# MÉTODOS E TÉCNICAS PARA PROGRAMAÇÃO EFICIENTE DE CLP'S EM APLICAÇÕES COMPLEXAS

C. M. Furukawa (*) M. Marchese (**) P. E. Miyagi (*)

(*) Escola Politécnica da USP
PMC - Mecatrônica
CP 8174 - CEP 05508

(**) Andersen Consulting
Arthur Andersen
R. Alexandre Dumas, 1981
CEP 04717 - São Paulo

## RESUMO

Hoje, o maior problema para a utilização racional dos modernos CLP's em sistemas de alta complexidade reside na concepção e na geração de seu software de controle. Este trabalho procura apontar meios para facilitar tal tarefa, diminuir seus custos e garantir ao software uma melhor qualidade. Para o nível de coordenação do sistema, introduz-se o MFG (Mark Flow Graph), cuja teoria é derivada das redes de Petri; para o nível mais baixo de controle, mostra-se como princípios de programação estruturada podem ser aplicados em CLP's.

## ABSTRACT

Currently, the main problem for the rational use of the modern PLC's on complex systems is related to its software conception and generation. This paper is intended to point means to make such work easier, decrease costs and guarantee quality to the software. For the system coordination level, the MFG (Mark Flow Graph) theory is introduced , which is derived from the Petri Nets theory; for the lower control level, it is shown how concepts of structured programming may be applied on PLC's.

**Keywords**: Sistemas de eventos discretos (SED); controle de SED; MFG; CLP; programação estruturada



## 1. INTRODUÇÃO

Os Controladores Lógicos Programáveis (ou simplesmente CLP's) foram concebidos no final dos anos 60 (Sekiguchi, 1988) para substituir os dispendiosos e nada práticos painéis de controle a relés. Desta forma, a aplicação inicial dos CLP's se restringiu ao controle por intertravamento e seqüenciamento de equipamentos.

Neste contexto, os CLP's foram extensivamente empregados em controle de equipamentos isolados ou grupos de dispositivos simples como válvulas, motores e bombas; os escassos recursos dos primeiros controladores não permitiam ir muito além.

No entanto, o emprego de poderosos microprocessadores na construção dos CLP's ampliaram enormemente a capacidade de controle e a gama de aplicação destes equipamentos. Seus recursos os direcionam a tarefas muito mais complexas, a ponto de ser possível controlar toda uma planta com um único CLP. Atualmente, CLP's podem ser interligados em rede, e trocar informações com um computador, constituindo assim um sistema altamente automatizado.

Porém, se por um lado o hardware do controlador já não é um fator limitante, por outro o papel do software de controle muda drasticamente nestas situações. O seu projeto e implementação exigem técnicas muito mais aprimoradas do que os diagramas de relés (*ladder logic*), herança dos painéis de relés. A aventura de se desenvolver um sistema de software deste porte sem qualquer método acaba por conduzir a um **custo de desenvolvimento** elevado. O custo é inchado em especial pelo tempo excessivo gasto por pessoal especializado em sua concepção, depuração e testes. Pode-se imaginar quão árdua e improdutiva seria a tarefa de se escrever, cegamente, um programa de 10.000 linhas em *ladder logic*.

Outro problema é a **baixa qualidade** do software resultante, traduzido pela baixa **confiabilidade** e **eficiência**. Estes programas tendem a ser **ininteligíveis**, o que onera o seu aprimoramento e manutenção. Como prever que efeitos colaterais a alteração de uma instrução poderia acarretar no restante do programa?

Em razão destes motivos, fica patente a limitação imposta pelo software à expansão da aplicação dos CLP's em problemas de maior complexidade. Este artigo procura integrar alguns métodos, técnicas e conceitos (oriundos da engenharia de software), numa tentativa de dar corpo a uma metodologia de desenvolvimento, a qual vise aumentar a eficiência do processo de programação dos CLP's.

## 2. CARACTERIZAÇÃO DE SISTEMAS COMPLEXOS

A análise de sistemas de maior complexidade destaca algumas características relevantes (Masuda, 1981):

Paralelismo: várias atividades paralelas devem ser conduzidas de forma coerente e harmoniosa, para garantir o correto funcionamento do sistema como um todo.

Concorrência: atividades paralelas muitas vezes acabam por competir por recursos comuns, gerando

conflitos e disputas internas ao sistema. Estes recursos devem ser compartilhados, muitas vezes em prejuizo de desempenhos individuais.

. Assincronismo: os eventos que demarcam a evolução das atividades possuem característica predominantemente assíncrona.

**Um Estudo de Caso**

Para ilustrar os problemas inerentes ao controle de sistemas deste tipo, vamos descrever uma pequena planta hipotética como exemplo. Trata-se de uma planta química, concebida para operar em lotes, e que se compõe de um tanque de solução base (**TB**), dois tanques de reagentes (**TR1, TR2**), dois reatores (**R1, R2**) e dois tanques para armazenamento de produtos finais (**TE1, TE2**). A figura 2.1 a esquematiza.

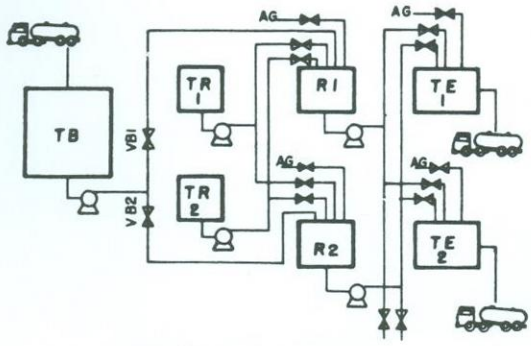


**Fig. 2.1: Sistema Exemplo.**

Cada reator opera de forma independente, reagindo quantidades determinadas de base e reagentes. Variando-se as proporções dos reagentes e o controle do caminho da reação, produz-se produtos diferentes em um mesmo reator. Os tanques de base e estoque são respectivamente recarregados e descarregados de tempos em tempos, em instantes aleatórios.

Assumiremos que os tanques de reagentes têm seus níveis mantidos sempre acima de limites mínimos (podemos tomar suas capacidades como infinitas). A transferência de reagente para os reatores é feita por bombas volumétricas, as quais não podem ser empregadas para alimentar ao mesmo tempo mais de um reator.

O volume de base mantido no primeiro tanque é irmanamente consumida pelos dois reatores. Os tanques de estoque também são livremente utilizados por estes, tomando-se o cuidado de não lhes transferir produtos diferentes dos que neles já se encontram. Sempre que houver necessidade, um reator solicita a lavagem de um **TE** para que haja a troca do produto a ser nele mantido.

Para este sistema, é possível enumerar as seguintes atividades principais:

. Controle do Tanque de Base: deve-se administrar devidamente o volume de base do **TB**, uma vez que este é compartilhado pelos reatores. O volume disponível é atualizado a cada requisição dos reatores e a cada recarga.

. Controle dos Reatores: os reatores são controlados de modo independente entre si. Podem estar produzindo produtos diferentes num mesmo instante, não estando sujeitos a tempos e parâmetros iguais.

Controle dos Tanques de Estoque: também operam de modo independente, já que podem armazenar produtos distintos, e suas descargas ocorrem a qualquer instante. Além disso, deve-se lavar um tanque quando da troca de seu conteúdo.

Lembrando as características citadas acima, vemos que ocorre *paralelismo* no controle tanto entre os controles individuais dos dois reatores ou dos tanques de estoque como entre as atividades de controle destes e mais a do tanque de base. Os dois reatores ainda *competem* entre si pelo uso de base, reagentes e **TE's**. E, por fim, todas estas atividades se desenvolvem de forma *assíncrona*.

**Níveis de Controle**

Uma análise mais atenta para o que até o momento temos chamado de *atividades* mostra que estas englobam várias ações menores. São ações de controle sobre válvulas, motores, etc, que concretamente são responsáveis pelo funcionamento da planta.

Em sistemas como este, dois níveis distintos de controle podem ser delineados: um nível de **macro-controle**, cuja principal atribuição é coordenar as diversas atividades que compõem o sistema; e um nível de micro-controle, onde se concentram as ações que perfazem as atividades. A nível micro, vamos encontrar problemas tradicionalmente resolvidos por meio de diagramas de relés, sobre as quais já se tem acumulada boa experiência.

Já a nível macro, o controle assume características típicas de **gerenciamento** ou **coordenação**, e é onde os problemas de paralelismo, concorrência e assincronismo devem ser resolvidos. Na realidade, estes dois níveis englobam várias camadas intermediárias de complexidade decrescente.

Desta forma, a questão chave é como quebrar sucessivamente o sistema em problemas de menor complexidade até se chegar a um ponto em que se tenha apenas ações controláveis por simples programas em *ladder logic*. A subdivisão do sistema certamente não pode ser conduzida a esmo: há de se procurar fazê-lo de forma a garantir **qualidade** (confiabilidade, eficiência e manutenibilidade) ao software final. É importante ainda que o software final seja **inteligível**, uma vez que isto deve implicar na facilidade de sua manutenção, em sua confiabilidade, e principalmente em menor tempo de desenvolvimento.

Uma forma de alcançar estes objetivos é desenvolver o software de forma *estruturada*. Em engenharia de software, a **programação estruturada** (Jensen, 81) há muito é uma filosofia consagrada de programação. Sua simplicidade permite que seja aplicada em praticamente qualquer sistema programável. No fundo, trata-se da aplicação em software de conceitos largamente empregados no projeto de outros sistemas físicos:

Abstração: por abstração, entende-se o processo de modelar um ente a partir de suas propriedades essenciais ou mais relevantes. Detalhes destas propriedades são intencionalmente suprimidos (ou *escondidos*), de modo a diminuir o grau de complexidade do sistema.

Concepção Hierárquica: o sistema é estruturado em níveis hierárquicos, onde um nível (ou camada) é subordinado apenas ao seu nível imediatamente superior, e por sua vez controla somente o seu nível logo abaixo.

. Modularidade: procura-se quebrar o sistema inteiro em subsistemas bem definidos e menores, pelo agrupamento de funções correlacionadas e afins. Um subsistema, uma vez construído, funciona como uma *caixa-preta*: é suficiente e completamente definido pelo seu comportamento Entrada/Saída.

A aplicação sistemática e eficiente destes conceitos necessita de novos conceitos e ferramentas específicos para os diferentes níveis (macro e micro) de projeto. A nível macro, por exemplo, busca-se retratar os requisitos de coordenação do sistema.

## 3. PROJETO E ANÁLISE DE SISTEMAS COMPLEXOS

As características de paralelismo, concorrência e assincronismo fazem dos sistemas complexos uma aplicação sob medida para as **Redes de Petri** (Peterson, 77,81), ou para outros modelos delas derivados. Em especial, para os problemas específicos dos sistemas de produção, a teoria do **Mark Flow Graph** (Miyagi, 89) - ou simplesmente MFG - se apresenta como uma boa opção.

### O Mark Flow Graph

O MFG se fundamenta na teoria de grafos e se constitui em uma ferramenta gráfica para modelamento, análise e projeto de sistemas complexos. A figura 3.1 mostra os elementos básicos de um grafo MFG.

Boxes podem ser empregados para representar estados possíveis. A presença de uma marca dentro de um box indica a condição de validade do estado a ele associado. A evolução do sistema no tempo é dada pela movimentação das marcas, que passam de um box para outro, seguindo a orientação dos arcos, e representam o seqüenciamento dos estados.

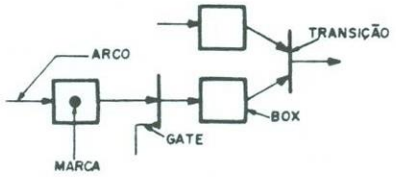


**Fig. 3.1: Elementos Gráficos do MFG.**

Marcas se movem somente quando se dá o **disparo** de uma transição. Para que uma transição possa disparar, é necessário que esta se encontre **habilitada**. Isto é:

. Todos os seus boxes de entrada devem estar marcados;
. Todos os boxes de saída devem estar vazios; e
. Os gates a ela conectados devem estar habilitados.

Uma transição ao disparar remove as marcas de todos os seus boxes de entrada e coloca uma marca em seus boxes de saída. A figura 3.2 ilustra um disparo.

### Modelamento e Projeto de Sistemas em MFG

Vamos ver como o MFG pode ser empregado para modelar um sistema real. Usaremos como exemplo a planta apresentada na figura 2.1. Inicialmente, além das atividades já apontadas de controle dos tanques e reatores, algumas outras devem ser acrescidas, como os procedimentos de energização (a ser feita

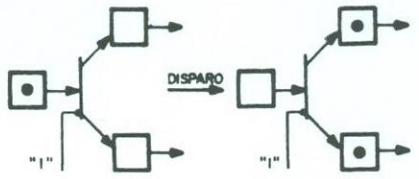


**Fig. 3.2: Disparo de uma Transição.**

sempre que a planta é posta em funcionamento), iniciação, parada total e desenergização. Chega-se então ao diagrama da figura 3.3.

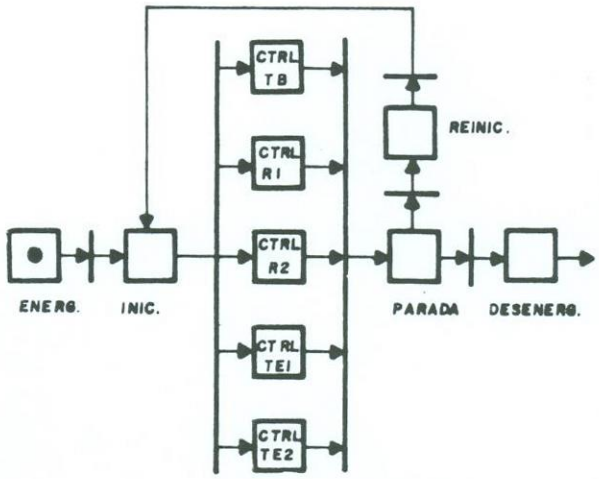


**Fig. 3.3: MFG da Planta Química.**

Como se pode ver, o grafo acima retrata apenas a forma com que as diferentes atividades devem ser coordenadas, sem apresentar qualquer indicação de como serão efetivamente controladas. Ou seja, encontram-se **abstraídos** os detalhes de suas implementações.

O passo seguinte é então detalhar melhor cada uma delas. O segundo nível de detalhamento das atividades de controle dos reatores, por exemplo, teria o aspecto da figura 3.4.

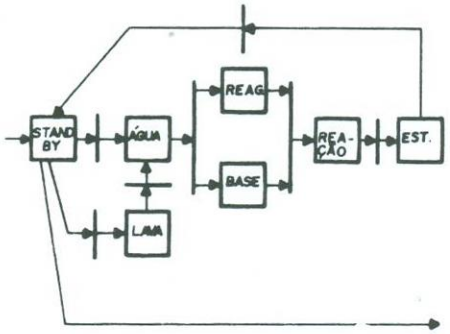


**Fig. 3.4: Controle do Reator 1.**

Resumidamente, as linhas de produção (centradas nos reatores) possuem um estado inicial onde permanecem paradas. A produção de um lote pode começar ou não com uma lavagem do reator. Vemos pelo grafo que o processo foi modelado segundo as seguintes premissas:

. O processo inicia com a entrada de água no reator;

. Base e reagente são transportados ao mesmo tempo para o reator; e

. A reação somente tem início com todos os componentes presentes.

Novamente, os detalhes de implementação destas sub-atividades não estão explícitos; e é esta abstração que nos permite interpretar e entender o modelo exposto no diagrama. O processo prossegue agora com o detalhamento das atividades do grafo da figura 3.4. Vê-se que atingimos agora um baixo nível de controle: nossa preocupação se reduz a acionar os dispositivos e equipamentos envolvidos em cada atividade, o que pode ser programado sem maiores dificuldades em diagramas *ladder*. Entretanto, isto somente será possível se houver algum mecanismo que assegure a coordenação das atividades.

**Coordenação de Atividades no CLP**

Para tanto, devemos implementar no CLP a mesma lógica de funcionamento do MFG. Uma forma de fazê-lo é encarar um grafo MFG como uma *máquina de estados estendida* (estendida porque, em dado instante, mais de um estado pode se encontrar válido). A construção desta máquina pode ser feita com a técnica *hot-bit*: associa-se um bit para cada estado/box do diagrama, e condiciona-se a execução das ações de uma atividade ao estado do bit associado, como ilustra a figura 3.5.


**Fig. 3.5: Controle dos Estados de uma Atividade.**

As mudanças de estados são feitas pelo disparo das transições. A figura 3.6 mostra como uma transição pode ser codificada por seqüências de *latch/unlatch* sobre os bits de estado correspondentes aos boxes MFG ligados à transição.

Convém ressaltar que a estrutura hierárquica montada com os níveis de detalhamento (do nível de maior até os de menor abstração) deve ser mantida nas máquinas de estado. Reportando-nos ao exemplo da planta hipotética, teria-se uma máquina principal que coordenaria outras sub-máquinas, responsáveis pela execução das atividades de energização, iniciação, controle de **TB**, controle de **R1**, etc.

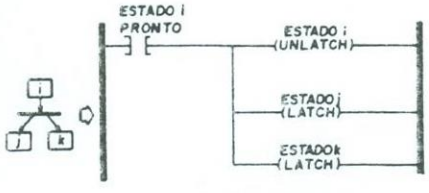


**Fig. 3.6: Codificação de uma Transição.**

Como se vê, a aplicação do modelo MFG na estruturação do problema assegura ao software uma arquitetura hierárquica. A hierarquia entre os níveis de detalhamento garante que uma sub-máquina somente dependa das suas sub-máquinas, não estando sujeita à forma com que as outras situadas no mesmo nível foram implementadas. Tal compartimentação aumenta a confiabilidade do sistema como um todo e facilita sua manutenção.

## 4. PROGRAMAÇÃO ESTRUTURADA EM CLP'S

A preocupação com estruturação deve ser mantida também a nível de programação das ações em *ladder logic*. No entanto, tal estruturação é dependente dos recursos de programação oferecidos pelo controlador. Recursos como:

a. Emprego de sub-rotinas (procedimentos, funções, sub-programas, etc);
b. Elementos para a montagem de estruturas de dados (vetores, registros, ponteiros, etc);
c. Endereçamento simbólico para dados e sub-rotinas;
d. Controle de escopo de símbolos;
e. Organização em módulos.

Os ítens acima foram citados em ordem crescente de poder de estruturação; o último (módulos) engloba todos os anteriores. Por módulo se entende uma unidade funcional de software, composta por sub-rotinas e estruturas de dados correlatas, associadas a símbolos (ou nomes). Símbolos aumentam a legibilidade do programa, tornando-o mais compreensível, e livram o programador da tarefa de administrar a memória do CLP. Alguns destes símbolos podem ser acessados por outros módulos (possuem *escopo aberto*), seja pela chamada de uma sub-rotina ou pelo acesso a uma estrutura de dados. Outros são mantidos propositalmente inacessíveis, para garantir a integridade do módulo: são dados e sub-rotinas ditos **encapsulados**.

Módulos são organizados em níveis hierárquicos: ou seja, um módulo usa os *serviços* oferecidos por módulos abaixo dele, e por sua vez oferece os seus serviços (sub-rotinas e estruturas de dados acessíveis) aos módulos da camada que lhe é superior. O encapsulamento, neste aspecto, age como implementação física do conceito de abstração, não permitindo que os níveis superiores tenham acesso aos detalhes de funcionamento do módulo. O emprego de módulos, portanto, se constitui em valioso subsídio na busca de qualidade para o sistema de software.

**Módulos em CLP's Convencionais**

No entanto, convém ressaltar que até o momento não temos conhecimento de nenhum CLP comercial (no Brasil) que ofereça este recurso de programação. O que pode ser encontrado da lista acima, no melhor dos casos, se restrige a sub-rotinas, algumas estruturas de dados básicas e associação de símbolos a endereços. Isto, porém, não chega a inviabilizar a programação estruturada para os CLP, pois tanto a modularidade como a programação estruturada devem ser tomadas como filosofias de projeto, a serem seguidas da melhor forma possível, dentro das limitações do sistema de programação de cada equipamento.

Vejamos como isto poderia ser feito em CLP's convencionais. Para isto, vamos novamente utilizar a planta hipotética da figura 2.1. Pensando inicialmente em um equipamento sem qualquer dos recursos listados acima, seríamos obrigados a montar a estrutura hierárquica esquematizada na figura 4.1, para o grafo MFG que foi descrito na figura 3.3.

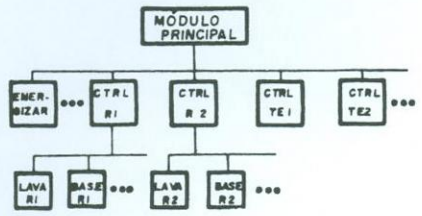


**Fig. 4.1: Estrutura de Módulos Dedicados.**

O sistema é composto por "módulos" específicos para cada uma das atividades. Assim, há por exemplo um módulo para o controle do reator 1 e outro para o reator 2, cada um deles com seus próprios módulos para lavagem, transporte de base, etc. Obviamente, não existe nenhum mecanismo que garanta o encapsulamento das camadas inferiores: o programa do "módulo principal" pode acessar livremente, por exemplo, os bits de controle de estados das máquinas que coordenam as atividades. É de responsabilidade do programador evitar que este tipo de coisa aconteça.

Note agora que as ações envolvidas no controle do reator 1 e do reator 2 são essencialmente as mesmas: mudam as válvulas, bombas e parâmetros do reator, mas a lógica de controle exigida é a mesma. Isto é, para transportar base para um dos reatores, por exemplo, sempre a bomba de base deve ser acionada e uma válvula (a do reator 1 ou 2) aberta.

### Emprego de Sub-rotinas

Suponha então que nos seja oferecida a possibilidade de se criar sub-rotinas dentro do software do CLP. Com isto, podemos implementar um único módulo, capaz de controlar indistintamente ambos reatores. Analogamente, verifica-se as mesmas características para o controle dos dois tanques de estoque. A figura 4.2 mostra a estrutura que o software passaria a ter.

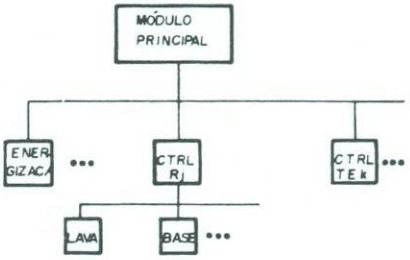


**Figura 4.2: Estrutura de Módulos Funcionais.**

Tem-se agora apenas um conjunto de módulos para o controle dos dois reatores e outro para os dois tanques de estoque. Vejamos como isto é feito, pensando no caso dos reatores. Os módulos de controle são acionados duas vezes em uma mesma *varredura* (ciclo de operação) quando ambos os reatores estão operando em paralelo: uma para o reator 1 e outra para o segundo. Lembrando que em um dado instante estes podem se encontrar em estados completamente diferentes, e que a máquina de controle é comum a ambos, o **contexto** de um reator deve ser salvo a cada varredura, de forma que possa ser corretamente recuperado na varredura seguinte.

Por contexto de um reator entendam-se os dados que definem o seu estado corrente, como o valor de suas saídas, entradas, atividade em andamento, etc. Como os módulos são acionados duas vezes em contextos diferentes, garante-se o perfeito funcionamento das duas linhas.

Seria algo como projetar dois filmes com um único projetor para dois espectadores diferentes: projeta-se um fotograma da fita 1 enquanto se tapa os olhos do espectador 2, troca-se a fita, projeta-se um fotograma agora da segunda fita sem deixar que o espectador 1 veja; volta-se a fita 1 exatamente na fotograma seguinte e repete-se o ciclo. Acreditando-se que as trocas possam ser feitas de forma muito rápida, tem-se que ambos assistirão perfeitamente seus filmes, sem qualquer interferência entre as imagens.

Este mecanismo, onde um mesmo segmento de programa é submetido concomitantemente a mais de uma tarefa, é conhecido como **reentrância** (Deitel, 84). No caso do CLP, tem-se uma simplificação: os módulos reentrantes, a cada chamada, são executados do início até o fim, sem interrupções, ao contrário do caso geral onde uma troca de contexto pode ocorrer a qualquer instante. Desta forma, temos apenas que nos preocupar em restaurar o contexto de uma das tarefas antes de acionar o módulo reentrante e novamente salvá-lo (com as atualizações efetuadas) após o término da execução.

### Imagens dos Pontos e Áreas de Trabalho

A implementação de código reentrante em CLP's deve ser rodeada de algumas precauções. Uma delas é dada pela dificuldade de se organizar as entradas e saídas em estruturas que possam ser endereçadas de forma indireta por um módulo. Isto é necessário, pois uma rotina reentrante deve funcionar para qualquer uma de suas atividades, e não pode portanto acessar um ponto de E/S de forma direta. Como exemplo, note que um transporte de base para o reator 1 envolve o acionamento da válvula **VB1**, enquanto que o mesmo transporte para o segundo reator deve acionar a válvula **VB2**. Assim, a rotina de transporte deve acionar uma válvula genérica **VBj**, onde **j** será igual a 1 ou 2, dependendo desta rotina estar rodando para o reator 1 ou 2.

Uma forma de superar este problema é fazer com que tais rotinas se limitem a tratar **imagens** dos pontos que mudam de uma atividade para outra. As imagens são posições de memória, tratadas como pontos de E/S pelo programa, mas que no entanto não se encontram conectados a qualquer dispositivo. Uma imagem seria empregada, por exemplo, para implementar a válvula genérica (e hipotética) **VBj**.

Desta forma, a chamada a uma rotina reentrante deve ser precedida de movimentações do valor dos pontos E/S da atividade em execução para as respectivas imagens; e ao término da execução outra movimentação deve ser feita em sentido contrário.

O conceito das imagens pode ser estendido a outros dados que também contribuam para formar o contexto das atividades. Informações como bits de estado, de sinalização, valores internos, constantes; enfim, tudo que não possa ser facilmente endereçado de forma indireta pela rotina reentrante. Estas informações e outras mais que não variem entre uma atividade e outra (e que portanto não pertençam a um contexto) devem ser, na medida do possível, agrupadas e organizadas, constituindo assim uma *área de trabalho*.

Em torno de uma área de trabalho, rotinas (reentrantes ou não) são agrupadas, formando um pequeno "módulo" de software. Se durante a criação do siste-

ma, mantiver-se o cuidado de não permitir que outras rotinas acessem diretamente esta área, teremos uma implementação a nível lógico do conceito de encapsulamento. Com isto, conseguimos construir o software em módulos funcionais, e desfrutar as vantagens que estes proporcionam.

## 5. NOTAS FINAIS

As idéias discutidas neste trabalho foram aplicadas na prática pelos autores em uma planta piloto que se encontra em construção nas dependências do Dep. de Engenharia Mecânica da Escola Politécnica da USP, em convênio com a Andersen Consulting.

A planta real é semelhante ao exemplo utilizado, sendo porém mais complexa; além dos elementos descritos no exemplo, conta com outros tanques adicionais. As ordens de produção não são fixas: são passadas na forma de receitas ao controlador por um microcomputador, através de uma rede de comunicação. A rede ainda permite o operador do micro monitorar o processo e controlar o início de operação, recuperação de falhas, e paradas (de emergência ou não) da planta.

Os resultados alcançados no desenvolvimento do software do CLP mostraram-se muito satisfatórios. Todo o sistema consumiu 3 kbytes de memória e 4 meses para ser projetado. escrito e testado.

A continuidade deste trabalho envolverá a melhor formalização dos elementos aqui descritos. Especial atenção deverá ser dispendida à concepção do nível de macro-controle. Vê-se que no tratamento de sistemas de complexidade extremamente elevadas, o poder de síntese do MFG pode ser aprimorado. Para estes casos, novas técnicas vêm sendo investigadas (Miyagi, 88); busca-se agora formas de transformá-las em ferramentas efetivas de projeto, que possam ser aplicadas na implementação do controle final dos processos, com o emprego de CLP's ou de qualquer outro equipamento que venha a ser concebido para este fim.

## REFERÊNCIAS BIBLIOGRÁFICAS


Deitel, H. M. (1984). **An Introduction to Operating Systems**, Addison-Wesley, USA.

Jensen, R.W. (1981). "Structured Programming". **IEEE Computer**, v.14, n.3, March, pp:31-48.

Masuda, R. & Hasegawa, K. (1981). "Mark Flow Graph and Its Application to Complex Sequential Control System", **13th Hawaii International Conference on System Science**, pp:194-203.

Miyagi, P.E. & Furukawa, C.M. (1988). "Mark Flow Graph e Production Flow Schema: Unificação e Estruturação de Controle". **Anais do 3o. Congresso Nacional de Automação Industrial (CONAI)**, São Paulo, SP.

Miyagi, P.E. et al (1989). "Mark Flow Graph (MFG) para o Modelamento e Controle de Sistemas de Eventos Discretos". **Monografias em Automação e Inteligência Artificial**, DEM/DEE, EPUSP, v.1, n1.

Peterson, J.L. (1977). "Petri Nets". **Computing Surveys**, v9, n3, pp:223-252, Sept.

Peterson, J.L. (1981). **Petri-net Theory and the Modelling of Systems**, Prentice-Hall, Englewood Cliffs, NJ.

Sekiguchi, T. et al (1988). **Sequential Control Engineering**. Society of Electrical Engineering (em japonês).

# 8º CONGRESSO BRASILEIRO DE AUTOMÁTICA



## Anais Vol. 1

de 10 a 14 de setembro de 1990
Centro Cultural Tancredo Neves
Belém - Pará - Brasil



SBA

IFAC