LEONARDO CHWIF

REDUÇÃO DE MODELOS DE SIMULAÇÃO DE EVENTOS DISCRETOS NA SUA CONCEPÇÃO: UMA ABORDAGEM CAUSAL

Tese apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do título de Doutor em Engenharia.

São Paulo

1999

LEONARDO CHWIF

REDUÇÃO DE MODELOS DE SIMULAÇÃO DE EVENTOS DISCRETOS NA SUA CONCEPÇÃO: UMA ABORDAGEM CAUSAL

Tese apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do título de Doutor em Engenharia.

Área de concentração:
Engenharia Mecânica (Mecatrônica)

Orientador:
Prof. Dr. Marcos Ribeiro Pereira Barretto.

São Paulo

1999

E eu vos digo que a vida é realmente escuridão, exceto quando há impulso.
E todo impulso é cego, exceto quando há saber.
E todo saber é vão, exceto quando há trabalho.
E todo o trabalho é vazio, exceto quando há amor.

Gibran Khalil Gibran

AGRADECIMENTOS

Ao meu orientador, Prof. Dr. Marcos Ribeiro Pereira Barretto, pelas diretrizes e apoio durante a realização deste trabalho.

Ao Prof. Dr. Ray J. Paul, pela sua excelente supervisão durante o período de estágio que realizei no exterior.

Ao Prof. John Salt, Tillal Eldabi e Hamad Odhabi, pelas discussões acerca do tema em questão.

À todos os membros do CASM (Centre for Applied Simulation Modelling) da Universidade de Brunel (Reino Unido), que me auxiliaram durante o peroído em que estive no exterior.

Aos meus pais, Sônia e José pelo incentivo e apoio. À minha esposa, Renata, pela paciência e compreensão durante a realização deste trabalho.

À CAPES, pela concessão da bolsa que suportou minha pesquisa.

A todos que diretamente ou indiretamente, contribuíram para a realização deste trabalho.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

AC ..... "Action Clusters"
ACAG ..... "Action Cluster Attribute Graph"
ACIG ..... "Action Cluster Incidence Graph"
AGV ..... "Automated Guided Vehicle"
APN ..... "Augmented Petri Nets"
CAP ..... "Condition Action Pairs"
CASM ..... "Computer Aided Simulation Modelling"
CFG ..... "Control Flow Graph"
CS ..... "Condition Specification"
DEVS ..... "Discrete Event System Specification"
EG ..... "Event Graphs"
FCFS ..... "First Come First Served"
FIFO ..... "First In First Out"
GIGO ..... "Garbage In Garbage Out"
GoM ..... "Graph of Models"
GSPN ..... "Generalized Stochastic Petri Nets"
HCFG ..... "Hierarchical Control Flow Graphs"
HIMASS ..... "Hierarchical Modeling and Simulation System"
HIRES ..... "HIerarchical REasoning Systems"
LIFO ..... "Last In Last Out"
LS ..... "Linguagens de Simulação"
MC ..... "Medida de Complexidade"
MD ..... "Medidas de Desempenho"
MIP ..... "Machine Inference Problem"
PN ..... "Process Networks"
QPT ..... "Qualitative Process Theory"
QR ..... "Qualitative Reasoning"
SC ..... "State Charts"
SMDE ..... "Simulation Model Development Environment"
SPN ..... "Stochastic Petri Nets"
TPN ..... "Temporized Petri Nets"
VDM ..... "Vienna Development Method"
WSCCS ..... "Weighted Synchronous Calculus of Communicating Systems"
X-ACD ..... "eXtended Activity Cycle Diagrams"

RESUMO

A simulação de Sistemas de Eventos Discretos é uma ferramenta bem conhecida e utilizada, por sua habilidade inerente de avaliar sistemas complexos e considerar seu comportamento dinâmico. Por outro lado, possui um inconveniente: o tempo para se completar um estudo de simulação é considerado longo, mesmo com o desenvolvimento atual de *hardware* e *software* de simulação. Assim, ações são tomadas em alguns casos de processos decisórios que dependem de simulações, antes do estudo apropriado de simulação ter sido completado. Neste cenário onde há uma demanda para se diminuir o tempo de resposta de um estudo de simulação, métodos de simplificação de modelos de simulação possuem um papel fundamental.

O objetivo deste trabalho é propor uma técnica para reduzir a complexidade de um modelo de simulação de eventos discretos, na fase de concepção de um estudo de simulação, implicando-se que os resultados computacionais do modelo não são conhecidos a priori para o processo de redução. Para permitir o ganho de tempo proposto, esta técnica deve ser passível de implementação em um computador. A escolha de uma técnica apropriada de representação de modelos de simulação, que suporte o processo de redução, também faz parte do escopo deste trabalho.

Os resultados obtidos mostraram a viabilidade da técnica de redução proposta, podendo auxiliar especialmente analistas não experientes na construção de modelos mais simples.

Apesar da importância deste assunto, a falta de pesquisa nesta área é uma constante. Este trabalho pode servir como incentivo a futuros pesquisadores que desejam adentrar neste tema.

ABSTRACT

Discrete Event Simulation is a very well known and utilised tool, because of its inherent ability to evaluate complex systems and to consider their dynamic behaviour. On the other hand it has a shortcoming: the time to complete a simulation study is considered to be long, even with current developments in hardware and simulation software. There are some cases in a decision making process which depends on simulation that actions have to be made without time to complete a proper simulation study. In this scenario, where there is a demand for decreasing simulation study response time, simplification methods for simulation models could play a key role.

The aim of this work is to propose a technique for reducing the complexity of a discrete event simulation model at the conceptual phase of simulation modelling i.e when the results from the simulation runs are not known a priori for the reduction process. In order to allow the proposed gain of time, this technique should be automatically performed by a computer. The choice of a proper simulation model representation technique to support the reduction process is also within the scope of this work.

The achieved results showed the feasibility of the proposed technique allowing to help specially non expert modellers in the construction of simpler models.

Despite of the importance of this subject, the lack of research in this area is surprisingly high. This work can serve as incentive for future researchers that would like to endeavor in this area.

CAPÍTULO 1
INTRODUÇÃO

1.1 INTRODUÇÃO

Simulação é um termo extremamente amplo e, basicamente, pode ser definido como o processo de elaboração de um modelo de um sistema real (ou hipotético) e a condução de experimentos com a finalidade de entender o comportamento de um sistema ou avaliar sua operação [SHANNON, 75]. Na grande maioria das vezes, o modelo é implementado em um computador. Portanto, usualmente, o termo "Simulação" é sinônimo de "Simulação Computacional" pois, embora esta possa ser efetuada manualmente, diante do número maciço de cálculos, o tempo gasto para tal seria inviável, em termos práticos.

Embora a simulação computacional esteja datada da década de 50, quando tinha basicamente fins militares, sua popularidade intensificou-se nesta última década, expandindo-se para outras áreas como manufatura e serviços entre engenheiros, administradores e mesmos leigos. Esta popularidade pode ser explicada por alguns fatores:

1. Desenvolvimento dos computadores, que apresentou crescimento extraordinário nos últimos anos;
2. Desenvolvimento de *softwares* de simulação, com interfaces homem-máquina mais "amigáveis" e com capacidade de processamento gráfico;
3. Natureza da simulação: capacidade de avaliar sistemas complexos e modelar seu comportamento dinâmico, sendo especialmente importante quando inexiste solução analítica (através de modelos matemáticos).

Ainda, segundo PIDD [98], pode-se enumerar algumas vantagens da simulação computacional, quando comparada como a experimentação direta de um sistema real. A primeira é em relação aos custos envolvidos: é muito menos oneroso efetuar experiências com o modelo de simulação do que com o sistema real, especialmente quando equipamentos de alto custo estão envolvidos. Também a experimentação de um sistema real envolve riscos, tanto materiais quanto humanos, o que não ocorre com um sistema simulado.

Apesar de suas vantagens, um problema prático enfrentado por um estudo de simulação é o tempo despendido. Este tempo é usualmente longo, quando comparado com outras técnicas (analíticas), ou mesmo quando a solução parte da experiência e bom senso do analista. Por este motivo, acredita-se que o estudo de técnicas de redução / simplificação de modelos de simulação seja de grande importância para contribuir com a diminuição deste tempo.

1.2 CONCEITOS E TERMOS UTILIZADOS.

Antes de prosseguir, torna-se necessário estabelecer alguns termos, com a finalidade de unificar os conceitos e permitir o melhor entendimento deste trabalho, em razão da ampla abrangência da simulação computacional. Os principais termos estão expostos a seguir.

O termo **sistema,** de acordo com GORDON [78], é utilizado numa grande variedade de modos, sendo difícil produzir uma definição suficientemente abrangente para cobrir vários usos e, ao mesmo tempo, concisa o suficiente para ter um propósito útil. Uma definição simples: um **sistema** é um agrupamento de partes que operam juntas, visando um objetivo em comum [FORRESTER, 68]. Em outras palavras, um **sistema** é uma combinação de componentes que atuam conjuntamente para a realização de uma

função que não é possível realizar com qualquer componente individual [CASSANDRAS, 93]. O **estado de um sistema** pode ser definido através de uma coleção de variáveis necessárias para descrevê-lo totalmente em um dado instante [LAW e KELTON, 91]. Esta coleção é denominada de **variáveis de estado**.

Do enfoque da teoria de sistemas, um **modelo** pode ser definido como uma representação das relações dos componentes de um sistema, sendo considerada como uma abstração, no sentido em que tende a se aproximar do verdadeiro comportamento do sistema.

Aqui cabe a discussão do que é um "bom" ou "mau" modelo. Um "bom" modelo, de uma maneira geral, é aquele que permite atingir os **objetivos da simulação** (conceito definido a seguir) com o mínimo custo. Segundo [HARREL e TUMAY, 94], o "bom" modelo deve possuir certas características, das quais duas estão dispostas abaixo:

1. É válido - no sentido de representar satisfatoriamente a realidade;
2. É mínimo - no sentido de incluir somente elementos que influenciam no problema a ser solucionado.

Os modelos para representar os sistemas podem ser classificados em: modelos simbólicos (ou icônicos), analíticos e modelos de simulação.

Os **modelos simbólicos** (ou icônicos) são constituídos de símbolos gráficos (como retângulos e retas) utilizados para dar noção de seqüência ou outras relações entre entidades. O fluxograma de processos de um sistema de manufatura ou o fluxograma de informações de um negócio são exemplos de modelos simbólicos. Embora estes modelos sejam muito úteis para entendimento e documentação do sistema, são fracos para análise quantitativa, pois só representam os sistemas de uma forma descritiva.

Os **modelos analíticos** basicamente podem ser reduzidos a um conjunto de equações que, ao serem resolvidas, permitem obter a solução esperada. São exemplos de

modelos analíticos: os modelos de programação dinâmica, as cadeias de Markov e os modelos de redes de fila [HARREL e TUMAY, 94]. Embora estes modelos forneçam soluções precisas, se o sistema a ser modelado for extremamente complexo, as soluções podem se tornar complicadas e, em muitos casos, têm de ser utilizadas hipóteses simplificadoras para que se resolva o modelo analiticamente. Isto pode levar à perda da validade do modelo, visto que o mesmo não conseguiu representar, satisfatoriamente, a realidade.

Os **modelos de simulação**, por sua vez, são implementados através do auxílio de um computador. Assim, como um modelo analítico pode ser representado por um conjunto de equações, um modelo de simulação pode ser representado através de uma linguagem de programação. Para tal, ao contrário dos modelos analíticos, os modelos de simulação são executados, ao invés de solucionados. As desvantagens dos modelos de simulação são: podem ser difíceis de se construir e podem levar a resultados menos precisos do que os modelos analíticos. Em contrapartida, são modelos excelentes para representar sistemas que possuem um número muito grande de variáveis e com dinâmica muito complexa (caso em que a aplicação de modelos analíticos possui restrições).

Um **modelo de simulação** é construído a partir dos **objetivos da simulação** (também denominados de **requisitos de simulação**). Estes podem ser traduzidos **pelas medidas de desempenho** do sistema, que são as variáveis de saída de interesse de um modelo de simulação. Deste modo, considerando um exemplo de um modelo de simulação de uma linha de montagem, este pode ter como objetivo a produtividade e a utilização da mão-de-obra. Portanto, as medidas de desempenho são a produção horária que sai da linha e as taxas de utilização de cada recurso humano na mesma.

A simulação computacional pode ser classificada basicamente em três categorias: **simulação monte carlo**, **simulação contínua** e **simulação de eventos discretos**, de

acordo com a taxonomia proposta por NANCE [93]. A **simulação monte carlo** é aquela onde um problema, notadamente não probabilístico, é solucionado através de um processo estocástico. Nesta, inexiste uma representação explícita do tempo. Já tanto a **simulação contínua** como a **simulação de eventos discretos** levam em conta o tempo. A primeira é utilizada para modelar sistemas cujo estado varia continuamente no tempo, como no exemplo de uma xícara de chá aquecendo. A **simulação contínua** se utiliza de equações diferenciais para o cálculo das mudanças das variáveis de estado ao longo do tempo. Por outro lado, **a simulação de eventos discretos** é utilizada para modelar sistemas que mudam o seu estado em pontos discretos no tempo, a partir da ocorrência de eventos, como é o caso de sistemas de manufatura. Para detalhes sobre sistemas de eventos discretos, ver HO [89]. Estes dois tipos de sistemas estão ilustrados pela figura 1.1. Em alguns casos, é necessário construir um modelo de simulação que compreenda aspectos de eventos contínuos e discretos. Neste caso, a simulação é denominada **simulação combinada** [PRITSKER, 86].

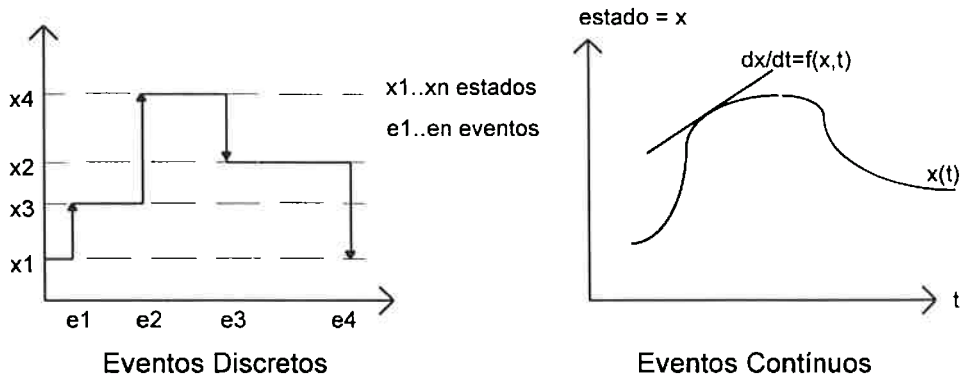


fig. 1.1 Eventos discretos x contínuos

Uma outra classificação pode ser feita em relação às variáveis com que os modelos de simulação trabalham. Os modelos de simulação que utilizam variáveis aleatórias são denominados **Modelos Estocásticos**, enquanto que os **Modelos Determinísticos** trabalham somente com variáveis do tipo não probabilísticas, e o

resultado da simulação é sempre o mesmo, não importando quantas vezes se "rode" o modelo. A grande maioria dos modelos de simulação são constituídos por variáveis estocásticas.

A partir deste ponto, quando não se mencionar o contrário, utiliza-se a palavra "Sistemas" como sinônimo do termo "Sistemas de Eventos Discretos" e o termo "Simulação", por sua vez, como "Simulação de Eventos Discretos".

Ao analisar um modelo de simulação, pode-se distinguir, basicamente, três elementos. Uma **entidade** é qualquer objeto de interesse do modelo. Um **atributo** é uma propriedade desta entidade. Qualquer processo que pode causar uma mudança no modelo é denominado de **atividade**. Assim em um modelo de simulação de manufatura, uma **entidade** pode ser as peças que entram no sistema, sua prioridade de produção pode ser um dos seus **atributos** e o seu processamento por uma máquina, uma **atividade**.

O modelo de simulação é implementado no computador mediante uma **estratégia da simulação** (os termos em inglês equivalentes são *World View* e *Conceptual Framework* [DERRICK e BALCI, 97]). A **estratégia da simulação**, de acordo com BALCI [88], é uma estrutura de conceitos e pontos de vista, onde a simulação é guiada para o desenvolvimento do modelo de simulação. Segundo PIDD [98], há,basicamente, quatro estratégias básicas da simulação: a de **evento**, a de **atividade**, a de **processo** e o **método das três fases.**

- Na **estratégia de evento**, o analista especifica quando ocorrem as ações no modelo. Esta estratégia é, portanto, baseada na programação das ações em certos instantes no tempo (localidade de tempo).
- Na **estratégia de atividade,** o analista especifica as causas para as ações ocorrerem no modelo, sendo estas baseadas nas pré-condições para uma determinada atividade ocorrer (localidade de estado).

- Na **estratégia de processo,** o analista especifica as entidades e descreve todas as sequências de ações de cada entidade, individualmente (localidade de objeto).
- O **método das três fases** é uma abordagem que mescla a **estratégia de evento** com a de **atividade**. Fases "A", "B" e "C" são introduzidas. A fase "B" checa pelo fim de atividades e a fase "C", as condições para as atividades ocorrerem. A fase "A", por sua vez, atualiza o tempo, avançando-o para o próximo evento.

Estas estratégias possuem vantagens e desvantagens, que não cabe aqui discutir. O que ocorre, na prática, é que a **estratégia de evento** é mais popular nos EUA, enquanto que a de **atividade** e o **método das três fases** são mais conhecidos no Reino Unido.

Para maiores detalhes sobre as **estratégias da simulação**, ver DERRICK, BALCI e NANCE [89].

Uma discussão mais detalhada sobre os diversos termos discutidos aqui pode ser encontrada em GORDON [78].

1.3 PROCESSO DE SIMULAÇÃO

Tendo-se definido que, para a análise de um dado sistema é o modelo de simulação que melhor se aplica, deve-se seguir certos passos, a fim de que o estudo de simulação seja bem sucedido. Estes passos ou processo são conhecidos na literatura como "metodologia de simulação" ou "ciclo de vida de um modelo de simulação" (LAW e MCCOMAS, [90], ULGEN et al, [94] e NORDGREN [94]).

De acordo com NANCE [83], ocorreu uma mudança de paradigma na comunidade de simulação de eventos discretos, a partir da década de 70, onde, até este período, a simulação era vista somente como a construção de um programa. Sabe-se que a simulação envolve muito mais que a simples construção de um programa, sendo esta atividade apenas uma, dentre as inúmeras atividades de um estudo de simulação.

1.3.1 CICLO DE VIDA DE UM MODELO DE SIMULAÇÃO

Basicamente, o desenvolvimento de um modelo de simulação compõe-se de três grandes etapas [PAUL e BALMER, 93]:

1. Concepção ou formulação do modelo;
2. Implementação do modelo;
3. Análise dos resultados do modelo.

Na primeira etapa ou fase, o analista de simulação deve entender claramente o sistema a ser simulado e os seus objetivos, através da discussão do problema com especialistas. Deve-se decidir qual é a abrangência do modelo e o nível de detalhe. Todas as hipóteses devem ser claramente estabelecidas. Após estas decisões, o modelo que está na mente do analista (**modelo abstrato**) deve ser representado de acordo com alguma técnica de representação de modelo de simulação (as técnicas mais utilizadas de representação de modelos de simulação são revistas no capítulo 2), a fim de torná-lo um **modelo conceitual**, de modo que outras pessoas possam entendê-lo.

Os dados de entrada também devem ser coletados nesta fase. De acordo com SHANNON [75], há uma constante interação entre a construção do modelo conceitual e a coleta dos dados. Não se pode negar a importância de se ter dados adequados para alimentar o modelo, sendo que a expressão GIGO (*Garbage in - Garbage out*) também é aplicável aos modelos de simulação [PEGDEN, SHANNON e SADOWSKI, 95]. No entanto, é importante ressaltar que o modelo é que deve dirigir a coleta de dados e não vice-versa [PIDD, 96]. Uma discussão mais aprofundada sobre a importância dos dados de entrada, pode ser encontrada em KELTON, SADOWISKI e SADOWSKI [98].

Na segunda etapa, o **modelo conceitual** é convertido em um **modelo computacional** através da implementação no computador, com a utilização de uma linguagem de simulação ou de um simulador comercial. Pode-se, ainda, codificar o

modelo de simulação em uma linguagem de programação geral, como o C ou Pascal, mas isto é altamente desaconselhável, dado o tempo que é despendido na programação de diversas rotinas já existentes nos simuladores comerciais. Isto deve ser feito somente em última instância caso não se consiga construir o modelo em uma linguagem de simulação ou num simulador. O **modelo computacional** deve, ainda, ser verificado contra **o modelo conceitual**, a fim de avaliar se está operando de acordo com o pretendido. Alguns resultados devem ser gerados para validar o modelo computacional, observando-se se o modelo é uma representação precisa da realidade (dentro dos objetivos da simulação). O assunto verificação e validação de modelos de simulação é bastante extenso e não cabe aqui expô-lo em detalhes. Consultar SARGENT [96] e BALCI [97], para um aprofundamento neste assunto.

Na terceira etapa, após a verificação e validação do **modelo computacional**, este está pronto para a realização dos experimentos, dando origem ao **modelo experimental** ou, de acordo com PAUL e BALMER [93], **modelo operacional.** São efetuadas várias "rodadas" do modelo e os resultados da simulação são analisados, utilizando-se diversas técnicas estatísticas e os resultados documentados. A partir destes resultados, conclusões e recomendações sobre o sistema podem ser geradas. Caso necessário (se o resultado da simulação não for satisfatório), o modelo pode ser modificado, e este ciclo é reiniciado. Estes passos estão mostrados na figura 1.2.

Apesar destes passos estarem dispostos em uma certa seqüência linear, isto não ocorre exatamente desta maneira em um estudo prático de simulação, e, segundo PAUL [92], podem haver várias iterações e realimentações no processo, a medida que o entendimento do problema muda.

Em um típico estudo de simulação, estima-se que 30-40% deste ciclo seja atribuído à implementação [LAW e MCCOMAS, 90].

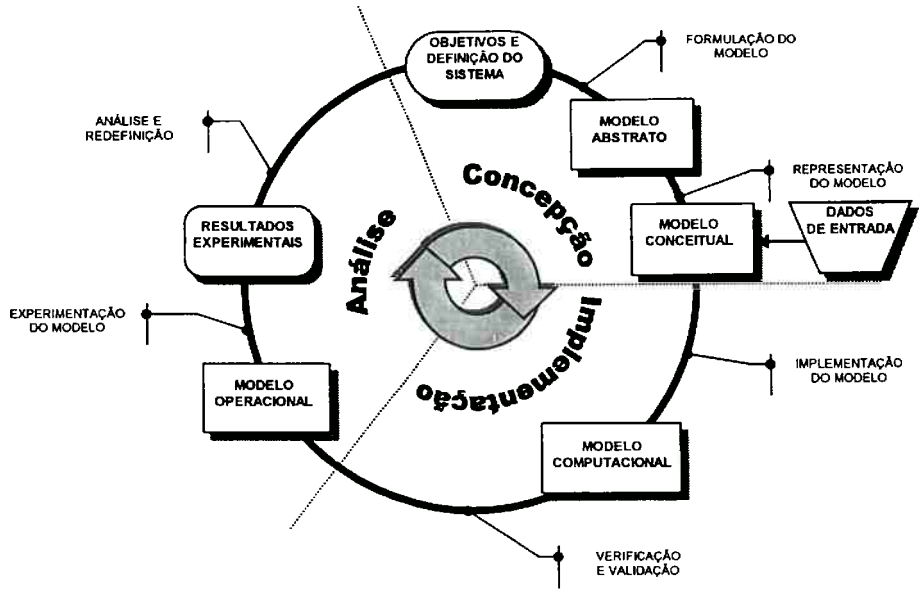


fig. 1.2 Ciclo de vida de um modelo de simulação

1.3.2 AMBIENTES DE SIMULAÇÃO

Segundo BALCI et al. [90], um Ambiente de Simulação é um "conjunto integrado de ferramentas de *hardware* e *software* que fornecem um suporte automatizado, a um custo eficiente, por todo o ciclo de vida de um modelo de simulação". Neste caso, o simulador em si seria apenas um de seus componentes de *software*.

BALCI e NANCE [92] propuseram uma arquitetura denominada SMDE (*Simulation Model Development Environment*). Esta arquitetura é composta, basicamente, de 4 camadas: *Hardware* e Sistema Operacional (0), Núcleo SMDE (1), SMDE mínimo (2) e SMDE (3). O SMDE mínimo contém ferramentas específicas para o desenvolvimento do modelo de simulação, como o administrador de projetos, administrador de pré-modelos, interpretador de linguagem, gerador de modelo, analisador de modelos e verificador de modelos, além de ferramentas de uso geral, como administrador de código fonte, sistema de *e-mail* e editor de texto. Esta arquitetura está ilustrada na figura 1.3.

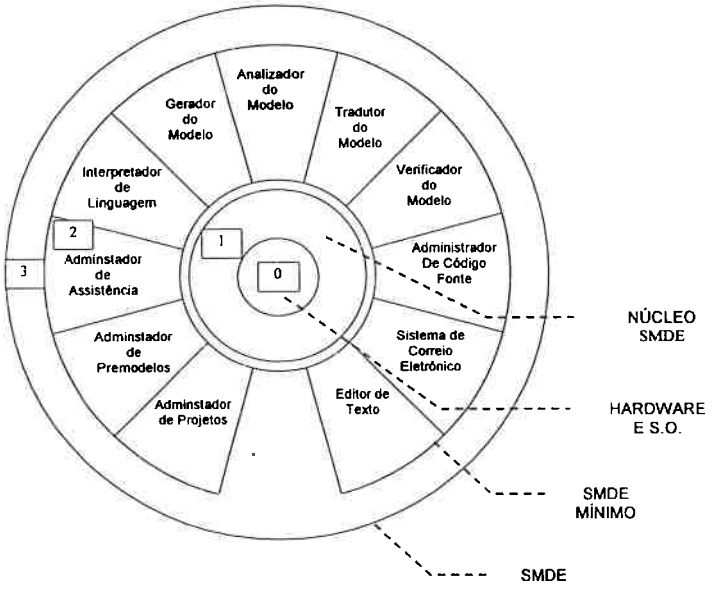


fig. 1.3 Arquitetura SMDE (BALCI e NANCE [92])

PAUL [92] também propôs um ambiente denominado CASM (*Computer Aided Simulation Modelling*), que consiste, basicamente, em uma série de geradores de programas de simulação interativos, com o objetivo principal de automatizar a formulação do problema (gerar o modelo conceitual a partir do modelo abstrato) e análise estatística dos dados de saída, utilizando-se diálogos em linguagem natural, sistemas especialistas e sistemas gráficos. Como outros ambientes de simulação, podemos citar o JADE [UNGER et. al. 86], KBSim [ROTHENBERG, 91] e ROSS [ROTHEMBERG, 88].

Embora a finalidade dos ambientes de simulação seja a de melhorar a qualidade dos modelos, aumentar a eficiência de um time de projeto de simulação e reduzir o tempo de desenvolvimento do modelo [BALCI e NANCE, 92], estes ainda carecem de produtos comerciais, tendo-se somente desenvolvidos sistemas protótipo em Universidades.

1.4 MODELAGEM VISANDO A SIMPLICIDADE DE MODELOS

Como foi comentado, o tempo do ciclo de um estudo de simulação é considerado grande. Uma das maneira de reduzí-lo é a criação de modelos mais simples, através da

utilização de uma possível técnica de redução de modelo. Os modelos simples possuem inúmeras vantagens, como será visto a seguir. No entanto, devido a vários fatores, modelos complexos são criados.

Na sequência, são discutidas as causas básicas deste fenômeno; a relação de um modelo simples com sua validade, com seu desempenho computacional e o potencial de redução do tempo de ciclo de vida, bem assim, a relação entre modelos simples e a experiência do analista.

1.4.1 VANTAGENS E DESVANTAGENS DOS MODELOS SIMPLES

Há um consenso na comunidade de simulação de que um modelo simples é sempre mais preferível a um modelo complexo. De fato, "modele simples – pense complicado" é um dos 5 princípios de modelagem em simulação, proposto por PIDD [96]. WARD [89] também vai nesta direção, dizendo que o modelo, tanto para o analista (criador do modelo), como para o usuário, deve ser simples, pois um modelo simples é mais fácil de compreender e é mais eficiente de se mudar, implementar e analisar. SALT [93] também comenta sobre este assunto e chega a afirmar que "a simplificação é a essência da simulação". Ainda aponta uma outra vantagem de um modelo simples: a facilidade de jogá-lo fora e começar novamente. Uma vez que um modelo complicado é criado, é difícil se livrar dele (é muito mais difícil de admitir uma falha em um modelo extremamente custoso do que em um modelo simples e barato).

ROBINSON [94] reporta que, em um caso real, um engenheiro automotivo economizou USD 150.000 em uma tarde de sexta feira. Um pequeno modelo de simulação foi desenvolvido e foram feitos experimentos que comprovaram a compra desnecessária de certos tipos de equipamentos. Como outro exemplo de um projeto real, a fim de provar quantos veículos eram necessários para mover os materiais em uma

indústria de manufatura, um modelo extremamente grande e complexo foi construído. Depois de nove meses, um monstro totalmente validado foi criado e foram realizados os experimentos. Foi possível encontrar uma economia de dois a três veículos no sistema, mas esta economia gerada era desprezível quando comparada ao custo total do projeto de simulação.

Apesar das inúmeras vantagens, um modelo simples pode ter alguns problemas. Tome o caso de um modelo de simulação de um sistema de AGV (*Automated Guided Vehicle*), que foi simplificado para lidar com somente 2 AGV's, ao invés de 5. Ao limitar o escopo de um modelo (o que, de acordo com a terminologia de ZEIGLER [76], significa limitar o *Experimental Frame* deste), apesar de ter-se resultados ainda válidos, o modelo reduzido perde a sua flexibilidade, pois só pode ser usado no caso específico.

Além disso, espera-se que um modelo complexo seja mais preciso do que um simples e podem haver casos em que um modelo simples demore mais tempo para ser construído [BROOKS e TOBIAS, 96]. As seções 1.4.4 e 1.4.5 fazem uma discussão mais detalhada respectivamente, sobre a relação modelo simples e validade do modelo e modelo simples e desempenho computacional.

1.4.2 RAZÕES PARA A EXISTÊNCIA DE MODELOS COMPLEXOS

Apesar da maioria concordar nas inúmeras vantagens de um modelo simples, muitos modelos complexos são criados. Pode-se tentar enumerar algumas causas, de ordem técnica e não técnica (fatores humanos), para que isto ocorra.

No tocante às causas não técnicas, há uma tendência de que o "complicado" e o "mais difícil" é mais valorizado. Alguns analistas tendem a criar modelos de simulação complicados e mais custosos para receber a aprovação da sua gerência, pois um modelo simples demais poderia colocar em risco seu posto, já que há, neste caso, uma conotação

de que "qualquer um poderia fazê-lo". Outra tendência é construir um modelo mais complicado, pois é possível em termos de capacidade computacional. Esta, aliada às facilidades dos *softwares* de simulação, vem crescendo em um ritmo muito veloz. A tentação de se criar um modelo mais complexo, pois há recursos disponíveis para tal, é muito grande.

Em relação a alguns fatores técnicos, pode-se citar [YIN e ZHOU, 89]:

1. Falta de entendimento do sistema real;
2. Deficiência na habilidade de modelar corretamente o problema;
3. Prática de programação pobre.

No entanto, é uma unanimidade, entre vários autores [INNIS e REXSTAD, 83; YIN e ZHOU, 89; SALT, 93], que a maior causa do crescimento de modelos complexos é a falta de **objetivos de simulação** claramente bem definidos. Segundo HENIZE [84], o objetivo de um modelo é de suma importância. Um modelo elaborado e preciso não tem valor algum se não é capaz de responder questões relevantes. Um modelo simples e até não tão preciso pode ser mais valioso se fornecer um melhor entendimento sobre o sistema a ser simulado. Para comprovar que, em muitas vezes, os objetivos da simulação são deixados em segundo plano, foi realizado um teste envolvendo algumas pessoas (12 ao todo) com pouca experiência em modelagem de simulação (menos de um ano), média experiência (1 a 5 anos) e grande experiência (mais de 5 anos). O problema fornecido pedia para estas pessoas construírem um modelo de simulação comunicativo, a partir de uma descrição do sistema. O sistema era, simplesmente, composto por três máquinas dispostas em série e o objetivo da simulação era de verificar qual a taxa de utilização da segunda máquina. Todas as pessoas envolvidas no exercício, sem exceção, descreveram o modelo contendo as três máquinas. Ora, se o objetivo da simulação era determinar a taxa de utilização da segunda máquina, como o processamento da terceira máquina não influi

no processamento das máquinas anteriores, o modelo para atender tal objetivo poderia conter somente as duas primeira máquinas. Embora este exemplo seja muito simples e o número de pessoas envolvidas não foi suficiente para uma conclusão mais precisa, em termos estatísticos, este teste demonstra claramente a tendência das pessoas em dar maior importância ao sistema a ser modelado do que aos seus objetivos. De fato, segundo ROBINSON [94], nos projetos reais de simulação há uma tendência de modelar "tudo", sem parar para considerar o que exatamente é necessário. O enunciado deste exercício se encontra disponível no Apêndice A.

1.4.3 MODELOS SIMPLES, ANALISTAS EXPERIENTES E INEXPERIENTES

Pode-se dividir os analistas de simulação em duas categorias básicas: a dos que se concentram mais na fase de concepção ou formulação do modelo e os que praticamente ignoram esta fase, construindo diretamente o modelo computacional. Este fato é também confirmado por CERIC e PAUL [89]. Enquanto o primeiro está mais preocupado em encontrar um modelo de simulação mais adequado ao problema, sendo os resultados uma consequência deste modelo, o objetivo maior do segundo tipo é ver os resultados. De fato, tem-se a tendência de não se concentrar na fase de concepção do modelo quando há uma intensa pressão para o atendimento de prazos (como o prazo de entrega de um relatório de simulação).

Se estes dois tipos identificados existem, a pergunta a se fazer é: em qual destes tipos se enquadram os Analistas Experientes? Uma possível resposta a esta pergunta vem dos trabalhos de WILLEMAIN [WILLEMAIN, 94; WILLEMAIN, 95]. Este fez uma experiência com 12 analistas experientes, requisitando que falassem o que estavam pensando, enquanto modelavam um problema. Ao analisar o padrão do pensamento falado (classificado em 6 categorias), ele achou que 60% do tempo dos analistas foram

gastos na estruturação do modelo, enquanto que apenas 10% estava relacionado a aspectos de implementação. A partir desta análise, embora não seja completamente realística (pois o tempo dado era pré-determinado e o processo de pensar falando não é o mesmo de pensar quieto), pode-se concluir que os analistas experientes se concentram mais na fase de concepção do modelo.

Em se concentrando na fase de concepção, a probabilidade de obtenção de um modelo mais simples por parte de um analista experiente é muito maior do que a de um analista inexperiente. Além do mais, o analista inexperiente possui a tendência de tentar incluir "o máximo possível" em um modelo de simulação, com medo de eliminar fatores que poderiam ter alguma relevância e evita pensar se estes fatores são realmente necessários. HENIZE [84] afirma que o analista experiente é aquele que aprendeu a distinguir e separar o que é essencial do que é meramente importante.

1.4.4 MODELO SIMPLES E VALIDADE DO MODELO

Foram apresentadas vários benefícios decorrentes de modelos de simulação simples. Por outro lado, podem haver alguns problemas em relação à criação de modelos simples.

Parafraseando Einstein, um modelo deve ser o mais simples possível, mas não o mais simples; deve ser complicado, se necessário, mas não muito. Um perigo da simplificação de modelos é a sua super-simplificação e a perda da sua validade, pois se forem omitidos fatores importantes no modelo, este pode falhar em tentar representar a realidade. Segundo ZEIGLER [84], um modelo mais refinado aparentemente representa mais validamente a realidade, embora, segundo SALT [93], seja também possível criar um modelo complexo e cheio de detalhes e completamente impreciso (não representa

fielmente a realidade). O grande desafio do desenvolvimento de uma técnica de redução de modelos de simulação seria que esta mantivesse a validade do modelo.

1.4.5 MODELOS SIMPLES, DESEMPENHO COMPUTACIONAL E REDUÇÃO DO TEMPO DO CICLO

De acordo com REXTAD e INNIS [85], um esforço considerável pode ser necessário na simplificação de um modelo existente. Se um processo de redução demandar muito tempo, podem haver ocasiões em que um modelo mais simples leva mais tempo para ser construído.

Não se encontra na literatura a relação clara entre a criação de modelos simples e o desempenho do modelo computacional / redução do tempo do ciclo de um estudo de simulação. Para tal foram realizados dois estudos (reportados em CHWIF, BARRETTO e SANTORO [98] e CHWIF e BARRETTO [98]), onde um modelo inicial foi simplificado, apesar de não se utilizar de uma técnica formal de redução de modelos e sim somente raciocinando sobre o sistema. Os resultados foram bastante animadores, sendo que, em um dos estudos, o modelo mais simples "rodava" 8 vezes mais rápido do que o modelo inicial. No outro estudo, o número de elementos do modelo de simulação, em relação ao modelo mais complicado, foi reduzido em 6 vezes e, o número de linhas de código, em 3. Neste caso, o tempo de ciclo total do estudo de simulação poderia ser reduzido de 39 horas para 17 horas, através da utilização de um modelo mais simples.

Embora estes trabalhos não permitam obter um resultado fechado, em se tratando da relação modelos simples / desempenho computacional / tempo do ciclo de um estudo de simulação, indicaram que se pode atingir resultados consideráveis em termos de ganho de desempenho computacional e tempo do ciclo de um estudo de simulação, o que é uma das maiores motivações do desenvolvimento deste trabalho. De acordo com PEDGREN, SHANNON e SADOWSKI [95], ter-se resultados (mesmo que aproximados) de um

modelo simples, antes do prazo de um estudo de simulação, é infinitamente superior a ter resultados de um modelo altamente complexo após o prazo.

Apesar da simulação computacional oferecer todas as vantagens descritas anteriormente, na prática, um problema constante de um estudo de simulação é que o desenvolvimento de um modelo de simulação leva um tempo considerável quando comparado ao desenvolvimento de outros tipos de modelos (analíticos, icônicos, etc.). Acredita-se que o estudo e desenvolvimento de técnicas que permitam a obtenção de modelos mais simples, além dos benefícios já descritos, tenha papel primordial na redução do tempo total de um estudo de simulação.

Na grande maioria dos casos, a redução de um modelo ocorre depois da sua análise, quando é feito um estudo de sensibilidade, a fim de verificar a relevância de determinados componentes. Por isto, acredita-se que, ao desenvolver uma técnica adequada de redução de modelos, logo na primeira fase do ciclo de vida (concepção), tenha se uma redução significativa na duração deste ciclo, especialmente nas etapas de implementação e análise, mesmo que haja um aumento do tempo na etapa ou fase de concepção do modelo. A figura 1.4 ilustra o descrito.

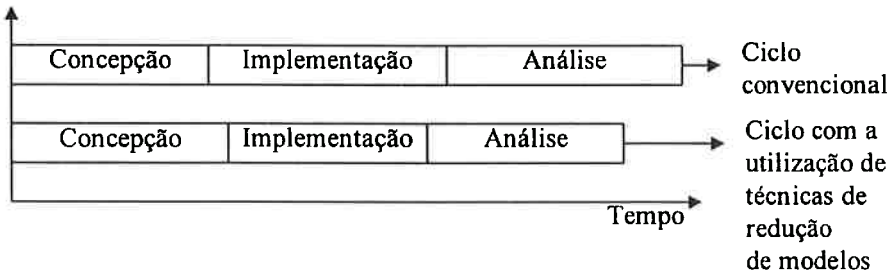


fig. 1.4 Comparação entre o ciclo de vida de um modelo de simulação com e sem técnicas de redução

Além das vantagens descritas anteriormente, tendo-se um modelo mais simples, a probabilidade de ocorrência de erros nas fases de implementação e análise é menor, o que indiretamente pode levar, também, à redução do tempo total do ciclo.

Na realidade, a motivação para reduzir a complexidade de modelos de simulação parece que mudou ao longo do tempo. Segundo LADY [81], a motivação principal para a redução de modelos, na década de 70, era a restrição computacional. Na década de 80, a motivação principal para a redução era o entendimento. Pode-se afirmar que, na década de 90 e no novo milênio, as motivações principais para a redução de modelos são o entendimento e, também, o tempo de desenvolvimento (a necessidade de se reduzir o tempo de um estudo de simulação).

1.5 OBJETIVOS E ESCOPO DO TRABALHO

O objetivo principal deste trabalho é o desenvolvimento de uma técnica de redução de modelos de simulação.

Baseada nas discussões anteriores, este processo de redução deve obedecer a alguns requisitos:

1. Deve depender, fundamentalmente, dos objetivos da simulação. Se os objetivos da simulação não estão claros, o processo de redução de modelos torna-se impraticável.
2. Deve ser realizado na fase de concepção do modelo, a fim de poder promover os ganhos de tempo descritos em todo o ciclo de vida de um modelo de simulação.
3. Deve garantir que o modelo mais simples gerado seja ainda válido, dentro dos objetivos da simulação. De nada adianta ter um modelo simples que não represente a realidade.
4. Deve ser passível de ser implementado e automatizado no computador, podendo constituir-se em um módulo do um ambiente de simulação. Isto porque, como o objetivo primordial do desenvolvimento de uma técnica de redução é diminuir o tempo de ciclo de um estudo de simulação, não seria válido se o processo de redução

em si tomasse muito tempo, portanto, o seu tempo de realização deve ser mínimo quando comparado a todo o tempo do ciclo do estudo de simulação.

O modelo de simulação deve ser descrito através de uma técnica adequada de representação de modelos de simulação. Como objetivos secundários (mas não menos importantes) deste trabalho, destacam-se:

1. A pesquisa do estado da arte de inúmeras técnicas de representação de modelos de simulação existentes na literatura.
2. Identificação dos requisitos básicos de uma técnica de representação de modelos de simulação, do ponto de vista da redução.
3. Escolha de uma técnica apropriada, do ponto de vista da redução.

A técnica de redução a ser desenvolvida deve ser aplicável a quaisquer tipos de modelos de simulação de eventos discretos. Os modelos de simulação de sistemas contínuos estão fora do escopo deste trabalho.

1.6 ORGANIZAÇÃO DO TEXTO

A partir deste ponto, o texto está organizado da seguinte maneira:

No Capítulo 2 (Revisão Bibliográfica) faz-se uma revisão do estado da arte da redução de modelos de simulação, com a abordagem da literatura relevante, além da discussão de alguns tópicos igualmente importantes ao assunto como, as medidas de complexidade de modelos de simulação e uma nova área de pesquisa, denominada de Modelagem Automatizada. Embora os modelos desta última sejam derivados, principalmente, de sistemas físicos (sistemas térmicos, hidráulicos, mecatrônicos, etc), muitas das suas idéias podem ser importantes ao assunto redução de modelos de simulação de eventos discretos. A partir das discussões realizadas, são identificados os requisitos que uma técnica de representação de modelos de simulação devem ter para

facilitar o processo de redução. Várias técnicas de representação de modelos de simulação na literatura são revistas e avaliadas em relação a estes requisitos.

O Capítulo 3 (Activity Cycle Diagrams / Condition Specification: Uma Abordagem Combinada) define a abordagem de representação de modelos de simulação adotada para a aplicação da técnica de redução, a partir das discussões realizadas no capítulo 2. A título ilustrativo, alguns exemplos de modelagem são mostrados, utilizando a abordagem adotada.

Algumas abordagens à redução de modelos são discutidas no Capítulo 4 (Técnica Proposta). O processo de redução é escolhido e definido e, baseada nesta definição, também é proposto o algoritmo de redução, aplicado sobre a representação discutida no capítulo anterior.

No Capítulo 5 (Estudos de Casos), é apresentada a aplicação da técnica desenvolvida no capítulo 4 em diversos modelos. Uma avaliação da complexidade do modelo antes e depois da aplicação da técnica proposta é feita, a fim de se verificar o seu comportamento e eficiência.

Finalmente, no Capítulo 6 (Conclusões), são discutidas as principais conclusões acerca deste trabalho. Igualmente, são abordadas as contribuições do mesmo, bem como, são feitas sugestões para futuros trabalhos.

CAPÍTULO 2
REVISÃO BIBLIOGRÁFICA

2.1 LITERATURA DE REDUÇÃO DE MODELOS DE SIMULAÇÃO

A simplicidade é discutida filosoficamente em BUNGE [63] em seu famoso livro "O Mito da Simplicidade". Neste, ele discute a máxima conhecida como o "Princípio de Ockham", atribuídos ao filósofo do século XIV, William of Ockham, que diz "Entia non sunt multiplicanda praeter necessitatem" (As entidades não devem ser multiplicadas sem necessidade). A busca por esta simplicidade esteve sempre presente na literatura. Procedimentos de redução de modelos foram criados em diversas áreas. Um exemplo de um processo de redução aplicado a sistemas de massa-mola está descrito em CHEN e GERADIN [97]. MOORE [81] propôs um método de redução na área da Teoria de Controle.

Na área de simulação de eventos discretos, apesar da importância do assunto, segundo BROOKS e TOBIAS [96] e BROOKS e TOBIAS [99], a redução de modelos obteve pouca atenção na literatura. Segundo SEVINC [91], "Não há nenhuma teoria referente à redução de modelos desenvolvida, e nenhum procedimento de caráter geral. Esta área, com menos de meia dúzia de artigos publicados, é um campo aberto para convidar a atenção dos pesquisadores em metodologias de simulação". Para se ter uma idéia da escassa literatura na área, nem uma terminologia adequada existe. Aqui utiliza-se o termo "redução de modelos de simulação", mas muitas vezes encontra-se, na literatura, o termo "simplificação de modelos de simulação" (ex. BROOKS e TOBIAS [99]) e "abstração de modelos de simulação" (ex. FRANZ [95]).

2.1.1 MÉTODOS DE REDUÇÃO DE MODELOS

Uma das primeiras tentativas de se encontrar um método para redução de modelos de simulação é atribuída a ZEIGLER [76]. Este propôs quatro métodos básicos de reduzir um modelo de simulação:

1. "Cortar" alguns componentes insignificantes do modelo;
2. Trocar partes do modelo por variáveis aleatórias;
3. Diminuir a faixa de variação das variáveis do modelos;
4. Agrupar partes do modelo em blocos.

YI e ZHOU [89], INNIS e REXTAD [83] e ROBINSON [94] listaram alguns métodos possíveis para reduzir um modelo de simulação. Estes são baseados nas experiências dos autores, sendo propostas regras específicas para a redução de modelos, como as explicitadas, anteriormente, em ZEIGLER [76]. A maioria destas regras podem ser classificadas segundo, basicamente, três categorias de simplificação explicitadas em PEDGREN, SHANNON e SADOWSKI, [95] (muito similares aos métodos de ZEIGLER [76]), que são: a omissão, a agregação e a substituição (de partes do modelo por variáveis aleatórias).

COURTOIS [85] discutiu alguns assuntos relacionados ao fator de escala do tempo, que pode levar a possíveis decomposições de modelos de simulação. No entanto, em todos os trabalhos descritos, nenhum consiste de um procedimento formal para a redução de modelos e estão longe de se constituir em uma metodologia.

SERVINC e FOO [90] e SERVINC [91] foram os que mais se aproximaram para a criação de um procedimento de fato para a redução de modelos de simulação, baseado na teoria desenvolvida por ZEIGLER [76] e ZEIGLER [84]. O procedimento tenta reduzir o número de estados no modelo, procurando agrupar estados vizinhos em um

único estado. No entanto, esta redução se dá tendo-se que "rodar" o modelo de simulação, e não na fase conceitual, onde a redução poderia gerar mais ganhos.

FRANZ [95] chegou a propor uma taxonomia para técnicas de "abstração de modelos de simulação" (foi utilizado este termo no sentido do termo "redução de modelos de simulação"), classificando-as em três grandes grupos: técnicas que mudam os limites do modelo, técnicas que mudam o comportamento do modelo e técnicas que modificam a forma do modelo. A maioria destas técnicas são aplicadas a modelos físicos (modelos de sistemas térmicos, hidráulicos, mecatrônicos, etc) e não a modelos de simulação (no item 2.1.3 – Modelagem Automatizada, o uso destas técnicas será melhor abordado). Uma técnica considerada de abstração de modelos de simulação é a metamodelagem. Um metamodelo de simulação é uma aproximação de um modelo de simulação mais complexo e é obtido através da análise das entradas e saídas. Na maioria dos casos, através desta análise, uma equação é gerada, relacionando estas entradas e saídas. O nome metamodelo vem justamente do fato de que estas equações são um modelo (matemático) de um outro modelo (de simulação). BARTON [94] faz uma revisão de metamodelos na literatura e novos procedimentos para a geração de metamodelos são apresentados em CAUGHLIN [96]. Como para gerar os metamodelos é necessário "rodar" o modelo de simulação, embora possam ser considerados modelos mais simples do que o inicial, a metamodelagem está fora do escopo deste trabalho. Isto porque a redução de modelos de simulação aqui se refere apenas à redução de modelos comunicativos, no início do ciclo de vida de um modelo de simulação.

FISHWICK [88] propôs determinados métodos para o processo de abstração. Estes métodos, combinados com definições de processos em diferentes níveis de abstração, constituem um grafo parcialmente ordenado, denominado de *Abstractions Network,* que, por sua vez, é utilizado para estudar o comportamento do modelo em

diferentes níveis de detalhes. Um ambiente de simulação denominado HIRES (*HIerarchical REasoning System*) foi desenvolvido para suportar este processo. No entanto, nem todos os modelos desenvolvidos são de eventos discretos (há modelos de computação gráfica e modelos contínuos neste sistema).

Acredita-se que a falta de pesquisas nesta área e o baixo desenvolvimento de técnicas de redução de modelos de simulação são originados por três causas básicas, sendo uma inerente à própria natureza dos modelos de simulação e duas originadas por paradigmas. No tocante à primeira causa, segundo ZEIGLER [84], pode-se fazer uma analogia de um sistema a um "prato de spaghetti". Se o sistema não for passível de decomposição, ao se puxar um único fio de "spagetthi", todos virão juntos. Um sistema passível de decomposição, por outro lado, é um sistema no qual se pode puxar um fio sem alterar o resto. Na maioria dos casos dos modelos de simulação, estes representam sistemas "parcialmente" passíveis de decomposição, onde "tudo está interligado". Esta característica torna os modelos de simulação, uma vez construídos, de difícil redução.

Um paradigma muito encontrado na literatura é que a modelagem (formulação do modelo conceitual) em simulação é considerada mais uma "arte" do que uma ciência. Visto desta forma, há uma inclinação para evitar estudos científicos na formulação do modelo. Contrariamente a esta idéia, GASS [87] chega a afirmar que "é preciso se livrar da muleta de que a modelagem é uma arte". Talvez por conta deste paradigma, há consideravelmente um maior número de publicações sobre as fases de implementação e análise do que em modelagem conceitual em simulação; FISHWICK [92] enfatiza esta afirmação. De fato, de acordo com PAUL [96]: "Apesar da tentativa de boas intenções, os textos de simulação tendem a ignorar o importante processo de formulação do problema e desenvolvimento lógico do modelo".

Outro paradigma é em relação ao aumento de capacidade dos computadores. Se forem analisados os trabalhos na literatura referentes a análise estrutural e técnicas de simplificação de modelos, a maioria destes encontra-se concentrados na década de 80 ou antes disto. Uma explicação para tal fato é que, na época, os computadores eram menos poderosos e a redução de modelos era um assunto mais importante, já que um modelo mais complexo poderia ser computacionalmente inviável de ser implementado. Com o desenvolvimento dos computadores, atualmente não há, praticamente, restrições para o desenvolvimento de modelos complexos, do ponto de vista da capacidade computacional.

2.1.2 COMPLEXIDADE DE MODELOS DE SIMULAÇÃO

Até este ponto discutiu-se acerca da existência de modelos simples e complexos, suas vantagens e desvantagens, embora em nenhum instante foi discutido o que se entende por um sistema complexo. Apesar do conceito de simplicidade ser intuitivo, uma questão a ser discutida é como se pode definir, objetivamente, a complexidade de um modelo de simulação e saber que um dado modelo é mais simples ou mais complexo do que outro. É o que se pretende abordar, a seguir.

2.1.2.1 DEFINIÇÃO DA COMPLEXIDADE DE UM MODELO

De acordo com BROOKS e TOBIAS [96], não há uma definição unânime aceita do que pode ser considerado um sistema complexo. Apresenta-se algumas definições de caráter geral:

1. A complexidade está relacionada com a dificuldade de se entender um sistema [GOLAY, SEONG e MANNO, 89].
2. Um sistema complexo é constituído de um grande número de partes que se interagem de uma maneira que não é simples [SIMON, 64].

3. Um sistema é complexo quando contém redundância suficiente para permitir seu funcionamento, apesar da presença de vários defeitos [GEORGE, 77], como é o caso de um sistema para fins espaciais que pode ter redundância setúpla.

Muitas vezes a complexidade de um modelo de simulação pode ser confundida com o nível de detalhe de um modelo. Embora um modelo mais detalhado seja um modelo mais complexo, o nível de detalhe é apenas um componente da complexidade de um modelo de simulação. Na realidade, a complexidade de um modelo de simulação pode ser composta de dois componentes: o **escopo** e o **nível** do modelo (**profundidade**).

O **escopo** do modelo pode ser considerado como o tamanho ou abrangência do modelo. Um modelo de manufatura pode conter a fábrica inteira, ou somente uma parte desta, dependendo do que se deseja avaliar. O escopo responde a pergunta: "O que modelar ?".

O **nível** do modelo, por sua vez, define a quantidade de detalhe do modelo (ou a profundidade do modelo). Pode-se modelar uma máquina somente como um tempo médio de processamento ou modelá-la com o tempo de processamento, taxa de quebras, sistemas de transportes que a alimenta, padrões de turnos, etc. Neste segundo caso, o nível de detalhe é muito maior, embora não se tenha alterado seu escopo (máquina). O nível responde a pergunta: "Como Modelar ?"

Para uma discussão mais detalhada sobre o escopo e nível do modelo, referir-se a ROBINSON [94].

A definição da melhor escolha do **escopo** e **nível** do modelo e, por conseguinte, sua complexidade, como já foi comentado anteriormente, é considerada por muitos uma "arte". No entanto, não se pode negar que a modelagem é um processo criativo. Portanto a escolha apropriada do **escopo** e **nível** adequados estão diretamente relacionados com criatividade e experiência do analista.

2.1.2.2. MEDIDAS DE COMPLEXIDADE

Segundo YUCESAN e SCHRUBEN [98], uma medida de complexidade de um modelo de simulação pode ser útil em estudos práticos de simulação, bem como no auxílio na pesquisa de metodologias de simulação. No primeiro caso, a medida de complexidade seria útil para avaliar se é possível atender os requisitos de custo e prazo, com um certo número de recursos, num projeto de simulação. No tocante às pesquisas, uma medida de complexidade poderia ser útil na classificação de modelos de simulação. Assim, numa demonstração de aplicabilidade de uma certa técnica, pode-se garantir que os modelos de teste cubram uma alta gama de complexidade.

Há, na literatura, diversas medidas de complexidade definidas. No entanto, a maioria delas lida somente com a complexidade de programas. Neste caso, uma das medidas mais conhecidas, dentro da engenharia de *software*, é devida a McCABE [76]. A partir da constatação de que um programa pode ser graficamente representado por um grafo de controle, onde os nós representam blocos de código sequencial e os arcos, ramificações no programa, McCABE [76] introduziu uma medida denominada "Número Ciclomático". Neste caso, esta medida de complexidade para um programa é definida como o número de arcos menos o número de vértices mais o número de componentes conectados no grafo de controle do programa (com um vértice adicional que une o primeiro ao último componente). "Número Ciclomático" é também igual ao máximo número de circuitos linearmente independentes neste grafo.

WALLACE [87] definiu uma série de características que uma medida para determinação da complexidade de modelos de simulação deveria obedecer. Algumas destas estão descritas abaixo:

1. Possuir generalidade de uso;
2. Avaliar as características dinâmicas dos modelos;

3. Ser simples e fácil de entender;
4. Permitir a comparação relativa entre vários modelos.

Infelizmente, há somente poucas publicações sobre medidas de complexidade no que se refere, especificamente, a modelos de simulação. ZEIGLER [76] definiu 6 medidas, mas estas apenas são aplicadas a dígrafos que representam máquinas sequenciais autônomas. Neste caso, algumas destas medidas de complexidade, definidas por este dígrafo, são o número de variáveis de estado, o número de arcos e o tamanho do maior subgrafo forte (para definição sobre dígrafos fortes e fracos, ver HARARY [65]).

Umas das primeiras tentativas de definir uma medida de complexidade mais geral de um modelo de simulação foi decorrente de EVANS, WALLACE e SUTERLAND [67]. Suas medidas incluem o número de diferentes tipos de entidades no modelo, a interação entre estas, a quantidade de eventos e outras quantidades. Embora tenham mostrado o conceito da métrica, estes não a expressaram em termos matemáticos.

WALLACE [87] definiu uma métrica, chamada CAT, baseada em uma representação gráfica derivada da técnica de representação de modelos de simulação, denominada *Condition Specification* (ver item 2.2.2.3 para uma explanação sobre esta técnica). Esta métrica pretende medir tanto a complexidade de transformação, como a de controle. A complexidade de tranformação e controle, segundo WALLACE [87], são dois elementos da complexidade, e uma medida de complexidade de modelos de simulação deve incorporar, necessariamente, estas duas características da complexidade. Para uma discussão mais detalhada sobre estes conceitos, ver WALLACE [85] .

Outras métricas, baseadas em representações gráficas de modelos de simulação, mais especificamente, os *Event Graph* (ver item 2.2.2.5 para uma discussão mais detalhada sobre os *Event Graphs*), foram propostas por YUCESAN e SCHRUBEN [98]. Uma das medidas mais fáceis e intuitivas é, simplesmente, o número de vértices de um

Event Graph. Outras medidas mais complexas são a razão arcos / vértices e a medida do número ciclomático, baseadas no trabalho, já discutido, de McCABE [76].

Estas medidas referem-se a propriedades estruturais de modelos de simulação comunicativos, embora possam inferir, indiretamente, sobre a complexidade do modelo computacional, em termos de linhas de código e tempo de execução.

Uma abordagem diferente, não baseada na teoria dos grafos, para medir a complexidade de modelos de simulação, parte da teoria da informação. GOLAY, SEONG e MANNO [89] utilizaram uma medida baseada na entropia, calculada através da informação do número de estados do sistema e as probabilidades de cada estado. O problema consiste, justamente, em se determinar as probabilidades de cada estado, o que a torna de difícil aplicação, na prática.

Apesar da existência de algumas medidas de complexidade propostas para modelos de simulação, não há uma medida que cubra todos os aspectos de nível de detalhe e seja aplicada a todos os modelos. Não há, também, nenhuma medida de complexidade padrão encontrada na literatura.

2.1.3 MODELAGEM AUTOMATIZADA (*AUTOMATED MODELLING*)

Segundo XIA e SMITH [96], a Modelagem Automatizada é uma nova área de pesquisa e está atraindo atenção de pesquisadores, nos últimos tempos. Também é uma área inter-disciplinar que envolve diversas teorias, como a de simulação, modelagem, inteligência artificial. Parte de seus trabalhos são inspirados na área de Raciocínio Qualitativo (*Qualitative Reasoning*), através da Teoria de Processos Qualitativos (QPT), onde os fenômenos físicos são modelados, não com variáveis quantitativas, mas sim com variáveis qualitativas. A maioria da teoria desenvolvida em *Qualitative Reasoning* pode ser atribuída a KUIPERS [94].

Embora os modelos abordados na modelagem automatizada não sejam exatamente de simulação de sistemas de eventos discretos e sim de modelos obtidos a partir de sistemas físicos (sistemas de engenharia, como sistemas termodinâmicos, mecatrônicos ou, mesmo, sistemas biológicos), seus objetivos são os mesmos da redução de modelos de simulação de eventos discretos. Conforme XIA e SMITH [96], o objetivo da Modelagem Automatizada é desenvolver ferramentas computacionais, a fim de gerar modelos bem formulados, a partir de informações estruturais, de uma forma automática. Estas ferramentas obedecem a princípios de modelagem genéricos, na construção de modelos, a fim de produzir o modelo mais simples, mas suficientemente adequado para um problema em questão.

Dois requisitos são importantes para conseguir atingir os objetivos da modelagem automatizada: uma boa biblioteca ou base de conhecimento de modelos de componentes e um conjunto de princípios de modelagem (XIA [94]).

Duas importantes publicações deram, praticamente, início à área, gerando-se duas abordagens distintas para a criação de modelos mais simples: a de ADDANKI, CREMONI e PENBERTHY [91], denominada Grafo de Modelos (*Graph of Models*), e a de FALKENHAINER e FORBUS [91], denominada de *Compositional Modelling.*

Na idéia básica do *Graph of Models (GoM)*, diversos modelos são pré-estabelecidos, criando-se um grafo, onde cada vértice do grafo corresponde a um modelo diferente. Os modelos são conectados, um ao outro, através de arcos, onde cada arco especifica o efeito da mudança de hipótese do modelo "origem" para o modelo "destino". ADDANKI, CREMONI e PENBERTHY [91] fornecem um exemplo da construção de um simples GoM, a partir de um mecanismo de abertura de porta. Neste exemplo, há um GoM contendo três modelos: um que não considera a distribuição de massa e atrito, outro que considera somente os efeitos do atrito e um terceiro que considera os efeitos de torsão

elástica do mecanismo. O problema consiste em selecionar, do GoM, o modelo mais simples, que represente o fenômeno corretamente. Na abordagem do GoM, define-se dois tipos de conflitos: os internos e os empíricos. Os conflitos internos ocorrem quando um valor fornecido, ou derivado do problema, é inconsistente com a hipótese do modelo. Por exemplo, se um modelo que assume uma hipótese de fluxo laminar e o seu resultado fornece um número de Reynolds maior do que 2300, há um conflito interno. O conflito empírico ocorre quando o resultado do modelo é muito diferente do esperado. Basicamente, a abordagem de Modelagem Automatizada, neste caso, consiste em selecionar um modelo do GoM, no qual os conflitos sejam eliminados. Como a estratégia parte dos modelos mais simples para os mais complicados, o modelo resultante é o mais simples que modela o fenômeno em questão. O grande inconveniente desta abordagem, além de dever-se construir todos os modelos (o espaço de modelos é definido explicitamente), é que esta necessita resolver os modelos e comparar seus resultados com o do sistema real. De acordo com XIA e SMITH [96], esta abordagem aparenta ser mais uma estratégia de controle de modelos do que propriamente uma técnica de criação de modelos.

Diferentemente da abordagem do *GoM*, no *Compositional Modelling* o objetivo é construir um modelo adequado, a partir de um conjunto de fragmentos do modelo. O conjunto de todos estes fragmentos é denominados de Teoria de Domínio (*Domain Theory*). Há diferentes fragmentos de modelos, uma para cada conjunto de condições e cada fragmento contém todas as informações necessárias para seu uso no modelo. Por exemplo, um cano pode ter fragmentos relacionados com sua habilidade em carregar fluídos, sua resistência mecânica, propriedades elétricas, etc. O processo do fluxo de fluído através deste cano pode ser descrito com muitos fragmentos: um para o fluxo não viscoso, outro para o laminar, outro para o turbulento, etc. Além dos conjuntos de

fragmentos, é fornecida a descrição do cenário a ser modelado, que contém a sua estrutura física e uma série de declarações a partir do seu comportamento (condições iniciais, limites de operação, etc.). Basicamente, o cenário definine o contorno do sistema. Tendo-se o conjunto de fragmentos e a descrição do cenário, o usuário pode definir uma questão para o sistema. Em um sistema térmico, por exemplo, uma questão poderia ser: "Como um aumento da taxa combustível /ar no forno pode afetar a quantidade de vapor fluindo no aquecedor ?". Esta questão é, então, expressa em uma linguagem própria de questão (*formal query language*) e o procedimento de modelagem automática consiste em escolher adequadamente os fragmentos e compô-los, de modo a responder esta pergunta. Este modelo, de acordo FALKENHAINER e FORBUS [91], deve ser o mais simples, para o uso pretendido, e coerente, não podendo ter inconsistências internas. *O Compositional Modelling* é somente aplicável a sistemas contínuos, representados por equações diferenciais ordinárias ou representações baseadas em QPT (*Qualitative Process Theory*).

WELD [92] propôs um procedimento baseado na abordagem GoM introduzindo o conceito de aproximação calculada (*fitting approximations*) para a obtenção de modelos mais simples. Um modelo P é uma aproximação calculada de um modelo M se P contiver um parâmetro exógeno, de modo que as previsões, utilizando o modelo M são praticamente as mesmas utilizando o modelo P, se o parâmetro da aproximação chegar a um determinado limite. Por exemplo, se P for um modelo mecânico, que inclui um barbante inelástico, e M, o mesmo modelo, mas inclui um barbante elástico com equações mais complicadas (e.g. f=kx), os dois modelos tendem a ser equivalentes, se a constante elástica tender para infinito. Este procedimento busca modelos mais simples a partir da definição de uma questão do tipo X < Y e um modelo M detalhado, selecionando um modelo mais simples P, que super-estima X e sub-estima Y. A diferença com a

introdução deste novo conceito é que a mudança entre modelos no GoM pode ser feita independentemente do domínio em questão.

Baseado nos trabalhos de *Compositional Modelling* (o modelo mais adequado é obtido, também, a partir de fragmentos) e do desenvolvimento do conceito de aproximações causais, NAYAK [94] e NAYAK e JOSKOWICZ [96] desenvolveram um procedimento de modelagem automatizada que não é NP-completo (eles demonstraram que o *Compositional Modelling* é um problema, em geral, intratável, que é aliviado utilizando o conceito de aproximações causais).

Estendendo o conceito de *Compositional Modelling* e de aproximações causais, LEVY, IWASAKI e FIKES [97], elaboraram o procedimento denominado *Relevance Reasoning* para encontrar o modelo de simulação mais apropriado. Fornecido uma quantidade "q", relevante para a questão a ser respondida, este procedimento segue as influências causais, determinando todas as variáveis que podem afetar esta quantidade "q". Ainda, cada fragmento contém declarações de relevância e o nível de detalhe do modelo pode ser determinado a partir do cheque destas. A diferença deste procedimento com o descrito anteriormente, é que este seleciona somente o fragmento mais simples possível em cada classe, e somente ajusta sua complexidade, se necessário. De acordo com LEVY, IWASAKI e FIKES [97], esta abordagem leva a uma economia grande na busca de fragmentos. Como, neste caso, o modelo gerado consiste em um modelo de simulação (de sistemas contínuos), IWASAKI e FIKES [97] resumem, perfeitamente, o problema de determinar as variáveis de interesse do modelo, sem rodar a simulação, o que pode representar um paradoxo: "Para simular, há a necessidade de formular um modelo. No entanto para formular um modelo apropriado [simples], há a necessidade de saber o que pode ocorrer no futuro, o que requer a simulação".

RICKEL e PORTER [97] reportaram um procedimento semelhante ao anterior. No entanto, neste caso, o modelo é simplificado a partir de considerações das diferentes velocidades dos processos no modelo. Eles utilizaram um exemplo derivado de um modelo que representa um sistema botânico, contendo vários processos. Neste caso, por exemplo, o processo de entrada de água através da membrana celular ocorre em segundos, o do soluto em minutos , o processo de crescimento da planta requer dias, enquanto que um processo ecológico, que aconteça ao seu redor, pode levar meses, ou mesmo anos. Dado a escala de tempo de interesse do fenômeno, os processos que levam um tempo muito maior do que esta escala são irrelevantes. Além da escala de tempo de interesse, são definidas as variáveis de interesse do modelo. O procedimento procura no grafo de influência (dígrafo que mostra como as variáveis do modelo se relacionam), as variáveis que estão dentro da escala de tempo de um certo fenômeno. A ênfase deste procedimento é mais semelhante à do GoM, pois, a partir de um modelo pré construído, determina-se o menor conjunto deste para gerar um modelo adequado à variável de interesse.

Outros procedimentos referentes a modelagem automatizada podem ser encontrados em WILSON e STEIN [92], XIA, LINKENS e BENNETT [93] e TOP et al. [95].

Embora a modelagem automatizada seja uma área que ainda está na infância e não totalmente relacionada à modelagem de sistemas de eventos discretos, alguns dos seus conceitos fundamentais estão diretamente relacionados ao tema deste trabalho.

2.2 REVISÃO DAS TÉCNICAS DE REPRESENTAÇÃO DE MODELOS DE SIMULAÇÃO

São várias as técnicas de representação de modelos comunicativos encontradas na literatura. De acordo com CERIC e PAUL [89] e CERIC e PAUL [92], pode-se enumerar

quatro vantagens em representar o modelo de simulação de acordo com uma técnica apropriada:

1. Fornecer melhor entendimento sobre a lógica de operação dos sistemas e interações entre seus componentes;
2. Aumentar a eficiência do desenvolvimento do modelo de simulação computacional.
3. Assegurar uma validação mais apropriada do modelo de simulação;
4. Servir como um meio de comunicação entre o analista e o usuário.

Neste ponto, torna-se necessário fazer uma distinção entre técnicas de representação baseadas em uma Linguagem de Simulação (LS) e técnicas independentes de LS. Segundo NANCE [83], na década de 70, começou a ocorrer uma mudança de foco na comunidade de simulação de eventos discretos, de uma abordagem centrada em "Programa de Simulação" para uma abordagem centrada no "Modelo de Simulação", onde a característica principal é o entendimento humano e não a direta codificação no computador.

As técnicas de representação de modelos de simulação podem ser classificadas, de acordo com a estratégia da simulação (as estratégias da simulação foram discutidas no item 1.2), em neutras (se não podem ser enquadradas em nenhuma estratégia) ou orientadas (se obedecem à determinada estratégia: atividade, evento, processo ou três fases).

As técnicas de representação podem ainda, possuir representação gráfica, sendo consideradas como técnicas diagramáticas, ou possuir uma sintaxe própria, como o caso de uma linguagem de especificação de modelos de simulação.

2.2.1 REQUISITOS BÁSICOS DAS TÉCNICAS DE REPRESENTAÇÃO DO PONTO DE VISTA DE REDUÇÃO DE MODELOS

No capítulo 1 foram identificados dois pontos de suma importância, no que se refere à redução de modelos de simulação. O primeiro é que o analista "experiente", na maioria dos casos, tende a obter, naturalmente, um modelo mais simples. Portanto, uma possível técnica de redução de modelos de simulação seria muito mais vantajosa para um analista inexperiente. Outro ponto é que umas das principais causas do crescimento de modelos complexos de simulação é a falta de objetivos da simulação bem definidos e práticas pobres de modelagem [INNIS e REXSTAD, 83; YIN e NAN, 89; SALT, 93].

A partir destas considerações, seria interessante que uma determinada técnica de representação obedecesse a dois requisitos:

1. Que a técnica de representação fosse de fácil entendimento e uso por parte de um analista inexperiente.
2. Que a técnica de representação indicasse, explicitamente, os objetivos da simulação (medidas de desempenho de interesse).

Seria, também, interessante que a técnica de representação possuísse representação gráfica, já que várias ferramentas oriundas da teoria de grafos poderiam ser aplicadas para sua simplificação (para textos introdutórios sobre a teoria de grafos ver [HARARY et al, 65; HARARY, 69]).

A seguir, as técnicas de representação de modelos de simulação mais comumente encontradas na literatura são apresentadas. Também estas são, posteriormente, avaliadas segundo os critérios aqui apresentados (1 - Fácil Entendimento e Uso, 2- Indica Objetivos da Simulação Explicitamente, 3- Possui Representação Gráfica).

2.2.2 TÉCNICAS DE REPRESENTAÇÃO DE MODELOS DE SIMULAÇÃO

2.2.2.1 ACTIVITY CYCLE DIAGRAMS

O *Activity Cycle Diagrams* ou, simplesmente, ACD, é uma técnica de representação disseminada, preponderantemente, no Reino Unido, desde da década de 1960. Originalmente, é baseada na idéia de TOCHTER de "Engrenagens Estocásticas" [TOCHTER, 63; PAUL e BALMER, 91; PIDD, 98].

O ACD é uma técnica diagramática que é capaz de modelar as interações entre as entidades (qualquer componente do sistema que retém sua identidade durante o passar do tempo), através da composição dos seus ciclos de vida. Uma entidade pode estar tanto num estado passivo (Fila) como num estado ativo (Atividade). O símbolo de "Atividade" ou "Fila" são os únicos símbolos que aparecem num grafo de ACD (ver figura 2.1).

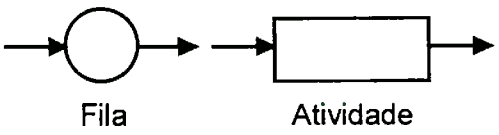


fig. 2.1 Elementos básicos da técnica de representação de ACD

É importante notar que a duração de uma atividade é sempre conhecida previamente (originada por alguma distribuição estocástica), um estado ativo normalmente envolve a cooperação de diferentes tipos de entidades e o tempo de permanência de uma entidade numa fila não pode ser estabelecido previamente.

Há algumas regras ou convenções para a construção do ACD:

1. Uma fila deve conter apenas um tipo de entidade.
2. Uma atividade sempre é antecedida por uma fila e vice-versa (quando não há razão para as entidades ficarem em fila antes de uma atividade, filas "fantasmas" devem ser incorporadas na representação).

3. Todos os ciclos de vida de cada entidade devem ser fechados.

O exemplo mais clássico e comum de uma representação em ACD é o exemplo do *Pub* Inglês. Neste caso, o modelo contém três entidades: os consumidores, os garçons e os copos. Os consumidores bebem ou esperam serem servidos. Os garçons estão servindo as bebidas ou estão parados. Os copos podem estar sendo utilizados pelos garçons enquanto servem as bebidas, sendo utilizados pelos consumidores enquanto estes bebem ou estarem esperando os garçons os encherem ou os consumidores os beberem. O ACD para esta versão simplificada do *Pub* está mostrada na figura 2.2.

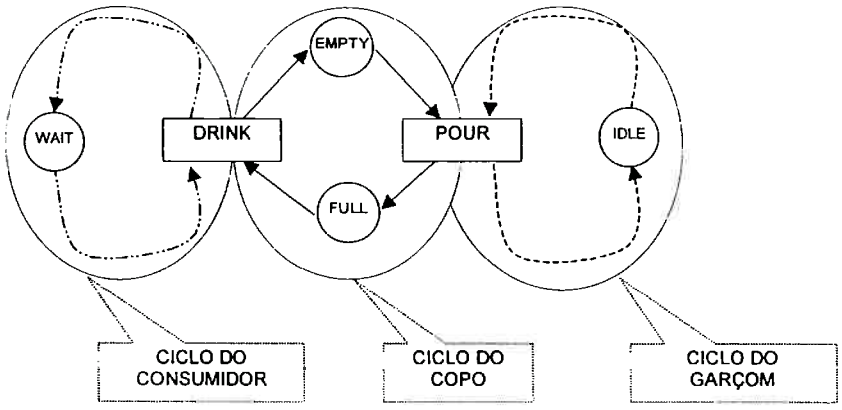


fig. 2.2 Representação em ACD do Pub Inglês na versão simplificada.

Se forem adicionadas "marcações" ao grafo de ACD (cada marcação representando uma única entidade), é possível efetuar uma simulação manual e, a posição de cada marcação, em uma atividade ou fila, em determinado instante, fornece o estado do sistema. Foi demonstrado, na literatura, que alguns métodos de execução de simulação são mais indicados para representações em ACD, como o método das Três Fases [PAUL, 1993].

Embora o ACD seja uma técnica de fácil aplicação, os diagramas obtidos podem ser extremamente grandes (com inúmeras conexões), especialmente em se tratando de sistema complexo. PLFUGHOEFT e MANUR [94], apresentaram uma concepção de ACD em multiplos níveis ou *layers*, com o intuito de fazer com que os diagramas fossem

menos complexos, baseados na idéia de composição hierárquica. Contudo, não chegaram a alterar sua simbologia básica. Baseado na idéia de X-ACD (*eXtended Activity Cycle Diagrams*), de POOLEY [91], KIENBAUM e PAUL [94], propuseram uma nova simbologia para representação hierárquica de modelos. Para tal, esta é baseada em um diagrama contendo 10 símbolos básicos (em oposição de 2 do ACD inicial). Para maiores detalhes sobre esta técnica, baseada em ACD, ver a KIENBAUM [95].

O ACD é também muito utilizado como o *front-end* de diversos "Geradores de Programas" (programas que produzem outros programas em uma linguagem de alto nível [MATHEWSON, 74]). Como alguns exemplos destes geradores de programas baseados em representação ACD, pode-se citar o CAPS/ECLS [CLEMENTSON, 91], DRAFT [MATHEWSON, 85] e VS7 [PAUL, 90].

Embora de concepção simples e de fácil aplicação e entendimento, é difícil capturar toda lógica de um modelo de simulação na representação em ACD, especialmente para os modelos mais complexos [MAK, 92]. Ainda o ACD não é usado como uma técnica formal de representação, apesar de DOMINGO [91] ter proposto uma descrição formal.

2.2.2.2 CONTROL FLOW GRAPHS

A técnica de representação baseada no *Control Flow Graph* ou CFG foi desenvolvida por COTA e SARGENT [90]. Seu objetivo principal é tornar a explícita a informação, para possibilitar o desenvolvimento de algoritmos para a execução de simulação em arquiteturas paralelas e distribuídas [COTA et al., 94].

Esta técnica é baseada em uma modificação da estratégia de processo, onde cada processo lógico é totalmente encapsulado e interage com outros componentes (ou outros processos), através de trocas de mensagens. Cada processo tem associado a ele um

conjunto de variáveis de estados e um conjunto de canais de entrada e saída. O comportamento de cada processo é especificado através de um CFG. Um CFG é um dígrafo estendido onde os vértices representam os estados possíveis de um processo (onde este aguarda alguma condição tornar verdadeira ou determinado período de tempo passar).

Os arcos deste dígrafo mostram as possíveis transições de estados. Associados a cada arco há três atributos: condição, prioridade e evento. A condição especifica se um arco pode ser escolhido para se efetuar a transição. A prioridade é utilizada para decidir qual arco irá efetuar a transição, caso dois arcos sejam candidatos no mesmo instante. O evento especifica a ação que é tomada durante a transição de estados (podendo modificar as variáveis de estado pertencentes a um processo ou enviar/receber mensagens de outros processos). Cada condição, por sua vez, pode ser de três tipos:

1. Condição Temporizada: torna-se verdadeira após determinado período de tempo ter passado (dado, provavelmente, por alguma distribuição aleatória);
2. Condição de Chegada: torna-se verdadeira caso um canal possua mensagens aguardando serem recebidas;
3. Condição de Expressão Booleana: torna-se verdadeira caso uma determinada expressão boolena, constituída de variáveis locais for verdadeira.

Para cada tipo de condição (temporizada, chegada, expressão boolena), a notação para os arcos é ilustrada pela figura 2.3.

Como um exemplo de modelagem por CFG, apresenta-se aqui o caso de um simples servidor com interrupção (*preemption*) descrito em FRITZ e SARGENT [95]. Neste caso, o servidor é capaz de lidar com duas classes de tarefas, uma de "alta prioridade" e outra de "baixa prioridade". As tarefas, em cada classe, são processadas através da regra FCFS (*First Come First Serve*). O servidor processa uma tarefa de alta

prioridade, se disponível, fazendo sempre isto até o término da tarefa (não há interrupção ou *preemption*). Se o servidor estiver processando uma tarefa com baixa prioridade, ao chegar uma tarefa de alta prioridade, o processamento da tarefa de baixa prioridade é interrompido e o servidor processa a de alta prioridade. Ao completar esta tarefa, e inexistindo outras tarefas de alta prioridade na fila, o processamento da tarefa de baixa prioridade, que foi interrompido, é retomado. A representação em CFG para este exemplo é mostrado na figura 2.4.

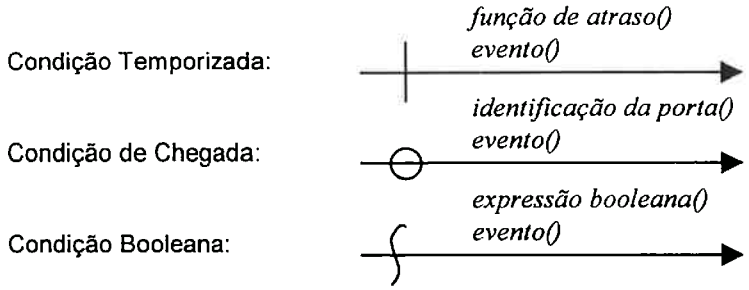


fig. 2.3 Notação para os arcos num CFG [FRITZ e SARGENT, 95].

Neste caso há, basicamente, quatro estados: I (repouso), quando o servidor não possui nenhuma tarefa, BL, quando o servidor está ocupado, processando uma tarefa de baixa prioridade e BH ou P, quando o servidor está processando uma tarefa de alta prioridade. Neste último estado (P), a tarefa de baixa prioridade foi interrompida, devido a chegada de uma de alta. As funções t-lo(), t-hi() especificam o tempo de processamento, enquanto que as condições lo-in, hi-in indicam a chegada, respectivamente, de tarefas de baixa e alta prioridade. Os números nos arcos (1,2) indicam a prioridade destes, sendo maior quanto menor o seu valor numérico.

FRITZ e SARGENT [95], também desenvolveram uma técnica de representação hierárquica, baseada no CFG, denominada HCFG (*Hierarchical Control Flow Graph Models*). Esta técnica se utiliza da mesma estratégia de processo do CFG e fornece dois tipos de estruturas hierárquicas que podem ser utilizadas concorrentemente. Um sistema

protótipo, que suporta esta técnica hierárquica, foi desenvolvido, denominado HIMASS (*Hierarchical Modeling and Simulation System)* [FRITZ, SARGENT e DAUM, 95]). Protótipos deste sistema podem ser encontrados para *download* (http://ocelot.cat.syr.edu/~srg/HIMASS.html).

De acordo com COTA, FRITZ e SARGENT [94], a aplicação do CFG é direta quando o sistema a ser modelado é simples, mas esta técnica pode se tornar extremamente complicada em se tratando de sistemas mais complexos.

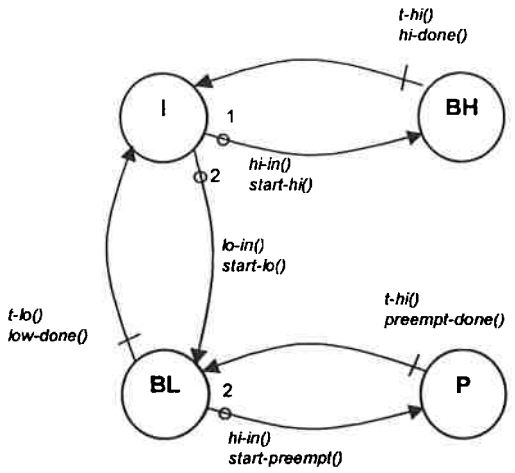


fig. 2.4 Representação em CFG para um servidor com preempção [FRITZ e SARGENT, 95]

2.2.2.3 CONDITION SPECIFICATION

OVERSTREET e NANCE [85] introduziram uma representação textual (linguagem de especificação de modelos de simulação), denominada *Condition Specification* (CS), com o objetivo de fornecer assistência ao analista na análise de modelos de simulação. O CS é baseado no *Conical Methodology*, também introduzido por NANCE [79] (uma abordagem que mistura definições *Top-Down* com especificações *Bottom-up*). Esta metodologia é embasada no paradigma de orientação por objetos [PAGE, 94]. Uma revisão atualizada pode ser encontrada em NANCE [94].

Basicamente, um modelo expresso em CS consiste de dois elementos básicos [PAGE, 94]: a descrição da comunicação do modelo com seu ambiente, composta pela especificação da interface (1) e a especificação de sua dinâmica, composta pela especificação dos objetos (2) e a especificação da transição (3). O CS define, também, a especificação de relatório (4) mas, para efeito das discussões, foca-se aqui as especificações 1,2,3.

A especificação da interface identifica os atributos (variáveis) de entrada e saída pelo nome, tipo de data e tipo de comunicação, que o modelo de simulação interfaceia com seu ambiente. Esta especificação deve possuir obrigatoriamente, no mínimo, um atributo de saída (medida de desempenho). A especificação dos objetos basicamente é uma lista de todos os objetos e seus atributos. A especificação da transição consiste em um conjunto de pares ordenados, denominados CAPs (*Condition-Action Pairs*). A condição é, simplesmente, uma expressão booleana que deveria ser avaliada durante a instanciação do modelo de simulação (execução). A estrutura do CAP é similar a das regras de produção de sistemas especialistas, dada sua estrutura *If - Then*. Se dois CAPs possuem exatamente a mesma condição, estes podem ser agrupados em ACs (*Action Clusters*). Também duas condições aparecem em toda especificação da transição: inicialização e finalização. A primeira é somente verdadeira no início da instanciação e, a última, somente no final. Do ponto de vista da estratégia de simulação, esta é uma técnica neutra, isto é, não é influenciada por nenhuma estratégia (processo, atividade, evento e três fases). No entanto, OVERSTREET e NANCE [86], demonstraram que esta pode ser decomposta, de tal maneira a se encaixar, perfeitamente, em uma determinada estratégia.

Toda a especificação é escrita como uma *syntaxe* do tipo *Pascal-Like*. Para a especificação da transição, o CS fornece várias primitivas, que podem ser vistas na tabela 2.1.

tab. 2.1 Sintaxe básica do Condition Specification [PAGE, 94]

FUNÇÃO	SINTAXE
Atribuir valor a variável	<objectName>.attribute:= newValueExpression
Setar um alarme	SET ALARM(alarmName, alarmTime <, argList>)
Verificar disparo de alarme	WHEN ALARM(alarmName)
Cancela alarme	CANCEL(alarmName <, alarmId>)
Criar entidades	CREATE(objectName<[instanceId]>)
Destruir entidades	DESTROY(objectName<[instanceId]>)
Ler variáveis	INPUT(attribList)
Escrever variáveis	OUTPUT(attribList)
Parar	STOP

A figura 2.5 ilustra a representação em CS de um modelo simples de fila M/M/1. Este exemplo é retirado de OVERSTREET et al. [94]. Pela própria sintaxe do CS e a dimensão do modelo, este exemplo não requer explicações adicionais. Note que a especificação da transição_possui 5 AC (*initialization, arrival, begin service, end service, termination*).

Para cada AC, os atributos são classificados de acordo com três categorias: **atributos de entrada**, **atributos de saída** e **atributos de controle**. De acordo com NANCE e OVERSTREET [87], um atributo x é um **atributo de controle** para um AC, se x aparece na condição deste AC. Um atributo x é um **atributo de saída** para um AC, se determinada ação do AC pode alterar o valor deste atributo. Um atributo x é um **atributo de entrada** para um AC, se o seu valor afeta os **atributos de saída**. Estes conceitos dão origem a toda a análise do CS, construída a partir de dígrafos. Para maiores detalhes, referir a OVERSTREET e NANCE [85], NANCE e OVERSTREET [88], OVERSTREET et al [87] e PAGE [94]. PAGE [94] também descreveu algoritmos para execução direta do CS, tanto em arquiteturas monoprocessadas como em arquiteturas paralelas.

Especificação da Interface para o modelo M/M/1:

```
Input Attributes
        Arrival_mean: positive real;
        Service_mean: positive real;
        Max_served: positive integer

Output Attributes
        Server_utilization: positive real;
```

Especificação dos objetos para o modelo M/M/1:

```
Environment
        arrival_mean            positive real;
        service_mean            positive real;
        system_time             positive real;
        arrival                 time-based signal;

Serverserver_status             (idle,busy);
        queue_size              nonnegative integer;
        num_served              nonnegative integer;
        max_served              nonnegative integer;
        end_of_service          timed-based signal;
```

Especificação da Transição para o modelo M/M/1:

```
{Initialization}
INITIALIZATION:
        INPUT(arrival_mean,service_mean,max_served);
        CREATE(server);
        Serverstatus=idle;
        Num_served:=0;
        System_time=0.0;
        SET ALARM(arrival,0);

{Arrival}
WHEN ALARM(arrival):
queue_size:=queue_size+1;
        SET ALARM(arrival,negexp(arrival_mean);

{Begin Service}
queue_size>0 andserver_status=idle:
        queue_size:=queue_size-1;
        server_status:=busy;
        SET ALARM(end_of_service,negexp(service_mean));

{End Service}
WHEN ALARM(end_of_service):
        server_status:=idle;
        num_served:=num_served+1;

{Termination}
num_served >=max_served:
        STOP
        PRINT REPORT
```

fig. 2.5 Representação em Condition Specification de um modelo M/M/1

2.2.2.4 DEVS (DISCRETE EVENT SYSTEM SPECIFICATION)

O *Discrete Event System Specification* (DEVS) é um formalismo (técnica de representação formal), inicialmente proposto por ZEIGLER[76] e ZEIGLER[84], baseado na teoria geral dos sistemas. Segundo ZEIGLER[76], embora uma representação informal possa comunicar a natureza essencial de um modelo, pode dar origem a certos problemas intrínsecos, como inconsistência, ambigüidade e a criação de um modelo incompleto.

O DEVS é baseado fortemente no conceito de estado. Basicamente, um modelo M representado em DEVS, é composto por 3 conjuntos (entradas, saídas e estados) e 4 funções. Matematicamente, pode ser escrito por:

$$M=<X,Y,S,\delta_{ext},\delta_{\phi},\lambda,t_a>$$

onde:

X é o conjunto de eventos internos de entrada

Y é o conjunto de eventos externos

S é o conjunto de estados sequenciais

δ_{ext} é a função de transição externa (QxX $\rightarrow$ S, onde Q={(s,e)| s $\in$ S, e $\in$ [0, $t_a(s)$])

δ_ϕ é a função de transição interna (S $\rightarrow$ S)

λ é a função de saída (S $\rightarrow$ Y)

t_a é a função de avanço do tempo (S $\rightarrow$ R^+_{oo})

Basicamente, a interpretação deste formalismo é a seguinte: um modelo, quando observado em um sistema simulado, muda de um estado para outro estado, em dois casos: (1) quando não ocorre nenhum evento externo para um determinado intervalo de tempo [t, t+ $t_a(s)$], e o modelo que estava em um estado s no tempo t, no tempo t+ $t_a(s)$ ele entrará no estado $d_\phi(s)$; (2) quando o modelo recebe um evento externo x, em um tempo t+e, e se o modelo estava no estado s no tempo t, então ele estará no estado $d_{ext}(s,e,x)$, imediatamente após o evento. Neste caso, se pressupõe que nenhum evento externo ocorreu no intervalo [t, t+e]. Finalmente, a função de saída efetua um mapeamento dos estados para os eventos de saída.

A figura 2.6 mostra um modelo expresso em DEVS de um servidor com estratégia FCFS (*First Come First Served*) e interrupção (*preemption*). Neste caso, uma tarefa que chega possui forma (a', i', p') onde a' é o nome, i' uma prioridade inteira e p' um valor real para o tempo de processamento.

ZEIGLER [84] também formaliza outros tipos de modelos, incluindo *Coupled Models*, isto é, modelos originados da junção de diversos modelos expressos em DEVS.

Apesar do grande poder de representação deste formalismo (em poder representar também sistemas contínuos, por exemplo), foi reconhecido pelo próprio ZEIGLER (ZEIGLER, HU e ROZEMBLIT [89]), que DEVS não é um modo prático de se representar modelos de simulação, já que se trata de um conjunto de conceitos teóricos.

$X_M = A_x I_x R_o^+$; $S_M = X^*_M$, $Y_M = A$

$\delta\phi(a,i,p)x) = x$ **para** $(a,i,p) \in X_M$, $x \in X^*_M$

$$\delta((a,i,p)x,e,(a',i',p')) = \begin{cases} (a',\ i',\ p')(a,i,p-e)x & \textbf{se } i'>i \textbf{ e } e<p \quad (1) \\ (a',\ i',\ p')x & \textbf{se } i'>i \textbf{ e } e=p \quad (2) \\ (a,\ i,\ p-e)PLACE((a',\ i',\ p')x)) & \textbf{caso cont.} \quad (3) \end{cases}$$

$$t((a,i,p)x) = p$$

$$\lambda(a,i,p),e,x) = \begin{cases} a & \textbf{se } p = e \quad (1) \\ \phi & \textbf{caso cont.} \quad (2) \end{cases}$$

$(\delta\phi(\Lambda) = \Lambda;\ t(\Lambda) = \infty,\ \lambda(\Lambda) = \phi)$

fig. 2.6 Representação em DEVS para servidor com interrupção [ZEIGLER, 76]

2.2.2.5 EVENT GRAPHS

SCHRUBEN [83] introduziu uma técnica de representação, baseada na estratégia de eventos a partir da observação de que, praticamente, inexistem, na literatura, técnicas de representação com esta estratégia.

Os *Event Graphs* (EG) podem ser aplicados para a representação de qualquer modelo de sistema de eventos discretos, utilizando apenas um símbolo [SCHRUBEN, 92]. Neste caso, os vértices de um *Event Graph* representam eventos que alteram os valores das variáveis de estado. Os arcos, por sua vez, representam condições onde um evento pode causar a ocorrência de outro evento, bem como determinar o intervalo de tempo a transcorrer entre dois eventos. A notação básica de um *Event Graph* pode ser visto na figura 2.7.

Esta notação pode ser interpretada da seguinte maneira: Se o evento A ocorrer, se a condição (i) for verdadeira, o evento B é programado para ocorrer em um tempo t, com o vetor de parâmetros j assumindo os valores correntes da expressão em k. Se, no entanto, o arco estiver indicado não como uma linha cheia, mas como uma linha tracejada, tem-se que, em t unidades de tempo, após a ocorrência de um evento A, qualquer ocorrência prevista do evento B será cancelada se a condição (i) for verdadeira no instante da

ocorrêcia de A. Note-se que *loops* são permitidos num *Event Graph* e pode haver mais de um arco conectando um par de eventos.

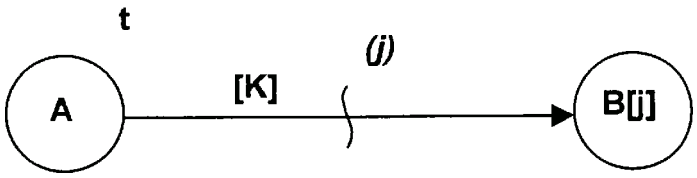


fig. 2.7 Representação básica de um Event Graph [YUCESAN e SCHRUBEN, 92]

O exemplo a seguir (figura 2.8) é retirado de YUCESAN e SCHRUBEN [92] e ilustra a representação de EG para um sistema de filas M/M/1. Neste caso, os consumidores chegam a cada ta unidade de tempo e são atendidos em ts unidades de tempo. As variáveis de estados são: Q (número de consumidores na fila), S é o status do servidor (0 = ocupado, 1 = livre), *id* é o número de identificação do consumidor, *in* representa o número de identificação do consumidor quando este esta sendo atendido, *nx* é o número de identificação do consumidor que é o próximo da fila, W[i] é o tempo total que o consumidor i gasta no sistema e CLK representa o relógio de simulação. As condições deste modelo são as seguintes:

1. Condição 1: S=1;
2. Condição 2: Q>0.

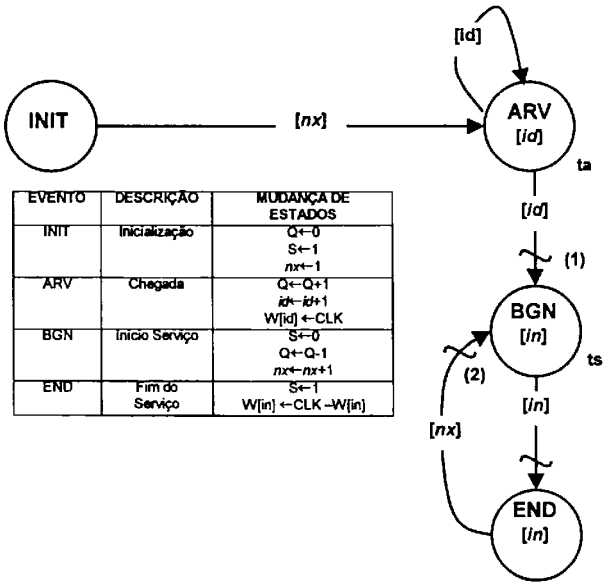
Uma série de outros exemplos de modelos representados por EG pode ser encontrada em BUSS [96].

Os *Event Graphs* podem, também, ser matematicamente definidos por um *Simulation Graph* [YUCESAN, 89]. Um *Simulation Graph* é uma quádrupla ordenada:

$G=<V(G), Es(G), Ec(G), \Psi g>$

onde V(G) é o conjunto de vértices (eventos), Es(G) é o conjunto de arcos cheios, Ec(G) é o conjunto de arcos tracejados (cancelamento de eventos) e Ψg é a função de transição.

Para detalhes, ver INGALLS, MORRICE e WHINSTON [96] e YUCESAN e SCHRUBEN [92]. Estes últimos autores ainda utilizaram o conceito de *Simulation Graphs* como base para avaliação de equivalência entre diferentes modelos. Eles conseguiram demonstrar que dois *Simulation Graphs* são estruturalmente equivalentes, se os grafos originados pela sua expansão (operação definida sobre *Simulation Graphs*) são isomórficos. Um resultado importante é que a equivalência estrutural de dois modelos de simulação é condição suficiente (mas não necessária) para equivalência comportamental. Contudo, este resultado ainda não foi implementado em um ambiente, sendo sua aplicação em modelos maiores, sujeito à pesquisas futuras. Algumas "Regras Práticas" de simplificação de EG foi colocada por YUCESAN e SCHRUBEN [92]. Estas simplificações, no entanto, não foram feitas à luz dos objetivos de simulação.



EVENTO	DESCRIÇÃO	MUDANÇA DE ESTADOS
INIT	Inicialização	Q←0 S←1 *nx*←1
ARV	Chegada	Q←Q+1 *id*←*id*+1 W[id] ←CLK
BGN	Início Serviço	S←0 Q←Q-1 *nx*←*nx*+1
END	Fim do Serviço	S←1 W[in] ←CLK –W[in]

fig. 2.8 Representação em EG do modelo M/M/1. [YUCESAN e SCHRUBEN, 92]

SCHRUBEN [90] e SCHRUBEN [92b] desenvolveram um sistema protótipo baseado no conceito de EG, denominado SIGMA.

2.2.2.6 REDES DE PETRI

As Redes de Petri tiveram sua origem com o trabalho de PETRI [66] e, desde a década de 70, tem-se originado uma enorme quantidade de publicações a seu respeito, especialmente em países europeus. Segundo PETERSON [77], a Rede de Petri é um modelo formal, abstrato do fluxo de informação. É uma ferramenta que é bem aplicada em sistemas que exibem paralelismo e concorrência.

Inicialmente, apresenta-se, aqui, o conceito básico de Redes de Petri e, a seguir, algumas de suas variações (que são inúmeras) mais apropriadas para a modelagem de sistemas de eventos discretos com fins de simulação.

Uma Rede de Petri básica (ordinária) é um grafo bipartido. Este grafo contém dois tipos de vértices: os "lugares" (representados, tipicamente, por círculos) e as "transições" (representadas por barras). Um arco do grafo existe somente entre lugares e transições. Se um arco que liga um lugar P a uma transição T existe, então P é denominado lugar de entrada e T, por sua vez, é a transição de saída. Analogamente, se um arco liga uma transição T' a um lugar P', P' é um lugar de saída para a transição T' e esta é uma transição de entrada para o lugar P'. Dado qualquer instante, somente os lugares podem conter 0 ou mais "marcações" (representados por pontos nos lugares). Uma Rede de Petri que contém marcação é denominada Rede de Petri Marcada. Uma certa distribuição de todas as marcas na rede constitui-se de uma "marcação" e determina o estado da rede em certo instante.

O comportamento dinâmico das marcas se dá quando uma transição é disparada. Uma transição só pode ser disparada se todos os lugares de entrada para uma determinada transição contiverem, pelo menos, uma marca em cada lugar, e o resultado do disparo faz com a marca seja depositada em um dos seus lugares de saída.

Matematicamente, a estrutura de uma Rede de Petri Simples (PN), pode ser expressa por :

$PN=<P,T,I(t),O(t),\mu>$

onde P representa o conjunto de lugares, T representa o conjunto de transições, I,O representam, respectivamente, o conjunto de lugares de entrada e saída para as transições e μ representa a marcação inicial da rede.

Segundo MURATA [89], uma das maiores vantagens das Redes de Petri é o seu suporte para a análise de muitas propriedades e problemas associados a sistemas concorrentes. Como algumas destas propriedades, pode-se citar a conservação, a alcançabilidade e a vivacidade. Muitas destas técnicas de análise são baseadas no conceito de "árvore de alcançabilidade" [PETERSON, 77].

Devido à não inclusão do tempo de processo, a Rede de Petri básica ou ordinária não é muito adequada para a representação de modelos de simulação. Muitas extensões foram criadas, sendo que algumas delas estão explicitadas a seguir:

1. **Redes de Petri Temporizadas (TPN):** nesta extensão, a duração de tempo δ pode ser associado a transições ou a lugares. No primeiro caso, se uma transição T dispara no tempo t, a marcação de entrada é removida e a marcação de saída aparece no tempo $t+\delta$. No segundo, se uma marcação aparece em P no instante t, então a transição posterior não pode ser disparada até o tempo $t+\delta$. Para detalhes sobre Rede de Petri Temporizadas, ver e KUMAR e HAROUS [90] e NICOL e ROY [91];
2. **Redes de Petri Aumentadas (APN):** além da temporização, as marcações da rede podem ser criadas, destruídas, divididas ou condensadas. Ainda possuem incorporadas outras características, como arcos de controle (ver COOLAHAN [83] e EVANS [88]);
3. **Redes de Petri Estocásticas (SPN / GSPN):** se uma Rede de Petri Temporizada possui associados tempos exponencialmente distribuídos, temos uma Rede de Petri

Estocástica. Se estas redes, por sua vez, possuírem extensões das APN que generalizam as Redes de Petri, temos uma Rede de Petri Estocástica Generalizada (GSPN). Referir a MOLLOY [82] e HASS e SCHEDLER [88], para detalhes;

4. **Simulation Nets**: o primeiro a reconhecer que as propriedades das Redes de Petri básica não permite uma representação ideal de modelos de simulação foi TORN [81]. Para tal, ele introduziu extensões que incluem: arcos inibitórios, arcos de testes, propriedades temporais para as transições, notação especial para representação de filas entre outras. Para detalhes sobre os *Simulation Nets,* ver, também, TORN [85] e TORN [91].

Um exemplo mais completo do *Pub* Inglês, representado por *Simulation Nets,* pode ser visto na figura 2.9. Neste caso, o arco inibitório (1) é utilizado para gerar as entidades temporárias no sistema. Um lugar com uma barra horizontal é utilizado para modelar a entidade temporária enquanto espera a atividade começar. O arco de teste (3) é usado para especificar uma condição: se *need* (número de bebidas que cada consumidor irá ainda tomar) é nulo, então este deixa o sistema; caso contrário, retorna ao balcão para ser servido.

MURATA [89] propôs algumas técnicas de simplificação de Redes de Petri. No entanto, estas simplificações têm a finalidade de facilitar sua análise estrutural e não são levados em consideração os objetivos da simulação.

HUTCHINSON [81] realizou uma comparação entre Redes de Petri e ACD, mostrando muita semelhança entre estas técnicas, já que os lugares e transições nas Redes de Petri podem corresponder a, respectivamente, filas e atividades no ACD. SCHRUBEN e YUCESAN [94] propuseram uma forma de se converter um modelo de simulação, representado em Redes de Petri, para *Event Graphs*.

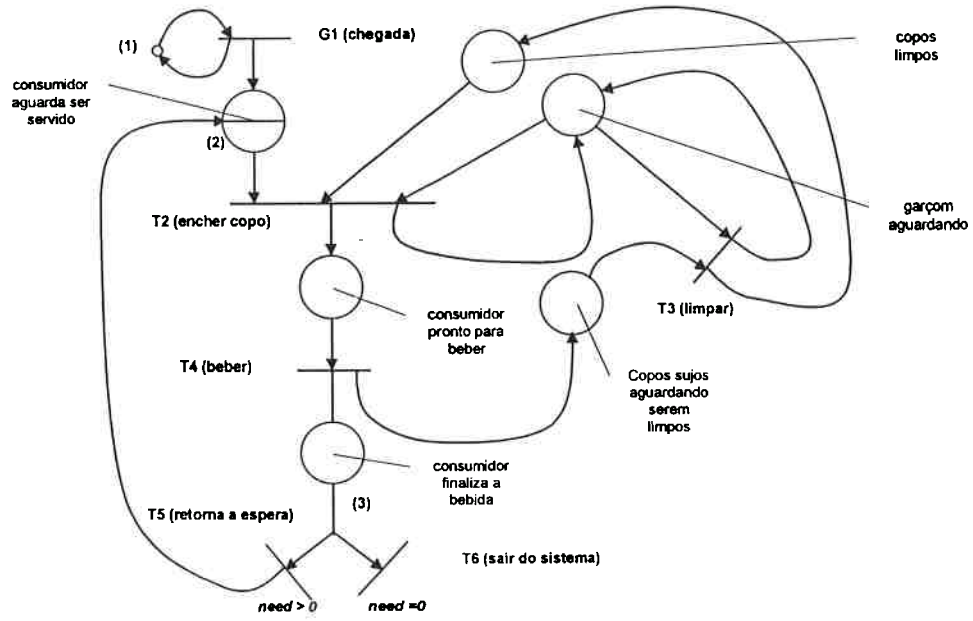


fig. 2.9 Representação em Simulation Nets do Pub Inglês. [DOMINGO, 91]

Esta técnica de representação foi a que gerou um maior número de sistemas computacionais. STORRLE [98] reporta a existência de quase uma centena de sistemas. Um grande número destes são descritos em http://ourworld.compuserve.com.homepage/wolf_garbe/petri-tools. Dentre alguns destes, pode-se citar o GreatSPN [BALBO e CHIOLA, 89], QPN-TOOL [BAUSE e KEMPER, 94], Time-NET [KELLING, 1995]. Um sistema denominado de DESIGN-CPN, baseado em Redes de Petri coloridas, pode ser obtido em http://www.daimi.aau.dk/designCPN/ de modo gratuíto.

2.2.2.7 STATE CHARTS

O State Chart (SC) é uma técnica proposta, inicialmente, por HARREL [87]. De acordo com GRUER, KOUKAM e MAZIGH [98], o SC é uma extensão dos autômatos finitos, que fornece uma decomposição hierárquica de estados, explícita representação da concorrência e comunicação em *Broadcast*. É, também, baseado no conceito de Higrafos

[HARREL, 88], muito utilizados na extensão de representações, como os diagramas entidade-relacionamento na especificação de bases de dados.

Os SC's são utilizados, basicamente, na especificação de sistemas reativos (sistemas que são caracterizados por serem dirigidos por eventos e tendo que reagir, continuamente, a estímulos externos e internos [HARREL et al, 1990]).

No diagrama de SC, os estados são representados por "Caixas", nomeadas com letras maiúsculas. Estes estados podem englobar outros e ser compostos por junções do tipo "E" ou "OU", já que os SC permitem construção hierárquica. Como os autômatos finitos, as transições entre estados são representadas por arcos orientados. A figura 2.10 ilustra um SC simples.

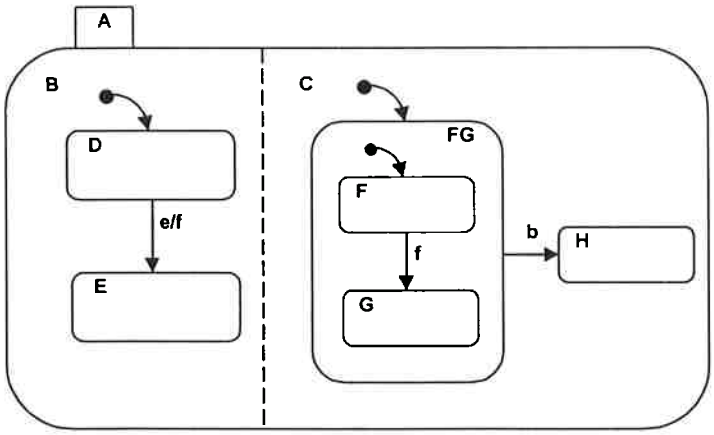


fig. 2.10 Representação de um State-Chart. [GRUER, KOUKAM e MAZIGH, 98]

Neste caso da figura 2.10, o super-estado "A" descreve a composicão paralela (junção "E") de dois processos concorrentes, definidos por B e C (este tipo de junção é representada no SC por uma linha pontilhada). O estado B, por sua vez, possui dois estados (D e E), e, como para estar no estado B, deve-se, obrigatoriamente, estar ou em D ou em E, este tipo de junção é denominada junção "OU". Um estado que possui apontado para ele uma seta marcada por um ponto indica o estado inicial ou *default* do SC. No caso da figura 2.10, os estados D, FG e F são os estados *default*. A sintaxe geral para as

transições obedece a notação e[c]/a, onde "e" é um evento, "c" uma condição e "a" uma ação. A ação é realizada depois da transição. Por exemplo, se o SC da figura 2.10 estiver na configuração (D,F) e o èvento e ocorrer, a ação f é imediatamente ativada, conduzindo, também, a transição do estado F para o estado G. Assim, a nova configuração é E,G.

O mesmo exemplo da seção anterior (Pub Inglês) é modelado agora utilizando a técnica do SC e está mostrado na figura 2.11.

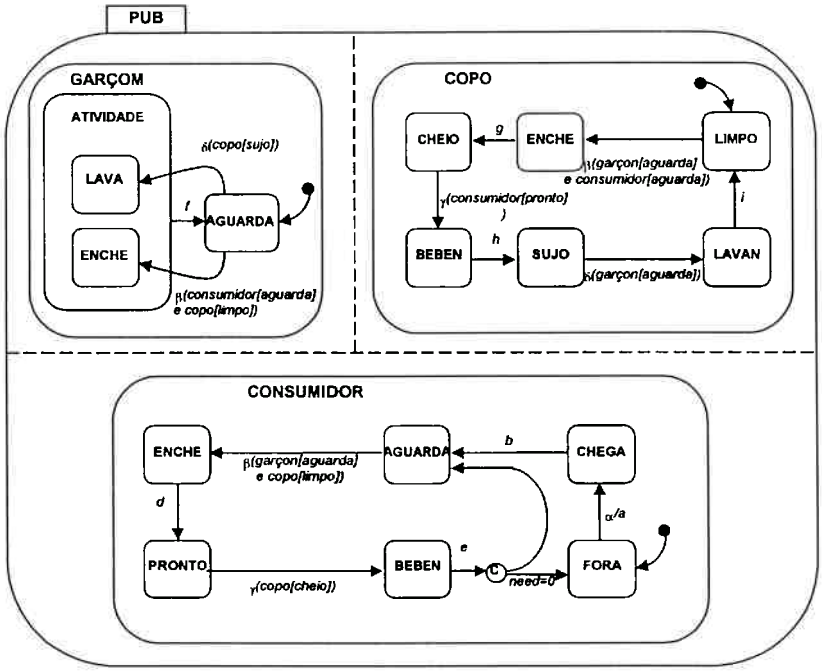


fig. 2.11 Representação em SC do Pub Inglês [DOMINGO, 91]

Note-se que, neste caso, devido à existência de uma condição de saída do cliente do Pub, tem-se o aparecimento de um C circunscrito, que denota, justamente, o desvio por entradas a estados condicionais.

Os SC mostram, claramente, a noção de paralelismo e, dada a sua concepção hierárquica, o diagrama se torna bem compacto. No entanto, a noção de fluxo fica prejudicada e é muito fácil para o analista se perder na evolução dos estados. Segundo DOMINGO [91], a aplicação direta dos SC para fins de simulação necessita incorporar mais algumas características, como a presença de estados diferenciados, passivos e ativos,

e extensões de temporizações. GRUER, KOUKAM e MAZIGH [98], propuseram algumas alterações nos SC para permitir análise quantitativa, no entanto, os SC ainda possuem pequeno uso na comunidade de simulação.

O STATEMATE é um sistema gráfico desenvolvido para a aplicação da representação de *State Chart* [HARREL, 95].

2.2.2.8 PROCESS NETWORKS

A técnica de representação por *Process Networks* (PN) é, sem dúvida, uma das formas mais antigas de representação de modelos de simulação [SCHUBEN, 92a]. Estas adotam, na sua grande maioria, a estratégia de processo. Embora sejam embasadas em representações gráficas, os PN são representações centradas em programas de simulação, portanto, dada uma representação gráfica de um modelo de simulação, expresso em PN, está-se associado a esta um "programa equivalente" descrito em alguma LS, pronto para ser executado no computador. Na realidade, a técnica de representação por PN não possui uma simbologia padrão e depende da LS que se está baseada. As PN são, portanto, uma classe ou um conjunto de técnicas de representação que possuem mesma natureza.

Uma das primeiras PN, criada há quase 4 décadas, está baseada no GPSS (*General Purpose Simulation System*). Uma introdução desta pode ser encontrada em SCHRIBER [91]. Neste caso, a PN é construída através de "blocos", que geram e processam entidades, manipulam filas e realizam desvios condicionais.

O *SLAM Network* foi desenvolvido por PRITSKER [86] e, além de permitir a representação de modelos na estratégia de processo, pode, também trabalhar por eventos ou em uma abordagem combinada (processo / evento). É muito próximo do GPSS, com algumas diferenças, como na passagem de tempo, que é modelada utilizando os arcos dos grafos, ao invés de utilizar vértices de avanço de tempo ou atraso.

A abordagem em PN de PEGDEN [PEGDEN et al., 90] segue, basicamente, os mesmos conceitos do GPSS e SLAM, sendo denominada de SIMAN.

Algumas destas representações em PN foram modificadas para ter um escopo mais específico, indo contra a idéia generalista das primeiras PN. O XCELL é uma técnica de PN, criada, especificamente, para a representação de sistemas produtivos (fábricas). Ver CONWAY et al. [92], para detalhes.

A figura 2.12 mostra um diagrama de blocos em GPSS de um sistema de filas M/M/1. Aqui não se pretende explicar sua simbologia, em razão do escopo do texto e, também, por sua extensão (há mais de 50 símbolos no GPSS). Além do que, o intuito é de fornecer o conceito básico deste tipo de técnica, sem a pretensão de ser exaustivo (cobrir todas existentes).

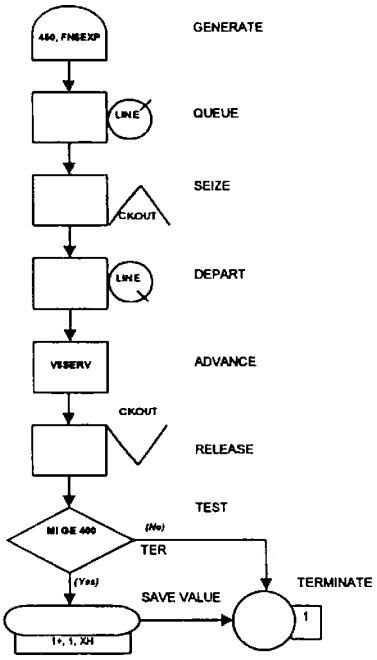


fig. 2.12 Representação em diagramas de blocos GPSS de um sistema M/M/1 [BANKS e CARSON, 84]

Em sendo baseadas em LS, outras técnicas de PN e seus respectivos avanços podem ser acompanhados, anualmente, nos Anais do *Winter Simulation Conference*, nos tutoriais sobre os softwares de simulação.

Embora sejam técnicas diagramáticas focadas em LS, fáceis de entender pelas maioria das pessoas, elas podem gerar dificuldades na validação e verificação do modelo correto, pois não são embasadas em nenhuma técnica mais formal. A "Ponte Conceitual" que estas técnicas promovem pode dar origem a uma "Fenda Conceitual" [PAGE, 95][1].

2.2.2.9 OUTRAS TÉCNICAS DE REPESENTAÇÃO

Há uma série de outras técnicas de representação de modelos de simulação, encontradas na literatura. Acredita-se que foram cobertas nas seções anteriores as que têm maior potencial de aplicação na prática (podendo ser utilizadas por não especialistas em simulação, inclusive). Apesar do caráter extremamente teórico, o formalismo DEVS foi incluído na seção 2.2.2.4, devido à sua grande importância no contexto da simulação, do ponto de vista da teoria geral dos sistemas.

Nesta seção, revisa-se, brevemente, algumas técnicas adicionais, classificadas segundo quatro abordagens básicas:

1. **Técnicas baseadas em álgebras específicas**. LACKNER [64] apresentou uma álgebra, denominada Cálculo de Mudança, para a representação de modelos de simulação. No entanto, de acordo com PAGE [95], sua álgebra é incompleta e difícil de usar. Mais recentemente, TOFTS [98] propôs uma álgebra baseada na Álgebra de Processos, denominada de WSCCS (*Weighted Synchronous Calculus of Communicating Systems*). Este ainda afirmou que uma linguagem de simulação orientada a processos pode ser totalmente representada de acordo com esta álgebra;
2. **Técnicas baseadas em lógicas computacionais**. Estas técnicas utilizam uma mistura de conceitos oriundos das áreas de inteligência artificial, engenharia de software e

[1] Segundo PAGE, uma LS pode "distorcer" o modo que o analista pensa naturalmente, obrigando-o a adaptar o modelo abstrato para a forma de um programa, podendo-se, por este motivo, gerar problemas na programação, verificação e validação de um modelo correto.

lógicas formais. NARAIN [91] descreve uma técnica, denominada DMOD, onde o modelo de simulação é visto como uma 7-tupla, consistindo de conjuntos de eventos, eventos iniciais e algumas relações ordenadas. ABRAMS et al. [91] apresentaram uma técnica, denominada UNITY, com o objetivo de facilitar a verificação formal de especificações de modelos. No entanto, tanto o DMOD, como o UNITY, ainda estão em fases iniciais de desenvolvimento. Uma outra técnica foi desenvolvida por RADIYA e SARGENT [91], procurando obter um fundamento lógico para os modelos de simulação, através do estabelecimento de conceitos fundamentais, de uma lógica modal de eventos discretos e um procedimento de simulação expressos em uma sublógica desta, para sua execução;

3. **Linguagens de especificação**. Como o *Condition Specification*, algumas linguagens de especificação foram propostas. A linguagem de especificação Z foi definida, na década de 70, por pesquisadores da Universidade de Oxford e redefinida, alguns anos mais tarde pela IBM. Para detalhes desta linguagem, ver SPIVEY [89]. O Vienna Development Method (VDM) é um outro exemplo de linguagem de especificação orientada a modelos [JONES, 90]. FRANKOWSKI e FRANTA [89] também desenvolveram uma linguagem de especificação orientada a processo. A grande maioria destas linguagens, no entanto, não possuem a finalidade de ser um meio de comunicação do modelo de simulação, sendo somente apropriadas para documentação e análise. Por exemplo, DOMINGO [91] reporta que o mesmo modelo do *Pub* Inglês, descrito anteriormente, torna-se muito grande quando especificado em linguagem Z (a especificação ocupou quase 20 páginas);
4. **Técnicas baseadas em processos de Markov**. GLYNN [87] sugeriu uma variante do processo de Markov, o GSMP (*Generalized Semi-Markov Process*), como uma base formal no estudo de Sistemas de Eventos Discretos. Para maiores detalhes, ver

DAMERDJI [93]. Embora possa representar qualquer sistema de eventos discretos, o seu formalismo é muito restritivo para suportar descrições de alto nível.

2.2.3 AVALIAÇÃO DAS TÉCNICAS DO PONTO DE VISTA DA REDUÇÃO DE MODELOS DE SIMULAÇÃO

Na seção 2.2.1, enumerou-se alguns pré-requisitos básicos que uma técnica de representação deve ter para permitir sua utilização por não especialistas e facilitar o processo de redução. A técnica deve ser de fácil entendimento e uso (1), indicar objetivos da simulação explicitamente (2) e possuir representação gráfica (3).

A tabela 2.2, desenvolvida por este autor, mostra as principais técnicas de representação de modelos de simulação e uma marca indica se a técnica apresenta o requisito em questão.

tab. 2.2 Comparativo entre as diversas técnicas de representação, do ponto de vista da redução de modelos

	1 ENTENDIMENTO E USO.	2 OBJETIVOS DA SIMULAÇÃO	3 REPRESENTAÇÃO GRÁFICA
ACTIVITY CYCLE DIAGRAMS	√		√
CONTROL FLOW GRAPHS			√
CONDITION SPECIFICATION		√	√(*)
DEVS		√	
EVENT GRAPHS			√
REDES DE PETRI	(**)		√
STATE CHARTS			√
PROCESS NETWORKS	√		√

(*) Embora o Condition Specification seja uma linguagem de especificação este possui grafos derivados (ACAG - *Action Cluster Attribute Graph* e o ACIG - *Action Cluster Incidence Graph*). Por isto, a marca com asterisco

(**) Nem todas as extensões de Redes de Petri são fáceis de entendimento e uso. A Rede de Petri básica, por exemplo, não é facilmente aplicável a modelos de simulação.

Note-se que somente o CS e o DEVS indicam, explicitamente, os objetivos da simulação (medidas de desempenho). O ACD e o PN, ao que tudo indica, parecem ser as técnicas de mais fácil entendimento e aprendizado (este fato é também confirmado por CERIC e PAUL [92]).

Como conclusão geral, nenhuma das técnicas atende, simultaneamente, aos três requisitos, pois algumas são melhores, em determinados aspectos, e outras, melhores em outros.

CAPÍTULO 3
ACTIVITY CYCLE DIAGRAMS / CONDITION SPECIFICATION: UMA ABORDAGEM COMBINADA

3.1 INTRODUÇÃO

No capítulo anterior foi efetuada uma revisão das principais técnicas de representação de modelos de simulação, observando-se, ainda, que um modelo de simulação deveria ser descrito através de uma técnica de representação adequada (a fim de permitir a atuação de um procedimento de redução). Estas técnicas de representação foram avaliadas de acordo com alguns requisitos do ponto de vista da redução, chegando-se a conclusão de que nenhuma destas técnicas os atendem plenamente.

Uma solução para tal problema poderia ser encontrada ao se utilizar uma abordagem mista, através de uma combinação de duas ou mais técnicas de representação. Observou-se que uma técnica baseada na combinação de *Activity Cycle Diagram* e *Condition Specification*, satisfaz os requisitos simultaneamente. Embora cada técnica tenha suas vantagens e desvantagens, uma poderia complementar a outra, facilitando, também, o entendimento do sistema a ser simulado.

A seguir, um melhor comparativo entre as principais vantagens e desvantagens das duas técnicas é apresentado bem como propõe-se um método de conversão de uma técnica para outra.

3.2 COMPARATIVO ENTRE ACD E CS

Como pode ser visto no item 2.2.2.1, as principais vantagens de se representar um modelo através da técnica de ACD são:

1. Simplicidade: um modelo de simulação conceitual pode ser desenvolvido a partir de dois símbolos somente [HLUPIC e PAUL, 94];
2. Habilidade de mostrar, explicitamente, as interações entre os objetos do sistema e seus fluxos;
3. Muito simples de se entender e utilizar. Como foi visto, ele também é utilizado como *front-end* de diversos geradores de programas.

Por outro lado, o ACD apresenta algumas desvantagens:

1. Diagramas se tornam ininteligíveis a medida que aumenta a complexidade do sistema a ser modelado (uma maneira de se contornar este problema é utilizar uma representação hierárquica, condensando ACD's em atividades de um ACD, em um nível superior, ou utilizar diretamente representações hierárquicas derivadas do ACD como o H-ACD);
2. É difícil capturar toda a lógica do modelo no formato ACD, especialmente em se tratando de um modelo com lógica complexa. Por exemplo, o ACD não mostra claramente disciplinas de fila ou atribuições condicionais de valores para atributos de entidades. Segundo PAUL [93], o ACD mostra o fluxo lógico mas não a lógica em profundidade.

Além disto, o ACD não é considerado como uma técnica formal de representação de modelos de simulação, apesar de algumas tentativas de formalizá-la [DOMINGO, 91]. O ACD é uma técnica que pode ser utilizada como base para a implementação de qualquer estratégia de simulação (evento, atividade, processo ou método das três fases), embora PAUL [93] tenha demonstrado que este combina, excepcionalmente bem, com o método das três fases.

O Condition Specification (CS), por sua vez, é um formalismo de especificação de modelos de simulação, que possui algumas vantagens:

1. Facilita o diagnóstico de um modelo de simulação, no que tange a diversas propriedades, medidas ou técnicas aplicadas ao CS [NANCE e OVERSTREET, 87]. Por exemplo, num modelo expresso em CS, pode-se avaliar a inicialização de um atibuto (se todos os requisitos para a atribuição inicial foram válidos), ou verificar a conectividade de um AC (se nenhum AC está isolado);
2. Indica, explicitamente, os objetivos do modelo de simulação expressos pelas medidas de desempenho;
3. Por se tratar de uma linguagem de especificação, o CS é muito útil como forma de documentação do modelo.

Segundo PAGE e NANCE [99], o CS foi desenvolvido com o objetivo de fornecer uma independência da representação do modelo, em relação à estratégia da simulação, que seja suficientemente expressiva para representar qualquer modelo e suficientemente concisa para facilitar o diagnóstico automático da representação. Se os requerimentos de diagnóstico do modelo conflitarem com sua expressividade, a escolha da sintaxe de linguagem dá prioridade ao diagnóstico do modelo. Por este motivo, um modelo expresso em CS pode ser, à primeira vista, ininteligível a um ser humano.

Em se tratando de uma técnica que não possui uma estrutura rígida para a construção do modelo de simulação conceitual (há várias maneiras de se descrever o mesmo modelo de simulação em CS), o analista tem que lidar com muitas entidades e informações, simultaneamente. Nos estudos psicológicos de MILLER [56], foi desenvolvido um teorema denominado *Hrair Limit*, que diz que o ser humano não consegue processar mais que 7±2 entidades simultaneamente. Esta parece ser uma explicação plausível para a facilidade do analista se perder na especificação do modelo de simulação em CS. Ainda, como o CS consiste em uma especificação *bottom-up* e sua sintaxe força uma definição exata dos tipos, geralmente demanda-se tempo e torna-se um

pouco maçante descrever um modelo diretamente em CS. De fato, de acordo com PAGE [95], a sua sintaxe exige, pelo menos, alguns mecanismos de *buffer* entre o CS e o analista.

Sumarizando, um modelo expresso em ACD é fácil de se construir e entender. No entanto não consegue capturar toda a lógica do modelo. Já o CS permite a descrição detalhada de toda a lógica do modelo e contém, explicitamente, os objetivos da simulação, sendo que o seu maior problema é a dificuldade da sua construção direta pelo analista, especialmente por um analista inexperiente. Como foi identificado que uma técnica de redução seria mais vantajosa para um analista inexperiente, tem-se aqui um impasse.

A idéia básica para resolver este impasse é, a partir de uma descrição do modelo de simulação em ACD, converter esta representação em CS de uma forma semi-automática, através da utilização de algumas regras de conversão. Esta conversão não poderia ser totalmente automática, já que, como foi visto, há informações no CS ausentes no ACD (por exemplo, os objetivos da simulação). Efetuando-se a conversão, o modelo a ser reduzido estaria expresso em CS, a partir de uma descrição em ACD, satisfazendo todos os requisitos de uma técnica de representação de modelos de simulação, com relação a redução. As regras de conversão de um modelo representado em ACD para um modelo em CS estão descritas a seguir.

3.3 REGRAS DE CONVERSÃO ACD/CS

As técnicas de ACD e CS foram revistas no capítulo 2, itens 2.2.2.1 e 2.2.2.3, respectivamente. Não há uma ordem exata para a aplicação das regras. Além disto, cada regra deve ser utilizada se cada caso a demandar.

- **Regra de Conversão de Entidade em Objeto**:

Cada tipo entidade no ACD pode ser considerado como um objeto diferente para a Especificação do Objeto. Desta maneira, no exemplo do *Pub Inglês* (itens 2.2.2.1, 2.2.2.6 e 2.2.2.7), têm-se 3 tipos de entidades (garçons, consumidores e copos) e, portanto, 3 objetos no CS.

- **Regra de Manipulação de Fila:**

Para cada vértice do tipo fila, no ACD, de nome Qx, defini-se Qx.size como um inteiro não negativo que mede o número de entidades existentes neste vértice em um dado instante da simulação.

Para o caso de se ter que modelar a seqüência das entidades na fila e políticas de fila, deve-se definir mais três atributos:

1. Qx.tail: Ponteiro inteiro que sempre aponta para o último elemento na fila;
2. Qx.head: Ponteiro inteiro que sempre aponta para o primeiro elemento na fila;
3. Qx.id[position]: *Array* inteiro que fornece o número de identificação ou o id da entidade na fila na Qx, na posição especificada por *position*. Por exemplo Qx.id[Qx.head] fornece o número de identificação do primeiro elemento na fila Qx.

Se a fila obedecer a política FIFO (*First in First Out*), caso chegue uma nova entidade na fila, tem-se:

Qx.tail:=Qx.tail+1.

Analogamente (ainda segundo o critério FIFO), se uma entidade partir da fila[2]:

Qx.head:=Qx.head+1.

A diferença Qx.tail - Qx.head fornece o número de elementos na fila Qx.size.

Analogamente, se a política da fila for diferente de FIFO, os atributos Qx.tail, Qx.head, Qx.size e Qx.id devem ser manipulados de acordo com a estratégia definida.

Se, inicialmente, não há nenhuma entidade na fila Qx, então Qx.size=Qx.head=Qx.tail=0.

Para facilitar a descrição do modelo, embora não faça parte da sintaxe básica do CS, pode-se definir alguns comandos adicionais para manipulação de filas:

1. ENQUEUE(id,queue_name, policy): este comando adiciona uma entidade com número de identificação "id" na fila, de nome "queue_name", segundo a política expressa por "policy". Se a estratégia de fila for FIFO, então ENQUEUE(id,queue,FIFO) é equivalente à seguinte descrição:

   queue.tail:=queue.tail+1;

   queue.size:=queue.size+1;

   queue.id[queue.tail]:=id.

2. DEQUEUE(queue_name): este comando retorna o id do elemento e decrementar o número de entidades na fila. DEQUEUE(queue) é equivalente as seguintes ações:

   queue.head:= queue.head+1;

   queue.size:= queue.size-1.

3. NONEMPTY(queue_name): retorna o valor "verdadeiro" se queue_name.size>0 e "falso", caso contrário.

4. RESET(queue_name): RESET(queue) tem o seguinte efeito:

   queue.head:=0;

   queue.tail:=0;

   queue.size:=0.

---

[2] Aqui adotar-se-á o critério de filas circulares, daí o motivo da variação na cabeceira da fila.

- **Regra de Manipulação de Atividade:**

Adicionalmente, pode ser definido, para cada vértice de atividade de um grafo de ACD, o atributo Ax.size (inteiro não negativo), medindo o número de atividades em progresso no vértice Ax. No exemplo do Pub, se a atividade "Beber" contiver um copo e um consumidor, Abeber.size = 1, pois embora haja duas entidades (o copo e o consumidor), somente uma atividade está se desenrolando. Se um vértice de atividade, em um grafo de ACD, possuir, unicamente, um tipo de entidade, somente neste caso Ax.size é igual ao número de entidades que existe neste vértice (execução paralela de uma mesma atividade).

- **Regra de construção do AC a partir de uma atividade:**

Para cada vértice de atividade no ACD (caso não se tenha a existência de desvios condicionais) têm-se 2 AC's (Action Clusters) na especificação da transição, como se segue:

Sejam Qin1, Qin2,...QinN, os vértices de fila que precedem um vértice de atividade Ax num ACD, e Qout1, Qout2, ...QoutN, os que sucedem o vértice Ax. Para cada vértice de atividade é definido um atributo End_Ax, do tipo sinal de tempo (alarme). Neste caso a avaliação WHEN ALARM(End_Ax) é verdadeira quando o instante que foi definido previamente o disparo deste alarme, através do comando SET ALARM(End_Ax, distribution(time)), coincide com o tempo corrente da simulação e falso caso contrário. O alarme é setado para disparar em um tempo gerado por uma certa distribuição a partir do instante atual. No caso de múltiplos sinais de tempo ou alarmes, End_Ax é um *array* indexado por Ax.size. A descrição dos dois AC's (Início de Atividade e Final de Atividade) estão dispostos a seguir:

```
{Início da Atividade Ax}
    (Qin1_size>0) and (Qin2_size>0) and .... (QinN_size) >0:
        Qin1_size:=Qin1_size-1;
        Qin2_size:=Qin2_size-1;
        ........
        QinN_size:=QinN_size-1;
        Ax_size:=Ax_size+1;
        SET ALARM(End_Ax[Ax_size], distribution(time));

    {Fim da Atividade Ax}
    WHEN ALARM(End_Ax[]):
        Qout1_size:=Qout1_size+1;
        Qout2_size:=Qin2_size+1;
        ........
        QoutN_size:=QoutN_size+1;
        Ax_size:=Ax_size-1;
```

- **Regra de Acompanhamento de Entidades**

Se no modelo há a necessidade de ter atributos de entidades[3], cada entidade no modelo deve receber, individualmente, um número de identificação ou id, sendo que o id varia entre 1 e N, onde N é o número total de entidades no modelo de simulação em um dado instante. Se N for constante, as entidades são ditas permanentes e, se N varia, as entidades são ditas temporárias. Voltando ao exemplo do Pub, se os consumidores entram e saem, estes são entidades temporárias, enquanto que os copos são entidades permanentes, pois não é alterada sua quantidade. Para cada atividade e tipo de entidade é definido um outro atributo ax.id[alarm#,{entity_type}], onde alarm# é o número do alarme. Se somente um tipo de entidade participa da atividade ou há a necessidade de avaliar o atributo de uma só entidade, o tipo da entidade ("entity type") pode ser omitido.

Deste modo, para cada transição no ACD, de fila para atividade e de atividade para fila, os id's devem ser atualizados para:

ax.id[alarm#]:=qx.id[position] (para o caso de transição de fila / atividade);

qx.id[position]:=ax.id[alarm#] (para o caso de transição de atividade / fila).

---

[3] Atributos de entidades são variáveis que acompanham cada entidade durante todos os instantes da simulação, enquanto que atributos são, simplesmente, quaisquer variáveis na descrição em CS.

Pode-se, opcionalmente, utilizar os comandos de manipulação de filas diretamente para a atribuição do id.

- **Regra do Desvio Condicional:**

Havendo "desvios condicionais", na representação em ACD, no mínimo um Action Cluster (AC) na representação em CS deve ser adicionado para cada condição. No caso da figura 3.1, a transição da especificação pode conter os seguintes AC's (AC1,AC2, AC3):

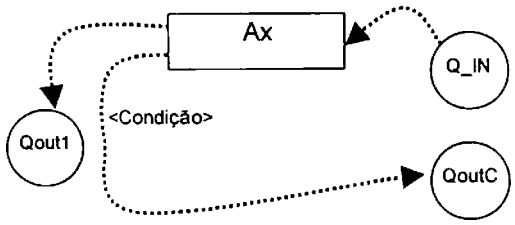


fig. 3.1 Desvio condicional em um ACD

```
AC1: Q_IN>0:
     Ax_size=Ax_size+1;
     <Outras ações>
     SET ALARM(end_Ax[i], distrib[time]);
AC2: WHEN ALARM (End_Ax[]) and (Condição=true):
     QoutC_size:=QoutC_size+1;
     Ax_size=Ax_size-1;
AC3: WHEN ALARM (End_Ax[]) and (Condição=false):
     Qout1_size:=Qout1+1;
     Ax_size:=Ax_size-1;
```

- **Regras Adicionais:**

1. Após a geração dos AC's, a partir das regras de conversão descritas, a geração do AC "termination" é imediata, a partir da condição de parada da instanciação do modelo.
2. AC "inicialization" é construído basicamente: (a) atribuindo os valores iniciais aos atributos determinados pelas regras de conversão, (b) Criando entidades através do comando "CREATE", (c) Lendo os atributos de entrada através do comando "INPUT"

e (d) Inicializando as filas através da atribuição direta de seus parâmetros ou da utilização dos comandos adicionais para manipulação de filas

3. A especificação da interface é construída listando e tipando (atribuindo os respectivos tipos) os atributos de entrada e saída (medidas de desempenho) do modelo. A partir da especificação da transição, a geração da especificação da interface é, portanto, imediata.
4. Para a geração da especificação dos objetos não há uma regra geral. O analista deve definir quais as classes de objetos adotadas (incluindo-se a classe *default* "ambiente") e classificar / tipar todos os atributos, de acordo com um ou outro objeto.
5. No caso do ACD possuir uma atividade e uma fila que se liga, única e exclusivamente, a esta atividade, como ilustra a figura 3.2, ao invés de se criar o atributo Qx.size, pode-se criar um atributo "Status" (eliminado-se Qx.size). Este tipo de construção pode ser aplicado no caso da modelagem de máquinas, por exemplo, onde o atributo "Status" pode assumir o valor "idle" ou "busy". Assim, Qx.size=0 seria equivalente a Status="Busy", enquanto que Qx.size = 1 seria equivalente a Status="Idle". Isto seria indicado para simplificar o caso de várias máquinas, onde a utilização do atributo, derivado diretamente da fila (Qx_size), necessitaria do mecanismo de acompanhamento da entidade, enquanto que, ao utilizar o atributo "Status", basta-se criar um "array", com tamanho igual ao número de máquinas (Status[1..n]).
6. Se uma fila Qx for do tipo *dummy* e Ax_prev, Ax_next forem respectivamente as atividades que a antecede e precede, Qx.size e outros atibutos desta fila podem ser eliminados. Neste caso o decremento de Ax_prev.size deverá causar imediatamente o incremento de Ax_next.size.

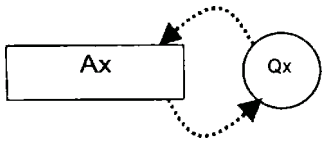


fig. 3.2 Outra forma de conversão de filas, utilizando um atributo "Status"

3.4 EXEMPLO: MODELO DO REPARADOR DE MÁQUINAS

O clássico exemplo do modelo do reparador de máquinas é uma variação do modelo denominado MIP (Machine Inference Problem), originado dos trabalhos de PALM [57] e COX e SMITH [61]. Este exemplo está ilustrado pela figura 3.3.

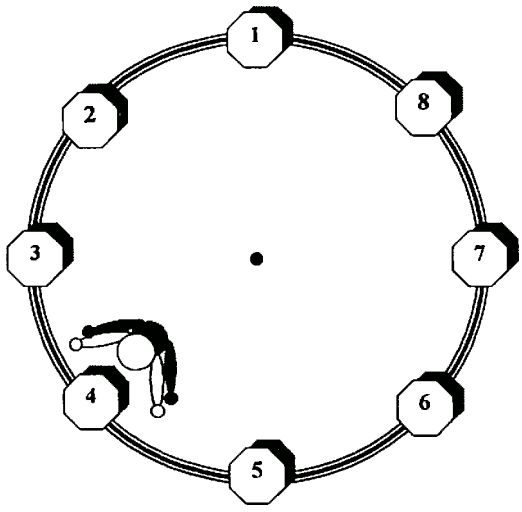


fig. 3.3 Configuração para o modelo do reparador de máquinas para N=8.

Neste exemplo, um único reparador de máquinas repara um grupo de N máquinas semi-automáticas idênticas. Estas máquinas falham, com tempo médio entre falhas (MTBF) exponencialmente distribuído, com média "mean_uptime". O reparador inicia em uma posição de espera e, quando uma ou mais máquinas requerem serviço, o reparador se desloca para uma máquina, de acordo com uma determinada estratégia. O tempo de reparo também segue uma distribuição exponencial (negativa), com média

"mean_repairtime". Após completar o serviço, o reparador vai para a próxima máquina que requer serviço ou vai para uma posição de espera. O objetivo do modelo é medir a porcentagem média do tempo em que as máquinas estão indisponíveis.

3.4.1 VERSÃO 1

Nesta versão, considera-se que o reparador, após consertar uma determinada máquina, fica esperando, no mesmo local, até outra máquina quebrar. O ACD para este caso é ilustrado pela figura 3.4. O tempo que o reparador gasta para se movimentar de uma máquina para outra está uniformemente distribuído entre "min_trv_time" e "max_trv_time". A estratégia de qual máquina se conserta primeiro é aleatória. Este exemplo foi retirado de CHWIF, PAUL e BARRETTO [99].

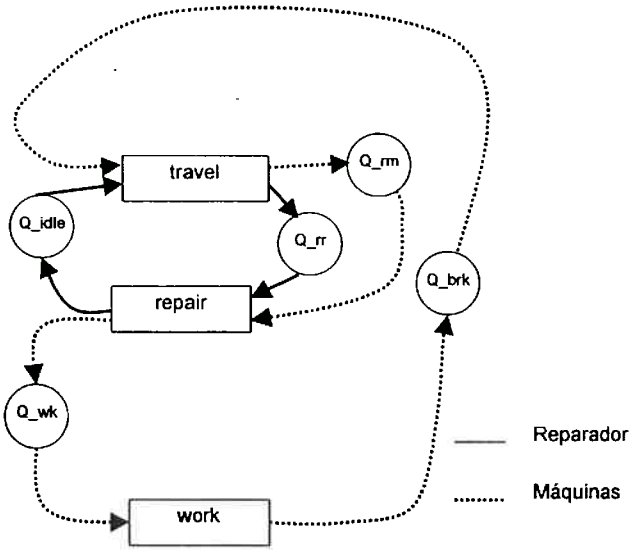


fig. 3.4 ACD para o exemplo do reparador de máquina (versão 1)

É interessante notar alguns pontos:

1. Pela própria característica do problema e das regras do ACD, as filas Q_rr, Q_rm e Q_wk são filas do tipo *dummies*. Isto pode simplificar a descrição do modelo em CS, utilizando a respectiva regra adicional explicitada anteriormente;

2. Embora, fisicamente, as máquinas estejam fixas ao solo e não se movimentem, logicamente, no ACD, elas participam da atividade "Travel" do reparador de máquinas, pois uma condição para que o reparador se dirija a máquina para efetuar o reparo é que ela esteja quebrada (existência de entidades na fila Q_brk).

Utilizando-se as regras descritas no item anterior, a representação do modelo em *Condition Specification* está mostrada a seguir:

Especificação da Interface:

Input:
n {number of machines} : positive integer;
mean_uptime : positive real;
mean_repairtime: positive real;
max_repair {max number of repairs}: positive integer;
min_trv_time: positive real;
max_trv_time: positive real;

Output:
mean_percent_downtime: nonnegative real;
total_uptime[1..n]: nonnegative real;
begin_uptime[1..n]: nonnegative real;

Especificação dos Objetos:

Environment:
system_time: :positive real;
N {number of machines} : positive integer;
mean_uptime : positive real;
mean_repairtime: positive real;
max_repair {max number of repairs}: positive integer;
min_trv_time: positive real;
max_trv_time: positive real;

Repairman:
End_Repair: time-based signal;
End_Travel: time-based signal;
Q_idle.size: nonnegative integer;
Q_rrsize: nonnegative integer;

Machine:
End_Work[1..n]: time-based signal;
Q_brk.size: nonnegative integer;
Q_rm.size: nonnegative integer;
Q_wk.size: nonnegative integer;

```
A_work.size: nonnegative integer;
begin_uptime[1..N]: nonnegative real;

Especificação da Transição:

{initialization}
INITIALIZATION:
        INPUT(N, max_repair, mean_uptime,
                mean_repairtime,min_trv_time,max_trv_time);
        FOR i=1 to N do
        CREATE(Machine[i]);
        END FOR
        CREATE(Repairman);
        num_repair:=0;
        Q_idle.size:=1;
        Q_rr.size:=0;
        Q_brk.size:=0;
        Q_rm.size:=0;
        Q_wk.size:=N;
        A_work.size:=0;

{termination}
numrepair>=max_repair:
        mean_percent_downtime:=1-(total_uptime/N)/system_time);
        STOP;

{wk1}
(Q_wk.size>0):
        Q_wk.size:=Q_wk.size-1;
        A_work.size:=A_work.size+1;
        begin_uptime[A_work.size]:=system_time;
        SET ALARM(End_Work[A_work.size],
                negexp(mean_uptime));

{wk2}
WHEN ALARM(End_Work[i:1..n]):
        total_uptime:= total_uptime + (system_time - begin_uptime[i]);
        Q_brk.size:=Q_brk.size+1;
        A_work.size:=A_work.size-1;

{trv1}
(Q_brk.size>0) and (Q_idle.size>0):
        Q_brk.size:=Q_brk.size-1;
        Q_idle.size:=Q_idle.size-1;
        SET ALARM(End_Travel, uniform(min_trv_time, max_trv_time));

{trv2}
WHEN ALARM(End_Travel):
        Q_rm.size:=Q_rm.size+1;
```

```
        Q_rr.size:=Q_rr.size+1;

{rep1}
(Q_rm.size>0) and (Q_rr.size>0)
        Q_rm.size:=Q_rm.size-1;
        Q_rr.size:=Q_rr.size-1;
        SET ALARM(End_repair, negexp(mean_repair));

{rep2}
WHEN ALARM (End_repair):
        Q_idle.size:=Q_idle.size+1;
        Q_wk.size:=Q_wk.size+1;
        numrepair:=numrepair+1;
```

3.4.2 VERSÃO 2

Nesta versão, mais complexa, o reparador de máquinas, quando nãc há nenhuma máquina a ser reparada, retorna para uma posição de espera localizada no centro do sistema. Tanto para ir desta posição a uma das máquinas, como para retornar ao centro, o reparador demora um tempo constante ("time_to_travel"). Como no caso anterior, o reparador demora um tempo uniformemente distribuído (mínimo de "min_trv_time" e máximo de "max_trv_time") para se movimentar de uma máquina a outra. A estratégia de qual máquina conserta primeiro também é aleatória. O ACD desta versão está ilustrada pela figura 3.5.

Nota-se a existência de dois desvios condicionais:

1. Um na atividade "repair": Quando o reparador termina de reparar a máquina, ele pode mover-se para outra máquina (caso haja, ainda, alguma máquina quebrada), pela atividade "travel0", ou o reparador pode retornar para a posição de repouso (caso não exista nenhuma máquina quebrada), através da atividade "travel2".
2. Um na atividade "work": Quando uma máquina acaba de quebrar, esta pode logicamente ir para a fila "Q_brk1", se o restante das máquinas estiverem funcionando ou ir para a fila "Q_brk2", pois, neste caso, o reparador está reparando outras máquinas.

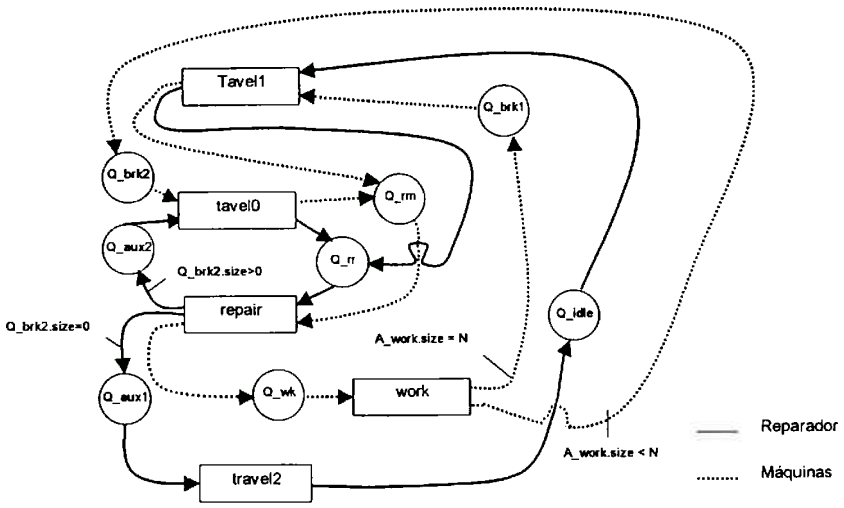


fig. 3.5 ACD para o exemplo do reparador de máquina (versão 2)

Nota-se, também, a existência de 3 atividades de transporte: "Travel0" se refere a movimentação do reparador indo de uma máquina a outra; "Travel1" refere-se a ida do reparador para a posição de uma máquina a partir da posição de repouso (centro do sistema) e "Travel2" corresponde a movimentação do reparador quando este acaba de consertar todas as máquinas paradas e retorna para a posição de repouso. A descrição do modelo em CS, a partir da descrição do mesmo em ACD, está exposto a seguir.

Especificação da Interface:

Input:
N {number of machines} : positive integer;
mean_uptime : positive real;
mean_repairtime: positive real;
max_repair {max number of repairs}: positive integer;
min_trv_time: positive real;
max_trv_time: positive real;
time_to_travel: positive real;

Output:
mean_percent_downtime: nonnegative real;
total_uptime[1..n]: nonnegative real;
begin_uptime[1..n]: nonnegative real;

Especificação dos Objetos:

Environment:
system_time: :positive real;
N {number of machines} : positive integer;

mean_uptime : positive real;
mean_repairtime: positive real;
max_repair {max number of repairs}: positive integer;
min_trv_time: positive real;
max_trv_time: positive real;
time_to_travel: positive real;

Repairman:
End_Repair: time-based signal;
End_Travel0: time-based signal;
End_Travel1: time-based signal;
End_Travel2: time-based signal;
Q_idle.size: nonnegative integer;
Q_rr.size: nonnegative integer;
Q_aux1.size: nonnegative integer;
Q_aux2.size: nonnegative integer;

Machine:
End_Work[1..N]: time-based signal;
Q_brk1.size: nonnegative integer;
Q_brk1.size: nonnegative integer;
Q_rm.size: nonnegative integer;
Q_wk.size: nonnegative integer;
A_work.size: nonnegative integer;
begin_uptime[1..N]: nonnegative real;

Especificação da Transição:

```
{initialization}
INITIALIZATION:
        INPUT(N, max_repair, mean_uptime,
                mean_repairtime,min_trv_time,max_trv_time, time_to_travel);
        FOR i=1 to N do
        CREATE(Machine[i]);
        END FOR
        CREATE(Repairman);
        num_repair:=0;
        Q_idle.size:=1;
        Q_rr.size:=0;
        Q_brk1.size:=0;
        Q_brk2.size:=0;
        Q_rm.size:=0;
        Q_aux1.size:=0;
        Q_aux2.size:=0;
        Q_wk.size:=N;
        A_work.size:=0;

{termination}
numrepair>=max_repair:
```

```
        mean_percent_downtime:=1-(total_uptime/N)/system_time);
        STOP;

{wk1}
Q_wk.size>0:
        Q_wk.size:=Q_wk.size-1;
        A_work.size:=A_work.size+1;
        begin_uptime[A_work.size]:=system_time;
        SET ALARM(End_Work[A_work.size],
                negexp(mean_uptime));

{wk2}
WHEN ALARM(End_Work[i:1..n]) AND (A_work.size=N):
        total_uptime:= total_uptime + (system_time - begin_uptime[i]);
        Q_brk1.size:=Q_brk1.size+1;
        A_work.size:=A_work.size-1;

{wk3}
WHEN ALARM(End_Work[i:1..n]) AND (A_work.size<N):
        total_uptime:= total_uptime + (system_time - begin_uptime[i]);
        Q_brk2.size:=Q_brk2.size+1;
        A_work.size:=A_work.size-1;

{trv1}
(Q_brk1.size>0) and (Q_idle.size>0):
        Q_brk1.size:=Q_brk1.size-1;
        Q_idle.size:=Q_idle.size-1;
        SET ALARM(End_Travel1,time_to_travel);

{trv2}
WHEN ALARM(End_Travel1):
        Q_rm.size:=Q_rm.size+1;
        Q_rr.size:=Q_rr.size+1;

{rep1}
(Q_rm.size>0) and (Q_rr.size>0)
        Q_rm.size:=Q_rm.size-1;
        Q_rr.size:=Q_rr.size-1;
        SET ALARM(End_repair, negexp(mean_repair));

{rep2}
WHEN ALARM (End_repair) AND (Q_brk2.size>0):
        Q_aux2.size:=Q_aux2.size+1;
        Q_wk.size:=Q_wk.size+1;
        numrepair:=numrepair+1;

{rep3}
WHEN ALARM (End_repair) AND (Q_brk2.size=0):
        Q_aux1.size:=Q_aux1.size+1;
```

```
        Q_wk.size:=Q_wk.size+1;
        numrepair:=numrepair+1;

{travel0_1}
(Q_aux2.size>0) AND (Q_brk2>0):
        Q_aux2.size:= Q_aux2.size-1;
        Q_brk2.size:= Q_brk2.size-1;
        SET ALARM(End_Travel0, uniform(min_trv_time, max_trv_time));

{travel0_2}

WHEN ALARM (End_Travel0):
        Q_rm.size:=Q_rm.size+1;
        Q_rr.size:=Q_rr.size+1;

{travel2_1}
(Q_aux1.size>0):
        Q_aux1.size:=Q_aux1.size-1;
        SET ALARM(End_Travel2, time_to_travel);

{travel2_2}
WHEN ALARM (End_Travel2):
        Qidle.size:=Qidle.size+1;
```

É interessante notar que as filas Q_wk, Q_aux1, Q_aux2, Q_rr, Q_rm são filas do tipo *dummy*, pois o tempo de permanência das entidades nelas é nulo. Portanto, as condições que envolvem estas filas no CS poderiam ser omitidas, diminuindo a representação. No entanto, por coerência e para mostrar a analogia com o ACD, foram mantidas todas as condições.

3.4.3 VERSÃO 3

Nesta terceira versão, basicamente a descrição do problema é parecida com da versão 2, inclusive a sua representação em ACD é a mesma (fig. 3.4). Muda-se, neste caso, a estratégia de conserto de uma máquina que quebrou, obedecendo, agora, a uma estratégia FIFO (*First In First Out*), ou seja, a primeira máquina que quebrou é a primeira a ser consertada. O tempo de movimentação do reparador de uma máquina a outra é considerado proporcional à distância entre as máquinas.

Para este caso, há a necessidade de se criar um número de identificação para cada máquina e mais um atributo denominado "last_repaired" (para fins do cálculo da distância entre as máquinas), que contém a identificação da última máquina reparada pelo operador. O tempo de movimentação do reparador é dado por uma função (o CS permite o uso de funções pré-definidas) denominada travel_time(x,y), onde x é o id da máquina de origem e y é o id da máquina destino. Para a manipulação de filas, foram utilizados alguns comandos não originais do CS, como o ENQUEUE e DEQUEUE (ver regra de manipulação de filas). O CS para este caso está descrito abaixo:

Especificação da Interface:

Input:
N {number of machines} : positive integer;
mean_uptime : positive real;
mean_repairtime: positive real;
max_repair {max number of repairs}: positive integer;
min_trv_time: positive real;
max_trv_time: positive real;
time_to_travel: positive real;

Output:
mean_percent_downtime: nonnegative real;
total_uptime[1..n]: nonnegative real;
begin_uptime[1..n]: nonnegative real;

Especificação dos Objetos:

Environment:
system_time: :positive real;
N {number of machines} : positive integer;
mean_uptime : positive real;
mean_repairtime: positive real;
max_repair {max number of repairs}: positive integer;
min_trv_time: positive real;
max_trv_time: positive real;
time_to_travel: positive real;

Repairman:
End_Repair: time-based signal;
End_Travel0: time-based signal;
End_Travel1: time-based signal;
End_Travel2: time-based signal;

Q_idle.size: nonnegative integer;
Q_rr.size: nonnegative integer;
Q_aux1.size: nonnegative integer;
Q_aux2.size: nonnegative integer;
last_repaired:: nonnegative integer;

Machine:
End_Work[1..n]: time-based signal;
Q_brk1.size: nonnegative integer;
Q_brk1.size: nonnegative integer;
Q_rm.size: nonnegative integer;
Q_wk.size: nonnegative integer;
A_work.size: nonnegative integer;
A_work.id: nonnegative integer;
A_travel2.id: nonnegative integer;
A_travel1.id: nonnegative integer;
A_travel0.id: nonnegative integer;
begin_uptime[1..N]: nonnegative real;

Especificação da Transição:

{initialization}
INITIALIZATION:
INPUT(N, max_repair, mean_uptime,
mean_repairtime,min_trv_time,max_trv_time, time_to_travel);
FOR i=1 to N do
CREATE(Machine[i]);
ENQUEUE(i, Q_wk, FIFO);
END FOR
CREATE(Repairman);
num_repair:=0;
Q_idle.size:=1;
Q_rr.size:=0;
Q_brk1.size:=0;
Q_brk2.size:=0;
Q_rm.size:=0;
Q_aux1.size:=0;
Q_aux2.size:=0;
Q_wk.size:=N;
A_work.size:=0;

{termination}
numrepair>=max_repair:
mean_percent_downtime:=1-(total_uptime/n)/system_time);
STOP;

{wk1}
(Q_wk.size>0):

```
        A_work.size:=A_work.size+1;
        A_work.id[A_work.size]:=DEQUEUE(Q_wk);
        begin_uptime[A_work.size]:=system_time;
        SET ALARM(End_Work[A_work.size],
                negexp(mean_uptime));

{wk2}
WHEN ALARM(End_Work[i:1..n]) AND (A_work.size=N):
        total_uptime:= total_uptime + (system_time - begin_uptime[i]);
        ENQUEUE(A_work.id[i], Q_brk1, FIFO);
        A_work.size:=A_work.size-1;

{wk3}
WHEN ALARM(End_Work[i:1..n]) AND (A_work.size<N):
        total_uptime:= total_uptime + (system_time - begin_uptime[i]);
        ENQUEUE(A_work.id[i], Q_brk2, FIFO);
        A_work.size:=A_work.size-1;

{trv1}
(Q_brk1.size>0) and (Q_idle.size>0):
        A_travel1.id:=DEQUEUE[Q_brk1];
        Q_idle.size:=Q_idle.size-1;
        SET ALARM(End_Travel1,time_to_travel);

{trv2}
WHEN ALARM(End_Travel1):
        ENQUEUE[A_travel1.id, Q_rm, FIFO);
        Q_rr.size:=Q_rr.size+1;

{rep1}
(Q_rm.size>0) and (Q_rr.size>0)
        Q_rr.size:=Q_rr.size-1;
        A_repair.id:=DEQUEUE[Q_rm];
        SET ALARM(End_repair, negexp(mean_repair));

{rep2}
WHEN ALARM (End_repair) AND (Q_brk2.size>0):
        Q_aux2.size:=Q_aux2.size+1;
        ENQUEUE(Arepair.id, Qwk, FIFO);
        numrepair:=numrepair+1;
        last_repaired:=A_repair.id;


{rep3}
WHEN ALARM (End_repair) AND (Q_brk2.size=0):
        Q_aux1.size:=Q_aux1.size+1;
        ENQUEUE(Arepair.id, Qwk, FIFO);
        numrepair:=numrepair+1;
        last_repaired:=A_repair.id;
```

```
{travel0_1}
(Q_aux2.size>0) AND (Q_brk2>0):
        Q_aux2.size:= Q_aux2.size-1;
        A_travel0.id:=DEQUEUE(Q_brk2);
        SET ALARM(End_Travel0, travel_time(last_repaired, A_travel0.id));

{travel0_2}

WHEN ALARM (End_Travel0):
        ENQUEUE(A_travel0.id, Q_rm, FIFO);
        Q_rr.size:=Q_rr.size+1;

{travel2_1}
(Q_aux1.size>0):
        Q_aux1.size:=Q_aux1.size-1;
        SET ALARM(End_Travel2, time_to_travel);

{travel2_2}
WHEN ALARM (End_Travel2):
        Qidle.size:=Qidle.size+1;
```

É interessante observar que, embora a representação em ACD não tenha mudado da versão 2 para a versão 3, a lógica do problema mudou, reforçando a afirmação que o ACD não consegue capturar toda a lógica do sistema.

CAPÍTULO 4
TÉCNICA PROPOSTA

4.1 DESCRIÇÃO GERAL

Até o presente momento foram discutidos vários tópicos referentes à redução de modelos e a importância de definir-se uma metodologia ou uma técnica com esta finalidade. No entanto, em nenhum instante foi explicitado, formalmente, o problema de redução, o que se faz aqui.

No capítulo 1, foram identificados basicamente 4 requisitos que uma técnica de redução deveria ter, a fim de diminuir o tempo do ciclo de um estudo de simulação:

1. Deve depender, fundamentalmente, dos objetivos da simulação;
2. Deve ser realizada na fase de concepção do modelo;
3. Deve garantir que o modelo mais simples gerado ainda seja válido;
4. Deve ser passível de ser implementada e automatizada por computador.

Pode-se enumerar, basicamente, duas abordagens para a obtenção de modelos simples: uma denominada **construtiva**, criaria um modelo mais simples diretamente; a outra, denominada **evolutiva** criaria um modelo adequado a partir de um modelo inicial.

Embora uma abordagem construtiva seja desejável, é muito difícil, neste caso, elaborar um procedimento que crie um modelo reduzido de forma automatizada, pois a modelagem é processo criativo.

Adota-se aqui uma abordagem evolutiva, ou seja, pressupõe-se a existência de um modelo inicial. Neste tipo de abordagem, ainda têm-se duas alternativas:

1. Criar um modelo mais complexo e depois simplificá-lo.
2. Produzir um modelo mais simples e depois adicionar detalhes a este.

A partir das discussões realizadas nos capítulos 1 e 2, foi identificado que uma técnica de redução de modelos de simulação seria mais útil ao analista inexperiente, já que ele tende a obter modelos mais complexos do que os experientes. Viu-se que é muito fácil criar modelos complexos e foram encontradas diversas razões para tal. Dados estes fatores, parece que a alternativa de se criar um modelo mais complexo e depois simplificá-lo, é a mais indicada. Portanto esta é a abordagem adotada.

4.2 DEFINIÇÃO DO PROCESSO DE REDUÇÃO.

Com base nos requisitos explicitados e na abordagem adotada, o processo de redução de um modelo de simulação pode ser definido como o processo de obtenção de um modelo mais simples, a partir de um mais complexo, que ainda seja válido, com respeito aos objetivos da simulação.

Formalmente, dado um modelo Mr, representado de acordo com uma determinada técnica de representação R, um conjunto de medidas de desempenho MD (que o modelo Mr é capaz de avaliar), e um conjunto de Hipóteses H sobre o modelo Mr, o problema da redução consiste em determinar um modelo mais simples Mr' que atenda os objetivos incluídos em MD. Mr' pode ser considerado um modelo mais simples do que Mr se e somente se:

MC(Mr') < MC(Mr), onde MC é uma medida de complexidade do modelo de simulação na técnica de representação R.

Ainda, se o modelo Mr for um modelo válido, o modelo Mr' também deve ser. Para tal, as medidas de desempenho avaliadas, utilizando Mr, devem ser estatisticamente equivalentes às do modelo Mr'. Em outras palavras, o modelo Mr' deve preservar o comportamento do modelo Mr no que se refere às medidas de interesse.

Como foi visto, o modelo para ser simplificado necessita estar representado de acordo com alguma técnica de representação de modelos de simulação. Para tal, foi adotada, como técnica de representação o *Condition Specification*, construída a partir do diagrama de ACD, abordada no capítulo anterior.

4.3 ALGORITMO PROPOSTO.

A idéia básica do algoritmo de redução, tendo-se em vista o(s) objetivo(s) da simulação (a partir de uma ou mais medidas de desempenho), é a verificação de quais atributos são capazes de influenciar as medidas de desempenho (MD). Este algoritmo se utiliza dos conceitos de **atributos de entrada**, **atributos de saída** e **atributos de controle**, de um modelo expresso em *Condition Specification*. Estes conceitos foram descritos no item 2.2.2.3.

O algoritmo de redução atua diretamente na especificação da transição, que é a especificação chave que determina como é gerado o comportamento do modelo. A partir da sua redução, a redução da **especificação da interface** bem como da **especificação dos objetos** se dá de uma forma direta. Por isto, concentra-se, aqui, na **especificação da transição**.

Desta maneira, a partir da **especificação da transição** de um modelo de simulação expresso em CS (derivado ou não do ACD), deve-se escrever todos os seus CAP's (*Condition Action Pairs*) em uma tabela, numerando cada CAP em uma ordem ascendente. Esta tabela assemelha-se à tabela 4.1.

A partir da tabela que contém todos os AC's do modelo, deve-se construir uma segunda tabela, denominada de tabela de "retroação". Esta tabela (tab. 4.2) possui o número do CAP na coluna 1, o atributo de interesse na coluna 2 e os atributos que o influenciam diretamente na coluna 3.

tab. 4.1 Especificação da transição em forma tabular

CAP #	NOME DO ACTION CLUSTER	CONDIÇÃO	AÇÕES
10	{initialization}	INITIALIZATION	Qidle.tail:=0
20	{initialization}	INITIALIZATION	Qtidle.head:=0
30	{initialization}	INITIALIZATION	Qtidle.size:=0
40	{initialization}	INITIALIZATION	cranewt[i]:=0
....	.....	.....	...
....	.....	.....	...
250	{fill}	Qtorp.size>1 and Qcrane.size>0 and pitt	Qtorp.head:=Qtorp.head+1
....	.....	.....	...

tab. 4.2 Tabela de retroação

CAP# (coluna 1)	RETROAÇÃO DE: (coluna 2)	INFLUENCIADO POR: (coluna 3)

O procedimento de redução inicia-se pela identificação de um atributo, o qual, na maioria dos casos, é um dos objetivos do modelo de simulação, expressos pelas medidas de desempenho. A partir deste atributo, deseja-se avaliar quais atributos o influenciam. Com base na descrição da especificação da transição na forma tabular, o algoritmo consiste em 2 fases: Fase de Busca (I) e Fase de Retroação (II), descritas a seguir.

I. Fase de Busca:

1. Identificar qual CAP contém o atributo de interesse (procurando em toda a especificação da transição) como um atributo de saída deste CAP. Escrever, na tabela de "retroação", o número deste CAP (coluna 1) e o nome deste atributo, na coluna 2;
2. Escrever, na coluna 3, todos os atributos que poderiam influenciar este atributo, identificado no passo 1, para este CAP específico. Os atributos que podem influenciá-lo consistem nos atributos de entrada e atributos de controle para este CAP;

3. Repetir os passos 1 e 2, até não encontrar mais nenhum CAP, que contenha o atributo de interesse como atributo de saída.

II. Fase de Retroação:

1. Selecionar um atributo que não foi selecionado anteriormente (ou seja, não aparece na coluna 2), a partir dos atributos listados na coluna 3. Se este atributo é um atributo exógeno (como o intervalo entre chegadas, por exemplo), este não pode ser influenciado por nenhum outro atributo, então, neste caso, pode-se marcá-lo com um asterisco "*", e selecionar um outro da coluna 3. Caso contrário, copiar este novo atributo na coluna 2.
2. Repetir a fase I, para este novo atributo.

- **Término do procedimento:**

Se não houver mais atributos na coluna 3 para efetuar a retroação, o algoritmo pára. Desta maneira, todos os CAP's identificados por este procedimento (e, portanto, seus respectivos atributos) são vitais para computar a medida de interesse e logo não devem ser removidos da transição da especificação

- **Obtenção do CS reduzido:**

Todos os CAP's remanescentes que não foram capturados pelo algoritmo e que não contenham nenhum comando do tipo FOR, NEXT, STOP, CREATE / DESTROY, INPUT / OUTPUT, podem ser removidos da especificação da transição, sem comprometer os objetivos da simulação. Para estes CAP's específicos, deve-se proceder conforme o descrito a seguir:

1. STOP: o CAP contendo o comando STOP que, por definição, indica o término da simulação, não deve ser eliminado;
2. FOR / NEXT: os CAP's que contenham os comandos FOR / NEXT podem ser eliminados, se os CAP's internos a este *loop* também o forem;

3. CREATE / DESTROY: os CAP's contendo os comandos CREATE / DESTROY só podem ser retirados da especificação da transição, se a entidade criada ou destruída através destes comandos for eliminada. A priori, estes CAP's podem ser eliminados manualmente pelo analista, a partir da constatação de que todos os atributos referentes a esta entidade não fazem parte da especificação da transição do modelo reduzido. Para uma eliminação automática destes comandos, os atributos devem estar descritos de acordo com o paradigma de orientação a objetos. A título ilustrativo, no modelo do reparador de máquinas mencionado no capítulo anterior, todos os atributos referentes ao reparador devem ser escritos como repairman.<nome do atributo>: repairman.Q_idle.size, repairman.numrepair, etc;
4. INPUT / OUTPUT: estes comandos listam os atributos de entrada (INPUT) e saída (OUTPUT) do modelo de simulação. Caso um atributo seja eliminado pelo algoritmo de retroação e for um argumento de um destes comandos, este deve ser eliminado da lista nos comandos INPUT ou OUTPUT.

Como algumas observações gerais:

1. O algoritmo de retroação pode partir de mais de um atributo, a fim de determinar o modelo mais simples. Neste caso, deve-se inicializar a tabela de retroação com todos estes atributos, simultaneamente;
2. Além das medidas de desempenho, pode-se acrescentar hipóteses adicionais sobre o modelo e fazer com que a algoritmo leve em conta estas hipóteses. Suponha-se que em um dos AC, haja a seguinte condição:

```
Qtorp.size>1 and Qcrane.size>0 and pitt
```

Se, a priori, o analista souber que para qualquer tempo da simulação Qcrane.size>0 (o que equivale a dizer que (Qcrane.size>0) = "true"), esta expressão é equivalente a:

Qtorp.size>1 and pitt

Neste caso, Qcrane.size, que era um atributo de controle para o CAP, não o é mais. Desta maneira, ao introduzir esta alteração na especificação da transição e efetuar a retroação, eliminam-se algumas "ligações", podendo-se ter um potencial maior de redução do modelo. Este mecanismo é utilizado em alguns modelos dos Estudos de Casos, apresentados no capítulo a seguir.

CAPÍTULO 5
ESTUDOS DE CASOS

5.1 INTRODUÇÃO

Este capítulo apresenta diversos estudos de casos, aplicando-se o algoritmo de redução desenvolvido no capítulo anterior. O modelo parte da sua descrição em ACD e mediante as regras vistas no capítulo 3, a descrição do modelo em CS é obtida para permitir a aplicação do algoritmo de redução.

Os casos partirão de níveis de complexidade menor para maior, e procura-se avaliar o desempenho do algoritmo para estes diversos níveis. Para tal, utiliza-se, aqui, uma medida de complexidade que é, simplesmente, o número de CAP's existente na especificação da transição de um modelo expresso em CS.

5.2 ESTUDOS DE CASOS

5.2.1 MÁQUINAS EM SÉRIE

Neste caso, o problema é o mesmo que foi apresentado no capítulo 1 (item 1.4.2), quando comentou-se sobre os objetivos da simulação. Em um "Job-Shop" há 3 máquinas: M1, M2 e M3. As peças chegam, aleatoriamente, ao sistema com taxa de chegada média de "lambda" minutos. Primeiramente são processadas na máquina M1, depois, na máquina M2 e, finalmente, na máquina M3 (sempre nesta seqüência). Após serem processadas na máquina M3, saem do sistema. Cada máquina pode processar uma peça por vez. O objetivo da simulação é determinar a taxa de utilização da máquina M2. Os tempos de processamento das máquinas M1, M2 e M3 seguem uma distribuição normal, com média de m1, m2 e m3 respectivamente e 15% de desvio padrão para cada média.

A figura 5.1 ilustra o modelo enunciado, expresso em ACD.

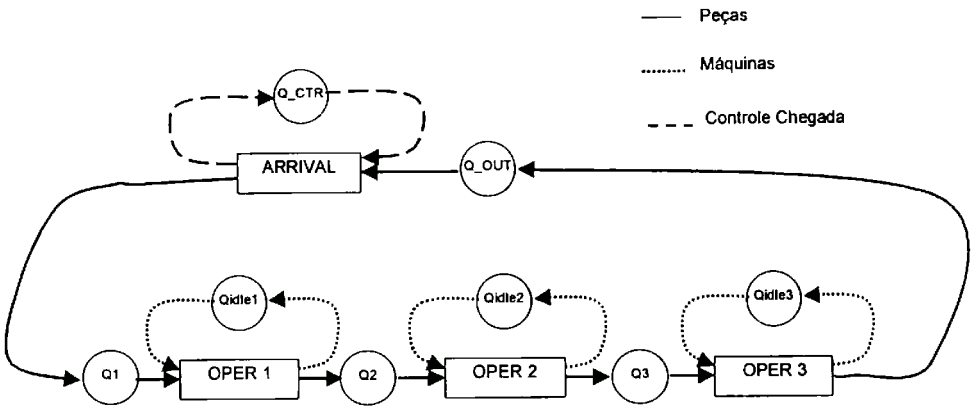


fig. 5.1 ACD completo do problema das máquinas em série

Para construir o modelo em CS, a partir da sua descrição em ACD, pode-se considerar que o mecanismo de porta não é necessário, pois somente um alarme (uma chegada) está previsto por vez. A fila de saída "Q_OUT" também não precisa ser modelada. A especificação da transição para este problema, já expressa de forma tabular pode ser vista na tabela 5.1.

tab. 5.1 Especificação da transição do modelo das máquinas em série.

CAP #	CONDIÇÃO	AÇÃO
1	INITIALIZATION:	Q1.size:=0
2	INITIALIZATION:	Q2.size:=0
3	INITIALIZATION:	Q3.size:=0
4	INITIALIZATION:	QIdle1.size:=1
5	INITIALIZATION:	QIdle2.size:=1
6	INITIALIZATION:	QIdle3.size:=1
7	INITIALIZATION:	SETALARM(End_arrive, negexp(lambda))
8	INITIALIZATION:	utiliz2:=0
9	INITIALIZATION:	produced_rate:=0
10	INITIALIZATION:	num_parts:=0
11	INITIALIZATION:	start_time2:=0
12	INITIALIZATION:	total_time2:=0
13	WHEN ALARM(End_arrive)	Q1.size:=Q1.size+1
14	WHEN ALARM(End_arrive)	SETALARM(End_arrive, negexp(lambda))
15	(QIdle1.size>0) and (Q1.size>0)	QIdle1.size:=QIdle1.size-1
16	(QIdle1.size>0) and (Q1.size>0)	Q1.size:=Q1.size-1
17	(QIdle1.size>0) and (Q1.size>0)	SETALARM(End_mach1, normal(m1,0.15*m1))
18	WHEN ALARM(End_mach1)	QIdle1.size:=QI1.size+1
19	WHEN ALARM(End_mach1)	Q2.size:=Q2.size+1
20	(Qidle2.size>0) and (Q2.size>0)	QIdle2.size:=QIdle2.size-1

21	(Qidle2.size>0) and (Q2.size>0)	Q2.size:=Q2.size-1
22	(Qidle2.size>0) and (Q2.size>0)	start_time:=sys_time
23	(Qidle2.size>0) and (Q2.size>0)	SETALARM(End_mach2, normal(m2,0.15*m2))
24	WHEN ALARM(End_mach2)	Q3.size:=Q3.size+1
25	WHEN ALARM(End_mach2)	Qidle2.size:=Qidle2.size+1
26	WHEN ALARM(End_mach2)	total_time2:=total_time2-(sys_time-start_time)
27	(Qide3.size>0) and (Q3.size>0)	Qidle3.size:=Qidle3.size-1
28	(Qide3.size>0) and (Q3.size>0)	Q3.size:=Q3.size-1
29	(Qide3.size>0) and (Q3.size>0)	SETALARM(End_mach3, normal(m3,0.15*m3))
30	WHEN (End_mach3)	num_parts:=num_parts+1
31	WHEN (End_mach3)	Qidle3.size:=Qidle3.size+1
32	sys_time>1000	produced_rate:=num_parts/sys_time
33	sys_time>1000	utilization2:=total_time2/sys_time
34	sys_time>1000	STOP

Aplica-se o procedimento de redução, considerando que só interessa a utilização da máquina 2 ("Utilization2"). Assim, incia-se a tabela de retroação identificando-se o(s) CAP's que contenham "Utilization2" como atributo de saída. O CAP de número 33 obedece a esta condição, e deve ser considerado na tabela de retroação. Observa-se que, para este CAP, "Utilization2" é influenciado pelos atributos "Total_time2" e "Sys_time" e portanto estes devem ser colocados na coluna 3 da tabela. Ainda, "Utilization2" também aparece como atributo de saída no CAP de número 8, que também deve ser inserido na tabela. Neste caso, porém, "Utilization2" é apenas inicializado, não sendo influenciado por nenhum outro atributo. O procedimento descrito, por sua vez deve ser repetido para o próximo atributo encontrado que é "Total_time2", e assim sucessivamente. O resultado completo do algoritmo de redução está mostrado na tabela 5.2.

tab. 5.2 Tabela de retroação para a taxa de utilização da máquina 2

CAP#	RETROAÇÃO DE:	INFLUENCIADO POR:
33	Utilization2	Total_time2, sys_time
8	Utilization2	-
26	Total_time2	Sys_time, start.time2, End.mach2
12	Total_time2	-
22	Start_time2	Sys_time, Qidle2.size, Q2.size
11	Start_time2	Sys_time, Qidle2.size, Q2.size
23	End_mach2	Qidle2.size, Q2_size, m2*
5	Qidle2_size	Q2.size
25	Qidle2_size	End_mach2
20	Qidle2_size	Qidle2.size, Q2.size
19	Q2.size	End_mach1
2	Q2.size	-

21	Q2.size	Qidle2.size, Q2.size
17	End_mach	Qidle1.size, Q1.size, m1*
4	Qidle_size	-
15	Qidle1.size	Qidle1_size, Q1_size
18	Qidle1.size	End_mach1
1	Q1.size	-
13	Q1.size	End_arrive
16	Q1.size	Qidle1.size, Q1.size
7	End_arrive	Lambda*
14	End_arrive	Qidle1.size, Q1.size, Lambda*

Portanto somente os CAPs de número 1, 4, 5 ,7 ,11 ,12 ,13 ,14 ,15 ,16 ,17 ,18, 20, 21, 22, 23, 25, 26 (determinados pela tabela de retroação) e 34 (referente ao comando STOP) afetam a utilização da máquina 2. Estes geram o seguinte modelo reduzido, mostrado na tabela 5.3.

tab. 5.3 Especificação da transição do modelo reduzido.

CAP #	CONDIÇÃO	AÇÃO
1	INITIALIZATION:	Q1.size:=0
2	INITIALIZATION:	Q2.size:=0
4	INITIALIZATION:	QIdle1.size:=1
5	INITIALIZATION:	QIdle2.size:=1
7	INITIALIZATION:	SETALARM(End_arrive, negexp(lambda))
8	INITIALIZATION:	utiliz2:=0
11	INITIALIZATION:	start_time2:=0
12	INITIALIZATION:	total_time2:=0
13	WHEN ALARM(End_arrive)	Q1.size:=Q1.size+1
14	WHEN ALARM(End_arrive)	SETALARM(End_arrive, negexp(lambda))
15	(QIdle1.size>0) and (Q1.size>0)	QIdle1.size:=QIdle1.size-1
16	(QIdle1.size>0) and (Q1.size>0)	Q1.size:=Q1.size-1
17	(QIdle1.size>0) and (Q1.size>0)	SETALARM(End_mach1,normal(m1,0.15*m1))
18	WHEN ALARM(End_mach1)	QIdle1.size:=QI1.size+1
19	WHEN ALARM(End_mach1)	Q2.size:=Q2.size+1
20	(Qidle2.size>0) and (Q2.size>0)	QIdle2.size:=QIdle2.size-1
21	(Qidle2.size>0) and (Q2.size>0)	Q2.size:=Q2.size-1
22	(Qidle2.size>0) and (Q2.size>0)	start_time:=sys_time
23	(Qidle2.size>0) and (Q2.size>0)	SETALARM(End_mach2,normal(m2,0.15*m2))
25	WHEN ALARM(End_mach2)	QIdle2.size:=QIdle2.size+1
26	WHEN ALARM(End_mach2)	total_time2:=total_time2-(sys_time-start_time)
33	sys_time>1000	utilization2:=total_time2/sys_time
34	sys_time>1000	STOP

O ACD equivalente do modelo reduzido, mostrado em CS, na tabela 5.3, pode ser visto na figura 5.2.

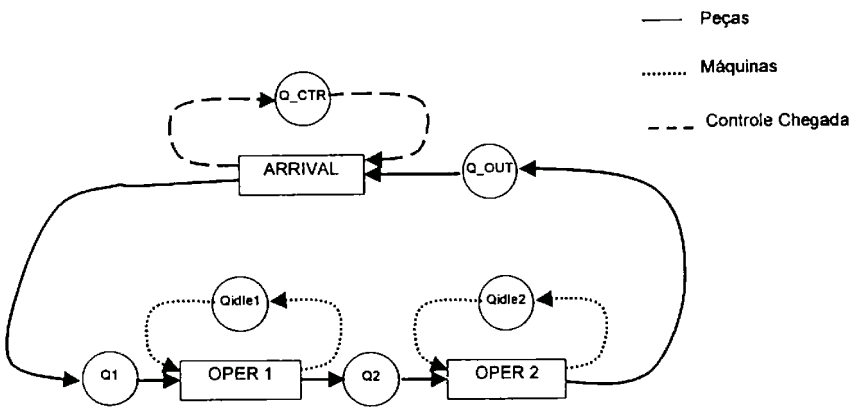


fig. 5.2 ACD reduzido para o modelo das máquinas em série.

Se compararmos o número de CAP's do modelo inicial com o reduzido, encontra-se 23/34, o equivalente à uma redução de 33 %.

Como foi visto, é possível efetuar a redução de um modelo, utilizando-se o algoritmo de retroação. Neste caso, a operação da máquina 3 não afeta a operação da máquina 2 e, o que parece que era intuitivo, foi verificado pela aplicação do algoritmo, reduzindo-se o problema. No entanto, nem sempre uma redução pode ser possível.

Suponha-se que agora há interesse em avaliar, além da taxa de utilização da máquina 2, a produção horária da planta (produced_rate). Se for aplicado o algoritmo de retroação para as duas medidas de desempenho, não se consegue simplificar o modelo inicial, já que as três máquinas influenciam a produção horária. A tabela de retroação, para este caso, pode ser vista no anexo A.

Supondo que novamente só interesse a taxa de utilização da máquina 2. Ao impor-se a restrição de que cada *buffer* ou fila intermediária só pode ter no máximo 20 peças (buffer finito), o CS mostrado pela tabela 5.1 foi modificado e pode ser visto no anexo B. Intuitivamente, neste caso, a máquina 3 pode influenciar a máquina 2, pois agora não se tem mais *buffers* infinitos. Efetuando o algoritmo de retroação, pode-se observar que, neste caso, não se consegue eliminar a máquina 3 (ver anexo B).

5.2.2 SUPERMERCADO

Neste exemplo, consumidores chegam ao supermercado com uma taxa média de chegada de "λ" (exponencialmente distribuído) e gastam um certo tempo fazendo compras, distribuído normalmente, com média "t" e desvio padrão "st". Neste supermercado há duas baias de serviço, com uma fila única, com tempo de serviço também distribuído exponencialmente, com média "μ". Os consumidores que, ao chegarem ao supermercado, virem uma fila de 4 ou mais pessoas esperando no caixa ou sendo atendidas, deixam o sistema. O objetivo do modelo é avaliar a proporção "p" dos consumidores que partem do sistema sem comprar nada e o tempo médio de permanência no sistema. Este exemplo é muito semelhante ao exemplo do *Grocery Store*, proposto por ZEIGLER[76]. A descrição do modelo em ACD está disposta na figura 5.3.

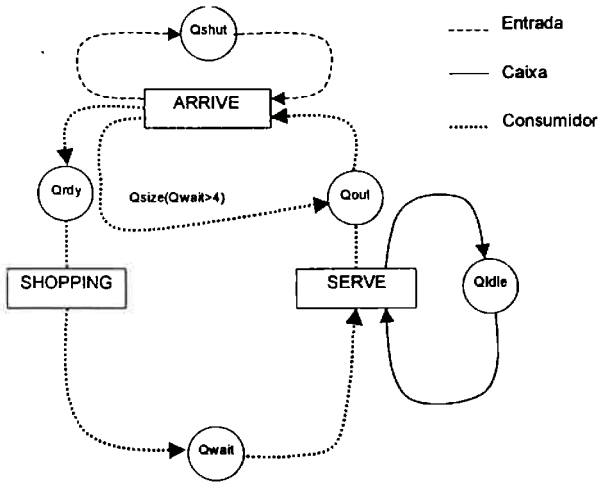


fig. 5.3 ACD para o modelo do supermercado.

Utilizando as regras de conversão, a especificação da transição, na forma tabular obtida, pode ser vista na tabela 5.4. Note se que, neste caso, foi utilizada uma variável de estado para o servidor denominada "status" (idle, busy), ao invés de se utilizar o atributo com a notação de fila (Qidlle). O efeito de aplicar qualquer um dos casos é o mesmo, como foi discutido no final do capítulo 4. Analogamente ao exemplo anterior, não é necessário modelar o mecanismo de porta e da fila de saída.

tab. 5.4 Especificação da transição do modelo do supermercado

CAP #	CONDIÇÃO	AÇÃO
1	INICIALIZATION	INPUT(s, μ, λ, t, st,max_time, m);
2	INICIALIZATION	FOR i=1 to s do
3	INICIALIZATION	CREATE(Server[i]);
4	INICIALIZATION	status[i]:=idle;
5	INICIALIZATION	END FOR
6	INICIALIZATION	Qrdy_size:=0;
7	INICIALIZATION	Qwait_size:=0;
8	INICIALIZATION	Ashopping_size:=0;
9	INICIALIZATION	just_arrive:=false;
10	INICIALIZATION	Wait_head:=0;
11	INICIALIZATION	Wait_tail:=0;
12	INICIALIZATION	Ready_head:=0;
13	INICIALIZATION	Ready_tail:=0;
14	INICIALIZATION	Cust_in:=0;
15	INICIALIZATION	Num_arrival:=0
16	INICIALIZATION	Num_served:=0
17	INICIALIZATION	SET ALARM(End_arrive, negexp(λ));
18	system_time>=max_time	p:=(num_arrivals-num_served)/num_arrivals;
19	system_time>=max_time	av_time:=sum(x:1..nserved, timeout[x]-timein[x])/nserved;
20	system_time>=max_time	STOP
21	WHEN ALARM(End_Arrive):	num_arrivals:=num_arrivals+1
22	WHEN ALARM(End_Arrive):	just_arrive:=true;
23	WHEN ALARM(End_Arrive):	SET ALARM(End_arrive, negexp(λ));
24	just_arrive and Qwait_size=<4	just_arrive:=false;
25	just_arrive and Qwait_size=<4	Qrdy_size:=Qrdy_size+1;
26	just_arrive and Qwait_size=<4	Cust_in:=cust_in+1;
27	just_arrive and Qwait_size=<4	Ready_ID[Ready_tail]:=cust_in
28	just_arrive and Qwait_size=<4	Ready_tail:=Ready_tail+1;
29	just_arrive and Qwait_size=<4	Time_in[cust_in]:=system_time;
30	Qrdy_size>0	Qrdy_size:=Qrdy_size-1;
31	Qrdy_size>0	Ashopping_size:=Ashopping_size+1;
32	Qrdy_size>0	Shopping_ID[Ashopping_size]:=Ready_ID[Ready_head];
33	Qrdy_size>0	Ready_head:=Ready_head+1;
34	Qrdy_size>0	SET ALARM(End_Shopping[Ashopping_size], normal(t, sd));
35	WHEN ALARM(End_Shopping[i:1..m])	Qwait_size:=Qwait_size+1;
36	WHEN ALARM(End_Shopping[i:1..m])	Wait_ID[wait_tail]:=Shopping_ID[i];
37	WHEN ALARM(End_Shopping[i:1..m])	Wait_tail:=Wait_tail+1;
38	WHEN ALARM(End_Shopping[i:1..m])	Ashopping.size:=Ashopping.size-1
39	(Qwait_size)>0 and (FOR SOME 1=<i<=s, status[i]=idle):	Qwait_size:=Qwait_size-1;
40	(Qwait_size)>0 and (FOR SOME 1=<i<=s, status[i]=idle):	Status[i]:=busy
41	(Qwait_size)>0 and (FOR SOME 1=<i<=s, status[i]=idle):	Serve_ID[i]:=Wait_ID[wait_head];
42	(Qwait_size)>0 and (FOR SOME 1=<i<=s, status[i]=idle):	Wait_head:=Wait_head+1;
43	(Qwait_size)>0 and (FOR SOME 1=<i<=s, status[i]=idle):	SET ALARM(End_Serve[i], negexp(μ));
44	WHEN ALARM(End_Serve[i:1..s]):	Num_served:=Num_served+1;
45	WHEN ALARM(End_Serve[i:1..s]):	Time_out[Serve_ID[i]]:=system_time
46	WHEN ALARM(End_Serve[i:1..s]):	Status[i]:=idle

Suponha, agora, que o objetivo do modelo de simulação seja avaliar a proporção dos consumidores que partem do sistema sem comprar nada, em relação ao total dos

consumidores que entraram no sistema ("p"). Ao efetuar o retroação desta medida, obtem-se o resultado expresso na tabela 5.5.

tab. 5.5 Tabela de retroação para a porcentagem de consumidores que partem sem comprar nada

CAP#	RETROAÇÃO DE:	INFLUENCIADO POR:
18	p.	num_arrivals, num_served
21	Num_arrivals	End_arrive
15	Num_arrival	-
44	Num_served	End_serve
16	Num_served	-
23	End_arrive	End_arrive, λ*
17	End_arrive	λ*
43	End_serve	μ*, Qwait_size, Status
35	Qwait_size	End_Shopping
39	Qwait_size	Qwait_size, Status
7	Qwait_size	-
46	Status	End_serve
40	Status	Qwait_size, Status
4	Status	-
34	End_shopping	Ashopping_size, t*, sd*
31	Ashoping_size	Qready_size
8	Ashoping_size	-
25	Qready_size	just_arrive, Qwait_size
30	Qready_size	Qready_size
6	Qready_size	-
22	Just_arrive	End_arrive
24	Just_arrive	Just_arrive, qwait_size
9	Just_arrive	-

A partir da tabela 5.5 e dos CAP's com comandos específicos, o conjunto dos CAP's {1, 2, 3, 4, 5, 6, 7, 8, 9, 15 ,16 ,17 ,18 ,20 ,21 ,22 ,23 ,24, 25, 30, 31, 34, 35, 39, 40, 43, 44,46} constituem o modelo reduzido.

Logo, para avaliar este objetivo (proporção "p" dos consumidores que partem do sistema sem comprar nada), não é necessário conhecer a "identidade" de cada consumidor (não é preciso representar consumidores individualmente) e, para tal, o algoritmo de redução "filtrou" todos os CAP's que são utilizados para efetuar o acompanhamento das identidades dos consumidores no modelo. Logo, a identidade dos consumidores é irrelevante em relação a este objetivo e não necessita ser modelada. Igualmente ao que se fez aqui, ZEIGLER [76], ao perceber este fato no seu modelo do *Grocery Store*, conseguiu criar um modelo reduzido (*Lumped Model*) do modelo inicial, não levando em

conta exatamente a identidade dos consumidores. No entanto, ZEIGLER não utilizou nenhuma metodologia de redução mais formal.

Neste caso, se compararmos a complexidade do modelo inicial em relação ao modelo final, encontraremos uma proporção de 28/46, o equivalente a uma redução de 40%. É interessante notar que não houve mudança na representação do modelo em ACD para este caso, pois o ACD não consegue representar a lógica da identificação de entidades.

Se, por outro lado, for feita a retroação do tempo médio que os consumidores gastam no sistema ("av_time"), o modelo inicial quase não pode ser reduzido, como pode ser visto no anexo C. Os CAP's resultantes são o conjunto {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 16, 17, 19, 20, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 43, 44, 45, 46}, fornecendo uma redução de apenas 5% (42/46).

5.2.3 *PUB* INGLÊS

Neste modelo, já descrito no capítulo 2, os consumidores chegam no Pub e aguardam os garçons servirem as bebidas. Após terminar de beber, podem beber mais ou sair do *Pub*. A taxa de chegada obedece uma distribuição exponencial (negativa), com média "arr_time". Na sua chegada, os consumidores decidem quantas bebidas irão beber, que é um valor amostrado uniformemente entre dois valores "x" e "y" (incluindo estes). O tempo de serviço obedece uma distibuição normal, com média "mean_s" e desvio padrão "sd_s". O tempo de consumo da bebida está uniformemente distribuído entre "a" e "b". A atividade de lavar copos demora um tempo constante, "wash_time". O atributo da entidade do consumidor "need" é decrementado da unidade a cada bebida que este toma. Se o valor deste atributo for "0", os consumidores saem do *Pub*. O ACD para este problema está mostrado na figura 5.4

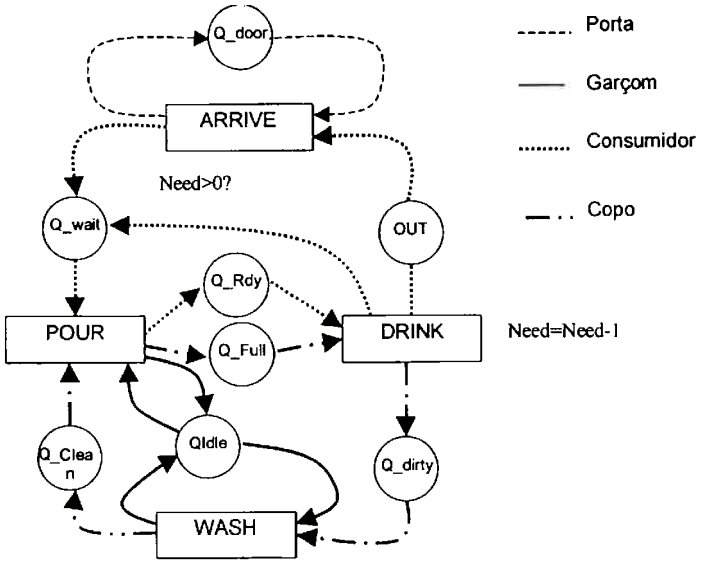


fig. 5.4 ACD para o modelo do Pub *Inglês*

Pode-se avaliar, por exemplo, as seguintes medidas de desempenho:

1. Utilização do garçon ("bu");
2. Porcentagem do tempo do garçon dedicado a servir a bebida ("pp");
3. Porcentagem do tempo do garçon dedicado a lavagem de copos ("pw");
4. Tempo médio de espera do consumidor na fila ("avgwt");
5. Disponibilidade de copos ("gav").

A partir da descrição em ACD, foi elaborado o modelo em CS, cuja transição da especificação está disposta na tabela 5.6. Neste caso, o mecanismo de "porta" no ACD e a fila de saída não foram modelados no CS. Percebendo-se que as filas Q_rdy e Q_full são filas *dumies*, a transição de entidades no CS vai diretamente da atividade "Pour" para a atividade "Drink". O AC "Initialization", por razões de espaço, foi omitido, pois este AC somente é responsável pela inicialização das variáveis e não determina a dinâmica do modelo. Além do que, o comportamento do algoritmo de retroação no AC "initialization" é análogo a qualquer outro AC, e sua atuação neste já foi demonstrada nos itens anteriores.

tab. 5.6 Especificação da transição do modelo do Pub *Inglês*

cap #	CONDIÇÃO	AÇÃO
1	Return	qwait.tail:=qwait.tail+1
2	Return	qwait.id[qwait.tail]:=Return.Id
3	Return	Return:=false
4	Return	Qwait.size:=Qwait.size+1
5	WHEN ALARM(End_Arrive)	num_cust:=num_cust+1
6	WHEN ALARM(End_Arrive)	Need[num_cust]:=uniform[x,y]
7	WHEN ALARM(End_Arrive)	time_in[num_cust]:=syst_time
8	WHEN ALARM(End_Arrive)	qwait.tail:=qwait.tail+1
9	WHEN ALARM(End_Arrive)	qwait.id[qwait.tail]:=num_cust
10	WHEN ALARM(End_Arrive)	Qwait.size:=Qwait.size+1
11	WHEN ALARM(End_Arrive)	SET ALARM(End_arrive, negexp(arr_time))
12	sys_time>max_time	bu:=(time_pour+time_wash)/syst_time
13	sys_time>max_time	pp:=time_pour/sys_time
14	sys_time>max_time	pw:=time_wash/sys_time
15	sys_time>max_time	gav:=1- ind_time/syst_time
16	sys_time>max_time	avgwt:=AVW/count
17	(Qwait.size>0) and (Qidle_Size>0) and (Qclean_size>0)	count:=count+1
18	(Qwait.size>0) and (Qidle_Size>0) and (Qclean_size>0)	Qclean.size:=Qclean.size-1
19	(Qwait.size>0) and (Qidle_Size>0) and (Qclean_size>0)	Qidle.size:=Qidle.size-1
20	(Qwait.size>0) and (Qidle_Size>0) and (Qclean_size>0)	Apour.size:=Apour.size+1
21	(Qwait.size>0) and (Qidle_Size>0) and (Qclean_size>0)	Apour.ID[Apour.size]:=Qwait.Id[Qwait.head]
22	(Qwait.size>0) and (Qidle_Size>0) and (Qclean_size>0)	Qwait.head:=Qwait.head+1
23	(Qwait.size>0) and (Qidle_Size>0) and (Qclean_size>0)	AVW:=AVW+syst_time - time_in[Apour.ID[Apour.size]]
24	(Qwait.size>0) and (Qidle_Size>0) and (Qclean_size>0)	time_1:=system_time
25	(Qwait.size>0) and (Qidle_Size>0) and (Qclean_size>0)	SET ALARM(End_pour[Apour.size], normal(mean_s,sd_s))
26	(Qwait.size>0) and (Qidle_Size>0) and (Qclean_size>0)	Qwait.size:=Qwait.size-1
27	WHEN ALARM(End_Pour[i])	Adrink.size:=Adrink_size+1
28	WHEN ALARM(End_Pour[i])	Adrink.ID[Adrink_size]:=Apour_ID[i]
29	WHEN ALARM(End_Pour[i])	Apour.size:=Apour.size-1
30	WHEN ALARM(End_Pour[i])	SET ALARM(End_drink[Adrink.size], uniform(a,b))
31	WHEN ALARM(End_Pour[i])	Qidle.size:=Qidle.size+1
32	WHEN ALARM(End_Pour[i])	Time_pour:=time_pour+system_time-time_1
33	WHEN ALARM (End_Drk[i])	Qdirty.size:=Qdirty_size+1
34	WHEN ALARM (End_Drk[i])	need[adrink.ID[i]]:=need[adrink.ID[i]]-1;
35	WHEN ALARM (End_Drk[i])	leaving[Adrink_size[i]]:=true
36	WHEN ALARM (End_Drk[i])	Adrink.size:=Adrink_size-1
37	For some i: leaving[i] and need[i]=0	leaving[i]:=false
38	For some i: leaving[i] and need[i]>0	Qwait.tail:=Qwait.tail+1
39	For some i: leaving[i] and need[i]>0	Qwait.ID[Qwait.tail]:=i.
40	For some i: leaving[i] and need[i]>0	leaving[i]:=false
41	For some i: leaving[i] and need[i]>0	Return:=true
42	For some i: leaving[i] and need[i]>0	Return.ID:=I
43	Qdirty.size>0 and Qidle_size>0	time2:=sys_time

44	Qdirty.size>0 and Qidle_size>0	Qdirty.size:=Qdirty_size-1
45	Qdirty.size>0 and Qidle_size>0	Qidle.size:=Qidle_size-1
46	Qdirty.size>0 and Qidle_size>0	SET ALARM(End_wach, wash_time)
47	WHEN ALARM (End_Wash)	time_wash:=time_wash+system_time-time2
48	WHEN ALARM (End_Wash)	Qidle.size:=Qidle.size+1
49	WHEN ALARM (End_Wash)	Qclean.size:=Qclean.size+1
50	Qclean.size=0	time3:=sys_time
51	Qclean.size=0	zero:=true
52	Qclean.size=0 and zero	ind_time=ind_time+sys_time-time3
53	Qclean.size=0 and zero	zero:=false

Suponha que, tendo este modelo, o analista esteja somente interessado em determinar o tempo médio dos consumidores na fila. Se for aplicado o algoritmo de retroação para este caso, os CAP's de interesse são: {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 16, 17, 18, 19, 20, 21, 22, 23, 25, 26, 27, 28, 29, 30, 31, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 45, 46, 48, 49} . O resultado do algoritmo de redução para este caso está mostrado na tabela 5.7.

tab. 5.7 Tabela de retroação para o tempo médio dos consumidores na fila

CAP#	RETROAÇÃO DE:	INFLUENCIADO POR:
16	Avgwt.	Avw, count
23	Avw	Syst_time, time_in, Apour_ID, Apour_size
10	Qwait_size	End_arrive
4	Qwait_size	Return
26	Qwait_size	Qwait_size, Qidle_size, Qclean_size
3	Return	-
41	Return	Leaving, need
35	Leaving	Adrk_size, End_drk
37	Leaving	Need
11	End_arrive	Arr_time*
19	Qidle_size	Qwait_size, Qidle_size, Qclean_size
31	Qidle_size	End_pour
45	Qidle_size	Qdirty_size
48	Qidle_size	End_wash
18	Qclean_size	Qwait_size, Qidle_size,
49	Qclean_size	End_wash
6	Need	End_drk, Adrink_ID
34	Need	End_drk; Adrink_ID
5	Num_cust	
25	End_pour	Apour_size, mean_s*, sd_s*
33	Qdirty_size	End_drk; Qidle_size
30	End_drk	Adrink_size, a*, b*
46	End_wash	Qdirty_size, Qidle_size, wash_time*
17	Count	Qwait_size, Qidle_size, Qclean_size
7	Time_in	Num_cust, End_arrive
21	Apour_ID	Apour_size, Qwait_ID, Qwait_head
20	Apour_size	Qwait_size, Qidle_size, Qclean_size
29	Apour_size	End_pour
28	Adrink.id	Adrink_size, Apour_ID, End_pour
27	Adrk_size	End_pour
36	Adrk_size	End_drk
2	Qwait.ID	Qwait_tail, Return_OD, Return

9	Qwait.ID	Qwait_tail, Num_Cust
39	Qwait_ID	Qwait_tail , leaving, need
22	Qwait_head	Qwait_head, Qwait_size, Qidle_size,
38	Qwait_tail	Leaving, need.
8	Qwait_tail	Leaving, end_need
1	Qwait_tail	Return
40	Laeving	Leaving, need
42	ReturnID	Leaving, leaving, need

Desta maneira, todos os CAP's que não foram encontrados no algoritmo de redução acima, não interferem no tempo médio de espera dos consumidores. Suponha-se, agora, a existência de uma hipótese adicional no modelo: os copos estão sempre disponíveis na fila Qclean (o que quer dizer que o número de copos não é uma restrição ao modelo). Traduzindo esta hipótese num formato de condição tem-se:

Copos sempre disponíveis → (Qclean_size>0) = true,

(0<systime<maxtime)

Com esta hipótese, é possível eliminar a condição apontada dos CAP's 17 e 26 (tabela 5.6) já que esta é sempre verdadeira para qualquer tempo da simulação. Se for efetuada uma nova retroação, verifica-se que os CAP's de número 18 e 49 não interferem nesta condição. Logo, os CAP's {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 16, 17, 19, 20, 21, 22, 23, 25, 26, 27, 28, 29, 30, 31, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 45, 46, 48} são todos os que interferem na medida de interesse, supondo a hipótese acima. Neste caso, conseguiu-se uma relação de redução de 38/54, ou seja, 29 % de redução em relação ao modelo inicial (não se computando o AC "Inicialization"). A utilização da hipótese simplificadora não resultou em ganhos substanciais neste modelo, no entanto, possui papel fundamental na redução do exemplo mostrado a seguir.

5.2.4 USINA

Este modelo representa a operação de uma usina de fundição de aço e foi baseado em um problema real. Nesta usina, há dois alto fornos, que derretem um certo volume

diário de ferro, despejam e enchem quantos torpedos estiverem disponíveis (torpedo é o nome dado a um carrinho especializado no transporte de ferro fundido, que anda sobre trilhos). Se nenhum torpedo estiver no momento exato do tombamento de um alto forno, o ferro fundido é despejado no chão e uma perda é produzida. Cada torpedo pode armazenar uma quantidade de ferro fundido (fundente), desde que não se ultrapasse seu limite de armazenagem. Todos os torpedos, contendo o fundente, vão para uma doca (pit), onde recipientes, movimentados por guindastes, são enchidos com o fundente contidos nos torpedos, um de cada vez. O recipiente pode conter, no máximo 100 toneladas de ferro fundido, que é o volume exato do forno de preparação do aço, que é alimentado por intermédio do guindaste. Há um certo número de fornos de preparação, que trabalham simultaneamente e que produzem o produto final da usina. A figura 5.5 mostra como é esquematicamente o seu layout.

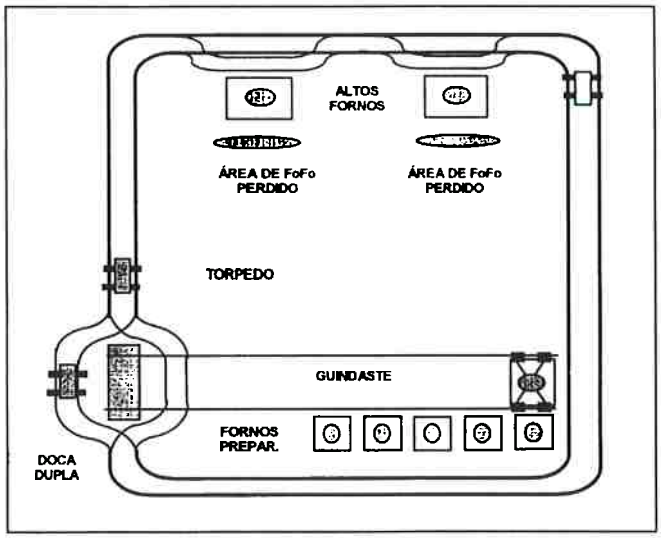


fig. 5.5 Layout esquemático (não em escala) da usina de fundição de aço.

Os dados necessários para a construção do modelo estão dispostos a seguir:

1. Altos fornos: são dois, ao todo. A massa do fundente produzido (em toneladas) para cada forno, segue uma distribuição normal, com média 380, e desvio padrão de 50. O

tempo entre cada descarga (excluindo 10 minutos de tempo de descarregamento) segue uma distribuição normal, com média de 110, e desvio padrão de 15 minutos;

2. Fornos de preparação (aço): cada forno contém exatamente 100 toneladas de aço. O tempo entre cada carga (incluindo o tempo de descarregamento, mas excluindo o tempo de carregamento) segue uma distribuição dada por uma constante de 50 minutos mais um tempo distribuído, exponencialmente, com média de 10 minutos;
3. Guindastes: o guindaste se move ao longo de um trilho e carrega um recipiente que pode armazenar até 100 toneladas de ferro fundido. Este recipiente é abastecido na doca por um torpedo de cada vez. A doca possui duas posições, a fim de evitar qualquer atraso, se um guindaste necessitar de mais de um torpedo para alimentá-lo. Se acontecer que dois torpedos sejam insuficientes para abastecer o recipiente do guindaste, o tempo que se leva para descarregar o segundo torpedo pode ser considerado suficiente para um terceiro torpedo (se existir) entrar no lugar do primeiro, pronto para descarregar. Somente um único guindaste pode estar na doca, de cada vez. O guindaste só sai da doca se estiver cheio com 100 toneladas. Como uma hipótese simplificadora, considere que o trilho suspenso do guindaste foi construído de tal forma a permitir que um guindaste passe sobre o outro, no caso de haver mais de um. Leva-se 5 minutos para um guindaste carregar 100 toneladas de ferro fundido para um forno de preparação de aço que está vazio. O guindaste não pode antecipar qual será o próximo forno a se descarregar, para tal deve aguardar na doca até um forno de preparação estar disponível para ser carregado. Leva-se dois minutos para o reservatório vazio preso ao guindaste retornar para a doca;
4. Torpedos: cada torpedo pode carregar até 300 toneladas de ferro fundido. Quando o alto forno esvaziou seu conteúdo no mínimo número de torpedos necessários (se disponíveis), todos os torpedos se dirigem à doca. Isto inclui torpedos parcialmente

cheios. O torpedo demora 10 minutos, em média, de acordo com uma distribuição de Poisson, para ir do alto forno até a doca. Estes só partem da doca quando seu conteúdo é totalmente esvaziado. O trajeto de retorno (doca para alto forno) leva 4 minutos (constante).

O objetivo maior deste modelo é verificar a quantidade total de ferro fundida perdida durante o processo "totwaste" (gerada se não há torpedo para carregar o ferro fundido do alto forno). O ACD deste modelo está mostrado na figura 5.6.

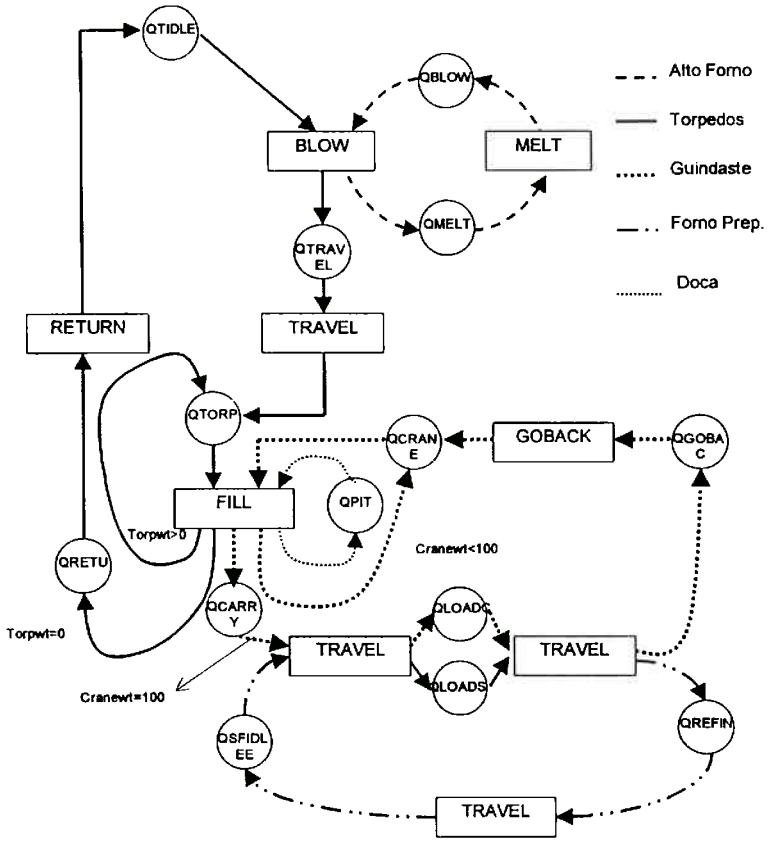


fig. 5.6 ACD para o modelo da usina

Por se tratar de um modelo extenso (esta especificação contém 221 CAP's), sua especificação da transição pode ser vista no anexo D.

Se for considerada a hipótese de que o guindaste esteja sempre disponível, ou seja, em termos de atributos do CS, Qcrane.size >0, para qualquer tempo de simulação, esta condição pode ser removida dos seus respectivos CAP's. Ao efetuar isto e aplicar o

algoritmo de retroação, obtem-se o modelo reduzido. A tabela de retroação e o modelo reduzido em CS obtido podem ser vistos, também, no anexo D. O ACD do modelo reduzido se assemelharia ao da figura 5.7

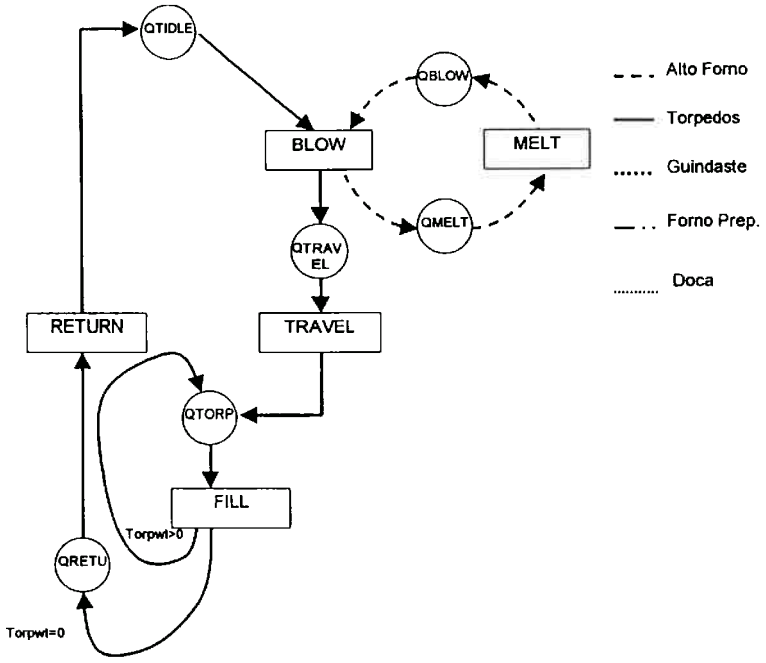


fig. 5.7 ACD reduzido para o modelo da usina.

Neste caso, o modelo reduzido possui 123 CAP's contra 221 do modelo inicial, o que fornece uma redução de 44%.

CAPÍTULO 6
CONCLUSÕES

6.1 CONCLUSÕES

A simulação de eventos discretos possui inúmeras vantagens. No entanto, sabe-se que esta é uma técnica que demanda um tempo maior quando comparado com outras técnicas (modelos analíticos, cálculos baseados na média, etc). Aliado a isto, a criação de modelos complexos é uma constante na prática, contrariamente ao que prega a comunidade de simulação. Embora o assunto redução de modelos de simulação seja de grande importância na geração de modelos mais simples e consequentemente, na diminuição do tempo de um estudo de simulação, o mesmo foi deixado um pouco de lado pelos pesquisadores, somente sendo comentado na superficialidade. Este fato foi evidenciado pela falta de trabalhos específicos que se encontrou publicados na área.

Este trabalho é uma das primeiras tentativas de buscar uma metodologia formal ao problema da redução de modelos de simulação, na sua fase de concepção.

A partir dos estudos de casos realizados, foi demonstrado que é possível efetuar a redução de um modelo de simulação, utilizando uma técnica de redução. A técnica de redução desenvolvida parte dos objetivos da simulação (e/ou de algumas hipóteses adicionais) e determina todas as suas variáveis de interesse, eliminando parte do modelo que não afeta os objetivos em questão. Pela realização dos estudos de casos, pode-se concluir:

1. O potencial de redução de um modelo de simulação existente independe da sua complexidade e sim dos objetivos da simulação e hipóteses efetuadas sobre eles, contrariamente à idéia intuitiva de que um modelo mais complexo tenha maior potencial de simplificação. Isto pode ser comprovado pela redução do modelo simples das

máquinas em série (item 5.2.1). Já para o caso da usina (item 5.2.4), se a hipótese não fosse assumida, o modelo não poderia ser reduzido.

2. Há casos em que o modelo não pode ser reduzido pela técnica proposta, pois todas as suas variáveis afetam a medida de desempenho em questão.

3. Dependendo dos objetivos do modelo / hipóteses, a técnica de redução pode levar a reduções consideráveis do modelo.

A princípio, desta forma não há garantias do que o algoritmo será bem sucedido, levando a reduções consideráveis do modelo. No entanto, a aplicação do algoritmo, a partir da descrição do modelo em CS, é direta, sendo que o maior gasto de tempo do analista reside na sua descrição. Deste maneira, mesmo não sendo possível a redução do modelo, a descrição do modelo de simulação em ACD e CS poderia forçar o analista a "repensar" o problema possibilitando o surgimento reduções, de uma maneira indireta.

6.2 PRINCIPAIS CONTRIBUIÇÕES DO TRABALHO

A principal contribuição do trabalho é, sem sombra de dúvida, o desenvolvimento da técnica de redução aplicada a modelos de sistemas de eventos discretos. No capítulo 1, foram identificados alguns requisitos que esta técnica devem obedecer, baseados nas discussões abordadas. Verifica-se, a seguir, o nível de atendimento da técnica desenvolvida, no que se refere a estes requisitos.

1. **Deve depender, fundamentalmente, dos objetivos da simulação**. Como foi verificado, o algoritmo de redução parte de um ou mais objetivos da simulação e segue as influências causais, de modo a apontar quais variáveis afetam estes objetivos. Logo, o algoritmo depende fundamentalmente destes, atendendo plenamente este requisito;

2. **Deve ser realizado na fase de concepção do modelo**. O algoritmo atua somente sobre o modelo conceitual de simulação, não sendo necessário rodar o modelo e conhecer, previamente, os seus resultados. Este requisito foi o mais difícil de atender, mas é o que promove o maior ganho, porque se o modelo for reduzido na fase de concepção, pode-se perder menos tempo em um estudo de simulação;
3. **Deve garantir que o modelo mais simples gerado seja ainda válido**. Como foi visto, o algoritmo de redução seleciona um sub-modelo do modelo inicial, que atenda aos objetivos da simulação. Nenhuma variável é eliminada, a não ser que não influa causalmente nos objetivos. Deste modo, o algoritmo desenvolvido preserva o comportamento do modelo reduzido, pelo menos no que tange às variáveis de interesse. Assim, se o modelo inicial for válido, pela garantia de preservação do comportamento, o modelo reduzido também será;
4. **Deve ser passível de ser implementado e automatizado no computador**. A partir da existência de um modelo expresso em CS, dos objetivos da simulação e ou de hipóteses adicionais sobre o modelo, é possível sua automação em um computador, por se tratar de passos seqüenciais bem definidos[4].

Portanto, a técnica de redução desenvolvida atende aos requisitos aqui descritos.

Como uma outra contribuição, houve o desenvolvimento das regras de conversão entre ACD e CS. Como foi visto, o CS não é uma técnica de representação para o analista trabalhar diretamente. Já o ACD é uma técnica de fácil entendimento e aplicação, sendo indicada a analistas não experientes; no entanto, não se consegue representar toda a lógica do sistema no ACD. A união destas duas técnicas facilitou ambos os lados, ou seja, a criação do modelo expresso em CS, de uma forma mais fácil, e a possibilidade de

[4] O algoritmo desenvolvido foi aplicado, manualmente, aos casos; mas isso não impede de implementá-lo em um computador.

descrever a lógica, não possível no ACD. Acredita-se que o analista ganhe com esta abordagem, pois o seu entendimento sobre o problema pode ser aumentado.

6.3 SUGESTÕES PARA TRABALHOS FUTUROS

Embora a técnica de redução permita obter o modelo reduzido, de acordo com os requisitos apresentados, ela atua eliminando ou "cortando" parte do modelo de simulação inicial, de acordo com os objetivos. Neste caso, o **escopo** do modelo é mudado, no entanto a sua **profundidade** não é alterada, o que é um outro componente da complexidade. Assim, um trabalho futuro seria o desenvolvimento de uma técnica de redução que atuasse também neste tipo de complexidade, obtendo modelos mais simples não só pelo corte ou eliminação de variáveis mais também pela agregação de variáveis. O grande desafio neste caso seria a obtenção de uma técnica de redução de agregação que ainda mantivesse a validade do modelo.

Outro trabalho futuro seria verificar como daria a implementação de técnicas de redução em sistemas de simulação, através do auxílio automático ao analista, na criação de modelos mais simples. Nos sistemas atuais, inexistem mecanismos de auxílio ao analista para este fim. A única característica presente, atualmente, é a possibilidade da utilização de "templates", onde a complexidade do modelo seria enclausurada numa "caixa preta", podendo ser utilizada como um bloco construtor do modelo com uma lógica e um comportamento pré-definidos. Embora esta abordagem diminua a complexidade "aparente" ou "visual" do modelo, não há nenhum mecanismo de inibição, corte ou agregação de partes do modelo. A idéia de um auxílio automático é "apontar" ao analista partes do modelo que poderiam ser inibidas, eliminadas ou agregadas. Indo nesta linha, um trabalho futuro mais imediato é analisar a aplicação do algoritmo de redução desenvolvido, em um ambiente de simulação gráfico. Neste caso, cada elemento gráfico

poderia estar associadas a descrições expressas em CS, e a eliminação de partes da descrição em CS, através da aplicação do algoritmo de retroação, poderia causar e eliminar (ou inibir) os respectivos elementos gráficos no modelo de simulação.

Outra idéia aplicada aos sistemas de simulação seria permitir que estes trabalhem com múltiplos modelos, segundo o paradigma do "Graph of Models" (GoM) discutido no item 2.1.3. Atualmente, os sistemas de simulação somente permitem a criação de um único modelo completo e não um modelo com várias "facetas". Acredita-se que a arquitetura do GoM, aplicada em sistemas de simulação tenha ganhos, tanto em versatilidade, como em documentação, já que as hipóteses de mudança de um modelo a outro deveriam ser obrigatoriamente explicitadas no sistema.

Como outros trabalhos futuros, não diretamente ligados a redução, poderíamos ter o desenvolvimento de regras de conversão entre outras técnicas de representação, como, por exemplo, Redes de Petri, para o CS. Desta maneira, a técnica de redução desenvolvida aqui poderia ser aplicada a estes modelos, desde que convertidos na representação em CS. Além disto, isto possibilitaria aplicar algumas facilidades que a representação em CS possui, tais como, permitir a análise automática da representação, permitir a execução do modelo, tanto em arquiteturas monoprocessadas, como em arquiteturas paralelas, melhorar a documentação do modelo e uma melhor análise da sua complexidade.

Finalmente espera-se que os resultados obtidos sirvam de incentivo à futuras pesquisas nesta área.

ANEXO A

tab. A.1 Tabela de retroação para o problema das máquinas em série, para a utilização da máquina 2 e para a produção total

CAP#	RETROAÇAO DE:	INFLUENCIADO POR:
33	Utilization2	Total_time2, sys_time
8	Utilization2	-
26	Total_time2	Sys_time, start.time2, End.mach2
12	Total_time2	-
22	Start_time2	Sys_time, Qidle2.size, Q2.size
11	Start_time2	Sys_time, Qidle2.size, Q2.size
23	End_mach2	Qidle2.size, Q2_size, m2*
5	Qidle2_size	Q2.size
25	Qidle2_size	End_mach2
20	Qidle2_size	Qidle2.size, Q2.size
19	Q2.size	End_mach1
2	Q2.size	-
21	Q2.size	Qidle2.size, Q2.size
17	End_mach	Qidle1.size, Q1.size, m1*
4	Qidle_size	-
15	Qidle1.size	Qidle1_size, Q1_size
18	Qidle1.size	End_mach1
1	Q1.size	-
13	Q1.size	End_arrive
16	Q1.size	Qidle1.size, Q1.size
7	End_arrive	-
14	End_arrive	Qidle1.size, Q1.size, Lambda*
32	Produced_rate	Num_parts, sys_time
9	Produced_rate	-
30	Num_parts	End_mach3
29	End_mach3	Qidle3.size, Q3.size, m3*
28	Q3.size	Qidle3.size, Q3.size
24	Q3.size	End_mach2
3	Q3.size	-
6	Qidle.size	-
27	Qidle3.size	Qidle3.size, Q3.size
31	Qidle3.size	End_mach3

ANEXO B

tab. B.1 Especificação da Transição na forma tabular para o problema das máquinas em série com restrição de buffer finito.

CAP #	CONDIÇÃO	AÇÃO
1	INITIALIZATION:	Q1.size:=0
2	INITIALIZATION:	Q2.size:=0
3	INITIALIZATION:	Q3.size:=0
4	INITIALIZATION:	QIdle1.size:=1
5	INITIALIZATION:	QIdle2.size:=1
6	INITIALIZATION:	QIdle3.size:=1
7	INITIALIZATION:	SETALARM(End_arrive, negexp(lambda))
8	INITIALIZATION:	utiliz2:=0
9	INITIALIZATION:	produced_rate:=0
10	INITIALIZATION:	num_parts:=0
11	INITIALIZATION:	start_time2:=0
12	INITIALIZATION:	total_time2:=0
13	WHEN ALARM(End_arrive) and (Q1.size =<20)	Q1.size:=Q1.size+1
14	WHEN ALARM(End_arrive) and (Q1.size =<20)	SETALARM(End_arrive, negexp(lambda))
15	(QIdle1.size>0) and (Q1.size>0)	QIdle1.size:=QIdle1.size-1
16	(QIdle1.size>0) and (Q1.size>0)	Q1.size:=Q1.size-1
17	(QIdle1.size>0) and (Q1.size>0)	SETALARM(End_mach1, normal(m1,0.15*m1))
18	WHEN ALARM(End_mach1) and (Q2.size =<20)	QIdle1.size:=QI1.size+1
19	WHEN ALARM(End_mach1) and (Q2.size =<20)	Q2.size:=Q2.size+1
20	(Qidle2.size>0) and (Q2.size>0)	QIdle2.size:=QIdle2.size-1
21	(Qidle2.size>0) and (Q2.size>0)	Q2.size:=Q2.size-1
22	(Qidle2.size>0) and (Q2.size>0)	start_time:=sys_time
23	(Qidle2.size>0) and (Q2.size>0)	SETALARM(End_mach2, normal(m2,0.15*m2))
24	WHEN ALARM(End_mach2) and (Q3.size =<20)	Q3.size:=Q3.size+1
25	WHEN ALARM(End_mach2) and (Q3.size =<20)	QIdle2.size:=QIdle2.size+1
26	WHEN ALARM(End_mach2) and (Q3.size =<20)	Total_time2:=total_time2-(sys_time-start_time)
27	(QIde3.size>0) and (Q3.size>0)	QIdle3.size:=QIdle3.size-1
28	(QIde3.size>0) and (Q3.size>0)	Q3.size:=Q3.size-1
29	(QIde3.size>0) and (Q3.size>0)	SETALARM(End_mach3, normal(m3,0.15*m3))
30	WHEN (End_mach3)	Num_parts:=num_parts+1
31	WHEN (End_mach3)	Qidle3.size:=QIdle3.size+1
32	sys_time>1000	produced_rate:=num_parts/sys_time
33	sys_time>1000	utilization2:=total_time2/sys_time
34	sys_time>1000	STOP

tab. B.2 Tabela de retroação na forma tabular para o problema das das máquinas em série com restrição de buffer finito.

CAP#	RETROAÇÃO DE:	INFLUENCIADO POR:
33	Utilization2	Total_time2, sys_time
8	Utilization2	-
26	Total_time2	Sys_time, start.time2, End.mach2, Q3.size
12	Total_time2	-
22	Start_time2	Sys_time, Qidle2.size, Q2.size
11	Start_time2	Sys_time, Qidle2.size, Q2.size
23	End_mach2	Qidle2.size, Q2_size, m2*
5	Qidle2_size	Q2.size
25	Qidle2_size	End_mach2
20	Qidle2_size	Qidle2.size, Q2.size
19	Q2.size	End_mach1
2	Q2.size	-
21	Q2.size	Qidle2.size, Q2.size
17	End_mach	Qidle1.size, Q1.size
4	Qidle_size	-
15	Qidle1.size	Qidle1_size, Q1_size
18	Qidle1.size	End_mach1
1	Q1.size	-
13	Q1.size	End_arrive
16	Q1.size	Qidle1.size, Q1.size
7	End_arrive	Lambda*
14	End_arrive	Qidle1.size, Q1.size, Lambda*
28	Q3.size	Qidle3.size, Q3.size.
24	Q3.size	End_mach2
3	Q3.size	-
6	Qidle.size	-
27	Qidle3.size	Qide3.size, Q3.size
31	Qidle3.size	End_mach3
29	End_mach3	Qidle.size, Q3.size, m3*

ANEXO C

tab. C.1 Tabela de retroação do tempo médio que os consumidores gastam no sistema, para o exemplo do supermercado.

CAP#	RETROAÇÃO DE:	INFLUENCIADO POR:
19	Av_time	Num_served, timeout, timein
44	Num_served	End_serve
16	Num_served	-
45	timeout	Serve_ID,
29	timein	Cust_in
40	Serve_ID	Qwait_size, Status, Wait_ID, Wait_head
26	Cust_in	Cust_in, just_arrive, Qwait_size
14	Cust_in	-
36	Wait_ID	Wait_tail, Shopping_ID, End_shopping
41	Wait_head	Qwait_size, Status
10	Wait_head	-
37	Wait_tail	End_Shopping
11	Wait_tail	-
32	Shopping_ID	Ashopping_size, Ready_ID, Ready_head
31	Ashopping_size	Ashopping_size, Qrdy_size
38	Ashopping_size	Ashopping_size, End_shopping
8	Ashopping_size	-
27	Ready_ID	Ready_tail, cust_in, just_arrive, Qwait_size
28	Ready_tail	Ready_tail, just_arrive, Qwait_size
13	Ready_tail	-
33	Ready_head	Qrdy_size, Ready_head
12	Ready_head	-
43	End_serve	μ*, Qwait_size, Status
22	Just_arrive	End_arrive
24	Just_arrive	Just_arrive, qwait_size
9	Just_arrive	-
35	Qwait_size	End_Shopping
39	Qwait_size	Qwait_size, Status
7	Qwait_size	-
46	Status	End_serve
40	Status	Qwait_size, Status
4	Status	-
34	End_shopping	Ashopping_size, t*, sd*
25	Qready_size	just_arrive, Qwait_size
30	Qready_size	Qready_size
6	Qready_size	-
23	End_arrive	End_arrive, λ*
17	End_arrive	λ*

ANEXO D

tab. D.1 Especificação da transição para o modelo da usina (inicial)

CAP #	AC	CONDIÇÃO	AÇÃO
7	{initial}	sys_time=0	Qmelt.tail:=0
8	{initial}	sys_time=0	Qmelt.size:=0
9	{initial}	sys_time=0	Qmelt.head:=0
10	{initial}	sys_time=0	For I:=1 to Nbf do
20	{initial}	sys_time=0	Create (Blast_furnace)
30	{initial}	sys_time=0	Qmelt.tail:=Qmelt.tail+1
40	{initial}	sys_time=0	Qmelt.size:=Qmelt.size+1
50	{initial}	sys_time=0	Qmelt.ID[Qmelt.tail]:=i.
60	{initial}	sys_time=0	caswt[i]:=0
70	{initial}	sys_time=0	End
71	{initial}	sys_time=0	Qcrane.size:=0
72	{initial}	sys_time=0	Qcrane.head:=0
73	{initial}	sys_time=0	Qcrane.tail:=0
80	{initial}	sys_time=0	For I:=1 to Ncr do
90	{initial}	sys_time=0	Create (Crane)
100	{initial}	sys_time=0	Qcrane.tail:=Qcrane.tail+1
110	{initial}	sys_time=0	Qcrane.size:=Qcrane.size+1
120	{initial}	sys_time=0	Qcrane.ID[Qmelt.tail]:=i.
130	{initial}	sys_time=0	cranewt[i]:=0
140	{initial}	sys_time=0	End
141	{initial}	sys_time=0	Qrefine.tail:=0
142	{initial}	sys_time=0	Qrefine.head:=0
150	{initial}	sys_time=0	For I:=1 to Nsf do
160	{initial}	sys_time=0	Create (SteelFurnaces)
170	{initial}	sys_time=0	Qrefine.tail:=Qrefine.tail+1
180	{initial}	sys_time=0	Qrefine.size:=Qrefine.size+1
190	{initial}	sys_time=0	Qrefine.ID[Qmelt.tail]:=i.
200	{initial}	sys_time=0	cranewt[i]:=0
210	{initial}	sys_time=0	End
219	{initial}	sys_time=0	Qidle.tail:=0
220	{initial}	sys_time=0	Qtidle.head:=0
221	{initial}	sys_time=0	Qtidle.size:=0
230	{initial}	sys_time=0	For I:=1 to Ntorp do
240	{initial}	sys_time=0	Create (Torpedoes)
250	{initial}	sys_time=0	Qtidle.tail:=Qtidle.tail+1
260	{initial}	sys_time=0	Qtidle.size:=Qtidle.size+1
270	{initial}	sys_time=0	Qtidle.ID[Qtidle.tail]:=i.
280	{initial}	sys_time=0	torpwt[i]:=0
290	{initial}	sys_time=0	End
300	{initial}	sys_time=0	Qtravel.head:=0
310	{initial}	sys_time=0	Qtravel.tail:=0
320	{initial}	sys_time=0	Qtravel.size:=0
330	{initial}	sys_time=0	Qreturn.head:=0

340	{initial}	sys_time=0	Qreturn.tail:=0
350	{initial}	sys_time=0	Qreturn.size:=0
360	{initial}	sys_time=0	Qtorp.head:=0
370	{initial}	sys_time=0	Qtorp.tail:=0
380	{initial}	sys_time=0	Qtorp.size:=0
420	{initial}	sys_time=0	Qgoback.head:=0
430	{initial}	sys_time=0	Qgoback.tail:=0
440	{initial}	sys_time=0	Qgoback.size:=0
450	{initial}	sys_time=0	Qcarry.size:=0
460	{initial}	sys_time=0	Qcarry.size:=0
470	{initial}	sys_time=0	Qcarry.size:=0
480	{initial}	sys_time=0	QloadC.size:=0
490	{initial}	sys_time=0	QloadC.size:=0
500	{initial}	sys_time=0	QloadC.size:=0
510	{initial}	sys_time=0	QloadS.size:=0
520	{initial}	sys_time=0	QloadS.head:=0
530	{initial}	sys_time=0	QloadS.tail:=0
540	{initial}	sys_time=0	Qsfidle.head:=0
550	{initial}	sys_time=0	Qsfidle.tail:=0
560	{initial}	sys_time=0	Qsfidle.size:=0
561	{initial}	sys_time=0	Qblow.head:=0
562	{initial}	sys_time=0	Qblow.tail:=0
563	{initial}	sys_time=0	Qblow.size:=0
570	{initial}	sys_time=0	Ablow.size:=0
580	{initial}	sys_time=0	Amelt.size:=0
590	{initial}	sys_time=0	Aretum.size:=0
600	{initial}	sys_time=0	Atravel.size:=0
620	{initial}	sys_time=0	Agoback.size:=0
630	{initial}	sys_time=0	Acarry.size:=0
640	{initial}	sys_time=0	Aload.size:=0
650	{initial}	sys_time=0	Arefine.size:=0
660	{initial}	sys_time=0	pitt:=true
670	{initial}	sys_time=0	totalwaste:=0
680	{melt1}	Qmelt.size>0	Amelt.size:=Amelt.size+1
690	{melt1}	Qmelt.size>0	Amelt.ID[Amelt_size]:=Qmelt.ID[Qmelt.head]
700	{melt1}	Qmelt.size>0	Qmelt.head:=Qmelt.head+1
710	{melt1}	Qmelt.size>0	Qmelt.size:=Qmelt.size-1
720	{melt1}	Qmelt.size>0	SET ALARM(End_melt[Amelt_size], normal(110,15))
730	{melt2}	WHEN ALARM(End_melt[i])	Qblow.tail:=Qblow.tail+1
740	{melt2}	WHEN ALARM(End_melt[i])	Qblow.size:=Qblow.size+1
750	{melt2}	WHEN ALARM(End_melt[i])	Qblow.ID[Qblow.tail]:=Amelt_ID[i]
760	{melt2}	WHEN ALARM(End_melt[i])	castwt[Qblow.ID[Qblow.tail]]:=normal[380,50]
770	{melt2}	WHEN ALARM(End_melt[i])	Amelt.size:=Amelt.size-1
780	{blow1}	Qtilde.size=1 and Qblow.size>0	Ablow.ID.torp[1]:=Qtidle.ID[Qtidle.head]
790	{blow1}	Qtilde.size=1 and Qblow.size>0	Qtidle.head:=Qtidle.head+1
800	{blow1}	Qtilde.size=1 and Qblow.size>0	Qtidle.size:=Qtidle.size-1
810	{blow1}	Qtilde.size=1 and Qblow.size>0	Ablow.ID.blastf[1]:=Qblow.ID[Qblow.head]
820	{blow1}	Qtilde.size=1 and Qblow.size>0	Qblow.head:=Qblow.head+1
830	{blow1}	Qtilde.size=1 and Qblow.size>0	Qblow.size:=Qblow.size-1
840	{blow1}	Qtilde.size=1 and Qblow.size>0	torpwt[Ablow.ID.torp[1]]:=castwt[Ablow.ID.blastf[1]]
850	{blow1}	Qtilde.size=1 and Qblow.size>0	totwaste:=totwaste+castwt[Ablow.ID.blastf[1]]-300

860	{blow1}	Qtilde.size=1 and Qblow.size>0	SET ALARM(End_blow1,10)
870	{blow0}	Qtidle=0 and Qblow.size>0	id_aux:=Qblow.ID
880	{blow0}	Qtidle=0 and Qblow.size>0	Qblow.head:=Qblow.head+1
890	{blow0}	Qtidle=0 and Qblow.size>0	Qblow.size:=Qblow.size-1
900	{blow0}	Qtidle=0 and Qblow.size>0	totaste:=totwaste+castwt[id_aux]
910	{blow0}	Qtidle=0 and Qblow.size>0	Qmelt.tail:=Qmelt.tail+1
920	{blow0}	Qtidle=0 and Qblow.size>0	Qmelt.size:=Qmelt.size+1
930	{blow0}	Qtidle=0 and Qblow.size>0	Qmelt.ID[Qmelt.tail]:=id_aux
940	{blow2}	Qtidle.size>1 and Qblow.size>0	Ablow.ID.torp[1]:=Qtidle.ID[Qtidle.head]
950	{blow2}	Qtidle.size>1 and Qblow.size>0	Qtidle.head:=Qtidle.head+1
960	{blow2}	Qtidle.size>1 and Qblow.size>0	Qtidle.size:=Qtidle.size-1
970	{blow2}	Qtidle.size>1 and Qblow.size>0	Ablow.ID.blastf[1]:=Qblow.ID[Qblow.head]
980	{blow2}	Qtidle.size>1 and Qblow.size>0	Qblow.head:=Qblow.head+1
990	{blow2}	Qtidle.size>1 and Qblow.size>0	Qblow.size:=Qblow.size-1
1000	{blow2}	Qtidle.size>1 and Qblow.size>0	torpwt[Ablow.ID.torp[1]]:=300
1010	{blow2}	Qtidle.size>1 and Qblow.size>0	Ablow.ID.torp[2]:=Qtidle.ID[Qtidle.head]
1020	{blow2}	Qtidle.size>1 and Qblow.size>0	Qtidle.head:=Qtidle.head+1
1030	{blow2}	Qtidle.size>1 and Qblow.size>0	Qtidle.size:=Qtidle.size-1
1040	{blow2}	Qtidle.size>1 and Qblow.size>0	torpwt[Ablow.ID.torp[2]]:=castwt[Ablow.ID.blastf[1]]-300
1050	{blow2}	Qtidle.size>1 and Qblow.size>0	SET ALARM(End_Blow2,10)
1060	{blow3}	WHEN ALARM(End_blow1)	Qtravel.tail:=Qtravel.tail+1
1070	{blow3}	WHEN ALARM(End_blow1)	Qtravel.size:=Qtravel.size+1
1080	{blow3}	WHEN ALARM(End_blow1)	Qtravel.ID[Qtravel.tail]:=Ablow.ID.torp[1]
1090	{blow3}	WHEN ALARM(End_blow1)	Qmelt.tail:=Qmelt.tail+1
1100	{blow3}	WHEN ALARM(End_blow1)	Qmelt.size:=Qmelt.size+1
1110	{blow3}	WHEN ALARM(End_blow1)	Qmelt.ID[Qmelt.tail]:=Ablow.ID.blastf[1]
1120	{blow4}	WHEN ALARM(End_blow2)	Qtravel.tail:=Qtravel.tail+1
1130	{blow4}	WHEN ALARM(End_blow2)	Qtravel.size:=Qtravel.size+1
1140	{blow4}	WHEN ALARM(End_blow2)	Qtravel.ID[Qtravel.tail]:=Ablow.ID.torp[1]
1150	{blow4}	WHEN ALARM(End_blow2)	Qtravel.tail:=Qtravel.tail+1
1160	{blow4}	WHEN ALARM(End_blow2)	Qtravel.size:=Qtravel.size+1
1170	{blow4}	WHEN ALARM(End_blow2)	Qtravel.ID[Qtravel.tail]:=Ablow.ID.torp[2]
1180	{blow4}	WHEN ALARM(End_blow2)	Qmelt.tail:=Qmelt.tail+1
1190	{blow4}	WHEN ALARM(End_blow2)	Qmelt.size:=Qmelt.size+1
1200	{blow4}	WHEN ALARM(End_blow2)	Qmelt.ID[Qmelt.tail]:=Amelt.ID.torp[1]
1210	{travel1}	Qtravel.size>0	Atravel.size:=Atravel.size+1
1220	{travel1}	Qtravel.size>0	Atravel.ID[Atravel.size]:=Qtravel.ID[Qtravel.head]
1230	{travel1}	Qtravel.size>0	Qtravel.head:=Qtravel.head+1
1240	{travel1}	Qtravel.size>0	Qtravel.size:=Qtravel.size-1
1250	{travel1}	Qtravel.size>0	SET ALARM(End_travel[Atravel.size],poisson(10))
1260	{travel2}	WHEN ALARM(End_travel[i])	Qtorp.ID[Qtorp.tail]:=Atravel.ID[i]
1270	{travel2}	WHEN ALARM(End_travel[i])	Qtorp.tail:=Qtorp.tail+1
1280	{travel2}	WHEN ALARM(End_travel[i])	Qtorp.size:=Qtorp.size+1
1290	{travel2}	WHEN ALARM(End_travel[i])	Atravel.size:=Atravel.size-1
1300	{return1}	Qreturn.size>0	Areturn.size:=Areturn.size+1
1310	{return1}	Qreturn.size>0	Areturn.ID[Areturn.size]:=Qreturn.ID[Qreturn.head]
1320	{return1}	Qreturn.size>0	Qreturn.head:=Qreturn.head+1
1330	{return1}	Qreturn.size>0	Qreturn.size:=Qreturn.size-1
1340	{return1}	Qreturn.size>0	SET ALARM(End_Return[Areturn.size],4)
1350	{return2}	WHEN ALARM(End_return[i])	Qtidle.tail:=Qtidle.tail+1
1360	{return2}	WHEN ALARM(End_return[i])	Qtidle.size:=Qtidle.size+1

1370	{return2}	WHEN ALARM(End_return[i])	Qtidle.ID[Qtidle.tail]:=Areturn.ID[i]
1380	{return2}	WHEN ALARM(End_return[i])	Areturn.size:=Areturn.size-1
1390	{fill}	Qtorp.size>1 and Qcrane.size>0 and pitt	pitt:=false
1400	{fill}	Qtorp.size>1 and Qcrane.size>0 and pitt	Afill.ID.torp:=Qtorp.ID[Qtorp.head]
1410	{fill}	Qtorp.size>1 and Qcrane.size>0 and pitt	Qtorp.head:=Qtorp.head+1
1420	{fill}	Qtorp.size>1 and Qcrane.size>0 and pitt	Qtorp.size:=Qtorp.size-1
1430	{fill}	Qtorp.size>1 and Qcrane.size>0 and pitt	Afill.ID.crane:=Qcrane.ID[Qcrane.head]
1440	{fill}	Qtorp.size>1 and Qcrane.size>0 and pitt	Qcrane.head:=Qcrane.head+1
1450	{fill}	Qtorp.size>1 and Qcrane.size>0 and pitt	Qcrane.size:=Qcrane.size-1
1459	{fill}	Qtorp.size>1 and Qcrane.size>0 and pitt	transf:=minof(100,torpwt[Afill.ID.torp] /*minimum of x,y*/
1460	{fill}	Qtorp.size>1 and Qcrane.size>0 and pitt	cranewt[Afill.ID.crane]:=cranewt[Afill.ID.crane]+transf
1470	{fill}	Qtorp.size>1 and Qcrane.size>0 and pitt	torpwt[Afill.ID.torp]:=torpwt[Afill.ID.torp]-transf
1480	{fill}	Qtorp.size>1 and Qcrane.size>0 and pitt	pitt:=true
1490	{fill}	Qtorp.size>1 and Qcrane.size>0 and pitt	SET ALARM(End_fill,5)
1500	{endfill1}	When (End_fill[i]) and torpwt[Afill.ID.torp]=0	Qreturn.tail:=Qreturn.tail+1
1510	{endfill1}	When (End_fill[i]) and torpwt[Afill.ID.torp]=0	Qreturn.size:=Qreturn.size+1
1520	{endfill1}	When (End_fill[i]) and torpwt[Afill.ID.torp]=0	Qreturn.ID[Qreturn.tail]:=Afill.ID.torp
1530	{endfill2}	When (End_fill[i]) and torpwt[Afill.ID.torp]>0	Qtorp.head:=Qtorp.head-1
1540	{endfill2}	When (End_fill[i]) and torpwt[Afill.ID.torp]>0	Qtorp.size:=Qtorp.size+1
1550	{endfill2}	When (End_fill[i]) and torpwt[Afill.ID.torp]>0	Qtorp.ID[Qtorp.head]:=Afill.ID.torp
1560	{endfill3}	When (End_fill[i]) and cranewt[Afill.ID.crane]<100	Qcrane.head:=Qcrane.head-1
1570	{endfill3}	When (End_fill[i]) and cranewt[Afill.ID.crane]<100	Qcrane.size:=Qcrane.size+1
1580	{endfill3}	When (End_fill[i]) and cranewt[Afill.ID.crane]<100	Qcrane.ID[Qcrane.head]:=Afill.ID.crane
1590	{endfill4}	When (End_fill[i]) and cranewt[Afill.ID.crane]=100	Qcarry.tail:=Qcarry.tail+1
1600	{endfill4}	When (End_fill[i]) and cranewt[Afill.ID.crane]=100	Qcarry.size:=Qcarry.size+1
1610	{endfill4}	When (End_fill[i]) and cranewt[Afill.ID.crane]=100	Qcarry.ID[Qcarry.tail]:=Afill.ID.crane
1620	{carry1}	Qcarry.size>0 and Qsfidle.size>0	Acarry.size:=Acarry.size+1
1630	{carry1}	Qcarry.size>0 and Qsfidle.size>0	Acarry.ID.crane:=Qcarry.ID[Qcarry.head]
1640	{carry1}	Qcarry.size>0 and Qsfidle.size>0	Qcarry.head:=Qcarry.head+1
1650	{carry1}	Qcarry.size>0 and Qsfidle.size>0	Qcarry.size:=Qcarry.size-1
1660	{carry1}	Qcarry.size>0 and Qsfidle.size>0	Acarry.ID.sf:=Qsfidle.ID[Qsfidle.head]
1670	{carry1}	Qcarry.size>0 and Qsfidle.size>0	Qsfidle.head:=Qsfidle.head+1
1680	{carry1}	Qcarry.size>0 and Qsfidle.size>0	Qsfidle.size:=Qsfidle.size-1
1690	{carry1}	Qcarry.size>0 and Qsfidle.size>0	SET ALARM(End_carry[Acarry_size],5)
1700	{carry2}	WHEN ALARM(End_carry[i])	Qloadc.tail:=qloadc.tail+1
1710	{carry2}	WHEN ALARM(End_carry[i])	Qloadc.size:=qloadc.size+1
1720	{carry2}	WHEN ALARM(End_carry[i])	Qloadc.ID[Qloadc.tail]:=Acarry.ID.crane
1730	{carry2}	WHEN ALARM(End_carry[i])	Qloads.tail:=qloads.tail+1
1740	{carry2}	WHEN ALARM(End_carry[i])	Qloads.size:=Qloads.size+1
1750	{carry2}	WHEN ALARM(End_carry[i])	Qloads.ID[Qloads.tail]:=Acarry.ID.sf

1760	{carry2}	WHEN ALARM(End_carry[i])	Acarry.size:=Acarry.size-1
1770	{load1}	Qloadc.size>0 and Qloads.size>0	Aload.size:=Aload.size+1
1780	{load1}	Qloadc.size>0 and Qloads.size>0	Aload.ID.crane[Aload.size]:=Qloadc.ID[Qloadc.head]
1790	{load1}	Qloadc.size>0 and Qloads.size>0	Qloadc.head:=Qloadc.head+1
1800	{load1}	Qloadc.size>0 and Qloads.size>0	Qloadc.size:=qloadc.size-1
1810	{load1}	Qloadc.size>0 and Qloads.size>0	Aload.ID.sf[Aload.size]:=Qloads.ID[Qloads.tail]
1820	{load1}	Qloadc.size>0 and Qloads.size>0	Qloads.head:=Qloads.head+1
1830	{load1}	Qloadc.size>0 and Qloads.size>0	Qloads.size:=Qloads.size-1
1840	{load1}	Qloadc.size>0 and Qloads.size>0	SET ALARM(End_load[Aload_size],5)
1850	{load2}	WHEN ALARM(End_load[i])	cranewt[Aload.ID.crane[i]]:=0
1860	{load2}	WHEN ALARM(End_load[i])	Qgoback.tail:=Qgoback.tail+1
1870	{load2}	WHEN ALARM(End_load[i])	Qgoback.size:=Qgoback.size+1
1880	{load2}	WHEN ALARM(End_load[i])	Qgoback.ID[Qgoback.tail]:=Aload.ID.crane[i]
1890	{load2}	WHEN ALARM(End_load[i])	Qrefine.tail:=Qrefine.tail+1
1900	{load2}	WHEN ALARM(End_load[i])	Qrefine.size:=Qrefine.size+1
1910	{load2}	WHEN ALARM(End_load[i])	Qrefine.ID[Qrefine.tail]:=Aload.ID.sf[i]
1920	{load2}	WHEN ALARM(End_load[i])	Aload.size:=Aload.size-1
1930	{refine1}	Qrefine.size>0	Arefine.size:=Arefine.size+1
1940	{refine1}	Qrefine.size>0	Arefine.ID[Arefine.size]:=Qrefine.ID[Qrefine.head]
1950	{refine1}	Qrefine.size>0	Qrefine.head:=Qrefine.head+1
1960	{refine1}	Qrefine.size>0	Qrefine.size:=Qrefine.size-1
1970	{refine1}	Qrefine.size>0	SETALARM(End_refine[Arefine_size],50+negexp[10])
1980	{refine2}	WHEN ALARM(End_refine[i])	Qsfidle.tail:=Qsfidle.tail+1
1990	{refine2}	WHEN ALARM(End_refine[i])	Qsfidlde.size:=Qsfidle.size+1
2000	{refine2}	WHEN ALARM(End_refine[i])	Qsfidle.ID[Qsfidle.tail]:=Arefine.ID[i]
2001	{refine2}	WHEN ALARM(End_refine[i])	Arefine.size:=Arefine.size-1
2010	{goback1}	Qgoback.size>0	Agoback.size:=Agoback.size+1
2020	{goback1}	Qgoback.size>0	Agoback.ID[Agoback.size]:=Qgoback.ID[Qgoback.head]
2030	{goback1}	Qgoback.size>0	Qgoback.head:=Qgoback.head+1
2040	{goback1}	Qgoback.size>0	Qgoback.size:=Qgoback.size-1
2050	{goback1}	Qgoback.size>0	SET ALARM(End_goback[Agoback_size],2)
2060	{goback2}	WHEN ALARM(End_goback[i])	Qcrane.tail:=Qcrane.tail+1
2070	{goback2}	WHEN ALARM(End_goback[i])	Qcrane.size:=Qcrane.size+1
2080	{goback2}	WHEN ALARM(End_goback[i])	Qcrane.ID[Qcrane.tail]:=Agoback.ID[i]
2090	{term}	sys_time>1000	STOP

tab. D.2 Tabela de retroação para o modelo da usina:

CAP #	RETROAÇAO DE:	INFLUENCIADO POR:
850	Totwaste	Qtidle.size, Qlow.size, Castwt, Ablow.ID.blasf[1]
670	Totwaste	-
900	Totwaste	Castwt, id_aux, Qtidle.size, Qblow.size
221	Qtidle.size	-
260	Qtidle.size	-
800	Qtidle.size	Qtidle.size, Qblow.size
960	Qtidle.size	Qtidle.size, Qblow.size
1030	Qtidle.size	Qtidle.size, Qblow.size
1360	Qtidle.size	End_return
563	Qblow.size	-
740	Qblow.size	End_melt
830	Qblow.size	Qtidle.size
890	Qblow.size	Qtidle.size, Qblow.size
990	Qblow.size	Qtidle.size, Qblow.size
60	Castwt	-
760	Castwt	Qblow.ID, Qblow.tail, End_melt
810	Ablow.ID.blasf[1]	Qblow.ID, Qblow.head, Qtidle.size, Qblow.size
970	Ablow.ID.blasf[1]	Qblow.ID, Qblow.head, Qtidle.size, Qblow.size
870	Id_aux	Qtidle.size, Qblow.size, Qblow.ID
1340	End_return	Qreturn.size,
720	End_melt	Amelt.sze, Qmelt.size
750	Qblow.ID	Qblow.tail, Amelt.ID
562	Qblow.tail	-
730	Qblow.tail	End_melt
561	Qblow.head	-
820	Qblow.head	Qtidle.size, Qblow.size
880	Qblow.head	Qtidle.size, Qblow.size
980	Qblow.head	Qtidle.size, Qblow.size
350	Qreturn.size	-
1330	Qreturn.size	Qreturn.size
1510	Qreturn.size	End_fill, Torpwt, Afill.ID.torp
590	Areturn.size	-
1300	Areturn.size	Qreturn.size
1380	Areturn.size	End.return
40	Qmelt.size	-
8	Qmelt.size	-
710	Qmelt.size	Qmelt.size
920	Qmelt.size	Qtidle.size, Qblow.size
1100	Qmelt.size	End.blow1
1190	Qmelt.size	End_blow2
580	Amelt.size	-
680	Amelt.size	Qmelt.size
770	Amelt.size	End_melt
690	Amelt.ID	Amelt.size, Qmelt.ID, Qmelt.head, Qmelt.size
1490	End_fill	Qtorp.size, pitt
280	Torpwt	-
840	Torpwt	Qtidle.size, Qblow.size, Ablow.ID.torp[1], castwt, Ablow.ID.blasf[1]
1040	Torpwt	Ablow.ID.torp[2], castwt, Ablow.ID.blasf[1]
1470	Torpwt	Afill.ID.torp, transf
780	Ablow.ID.torp[1]	Qtidle.size, Qblow.size, Qtidle.ID, Qtidle.head
940	Ablow.ID.torp[1]	Qtidle.size, Qblow.size, Qtidle.ID, Qtidle.head
1010	Ablow.ID.torp[1]	Qtidle.size, Qblow.size, Qtidle.ID, Qtidle.head
1400	Afill.ID.torp	Qtorp.size, pitt, Qtorp.ID, Qtorp.head
860	End_blow1	Qtidle.size, Qblow.size
1050	End_blow2	Qtidle.size, Qblow.size
930	Qmelt.ID	Qmelt.tail, id_aux, Qtidle.size, Qblow.size
1000	Torpwt	Ablow.ID.torp[1], Qtidle.size
1110	Qmelt.ID	Qmelt.tail, Amelt.ID.torp[1], Ablow.ID,bastf[1], End_blow1
1200	Qmelt.ID	Qmelt.tail, Amelt.ID.torp[1], End_blow2
50	Qmelt.ID	Qmelt.tail
9	Qmelt.head	-
700	Qmelt.head	Qmelt.size
30	Qmelt.tail	-
910	Qmelt.tail	Qtidle.size, Qblow.size

1090	Qmelt.tail	End_blow1
1180	Qmelt.tail	End_blow2, Qmelt.tail
220	Qtidle.head	-
790	Qtidle.head	Qtidle.size, Qblow.size
950	Qtidle.head	Qtidle.size, Qblow.size
1020	Qtidle.head	Qtidle.size, Qblow.size
250	Qtidle.tail	-
1350	Qtidle.tail	End_Return
270	Qtidle.ID	-
1370	Qtidle.ID	Qtidle.tail, End_retun, Areturn.ID
380	Qtorp.size	-
1280	Qtorp.size	End_travel
1420	Qtorp.size	Qtorp.size, pitt
1540	Qtorp.size	End_fill, Afill.ID.torp, torpwt
1260	Qtorp.ID	Qtorp.tail, Atravel.ID, End_travel
1550	Qtorp.ID	Qtorp.head, Afill.ID.torp, torpwt, End_fill
360	Qtorp.head	-
1410	Qtorp.head	Qtorp.size, pitt
1530	Qtorp.head	End.fill, Afill.ID.torp, Qtorp.head
370	Qtorp.tail	-
1270	Qtorp.tail	End_travel
660	Pitt	-
1390	Pitt	Qtorp.size, pitt
1480	Pitt	Qtorp.size, pitt
1459	Transf	Torpwt, Afill.ID.torp, Qtorp.size, pitt
1250	End_travel	Qtravel.size, Atravel.size
600	Atravel.size	-
1210	Atravel.size	Qtravel.size, Atravel.size
1290	Atravel.size	End_travel
1220	Atravel.ID	Qtravel.size, Qtravel.ID, Qreturn.head, Qreturn.size
1310	Areturn.ID	Areturn.size, Qreturn.ID, Qreturn.head, Qreturn.size
330	Qreturn.head	-
1320	Qreturn.head	Qreturn.size
1520	Qreturn.ID	Qreturn.tail, Afill.ID.torp, End_fill, Torpwt
340	Qreturn.tail	-
1500	Qreturn.tail	End_fill, torpwt, Afill.ID.torp, Qreturn.tail
320	Qtravel.size	-
1070	Qtravel.size	End_blow1
1130	Qtravel.size	End_blow2
1160	Qtravel.size	End_blow2
1240	Qtravel.size	Qtravel.size
1080	Qtravel.ID	Qtravel.tail, Ablow.ID.torp[1], End_blow
1170	Qtravel.ID	Qtravel.ID, Qtravel.tail, Ablow.ID.torp[2]
1140	Qtravel.ID	Qtravel.tail, Ablow.ID.torp[1]
310	Qtravel.tail	-
1060	Qtravel.tail	End_blow1
1120	Qtravel.tail	End_blow2
1150	Qtravel.tail	Qtravel.tail, End_blow2
300	Qtravel.head	-
1230	Qtravel.head	Qtravel.head, Qtravel.size

tab. D.3 Especificação da transição para o modelo da usina (reduzido):

CAP #	AC	CONDIÇÃO	AÇÃO
7	{initial}	sys_time=0	Qmelt.tail:=0
8	{initial}	sys_time=0	Qmelt.size:=0
9	{initial}	sys_time=0	Qmelt.head:=0
10	{initial}	sys_time=0	For I:=1 to Nbf do
20	{initial}	sys_time=0	Create (Blast_furnace)
30	{initial}	sys_time=0	Qmelt.tail:=Qmelt.tail+1
40	{initial}	sys_time=0	Qmelt.size:=Qmelt.size+1
50	{initial}	sys_time=0	Qmelt.ID[Qmelt.tail]:=i.
60	{initial}	sys_time=0	caswt[i]:=0
70	{initial}	sys_time=0	End
219	{initial}	sys_time=0	Qidle.tail:=0
220	{initial}	sys_time=0	Qtidle.head:=0
221	{initial}	sys_time=0	Qtidle.size:=0
230	{initial}	sys_time=0	For I:=1 to Ntorp do
240	{initial}	sys_time=0	Create (Torpedoes)
250	{initial}	sys_time=0	Qtidle.tail:=Qtidle.tail+1
260	{initial}	sys_time=0	Qtidle.size:=Qtidle.size+1
270	{initial}	sys_time=0	Qtidle.ID[Qtidle.tail]:=i.
280	{initial}	sys_time=0	torpwt[i]:=0
290	{initial}	sys_time=0	End
300	{initial}	sys_time=0	Qtravel.head:=0
310	{initial}	sys_time=0	Qtravel.tail:=0
320	{initial}	sys_time=0	Qtravel.size:=0
330	{initial}	sys_time=0	Qreturn.head:=0
340	{initial}	sys_time=0	Qreturn.tail:=0
350	{initial}	sys_time=0	Qreturn.size:=0
360	{initial}	sys_time=0	Qtorp.head:=0
370	{initial}	sys_time=0	Qtorp.tail:=0
380	{initial}	sys_time=0	Qtorp.size:=0
561	{initial}	sys_time=0	Qblow.head:=0
562	{initial}	sys_time=0	Qblow.tail:=0
563	{initial}	sys_time=0	Qblow.size:=0
580	{initial}	sys_time=0	Amelt.size:=0
590	{initial}	sys_time=0	Areturn.size:=0
600	{initial}	sys_time=0	Atravel.size:=0
660	{initial}	sys_time=0	pitt:=true
670	{initial}	sys_time=0	totalwaste:=0
680	{melt1}	Qmelt.size>0	Amelt.size:=Amelt.size+1
690	{melt1}	Qmelt.size>0	Amelt.ID[Amelt_size]:=Qmelt.ID[Qmelt.head]
700	{melt1}	Qmelt.size>0	Qmelt.head:=Qmelt.head+1
710	{melt1}	Qmelt.size>0	Qmelt.size:=Qmelt.size-1
720	{melt1}	Qmelt.size>0	SET ALARM(End_melt[Amelt_size], normal(110,15))
730	{melt2}	WHEN ALARM(End_melt[i])	Qblow.tail:=Qblow.tail+1
740	{melt2}	WHEN ALARM(End_melt[i])	Qblow.size:=Qblow.size+1
750	{melt2}	WHEN ALARM(End_melt[i])	Qblow.ID[Qblow.tail]:=Amelt_ID[i]
760	{melt2}	WHEN ALARM(End_melt[i])	castwt[Qblow.ID[Qblow.tail]]:=normal[380,50]

770	{melt2}	WHEN ALARM(End_melt[i])	Amelt.size:=Amelt.size-1
780	{blow1}	Qtilde.size=1 and Qblow.size>0	Ablow.ID.torp[1]:=Qtidle.ID[Qtidle.head]
790	{blow1}	Qtilde.size=1 and Qblow.size>0	Qtidle.head:=Qtidle.head+1
800	{blow1}	Qtilde.size=1 and Qblow.size>0	Qtidle.size:=Qtidle.size-1
810	{blow1}	Qtilde.size=1 and Qblow.size>0	Ablow.ID.blastf[1]:=Qblow.ID[Qblow.head]
820	{blow1}	Qtilde.size=1 and Qblow.size>0	Qblow.head:=Qblow.head+1
830	{blow1}	Qtilde.size=1 and Qblow.size>0	Qblow.size:=Qblow.size-1
840	{blow1}	Qtilde.size=1 and Qblow.size>0	torpwt[Ablow.ID.torp[1]]:=castwt[Ablow.ID.blastf[1]]
850	{blow1}	Qtilde.size=1 and Qblow.size>0	totwaste:=totwaste+castwt[Ablow.ID.blastf[1]]-300
860	{blow1}	Qtilde.size=1 and Qblow.size>0	SET ALARM(End_blow1,10)
870	{blow0}	Qtidle=0 and Qblow.size>0	id_aux:=Qblow.ID
880	{blow0}	Qtidle=0 and Qblow.size>0	Qblow.head:=Qblow.head+1
890	{blow0}	Qtidle=0 and Qblow.size>0	Qblow.size:=Qblow.size-1
900	{blow0}	Qtidle=0 and Qblow.size>0	totaste:=totwaste+castwt[id_aux]
910	{blow0}	Qtidle=0 and Qblow.size>0	Qmelt.tail:=Qmelt.tail+1
920	{blow0}	Qtidle=0 and Qblow.size>0	Qmelt.size:=Qmelt.size+1
930	{blow0}	Qtidle=0 and Qblow.size>0	Qmelt.ID[Qmelt.tail]:=id_aux
940	{blow2}	Qtidle.size>1 and Qblow.size>0	Ablow.ID.torp[1]:=Qtidle.ID[Qtidle.head]
950	{blow2}	Qtidle.size>1 and Qblow.size>0	Qtidle.head:=Qtidle.head+1
960	{blow2}	Qtidle.size>1 and Qblow.size>0	Qtidle.size:=Qtidle.size-1
970	{blow2}	Qtidle.size>1 and Qblow.size>0	Ablow.ID.blastf[1]:=Qblow.ID[Qblow.head]
980	{blow2}	Qtidle.size>1 and Qblow.size>0	Qblow.head:=Qblow.head+1
990	{blow2}	Qtidle.size>1 and Qblow.size>0	Qblow.size:=Qblow.size-1
1000	{blow2}	Qtidle.size>1 and Qblow.size>0	torpwt[Ablow.ID.torp[1]]:=300
1010	{blow2}	Qtidle.size>1 and Qblow.size>0	Ablow.ID.torp[2]:=Qtidle.ID[Qtidle.head]
1020	{blow2}	Qtidle.size>1 and Qblow.size>0	Qtidle.head:=Qtidle.head+1
1030	{blow2}	Qtidle.size>1 and Qblow.size>0	Qtidle.size:=Qtidle.size-1
1040	{blow2}	Qtidle.size>1 and Qblow.size>0	torpwt[Ablow.ID.torp[2]]:=castwt[Ablow.ID.blastf[1]]-300
1050	{blow2}	Qtidle.size>1 and Qblow.size>0	SET ALARM(End_Blow2,10)
1060	{blow3}	WHEN ALARM(End_blow1)	Qtravel.tail:=Qtravel.tail+1
1070	{blow3}	WHEN ALARM(End_blow1)	Qtravel.size:=Qtravel.size+1
1080	{blow3}	WHEN ALARM(End_blow1)	Qtravel.ID[Qtravel.tail]:=Ablow.ID.torp[1]
1090	{blow3}	WHEN ALARM(End_blow1)	Qmelt.tail:=Qmelt.tail+1
1100	{blow3}	WHEN ALARM(End_blow1)	Qmelt.size:=Qmelt.size+1
1110	{blow3}	WHEN ALARM(End_blow1)	Qmelt.ID[Qmelt.tail]:=Ablow.ID.blastf[1]
1120	{blow4}	WHEN ALARM(End_blow2)	Qtravel.tail:=Qtravel.tail+1
1130	{blow4}	WHEN ALARM(End_blow2)	Qtravel.size:=Qtravel.size+1
1140	{blow4}	WHEN ALARM(End_blow2)	Qtravel.ID[Qtravel.tail]:=Ablow.ID.torp[1]
1150	{blow4}	WHEN ALARM(End_blow2)	Qtravel.tail:=Qtravel.tail+1
1160	{blow4}	WHEN ALARM(End_blow2)	Qtravel.size:=Qtravel.size+1
1170	{blow4}	WHEN ALARM(End_blow2)	Qtravel.ID[Qtravel.tail]:=Ablow.ID.torp[2]
1180	{blow4}	WHEN ALARM(End_blow2)	Qmelt.tail:=Qmelt.tail+1
1190	{blow4}	WHEN ALARM(End_blow2)	Qmelt.size:=Qmelt.size+1
1200	{blow4}	WHEN ALARM(End_blow2)	Qmelt.ID[Qmelt.tail]:=Amelt.ID.torp[1]
1210	{travel1}	Qtravel.size>0	Atravel.size:=Atravel.size+1
1220	{travel1}	Qtravel.size>0	Atravel.ID[Atravel.size]:=Qtravel.ID[Qtravel.head]
1230	{travel1}	Qtravel.size>0	Qtravel.head:=Qtravel.head+1
1240	{travel1}	Qtravel.size>0	Qtravel.size:=Qtravel.size-1
1250	{travel1}	Qtravel.size>0	SET ALARM(End_travel[Atravel.size],poisson(10))
1260	{travel2}	WHEN ALARM(End_travel[i])	Qtorp.ID[Qtorp.tail]:=Atravel.ID[i]
1270	{travel2}	WHEN ALARM(End_travel[i])	Qtorp.tail:=Qtorp.tail+1

1280	{travel2}	WHEN ALARM(End_travel[i])	Qtorp.size:=Qtorp.size+1
1290	{travel2}	WHEN ALARM(End_travel[i])	Atravel.size:=Atravel.size-1
1300	{return1}	Qreturn.size>0	Areturn.size:=Areturn.size+1
1310	{return1}	Qreturn.size>0	Areturn.ID[Areturn.size]:=Qreturn.ID[Qreturn.head]
1320	{return1}	Qreturn.size>0	Qreturn.head:=Qreturn.head+1
1330	{return1}	Qreturn.size>0	Qreturn.size:=Qreturn.size-1
1340	{return1}	Qreturn.size>0	SET ALARM(End_Return[Areturn.size],4)
1350	{return2}	WHEN ALARM(End_return[i])	Qtidle.tail:=Qtidle.tail+1
1360	{return2}	WHEN ALARM(End_return[i])	Qtidle.size:=Qtidle.size+1
1370	{return2}	WHEN ALARM(End_return[i])	Qtidle.ID[Qtidle.tail]:=Areturn.ID[i]
1380	{return2}	WHEN ALARM(End_return[i])	Areturn.size:=Areturn.size-1
1390	{fill}	Qtorp.size>1 and Qcrane.size>0 and pitt	pitt:=false
1400	{fill}	Qtorp.size>1 and Qcrane.size>0 and pitt	Afill.ID.torp:=Qtorp.ID[Qtorp.head]
1410	{fill}	Qtorp.size>1 and Qcrane.size>0 and pitt	Qtorp.head:=Qtorp.head+1
1420	{fill}	Qtorp.size>1 and Qcrane.size>0 and pitt	Qtorp.size:=Qtorp.size-1
1459	{fill}	Qtorp.size>1 and Qcrane.size>0 and pitt	transf:=minof(100,torpwt[Afill.ID.torp] /*minimum of x,y*/
1470	{fill}	Qtorp.size>1 and Qcrane.size>0 and pitt	torpwt[Afill.ID.torp]:=torpwt[Afill.ID.torp]-transf
1480	{fill}	Qtorp.size>1 and Qcrane.size>0 and pitt	pitt:=true
1490	{fill}	Qtorp.size>1 and Qcrane.size>0 and pitt	SET ALARM(End_fill,5)
1500	{endfill1}	When (End_fill[i]) and torpwt[Afill.ID.torp]=0	Qreturn.tail:=Qreturn.tail+1
1510	{endfill1}	When (End_fill[i]) and torpwt[Afill.ID.torp]=0	Qreturn.size:=Qreturn.size+1
1520	{endfill1}	When (End_fill[i]) and torpwt[Afill.ID.torp]=0	Qreturn.ID[Qreturn.tail]:=Afill.ID.torp
1530	{endfill2}	When (End_fill[i]) and torpwt[Afill.ID.torp]>0	Qtorp.head:=Qtorp.head-1
1540	{endfill2}	When (End_fill[i]) and torpwt[Afill.ID.torp]>0	Qtorp.size:=Qtorp.size+1
1550	{endfill2}	When (End_fill[i]) and torpwt[Afill.ID.torp]>0	Qtorp.ID[Qtorp.head]:=Afill.ID.torp
2090	{term}	sys_time>1000	STOP

REFERÊNCIAS BIBLIOGRÁFICAS

ABRAMS et al. "Linking Simulation Model Specification and Parallel Execution Through UNITY", In: *Proceedings of the Winter Simulation Conference*, pp. 233-242, 1991.

ADDANKI, S; CREMONI, R; PENBERTHY, J.S. "Graph of Models", *Artificial Intelligence*, v. 51, pp. 145-177, 1991.

BALBO G.; CHIOLA, G. "Stochastic Petri Nets Simulation", In: *Proceedings of the Winter Simulation Conference*, p. 266-276, 1989.

BALCI, O. "The Implementation of Four Conceptual Frameworks for Simulation Modelling in high-level languages", In: *Proceedings of the Winter Simulation Conference*, pp. 287-295, 1988.

BALCI, O. "Verification, Validation and Accreditation of Simulation Models", In: *Proceedings of the Winter Simulation Conference*, p. 135-141, 1997.

BALCI, O. et al. "Model Generation Issues in a Simulation Support Environment", In: *Proceedings of the Winter Simulation Conference*, p. 257-263, 1990.

BALCI, O.; NANCE, R.E. "The Simulation Model Development Environment: Na Overview", In: *Proceedings of the Winter Simulation Conference*, p. 726-736, 1992.

BANKS, J; CARSON, J. "Discrete-Event Systems Simulation", Prentice-Hall, Englewood Cliffs, 1984.

BARTON, R. R. "Metamodeling: A State of the Art Review", In: *Proceedings of the Winter Simulation Conference*, 237-244, 1994.

BAUSE, F; KEMPER, P. "QPN-Tool for the Qualitative and Quantitative Analysis of Queuing Petri Nets", In: *Proceedings of the 7th International Conference on Computer Performance Evaluation, Modelling Techniques and Tools*, pp. 321-334, 1994.

BROOKS, R. J.; TOBIAS, A. M. "Choosing the Best Model: Level of Detail, Complexity and Model Performance", *Mathl. Comput. Mpdelling*, v. 24, n. 4, pp. 1-14, 1996.

BROOKS, R. J.; TOBIAS, A. M. "Methods and Benefits of Simplification in Simulation", In: *Proceedings of UKSIM*, United Kingdom Simulation Society, pp. 88-92, 1999.

BUNGE, M. "The Myth of Simplicity: Problems of Scientific Philosophy", Prentice-Hall, Englewood Cliffs, New Jersey, 1963.

CASSANDRAS, C. "Discrete-Event Systems", Akren A.I. Publishers, 1993.

CAUGHLIN, D. "Parameter Identification Methods for Metamodeling Simulations", In: *Proceedings of the Winter Simulation Conference*, 756-763, 1996.

CERIC, V. "Hierarchical Abilities of Diagrammatic Representations of Discrete Event Simulation Model", In: *Proceedings of the Winter Simulation Conference*, p. 589-594, 1994.

CERIC, V.; PAUL, R. J. "Diagrammatic Representations of the Conceptual Simulation Model for Discrete Event Systems", *Mathematics and Computers in Simulation*, v. 34, p. 317-324, 1992.

CERIC, V.; PAUL, R. J. "Preliminary Investigations into Simulation Model Representation", *11th International Symposium Computer at the University*, Cavtat, 1989.

CHEN, S.L.; GERADIN, M; "An Exact Model Reduction procedure for Mechanical Systems", *Computational Methods Applied to Mechanical Engineering*, 143, pp. 69-78, 1997.

CHWIF, L. BARRETTO M.R.; SANTORO, M. C. "Model Reduction – Some Results", In: *Proceedings of the 31st Annual Simulation Sysmposium*, pp.120-125, 1998.

CHWIF, L.; BARRETTO, M.R.P. "Model Reduction Applied to a Kanban Operation Case Study", Autofact'98, Technical Paper, 1998.

CHWIF, L; PAUL, R.J., BARRETTO, M.R.P, "Combining the Best of the two: An Activity Cycle Diagram / Condition Specification Approach, In: *Proceedings of the UKSIM, United Kingdom Simulation Society*, pp. 93-98, 1999.

CLEMENTSON, A.T, "The ECSL Plus System Manual", The Chestnuts, Princes Road, Windermere, Cumbria, U.K., 1991.

CONWAY, R. W. et al. "The XCELL + Factory Modeling System", Realease 4, Scientific Press, 1992.

COOLAHAN, J.E. "Timing Requirements for Time Driven Systems Using Augmented Petri Nets", *IEEE Transactions on Software Engineering*, v. 9, n. 5, pp. 603-615, 1983.

COTA, B.A; FRITZ, D.G; SARGENT, R.G. "Control Flow Graphs as a Representation Language", In: *Proceedings of the Winter Simulation Conference*, p. 555-559 ,1994.

COTA, B.A; SARGENT, R.G. "Control Flow Graphs: A Method of Model Representation for Parallel Discrete Event Simulation". CASE Center Technical Report 9026, Syracuse University, Syracuse, NY, 1990.

COURTOIS, P.J "On Time and Space Decomposition of Complex Structures", *Communications of ACM*, v. 28, n.6, pp. 591-603, 1985.

COX, D.R.; SMITH, W.L. "Queues", Methuen and Company, Ltd., 1961

DAMERDJI, H. "Parametric Inference for Generalized Semi-Markov Processes", In: *Proceedings of the Winter Simulation Conference*, pp. 323-328, 1993.

DERRICK, E. J; BALCI, O; "DOMINO: A Multifaceted Conceptual Framework for Visual Simulation Modeling", *INFOR*, v. 35, n. 2, pp. 93-120, 1997.

DERRICK, E. J; BALCI, O; NANCE, R. E. "A Comparison of Selected Conceptual Frameworks for Simulation Modeling", In: *Proceedings of the Winter Simulation Conference*, p. 711-719, 1989.

DOMINGO, L. T. "Formal Methods in Specifying Discrete Events Simulation Models", Unpublished Ph.D. Thesis. University of London, England, 1991.

EVANS, G.W.; WALLACE, G. F.; SUTERLAND, G.L. "Simulation Using Digital Computers", Prentice Hall. Englewood Cliffs, NJ, 1967.

EVANS, J.B. "Structures of Discrete Event Simulation: An Introduction to the Engagement Strategy", Ellis Horwood, Chichester, 1988.

FALKENHAINER, B.; FORBUS, K.D. "Compositional Modeling: Finding the Right Model for the Job", *Artificial Intelligence*, v. 51, pp. 95-143, 1991.

FISHWICK, P.A. "An Integrated Approach to System Modeling Using a Synthesis of Artificial Intelligence, Software Engineering and Simulation Methodologies", *ACM Transactions on Modeling and Computer Simulation*, v.2, n. 4, 307-330, 1992.

FORRESTER, J. W. "Principles of Systems", Wright-Allen Press, Inc., Cambridge, Mass, 1968.

FRANTZ, F. K. "A Taxonomy of Model Abstraction Techniques", In: *Proceedings of the Winter Simulation Conference*, p. 1413-1420, 1995

FRITZ, D.G.; SARGENT, R.G., "An Overview of Hierarchical Control Flow Graph Models", In: *Proceedings of the Winter Simulation Conference*, p. 1347-1355 ,1995.

FRITZ, D.G.; SARGENT, R.G., DAUM, T., "An Overview of Hi-mass (Hierarchical Modeling and Simulation Systems)", In: *Proceedings of the Winter Simulation Conference*, p. 1356-1363 ,1995.

GEORGE, L. "Tests for System Complexity", *International Journal of General Systems*, v. 3, pp. 253-258, 1977.

GLYNN, P. "A GSMP Formalism for Discrete Event Systems", In: *Proceedings of the IEEE*, v. 77, n.1, pp. 14-23, 1989.

GOLAY, M. P.; SEONG, P.H.; MANNO, V.P. "A Measure of the Difficulty of Systems Diagnosis and its Relationship to Complexity", *International Journal of General Systems*, v. 16, n. 1, pp. 1-23, 1989.

GORDON, G. "System Simulation", Prentice-Hall, Englewood Cliffs, 1978.

GRUER, P.; KOUKAM, A.; MAZIGH,B. "Modeling and Quantitative Analysis of Discrete Event Systems: A Statecharts based Approach", *Simulation Practice and Theory*, v. 6, p. 397-411, 1998.

HARARY, F. "Graph Theory", Massachussets, Adison-Wesley, 1969.

HARARY, F. et al. "Structural Models: An Introduction to the Theory of Directed Digraphs", John Wiley & Sons, 1965.

HARREL D. et al "STATEMATE: A Working Environment for the Development of Complex Reactive Systems", *IEEE Trans. of Software Engineering*, v. 16, n. 4, 1990.

HARREL, C.; TUMAY, K. "Simulation Made Easy", Engineering & Management press, 1994.

HARREL, D. "On Visual Formalisms", *Communications of the ACM*, v. 31, n. 5, pp. 514 – 530.

HARREL, D. "Statecharts: A visual formalism for Complex Systems", *Sci. Comput. Program*, v. 8, n.3, pp. 231-274, 1987.

HARREL, D. STATEMATE, version 6.0, Analyser Refererence Manual, I-logix, 1995.

HASS, P.J; SHEDLER, G.S. "Modeling Power of Stochastic Petri Nets for Simulation", *Probability in the Engineering and Information Sciences*, v.2, pp. 435-459, 1988.

HENIZE, J. "Critical Issues in Evaluating Socio-Economic Models", In: *Simulation and Model Based Methodologies: An Integrative View*, Ed. T.I. Oren, B.P. Zeigler, M.S Elzas, 1984.

HLUPIC, V.;PAUL, R.J. "Simulation Modelling of Flexible Manufacturing Systems using Activity Cycle Diagrams", *J. Opl Res. Soc.*, v. 45, n. 9, pp 1011-1023, 1994.

HO, Y. "Scanning the Issue - Dynamics of Discrete Event Systems",In: *Proceedings of the IEEE*, v. 77, n. 1, pp. 3 - 7, January, 1989.

HUTCHINSON, G., K. "The Automation of Simulation", In: *Proceedings of the Winter Simulation Conference*, p. 489-495,1981.

INGALLS, R.G.; MORRICE, D.J; WHINSTON, A. B. "Eliminating Canceling Edges from the Simulation Graph Model Methodology", In: *Proceedings of the Winter Simulation Conference*, p. 825-832, 1996.

INNIS, G; REXSTAD, E. "Simulation Model Simplification", *Simulation*, pp. 7-15, July, 1983.

KELLING, C. "TimeNET-SIM – A parallel Simulator for Stochastic Petri Nets". In: *Proceedings of 28th Annual Simulation Symposium*, Phoenix, Arizona, USA, 1995.

KELTON, D.W., SADOWSKI, R. P.; SADOWSKI, D. A. "Simulation with Arena", McGraw-Hill, 1998.

KIENBAUM, G. "A Framework for Automatic Generation of Object-Oriented Simulation Models", Unpublished Ph.D. Thesis. Brunel University, 1995.

KIENBAUM, G; PAUL, R. J., "H-ACD: Hierarchical Activity Cycle Diagrams for Object-Oriented Simulation Modelling", In: *Proceedings of the Winter Simulation Conference*, p. 600-610 ,1994.

KUIPERS, B. "Qualitative Reasoning – Modeling and Simulation with Incomplete Knowledge", Massachusetts Institute of Technology, 1994.

KUMAR, D.; HAROUS, S. "An Approach Towards Distributed Simulation of Timed Petri Nets", In: *Proceedings of the Winter Simulation Conference*, pp. 428-435, 1990.

LACKER, M.R. "Digital Simulation and Systems Theory", System Development Corporation, Santa Monica, CA, 1964.

LADY, G.M "On Organizing Analysis", In: *Computer-Assisted Analysis and Model Simplification*, Edited by Harvey J. Greenberg and John S. Maybee, p. 1-15, 1981.

LAW, A. M. : MCCOMAS, M. G "Secrets of Succesful Simulation Studies", *Industrial Engineering*, pp.47-72, May, 1990.

LAW, A.M.; KELTON, D. W. "Simulation Modeling & Analysis", McGraw-Hill, Inc, 1991.

LEVY, A.Y; IWASAKI, Y, FIKES, R; "Automated Model Selection for Simulation based on Relevance Reasoning", *Artificial Intelligence*, n. 96, p. 351-394, 1997.

MAK, H. Y. "Systems Dynamics and Discrete Event Simulation Modelling", Unpublished Ph.D. Thesis. University of London, England, 1992.

MATHEWSON, S.C. "Simulation Program Generators: Code and Animation on a PC", *Op. Res. Society*, v.36, n. 7, pp. 583-589, 1985.

MATHEWSON, S.C. "Simulation Program Generators", *Simulation*, v. 23, n.6 pp. 181-189, 1974.

MCCABE, T. J "A Complexity Measure", *IEEE Transactions on Software Engineering*, v. 2, pp. 308-320, 1976.

MILLER, G.A. "The Magical Number Seven, Plus or minor Two: Some Limits on our Capacity for Processing Information", *The Psychological Review*, v. 63, pp. 81-87, 1956.

MOLLOY, M.K. "Performance Analysis using Stochastic Petri Nets", *IEEE Transactions on Computers*, v. 31, n. 9, pp. 913-917, 1982.

MOORE, B. C. "Principal Component Analysis in Linear Systems: Controllability, Observability, and Model Reduction", *IEEE Transactions on Automatic Control*, v. 26, n. 1, pp. 17-31,1981.

NANCE, R. E. "A History of Discrete Event Simulation Programming Languages", In: *Proceedings of the Second ACM SIGPLAN History of Programming Languages Conferences*, Cambridge, MA, 28(3), pp. 149-175, 1993.

NANCE, R. E., "Model Representation in Discrete Event Simulation: Prospects for Developing Documentation Standarts", *Current Issues in Computer Simulation*, Academic Press, New York, pp. 83-97, 1979.

NANCE, R. E.; "The Conical Methodology and the Evolution of Simulation Model Development", *Annals of Operations Research*, v. 53, pp. 1-45, 1994.

NANCE, R. E.; OVERSTREET, C. M, "Diagnostic Assistance using digraph representations of discrete event simulation Model specifications", *Transactions of the Society for Computer Simulation*, v. 4, n. 1, pp. 33-57, 1988.

NANCE, R. E.; OVERSTREET, C. M, "Exploring the Forms of Model Diagnosis in a Simulation Support Environmet", In: *Proceedings of the Winter Simulation Conference*, pp. 590 – 596, 1987.

NANCE, R. E.; OVERSTREET, C. M, PAGE, E. "Redundancy in Model Specifications for Discrete Event Simulation", Unpublished Paper, 1999.

NANCE, R.E "A Tutorial View of Simulation Model Development", In: *Proceedings of the Winter Simulation Conference*, pp. 325-331, 1983.

NARAIN, S. "An Axiomatic Basis for General Discrete-Event Modeling", In: *Proceedings of the Winter Simulation Conference*, pp. 1073-1082, 1992.

NAYAK, P. "Causal Approximations", *Artificial Intelligence*, v. 70, pp. 277-334, 1994.

NAYAK, P; JOSKOWICZ, L. "Efficient Compositional Modeling for Generating Causal Explanations", *Artificial Intelligence*, 83, pp. 193-227, 1996.

NICOL, D.M; ROY, S. "Parallel Simulation of Timed Petri Nets", *In: Proceedings of the Winter Simulation Conference*, pp. 574-583, 1991.

NORDGREN, W. B. "Steps to Sucessful Simulation Project Management", F&H Simulations, Inc, 1994

OVERSTREET M.C.; NANCE, R. E., "World View Based Discrete Event Model Simplification", In: *Modelling and Simulation Methodology in the Artificial Intelligence Era*, Edited by M.S. Elzas, pp.165 – 179, 1986.

OVERSTREET, M.C; NANCE, R.E, "A Specification Language to Assist in Analysis of Discrete Event Simulation Models", *Communications of the ACM*, pp. 191-201, 1985.

OVERSTREET, M.C; PAGE, E.H; NANCE, R.E, "Model Diagnosis Using the Condition Specification: From Conceptualization to Implementation", In: *Proceedings of the Winter Simulation Conference*, pp. 566-573, 1994.

PAGE, H. E. "Simulation Modeling Methodology: Principles and Etiology of Decision Support", Unpublished Ph.D. Thesis. Virginia Polytechnic Institute and State University, Virginia, 1994.

PAGE, H. E.; NANCE, R.E. "Incorporating Support for Model Execution within the Condition Specification", *Unpublisehd Paper*, 1999.

PALM, D.C. "The Distribution of Repairmen in Servicing Automatic Machines", *Industritidningen Norden,* 175, p. 75, 1947.

PAUL R. J. "VS7& sysPack User Guide", sysPACK Ltd, London, U.K. 1990.

PAUL, R. J. "Activity Cycle Diagrams and the Three Phase Approach", In: *Proceedings of the Winter Simulation Conference*, pp. 123-131, 1993.

PAUL, R. J. "The Computer Aided Simulation Modeling Environment: An Overview", In*: Proceedings of the Winter Simulation Conference*, pp. 737-745, 1992.

PAUL, R. J.; BALMER, D.W. "Simulation Modelling", Chartwell-Bratt, London, 1993.

PEDGREN, D. C.; SHANNON, R. E.; SADOWSKI, R. P. "Introduction to Simulation using Siman", McGraw-Hill, 1995.

PETERSON, T.L "Petri Nets", *ACM Computing Surveys*, v. 9, n.3, pp. 223-252, 1977.

PETRI, C.A., "Communication with Automata", New York: Griffiss Air Force Base. Tech. Report. RADC-TR-65-377, v. 1, Suppl.1, 1966.

PFLUGHOEFT, K.A.; MANUR, K. "Multi-Layered Activity Cycle Diagrams and Their Convertion into Activity-Based Simulation Code", In: *Proceedings of the Winter Simulation Conference*, p. 595-599 ,1994.

PIDD, M. "Computer Simulation in Management Sciences", John Wiley and Sons, Chichester, 4th edition, 1998.

PIDD, M. "Five Simple Principles of Modelling", *In: Proceedings of the Winter Simulation Conference,* p. 721-728, 1996.

POOLEY, R. J. "Towards a Standard for Hierarchical Process Oriented Discrete Event Simulation Diagrams. PART III: Agregarion and Hierarchical Modelling", *Transactions of the Society for Computer Simulation*, v. 8, n.1, pp. 33-41, 1991.

PRITSKER, A. B. , - "Introduction to Simulation and Slam", John Wiley & Sons, 1986.

RADYA, A.A, SARGENT, R.G. "A Logic-Based Foundation of Discrete Event Modeling and Simulation", *ACM Transactions on Modeling and Computer Simulation*, v. 4, n. 1, p. 3-51, 1994.

REXTAD, E.; INNIS, G.S. "Model Simplification – Three Applications", *Ecological Modelling*, n. 27 v ½, pp. 1-13, 1985.

RICKEL, J ; PORTER B. "Automated Modeling of Complex Systems to Answer Prediction Questions", *Artificial Intelligence*, v. 93, pp. 201-260, 1997.

ROBINSON, S. "Successful Simulation – A Practical Approach to Simulation Projects", McGraw-Hill, London, 1994.

ROTHEMBERG, J. "Knowledge-Based Simulator at RAND", *Simuletter*, v. 19 n. 2, pp. 54-59, 1988.

ROTHEMBERG, J. "Tutorial: Artificial Intelligence and Simulation", In: *Proceedings of the Winter Simulation Conference*, p. 218-222, 1991.

SALT, J. D. "Keynote Address: Simulation Should be Easy and Fun !", In: *Proceedings of the Winter Simulation Conference,* p. 1-5, 1993.

SARGENT, R. G. "Verifying and Validating Simulation Models", In: *Proceedings of the Winter Simulation Conference*, pp. 55-64, 1996.

SCHRIBER, T. J. "An Introduction to Simulation Using GPSS / H", John Wiley and Sons, 1991.

SCHRUBEN, L. "Simulation Modeling with Event Graphs", *Communications of the ACM*, v.26, n.11, 1983.

SCHRUBEN, L. W. "Graphical Model Structures for Discrete Event Simulation", In: *Proceedings of the Winter Simulation Conference*, pp. 241-245, 1992a.

SCHRUBEN, L. W. "Modeling Priority Queues with Entity Lists: A Sigma Tutorial", In: *Proceedings of the Winter Simulation Conference*, pp. 380-384, 1992b.

SCHRUBEN, L.W. " Simulation Graphical Modeling and Analysis (Sigma) Tutorial", In: *Proceedings of the Winter Simulation Conference*, pp. 158-161, 1990.

SCRUBEN, L; YUCESAN, E. "Transforming Petri Nets into Eventy Graphs Models", In: *Proceedings of the Winter Simulation Conference*, pp. 560-565, 1994.

SEVINC, S. ; FOO, N.Y. "Discrete Event Model Simplification via State Classification", In: *AI and Simulation: Theory and Applications*, Ed. W. Webster and R. Uttamsingh, Simulation Series, v. 22, n. 3, April, 1990.

SEVINC, S. "Theories of Discrete Event Model Abstraction", In: *Proceedings of the Winter Simulation Conference*, p. 1115-1119, 1991

SHANNON, R.E, "Systems Simulation – The Art and Science", Prentice-Hall, 1975.

SIMON, H.A. "The Architecture of Complexity", *General Systems Yearbook*, v. 16, pp. 63-76, 1964.

STORRLE, H. "An Evaluation of High-End Tools for Petri-Nets", Techical Report #9802, June, 1998.

TOCHTER, K.D. "The art of simulation", English Universities Press, 1963.

TOFTS, C. "Exact, Analytic, and Locally Approximate Solutions to Discrete Event-Simulation Problems", *Simulation Practice and Theory*, v. 6, pp. 721-759, 1998.

TOP, J et al. "Structure and Use of a Library for Physical Systems Models", *Proceedings of ICBGM*, pp. 97-102, 1995.

TORN, A.A. "Simulation Graphs: A General Tool for Modeling Simulation Designs", *Simulation*, v. 37, n. 6, pp. 187-194, 1981.

TORN, A.A. "Simulation Nets – A Simulation Modelling and Validation Tool", *Simulation*, v. 45, n. 2, pp. 71-75, 1985

TORN, A.A. "The Simulation Net Approach to Modelling and Simulation", *Simulation*, v. 57, n. 3, pp. 196-198, 1991.

ULGEN, O. M. et al. "Simulation Methodology - A Practioner's Perspective", *International Journal of Industrial Engineering, Aplications and Practice*, v.1, n.2, June, 1994.

UNGER, B.A et al. "The JADE Aproach to Distributed Software Development", Proceedings of the Conference on Ai Applied to Simulation, pp. 474-484, 1986.

WALLACE, J.C. "The Control and Transformation Metric: Toward the Measurement of Simulation Model Complexity", In: *Proceedings of the Winter Simulation Conference*, 597-603, 1987.

WALLACE, J.C. "The Control and Transformation Metric: A Basis for Measuring Model Complexity", M.S. Thesis, Department of Computer Science, Virginia Tech, Blacksburg, V, 1985.

WARD, S.C. "Arguments for Constructively Simple Models", *J. Opl. Res. Soc.* v. 40, n. 2, pp. 141-153, 1989.

WILLEMAIN, T.R. "Insights on Modeling from a Dozen Experts", *Operations Research*, v. 42, n. 2, 1994.

WILLEMAIN, T.R. "Model Formulation: What Experts Think about and When", *Operations Research*, v. 43, n. 6, 1995.

WILSON, B.H.; STEIN, J.L. "An Algorithm for Obtaining Minimum-Order Models of Distributed and Discrete Systems" *Automated Modelling*, ASME, pp.37-46, 1992.

XIA, S. "Formulation of Generic Principles of Modelling Industrial Systems", *Int. J. Systems Analysis, Modelling and Simulation*, v. 15, pp. 283-291, 1994.

XIA, S. SMITH, N, "Automated Modelling: a Discussion and Review", *The Knowledge Engineering Review*, v. 11, n. 2, p.137-160, 1996.

XIA, S.; LINKENS, D. A.; BENNETT, S. "Automatic Modelling and Analysis of Dynamic Physical Systems Using Qualitative Reasoning and Bond Graphs", *Intelligent Systems Engineering*, v. 3, n. 2, pp. 201-212, 1993.

YIN H.Y.; ZHOU Z. N. "Simplification Techniques of Simulation Models", In: *Proceedings of Beijing International Conference on System Simulation and Scientific Computing*, pp. 23-26, 1989.

YUCESAN, E. "Simulation Graphs for the Design and Analysis of Discrete Event Simulation Models", Unpublished Ph.D. Dissertation. Cornell University, Ithaca, NY, 1989.

YUCESAN, E.; SCHRUBEN, L. "Structural and Behavioral Equivalence of Simulation Models", *ACM Transactions on Modelling and Computer Simulation*, v. 2, n. 1, 1992.

YUCESAN, E.; SCHUBEN, L. "Complexity of Simulation Models: A Graph Theoretic Approach", *INFORMS Journal on Computing*, v. 10, n. 2, pp. 94-106, 1998.

ZEIGLER, B.P. "Multifacetted Modeling and Discrete Event Simulation", Academic Press, 1984.

ZEIGLER, B.P. "Theory of Modelling and Simulation", John Wiley, 1976.

ZEIGLER, B.P; HU, J.; ROZEMBLIT, J.W. "Hierarchical Modular Modeling in DEVS-Scheme",In: *Proceedings of the Winter Simulation Conference*, pp. 84-89, 1989

APÊNDICE A

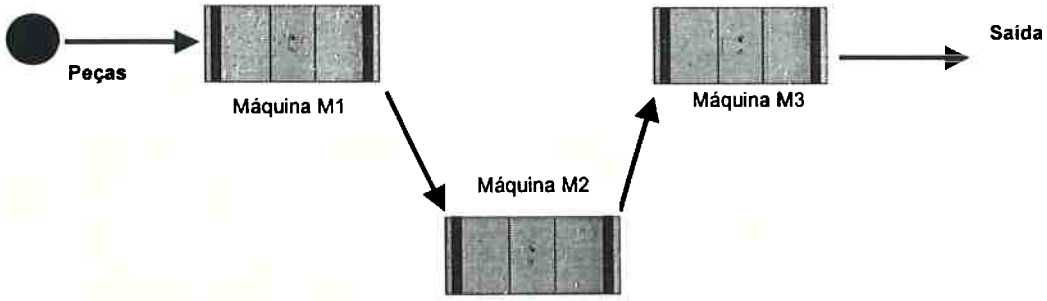
EXERCÍCIO DE MODELAGEM

Nome:	Idade:	Sexo: M / F

Experiência em Modelagem de Simulação (marque onde apropriado)	Menos de 6 meses	De 6 mes. a 1 ano	De 1 ano a 2 anos	De 2 anos a 5 anos	Mais de 5 anos

O objetivo deste exercício simples é avaliar como pessoas diferentes entendem e interpretam uma certa descrição e verificar como o modelo conceitual é construído.

DESCRIÇÃO DO PROBLEMA (MUITO SIMPLES): Num "Job-Shop" há 3 máquinas M1, M2 e M3. As peças chegam aleatoriamente no sistema com taxa de chegada média de 2.12 minutos. Primeiramente são processadas na máquina M1, depois na máquina M2 e finalmente na máquina M3 (Sempre nesta seqüência). Após ser processada na máquina M3, estas saem do sistema. Cada máquina pode processar uma peça por vez. O gerente da produção requisitou um estudo de simulação a fim de determinar a taxa de utilização da máquina M2. Observações: Tempos de processamento das máquinas M1, M2 e M3 seguem uma distribuição normal com média de 1.70, 1.55 e 1.62 respectivamente e 15% de desvio padrão para cada média. As peças podem ser armazenadas em estoques próximos das máquinas. A figura abaixo ilustra esta situação:



Favor Desenhar abaixo o modelo de simulação representado em ACD desta descrição: