no transporte. Os nomes são utilizados nos relatórios gerados e podem ter, cada um, no máximo 8 caracteres. A prioridade das placas é utilizada pelos transportadores durante a simulação. Quanto maior o valor atribuído, mais alta é a prioridade (máximo = 255).

Pergunta	**Resposta**	
p1. Nome:	PCI_NUA	*(47)*
p1. Prioridade no Tranporte:	1	*(48)*
p2. Nome:	PCI_MONT	*(49)*
p2. Prioridade no Tranporte:	0	*(50)*
p3. Nome:	PCI_SOLD	*(51)*
p3. Prioridade no Tranporte:	0	*(52)*
p4. Nome:	CHIP_PEQ	*(53)*
p4. Prioridade no Tranporte:	0	*(54)*
p5. Nome:	CHIP_GRD	*(55)*
p5. Prioridade no Tranporte:	0	*(56)*

Nota-se que as placas nuas (**p1**) têm maior prioridade no transporte que as demais. Isto é necessário para que o robô manipulador de PCIs (**r2**) priorize o fornecimento de placas nuas ao suporte de montagem (**m2**).

4.5 Simulação

A etapa de simulação é iniciada a partir de um comando do menu principal. Os tempos iniciais e finais de simulação devem ser especificados. O sistema confere a consistência do modelo e, se nenhum erro for detectado, o processo de simulação é iniciado.

A animação gráfica suportada pelo protótipo implementado é bastante simples. Apenas os movimentos dos braços dos robôs são representados na tela. Com este efeito de animação, a concorrência nas operações dos robôs pode ser verificada, como ilustrado na Fig. 4.4. Nesta figura, o robô **r1** insere um componente na placa presa ao suporte ao mesmo tempo em que o robô **r2** opera uma placa no tanque de solda.

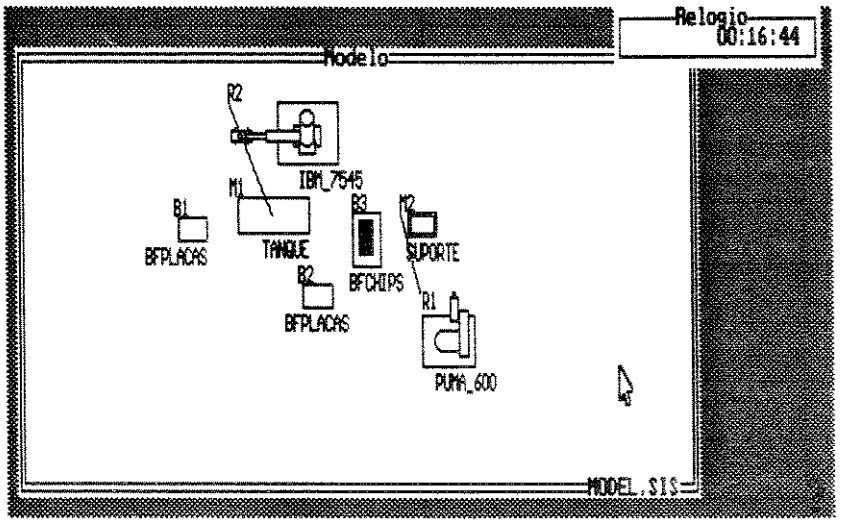

Figura 4.4: Concorrência nas operações dos robôs.

A simulação é encerrada quando o tempo final especificado é atingido ou quando a fila de eventos é encontrada vazia. No caso do exemplo apresentado, o simulador interrompeu o processo depois de aproximadamente 8 minutos de simulação (Fig. 4.5).

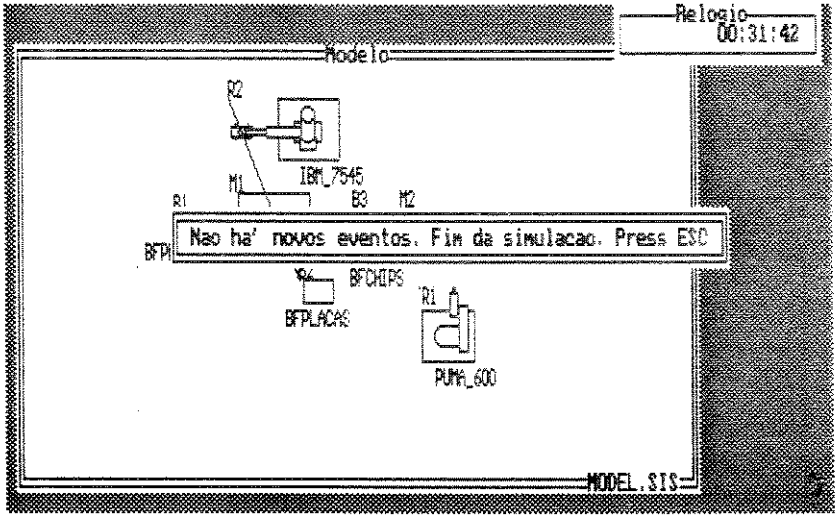

Figura 4.5: Término da simulação.

4.6 Análise de Desempenho da Célula

Os resultados obtidos com a simulação da célula de montagem apresentada são ilustrados na Fig. 4.6. Nota-se que o robô de inserção de *chips* (**r1**) está subutilizado, enquanto que o robô de manipulação de placas (**r2**) trabalha quase ao extremo de sua capacidade. O grau de utilização do tanque de solda (**m1**) também foi verificado ser muito baixo (aproximadamente 10%). O suporte de montagem (**m2**) teve 0% de utilização devido ao fato do seu tempo de operação ter sido especificado nulo.

```
Tempo total de simulação:      00:26:26 (HH:MM:SS)
Nº de partes produzidas:       10 PCI_SOLD

Utilização:
   r1 = 46.0%
   r2 = 97.2%
   m1 = 10.4%
   m2 = 0%
```

Figura 4.6: Resultados da simulação (célula com 2 robôs).

Com o objetivo de comparação, o mesmo processo de montagem foi simulado com apenas um robô (**r3**), supondo que este seja capaz de manipular placas e componentes com a mesma habilidade que os dois robôs utilizados no primeiro exemplo. Os resultados obtidos com esta nova configuração são ilustrados na Fig. 4.7.

```
Tempo total de simulação:      00:35:44 (HH:MM:SS)
Nº de partes produzidas:       10 PCI_SOLD

Utilização:
   r3 = 100%
   m1 = 6.5%
   m2 = 0%
```

Figura 4.7: Resultados da simulação (célula com 1 robô).

Verifica-se que a célula com um robôs foi apenas 35% mais lenta que a célula com dois robôs. Se considerarmos os preços dos três tipos de robôs equivalentes, a opção mais rentável seria contruir duas células independentes, cada uma com apenas um robô similar ao utilizado na segunda célula simulada.

4.7 Resumo

Neste capítulo, foram apresentados os resultados obtidos com a modelagem e simulação de uma célula de manufatura hipotética utilizando-se o protótipo implementado. Duas configurações desta célula foram simuladas e os respectivos resultados, comparados.

Capítulo 5

DISCUSSÃO E CONCLUSÕES

5.1 Introdução

Neste capítulo, são apresentadas as conclusões acerca do simulador implementado bem como algumas sugestões para continuidade deste trabalho.

5.2 Comentários e Considerações Gerais

O protótipo do SISMA foi desenvolvido utilizando 20.342 linhas de código fonte em Pascal e 663 linhas em Assembly, resultando em um arquivo executável com 156.448 bytes de tamanho[14]. Da memória disponível do microcomputador, o programa utiliza aproximadamente 220 KBytes, deixando o restante para armazenamento do modelo simulado.

A implementação do SISMA foi estruturada em 41 módulos (arquivos-fonte) com atribuições bem definidas e código bem comentado. A documentação detalhada do projeto pode ser encontrada em [Borb92].

Devido às características de reutilizabilidade da programação orientada por objetos, o SISMA tem como subproduto uma *toolbox* PASCAL para desenvolvimento de programas de simulação. A documentação necessária para utilização desta biblioteca de classes e procedimentos também pode ser encontrada em [Borb92].

5.3 Sugestões para Trabalhos Futuros

Nem todas as características especificadas neste trabalho foram implementadas no protótipo desenvolvido. Os esforços foram principalmente concentrados no desenvolvimento do sistema de modelagem (edição e parametrização do modelo), do núcleo de simulação e dos principais elementos da estrutura hierárquica de classes especificada. Os demais sistemas, principalmente os de análise e animação gráfica, foram implementados de maneira bastante simples, deixando-os preparados para futuros aperfeiçoamentos.

As principais características que poderão ser incluídas em novos protótipos são:

14. *Overlays* não foram utilizados apesar de serem suportados pela linguagem.

- Implementação e validação de novas classes

  AGVs, esteiras, empilhadeiras e demais classes especificadas na seção 3.5.3 poderiam ser implementadas. A validação destas novas classes poderá ser feita com simulações da célula de manufatura atualmente em desenvolvimento no Laborátorio de Computação e Automação Industrial da FEE/UNICAMP.

- Suporte a escalonamento de tarefas nos processadores

  O protótipo desenvolvido suporta apenas uma tarefa por processador. Entretanto, a estrutura de camadas apresentada na seção 3.6.7 foi implementada. Neste protótipo, o controlador da célula, depois de receber a mensagem "Tarefa Concluída" do processador, envia-lhe a mesma tarefa de produção novamente. Novos protótipos poderiam elaborar no sistema de parametrização uma sintaxe que permitisse ao usuário definir um conjunto de processos de transformações a serem desempenhadas por cada processador e a sequência de atribuição destas tarefas. Esta definição representaria o sequenciamento de produção do diagrama ilustrado na Fig. 3.23.

- Aperfeiçoamento do sistema de análise de desempenho

  A análise de simulação efetuada pelo protótipo implementado envolve apenas a utilização dos processadores e transportadores. A especificação apresentada neste trabalho indica outras estatísticas que poderiam ser incluídas nesta análise e sugere o uso de gráficos e histogramas nos resultados.

- Suporte a grupos

  No protótipo implementado, cada vértice ou arco das rotas de produção (seção 3.5.4) suporta apenas um objeto. A especificação entretanto prevê a definição de grupos de buffers, transportadores ou processadores não só na definição das rotas de produção como também na análise da simulação.

- Parametrização individual dos objetos

  Além do sistema de parametrização de direção própria já implementado (seção 3.5.2), novos protótipos poderiam possibilitar a edição isolada de parâmetros de forma individual para cada objeto do modelo.

- Facilidades de manutenção da biblioteca de objetos

  A biblioteca de objetos utilizada pelo protótipo foi criada por um programa especialmente desenvolvido, dedicado a esta função. Um sistema para manutenção de bibliotecas, incluindo um editor de ícones, poderia ser incorporado ao SISMA.

• Definição gráfica das rotas de produção

Ao invés da sintaxe adotada pelo protótipo (pág. 48), a definição das rotas de produção poderia ser feita por roteamento gráfico na tela do computador com o auxílio do *mouse*.

• Utilização de cores e maior definição gráfica

O protótipo implementado utiliza o modo gráfico CGA de 640x200 pontos de resolução e apenas 2 cores. Placas controladoras de vídeo mais modernas como a EGA e VGA suportam pelo menos 16 cores simultâneas e resolução 640x350 pontos (640x480 no caso da VGA). Apesar do protótipo ser compatível com estes tipos de monitores, ele não utiliza todos os recursos disponíveis, permanecendo no modo CGA em qualquer um deles. Novos protótipos poderiam detectar o tipo de *hardware* gráfico do computador e utilizar a maior resolução gráfica disponível e todo o conjunto de cores.

• Visualização dos estados dos objetos

Novos protótipos poderiam utilizar uma representação gráfica para ilustrar os estados dos objetos do modelo durante a simulação. Desta maneira, o usuário poderia acompanhar, por exemplo, a quantidade de partes em um buffer, o estado de operação de uma máquina, o tipo de parte que um robô está manipulando, etc. As representações poderiam ser feitas por mudanças de ícone, pela alteração de sua cor ou por algum valor ou mensagem colocada próxima do objeto.

• Suporte a FDPs na determinação de parâmetros

No protótipos desenvolvido, a especificação dos parâmetros de tempo é feita com valores exatos. Utilizando as funções de distribuições de probabilidade (FDP) já implementadas (seção 3.6.8), novos protótipos poderiam possibilitar esta especificação através de uma valor médio e de uma modificação (A ± B), semelhante ao especificado para os parâmetros associados a um ponto de entrada (pág. 65).

• Redação detalhada do arquivo de ajuda

Para todos os contextos do programa implementado, existem chaves de ajuda definidas, faltando apenas redigir os textos de ajuda relativos a cada um destes contextos.

Esforços mais significativos em direção ao desenvolvimento de um sistema de modelagem e simulação profissional podem ser efetuados. Neste sentido, algumas sugestões para novos trabalhos são apresentadas a seguir.

- Transportar o SISMA para ambiente Windows utilizando o *Turbo Pascal for Windows* lançado recentemente pela Borland International [Lin91]. Neste ambiente, o desenvolvimento da interface gráfica passaria a ser bastante facilitado devido ao fato do Windows já incorporar os conceitos de ícones, janelas, menus e sistema de ajuda por hipertexto [Borl91b];

- Validar o sistema com aplicações industriais reais;

- Prover suporte a regras de produção dinâmicas e bases de conhecimento. Poderia ser desenvolvida para o SISMA uma máquina de inferência em Pascal ou uma interface com a linguagem Prolog ([Ari87] e [Ari88]);

- Aprimorar os efeitos de animação utilizando técnicas de computação gráfica;

5.4 Conclusões

Neste trabalho, o SISMA foi validado com uma célula de manufatura hipotética extraída da literatura [Lev89]. Foi demonstrado que a arquitetura proposta é adequada para simulação de sistemas de manufatura complexos. Desta forma, o protótipo implementado e a especificação apresentada servem como ponto de partida para o desenvolvimento de um sistema com grande potencial em aplicações acadêmicas e industriais.

REFERÊNCIAS BIBLIOGRÁFICAS

[Ari87] ARITY Corp. (1987), Using the Arity/PROLOG Interpreter and Compiler.

[Ari88] ARITY Corp. (1988), The Arity/PROLOG Language Reference Manual.

[Bak74] BAKER, K. R. (1974), Introduction to Sequencing and Scheduling, John Wiley, N.Y, USA.

[Bel87] BELL, P., O'KEEFE, R. (1987), "Visual Interactive Simulation - History, Recent Developments, and Major Issues", Simulation, vol. 49, n. 3, pp. 109-116.

[Borb90] BORBA, R. G. (1990), "Interfaces com Usuário Orientadas por Objetos - Guia de Implementação", Nota Técnica, CPqD-Telebrás, Campinas-SP.

[Borb92] BORBA, R. G. (1992), "Estrutura de Implementação e Descrição de Projeto do SISMA", Relatório Técnico RT-DCA 005/92, FEE, UNICAMP, Campinas-SP.

[Borl89a] BORLAND International (1989), Turbo Pascal 5.5 Object-Oriented Programming Guide, Borland International, Inc., Scotts Valley, CA, USA.

[Borl89b] BORLAND International (1989), Turbo Pascal User's Guide Version 5.0, Borland International, Inc., Scotts Valley, CA, USA.

[Borl90] BORLAND International (1990), Turbo Pascal Reference Guide Version 6.0, Borland International, Inc., Scotts Valley, CA, USA.

[Borl91a] BORLAND International (1991), Turbo Pascal For Windows Reference Guide, Borland International, Inc., Scotts Valley, CA, USA.

[Borl91b] BORLAND International (1991), Borland Languages Help Compiler, Borland International, Inc., Scotts Valley, CA, USA.

[Borl91c] BORLAND International (1991), Borland C++ Reference Guide Version 2.0, Borland International, Inc., Scotts Valley, CA, USA.

[Can89] CANNING, P. S., COOK, W. R., HILL, W. L. e OLTHOFF, W. G. (1989), "Interfaces for Strongly-Typed Object-Oriented Programming", SIGPLAN NOTICES OOPSLA'89 Proceedings, vol. 24, n. 10, pp. 457-467.

[Car90] CARDELLI, L. (1990), "A Semantics of Multiple Inheritance", Readings in Object-Oriented Database Systems.

[Che85] CHENG, T. (1985), "Simulation of Flexible Manufacturing Systems", Simulation, vol. 45, n. 6, pp. 299-302.

[Cho65] CHORAFAS, D. (1965), Systems and Simulation, New York: Academic Press, Inc., USA.

[Cox87] COX, S. (1987), "Interactive Graphics in GPSS/PC", Simulation, vol.49, n.3, pp. 117-122.

[Dah66] DAHL, O-J. e NYGAARD, K. (1966), "SIMULA - An ALGOL Based Simulation Language", Communications of the ACM, vol. 9, n.9, pp. 671-678.

[Die90] DIEβL, G., SCHULZ, G. e WINKLER, J. F. H. (1990), "Object-CHILL: The Road to Object Oriented Programming with CHILL", Proceedings of the 5th CHILL Conference, pp. 118-125.

[Eld90] ELDREDGE, D. L., McGREGOR, J. D. e SUMMERS, M. K. (1990), "Applying the object-oriented pradigm to discrete event simulations using the C++ language", Simulation, vol. 54, n. 2, pp. 83-91.

[Eng85] ENGELKE, H. et al. (1985), "Integrated Manufacturing Modeling System", IBM J. Res. Develop., vol. 29, n. 4, pp. 343-355.

[Fis73] FISHMAN, G. S. (1973), Concepts and Methods in Discrete Event Simulation, John Wiley, N.Y., USA.

[Fis78] FISHMAN, G. S. (1978), Principles of Discrete Event Simulation, John Wiley, N.Y., USA.

[Gol83] GOLDBERG, A. e ROBSON, D. (1983), SMALLTALK-80: The Language and Its Implementation, Addison-Wesley, USA.

[Gol84] GOLDBERG, A. e ROBSON, D. (1983), SMALLTALK-80 : The Interactive Programming Environment, Addison-Wesley, USA.

[Goo87] GOODNIGHT, H. T. (1987), "Designing a Simulation Interface for the Industrial Engineer", Simulation '87 Long Beach, CA, Oct.

[Gor75] GORDON, G. (1975), The application of GPSS V to Discrete System Simulation, Prentice Hall, Inc., Englewood Cliffs, N.J., USA.

[Gor78] GORDON, G. (1975), System Simulation, Prentice Hall, Inc., Englewood Cliffs, N.J., USA.

[Har75] HARTLEY, M. G., Digital Simulation Methods, Peter P. Ltd., England.

[Jen85] JENSEN, K. e WIRTH, N. (1985), Pascal: User manual and report: revised for the ISO Pascal standard, 3ª ed., Springer-Verlag, USA.

[Kus86] KUSIAK, A. (1986), "Efficient implementation of Johnson's scheduling algorithm", IIE Transactions, vol. 18, n. 2, pp. 215-216.

[Kus90] KUSIAK, A. (1990), Intelligent Manufacturing Systems, Prentice Hall, Inc., Englewood Cliffs, N.J., USA.

[Law85] LAWLER, E. L., LENSTRA, J. K., RINNOOY KAN, A. H. G. e SHMOYS, D. B., The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization, John Wiley, N.Y., 1985.

[Lev89] LEVAS, A. e JAYARAMAN, R. (1989), "WADE: An Object-Oriented Environment for Modeling and Simulation of Workcell Applications", IEEE Transactions on Robotics and Automation, vol. 5, n. 3, pp. 324-336.

[Lin91] LINDERHOLM, O. (1991), "Turbo Pascal Makes Windows Programming a Breeze", BYTE, vol. 16, n. 6, pp. 58-60.

[Lue89] LUEHRMANN, M. e BYRKETT, D. (1989), "A Pilot Study of the Impact of Animation on Decision-Making", Simulation, vol. 53, n. 4, pp. 153-158.

[Mag87] MAGNETAT-THALMANN, N. e THALMANN, D. (1987), "Procedural Animation Blocks in Discrete Simulation", Simulation, vol. 49, n. 3, pp. 102-108.

[Mar69] MARTENS, H. R. (1969), "A Comparative Study of Digital Integration Methods", Simulation, vol. 12, n. 2, pp. 87-96.

[Oga70] OGATA, K. (1970), Modern Control Engineering, Prentice Hall, Inc., Englewood Cliffs, N.J., USA.

[Peg85] PEGDEN, C. D. (1985), Introduction to SIMAN, Systems Modeling Corporation, USA.

[Pro86] PROSOFT Inc. (1986), Fontasy User's Manual.

[Pru90] PRUETT, J. M. e VASUDEV, V. K. (1990), "MOSES: Manufacturing Organization Simulation and Evaluation System", Simulation, vol. 54, n. 1, pp. 37-45.

[Red86] REDDY, R. et al. (1986), "The Knowledge-Based Simulation System", IEEE Software, vol. 3, n. 2, pp. 26-37.

[Red87] REDDY, R. (1987), "Epistemology of Knowledge Based Simulation", Simulation, vol. 48, n. 4, pp. 162-166.

[Sch89] SCHROER, B. J. (1989), "A Simulation Assistant for Modeling Manufacturing Systems", Simulation, vol. 53, Nr. 5, pp. 201-206.

[Sco90] SCORTESSE, A. (1990), "OO-CHILL: Integrating the object paradigma into CHILL", Proceedings of the 5th CHILL Conference, pp. 111-117.

[Sha85] SHANNON, R. E., MAYER, R. e ADELSBERGER, H. H. (1987), "Expert Systems and Simulation", Simulation, vol. 44, n. 6, pp. 275-284.

[Sha88] SHANNON, R. E. (1988), "Knowledge Based Simulation Technics for Manufacturing", Int. J. Prod. Res., vol. 26, n. 5.

[She90] SHEARN, D. C. S. (1990), "PASSIM - A Pascal discrete event simulation program generator", Simulation, vol. 55, n. 1, pp. 31-38.

[SIM86] SIMULATION (1986), "Catalog of Simulation Software", Simulation, vol. 47, n. 4, pp. 152-165.

[SIM87] SIMULATION (1987), "Addendum to the Simulation Software Catalog", Simulation, vol. 48, n. 2, pp. 69-73.

[Smi87] SMITH, R. e PLATT, L. (1987), "Benefits of Animation in the Simulation of a Machining and Assembly Line", Simulation, vol. 48, n. 1, pp. 28-30.

[Sta77] STALLMAN, R. M. e SUSSMAN, G. J. (1977), "Forward Reasoning and Dependency-Directed Backtracking in a System for Computer-Aided Circuit Analysis", Artificial Intelligence, vol. 9, pp. 135-196.

[Ste82] STEFIC, M. et al. (1982), "The Organization of Expert Systems: A Prescritive Tutorial", Artificial Intelligence, vol. 18, n. 2, pp. 135-173.

[Tan87] TANENBAUM, A. S. (1987), Operating Systems: design and implementation, Englewood Cliffs, Prentice-Hall, USA.

[Tav89] TAVERNINI, L. (1989), "A User-Friendly Interactive Turbo Pascal Simulation Toolkit", Simulation, vol. 53, n. 2, pp. 45-55.

[Zdo90] ZDONIK, S. B. e MAIER, D. (1990), Fundamentals of Object-Oriented Databases, Readings in Object- Oriented Database Systems, Morgan K. Publishers, Inc., USA.

[Zim87] ZIMMERS, E. W. e VALENZUELA, C. A. (1987), "Building a Flexible Manufacturing Systems (FMS) Simulation in a PC Environment", Simulation '87 Long Beach, CA, Oct.