MODELAMENTO DE PROCESSOS DIGITAIS COM REDES DE INSTÂNCIAS

por

Flávio Rech Wagner

RT nº 006 CPGCC/UFRGS março/84

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

SUMÁRIO

RESUMO

Este relatório apresenta o conceito de redes de instâncias. É mostrado que sistemas onde se desenrolam processos digitais podem ser modelados como redes de instâncias. Redes de Petri são utilizadas para descrever a estrutura de causalidade de sistemas assim modelados. O conceito de instância é generalizado para o de instância programável, para se permitir o modelamento de blocos obtidos na partição de um sistema que executa um programa qualquer (possivelmente não-sequencial). É mostrado que sistemas digitais podem ser modelados por redes de instâncias em diferentes níveis de abstração e realização.

Palavras-chave: processos digitais, redes de instâncias, redes de Petri, sistemas programados, instâncias programadas, realização direta e indireta de redes de instâncias.

ABSTRACT

This report presents the concept of nets of agencies. It is shown that systems, where digital processes are ocurring, can be modelled as nets of agencies. Petri nets are used to describe the structure of causality of systems modelled in this way. The concept of agency is generalized to that of programmable agencies, in order to permit the modelling of blocks obtained in the partition of systems which execute programs (possibly non-sequential ones). It is shown that digital systems can be modelled by nets of agencies at different levels of abstraction and implementation.

Keywords: digital processes, nets of agencies, Petri nets, programmed systems, programmed agencies, direct and indirect realization of nets of agencies.

1. REDES DE INSTÂNCIAS

Redes de instâncias [WEN 82] [WEN 83] são grafos bipartites, onde existem elementos de duas classes: instâncias e memórias. Instâncias são elementos ativos, que como resultado de uma função qualquer modificam o conteúdo das memórias. Estas são elementos passivos, sem capacidade de processamento, que mostram imediatamente um novo valor de acordo com as ordens vindas das instâncias. Memórias são locais de observação das alterações no estado do sistema modelado pela rede de instâncias. Elas não precisam corresponder necessariamente a memórias do sistema real sendo modelado. Se temos um sistema de software, então os locais de observação são as variáveis do programa, que realmente correspondem a memórias. Se por outro lado temos um sistema de hardware, locais de observação são por exemplo os sinais de saída das portas lógicas, que não correspondem a memórias reais. Para que o valor destes sinais seja no entanto observável no modelo, ele é armazenado numa variável, i.e., o sinal é modelado por uma memória. Por este motivo as denominações "memória" e "variável" são usadas indistintamente.

Para salientar o caráter de grafo bipartite da rede, instâncias são representadas por retângulos e memórias por círculos.

Instâncias se comunicam com o resto do sistema sempre através de suas variáveis (memórias) de ligação, como na Fig. 1a. A função de uma instância pode ser explicada pelo seu comportamento de entrada/saída, i.e., através das possíveis relações entre os valores de suas variáveis de ligação. Variáveis de estado podem precisar ser introduzidas para que a função da instância seja explicável. A interconexão de variáveis de ligação de diferentes instâncias significa uma igualdade de tipo e valor, em todos os instantes de tempo, destas variáveis de ligação. Por este motivo pode-se repretar variáveis de ligação interconectadas como uma única variável, como na Fig. 1b.

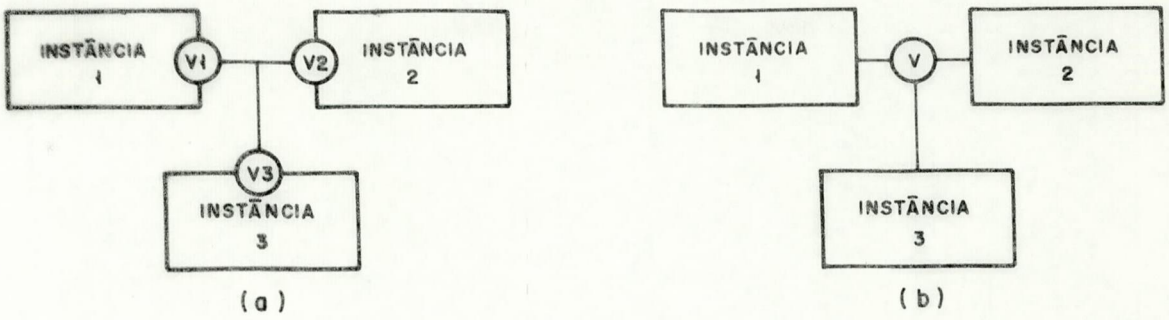


Figura 1

A representação da Fig. 1 não ilustra os tipos de relações entre instâncias e variáveis. Para isto utiliza-se um grafo de fluxo de dados, como na Fig. 2. Uma instância tem 3 tipos diferentes de acesso a variáveis do sistema: acesso de leitura, quando a instância só pode ler o conteúdo da variável, acesso de escrita, quando a instância só pode escrever um novo conteúdo na variável, sem conhecer o seu conteúdo anterior, e acesso de modificação, quando a instância escreve um novo conteúdo que pode depender do anterior. Denomina-se entorno de uma instância o conjunto de todas as variáveis do sistema às quais a instância tem acesso. Nesta definição não se inclui as variáveis de estado da instância.

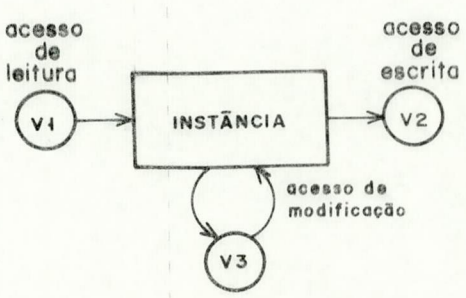
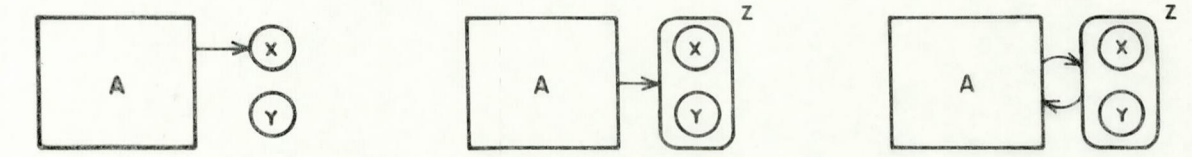


Figura 2- Grafo de fluxo de dados

A Fig. 3 explica a necessidade de se introduzir o conceito de acesso de modificação. A instância A tem acesso de escrita à variável X e nenhum acesso à variável Y, como se vê na Fig. 3a. Por razões de modelamento se quer unificar as variáveis X e Y numa única variável Z. A nova situação não pode ser modelada como na Fig. 3b, pois se teria a errônea interpretação de que A pode escrever um novo conteúdo em Y. É preciso que a ação de A sobre Z mantenha o valor de Y inalterado. Isto só é possível se imaginarmos que A pode ler o valor de Z e escrever um novo, no qual parte de Z (correspondente a Y) é mantida com valor idêntico ao anterior. A Fig. 3c modela corretamente esta situação.



(a) variáveis isoladas

(b) variáveis concatenadas, modelamento incorreto

(c) variáveis concatenadas, modelamento correto

Figura 3

Suponhamos que X e Y sejam valores binários, concatenados para formar uma variável Z de 2 bits, onde X é o bit mais significativo. A ação X ← 0 da instância A corresponde a uma ação Z ← Z AND 01, necessária para se preservar o valor de Y. Aqui reconhece-se a necessidade de A precisar modificar Z, embora ela só possa escrever em X.

Por processos digitais [WEN 83] entendemos aqueles nos quais as variáveis têm valores discretos e as alterações no estado do sistema ocorrem em instantes de tempo que também só assumem valores discretos. Alterações discretas no estado do sistema são eventos. Processos digitais são portanto seqüências de eventos. Dois eventos podem ser independentes entre si, e portanto ocorrer em qualquer relação

de tempo, ou podem ter uma relação de causalidade. Um processo digital em particular pode ser descrito se for feito o protocolo de todos os eventos nele acontecidos. Uma classe de processos digitais (conjunto de processos possíveis de ocorrer num dado sistema), por seu turno, pode ser descrita se conhecermos a estrutura de causalidade que explica a geração de todos os eventos no sistema de acordo com as relações de causalidade entre eles. A partir da estrutura de causalidade pode-se portanto prever todas as seqüências possíveis de eventos para o sistema.

Redes de instâncias servem ao modelamento de sistemas onde se desenrolam processos digitais, devido a duas propriedades suas: 1) as memórias podem armazenar o estado do sistema e pode-se portanto observar a ocorrência de todos os eventos do processo pela observação do conteúdo das memórias (variáveis de estado das instâncias também são modeladas como memórias); 2) as instâncias são capazes de gerar os eventos de acordo com a estrutura de causalidade do sistema. Para isto elas precisam alterar os valores das memórias. Define-se domínio de responsabilidade de uma instância como o conjunto de variáveis do sistema por cujos valores a instância é responsável num dado instante de tempo. O domínio de responsabilidade de uma instância deve ser obviamente um subconjunto do entorno da instância, e necessariamente um subconjunto do conjunto de variáveis às quais a instância tem acesso de escrita ou de modificação. A função de uma instância pode então ser definida através de duas funções de atribuição de valores:

novo valor das variáveis do domínio de responsabilidade em $t_{i+1} := \delta$(entorno da instância e variáveis de estado em t_i),

e

novo valor das variáveis de estado em $t_{i+1} :=$ ω (entorno da instância e variáveis de estado em t_i)

Nestas funções pode-se reconhecer as funções δ e ω de um autômato genérico. Através do conjunto de funções δ e ω de todas as instâncias presentes no sistema tem-se a estrutura de causalidade que define o processo digital.

Os níveis tradicionais de projeto de sistemas digitais programáveis correspondem todos a processos digitais (excetuando-se o nível de circuitos elétricos), e podem portanto sempre ser modelados como redes de instâncias. Na Fig. 4 vemos que instâncias e memórias podem ser elementos muito simples, como portas lógicas combinacionais e sinais lógicos, no nível lógico, ou muito complexos, como uma unidade de recuperação de informações e um banco de dados, no nível de software de aplicação, passando por elementos de complexidade média, como registradores e vetores binários, no nível de transferência entre registradores (RT).

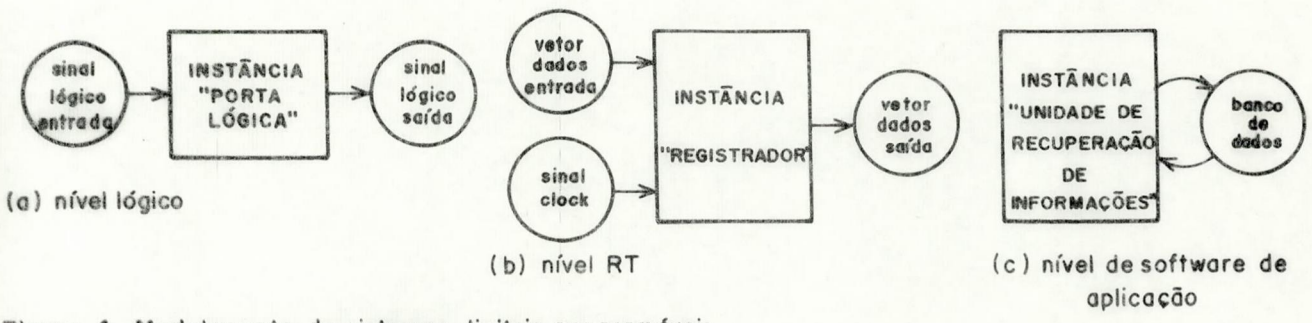


Figura 4 - Modelamento de sistemas digitais programáveis

Deve ficar bem claro que uma rede de instâncias é apenas um modelo estático de um sistema. Para que se tenha a descrição completa do sistema, é necessário acrescentar a esta visão estática a estrutura de causalidade dos eventos que ocorrem no sistema. Em outras palavras, é preciso que se descreva a função das instâncias. A Fig. 5 mostra um sis

tema composto por 3 instâncias, "produtor", "consumidor" e "observador", que se comunicam através de uma variável V, com domínio de valores {vazio, ocupado e não observado, ocupado e observado}. Esta figura não nos diz ainda nada a respeito de como atuam as instâncias. A Fig. 6 mostra 2 possíveis estruturas de causalidade para este sistema. Faz-se uso de redes de Petri [PET 76] para se descrever estruturas de causalidade.

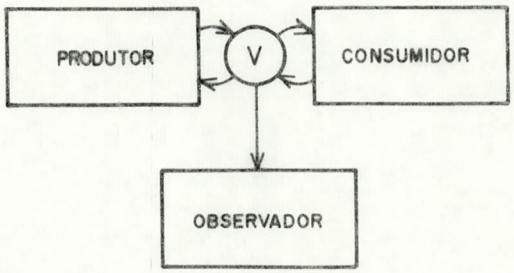
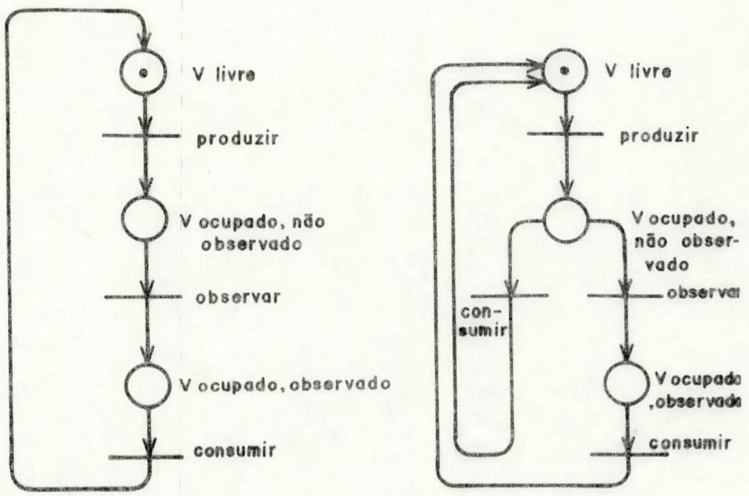


Figura 5



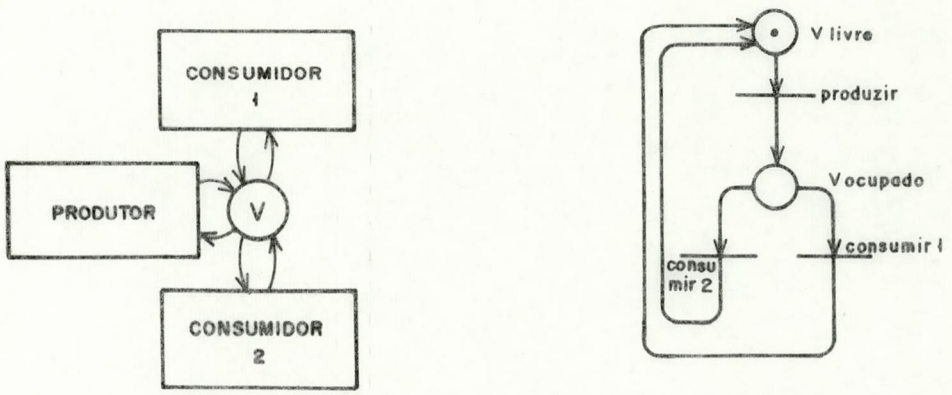
(a) relação causal entre as ações "observar" e "consumir"

(b) ações "observar" e "consumir" são independentes

Figura 6 - Estruturas de causalidade para o sistema da figura 5

Na estrutura da Fig. 6a, impõe-se uma relação de causalidade entre as ações "observar" e "consumir". Na estrutura da Fig. 6b esta relação não existe, e uma ação "consumir" pode ocorrer mesmo sem a ocorrência da ação "observar". Neste caso diz-se haver um conflito de reconhecimento [WEN 82] entre as ações "consumir" e "observar". A ocorrência de uma delas (consumir) destrói a condição para a existência da outra, mas o inverso não é verdadeiro.

Redes de instâncias podem possuir também conflitos de responsabilidade [WEN 82], quando os domínios de responsabilidade de duas ou mais instâncias se superpõem num dado instante. Este é o caso do sistema da Fig. 7a, onde existem duas instâncias "consumidor". O conflito é reconhecido na estrutura de causalidade na Fig. 7b. A ocorrência de qualquer uma das ações "consumir" destrói a condição para o disparo da transição correspondente à outra ação "consumir".



(a) grafo de fluxo de dados (b) estrutura de causalidade

Figura 7- Sistema com conflito de responsabilidade

Domínios de responsabilidade podem ser variantes ou invariantes no tempo. No sistema da Fig. 5 a variável V em diferentes instantes de tempo pertence ao domínio de responsabilidade de diferentes instâncias. Segundo a estrutura de causalidade da Fig. 6a, V está no domínio de responsabilidade

- do produtor, quando o seu valor é "livre",
- do observador, quando o seu valor é "ocupado, não observado", ou
- do consumidor, quando o seu valor é "ocupado, observado".

Importante é que a cada instante de tempo o conjunto dos domínios de responsabilidade seja uma partição do conjunto de variáveis do sistema. Em caso contrário temos um conflito de responsabilidade.

Redes de instâncias correspondentes a sistemas digitais descritos no nível lógico e no nível RT têm domínios de responsabilidade invariantes no tempo. Cada porta lógica ou elemento do nível RT é sempre responsável pelas suas variáveis de saída e nunca responsável por suas variáveis de entrada.

2. INSTÂNCIAS PROGRAMÁVEIS

Neste capítulo iremos tratar do modelamento de sistemas cuja função é dada por um programa não-sequencial. Programas não sequenciais podem ser representados por redes de Petri [WEN 79], onde transições correspondem a instruções do programa e conflitos na rede são resolvidos testando-se variáveis do programa. A Fig. 8 dá o exemplo de um programa assim modelado.

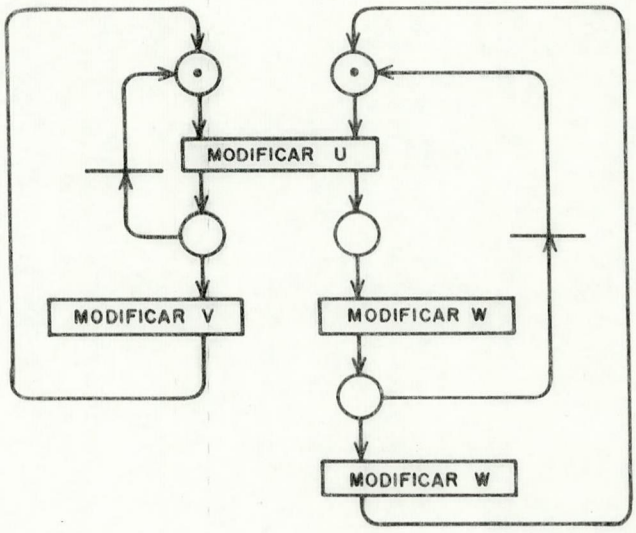


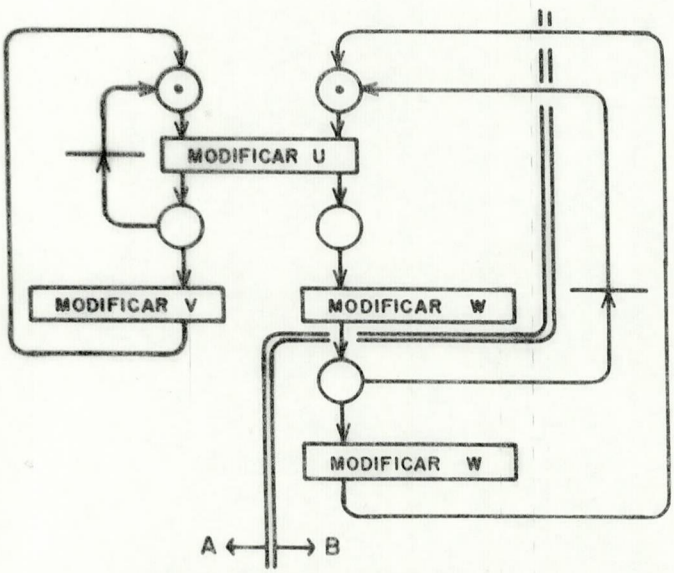
Figura 8 - Programa não sequencial modelado por redes de Petri

No caso de um sistema complexo, pode-se particionar o programa em blocos, de acordo com critérios semânticos, de modo a se tratar problemas de complexidade menor. A Fig. 9a mostra uma possível partição do programa da Fig. 8. Blocos obtidos numa tal partição podem

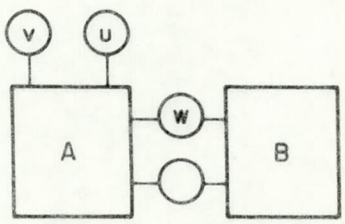
a. ter acesso a variáveis comuns, e
b. alterar a marcação de outros blocos.

Para a representação da comunicação entre blocos, introduz-se canais. Para cada variável do programa existe

um canal. No exemplo da Fig. 9, vemos que apenas o bloco A tem acesso às variáveis u e v, mas que tanto A como B tem acesso à variável w, estabelecendo-se assim uma comunicação entre ambos. Este canal não é no entanto suficiente para descrever a comunicação entre A e B. Há 2 alternativas para a ação de B: modificar w ou não fazer nada. O bloco A não pode portanto decidir se B terminou ou não sua ação, apenas olhando para a variável w.



(a) partição do programa da figura 8



(b) comunicação entre os blocos

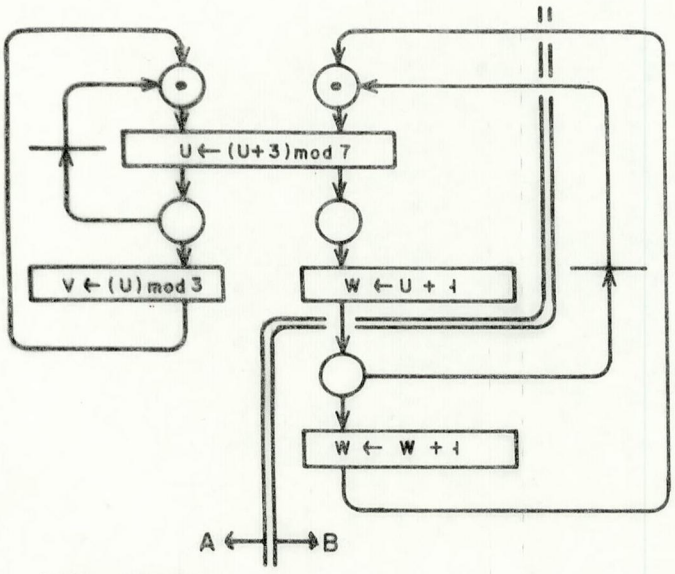
Figura 9

O canal adicional introduzido para representar a comunicação entre A e B corresponde à colocação pelo bloco B de uma marca no lugar L do programa do bloco A. Temos assim 2 tipos de comunicação entre os blocos:

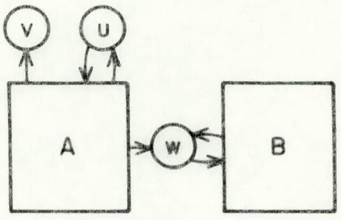
- comunicação por operandos, quando blocos têm acesso a variáveis comuns, e

- comunicação por marcas, quando um bloco altera a marcação da rede de Petri de outro bloco.

Um grafo de fluxo de dados, tal como introduzido no capítulo anterior, não pode modelar esta situação. Na Fig. 10 temos um exemplo concreto de programa para o sistema da Fig. 8 e o grafo de fluxo de dados correspondente. O canal para o modelamento da comunicação por marcas não aparece neste grafo. Em outras palavras, redes de instâncias prevêm apenas comunicação por operandos entre as instâncias.



(a) programa



(b) grafo de fluxo de dados

Figura 10

Para modelar a nova situação, introduz-se o conceito de instâncias programadas [WEN 80] que podem ter entre si tanto comunicação por operandos como comunicação por marcas. Uma instância programada tem genericamente a estrutura interna indicada na Fig. 11. Ela possui uma memória P para o programa (possivelmente não-sequencial) e uma unidade de execução do programa. Como programas são representados por redes de Petri, a instância programada necessita ainda uma memória M para a marcação da rede. Note-se que as variáveis do programa residem fora das instâncias programadas. A unidade de execução pode ter uma memória auxiliar interna para permitir a sequencialização da execução de uma instru-

ção, mas no âmbito deste trabalho supõe-se a unidade de execução capaz de executar uma instrução num único passo.

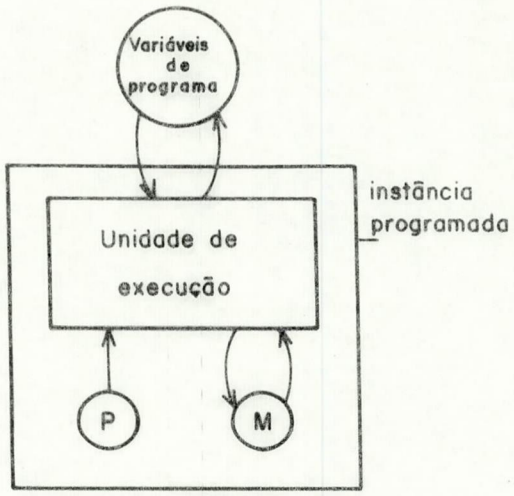
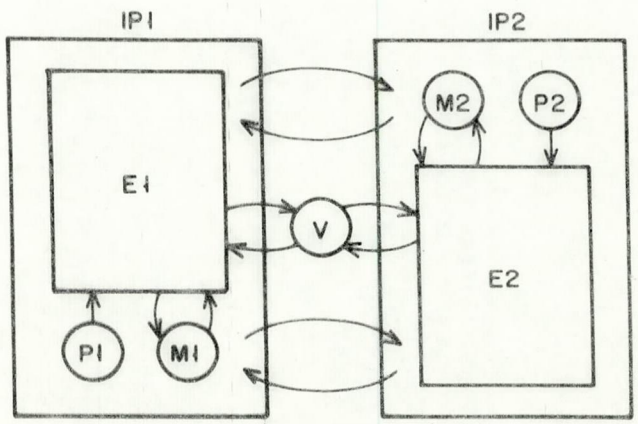


Figura 11

A Fig. 12 mostra a comunicação entre duas instâncias programadas IP1 e IP2. Comunicação por operandos significa que ambas unidades de execução E1 e E2 têm acesso a uma variável de programa comum V. Comunicação por marcas significa que uma unidade de execução tem acesso à memória M da outra instância programada. Note-se que num nível maior de detalhamento uma instância programada é modelada como uma rede de instâncias e que comunicação por marcas entre instâncias programadas significa comunicação por operandos entre instâncias.



Fi - Comunicação entre instâncias programadas

3. MODELAMENTO DE SISTEMAS DIGITAIS PROGRAMADOS

A Fig. 13 mostra o exemplo de um sistema que executa um programa não-sequencial contendo 4 instruções A, B, C e D [WEN 83]. Na Fig. 13a este sistema foi particionado de modo a resultarem dois programas sequenciais que se comunicam entre si. Supondo que as instruções A e B façam referência a variáveis locais L_1, diversas daquelas referidas por C e D (L_2), vê-se na Fig. 13b que entre as instâncias programadas correspondentes a estes dois programas existe apenas comunicação por marcas.

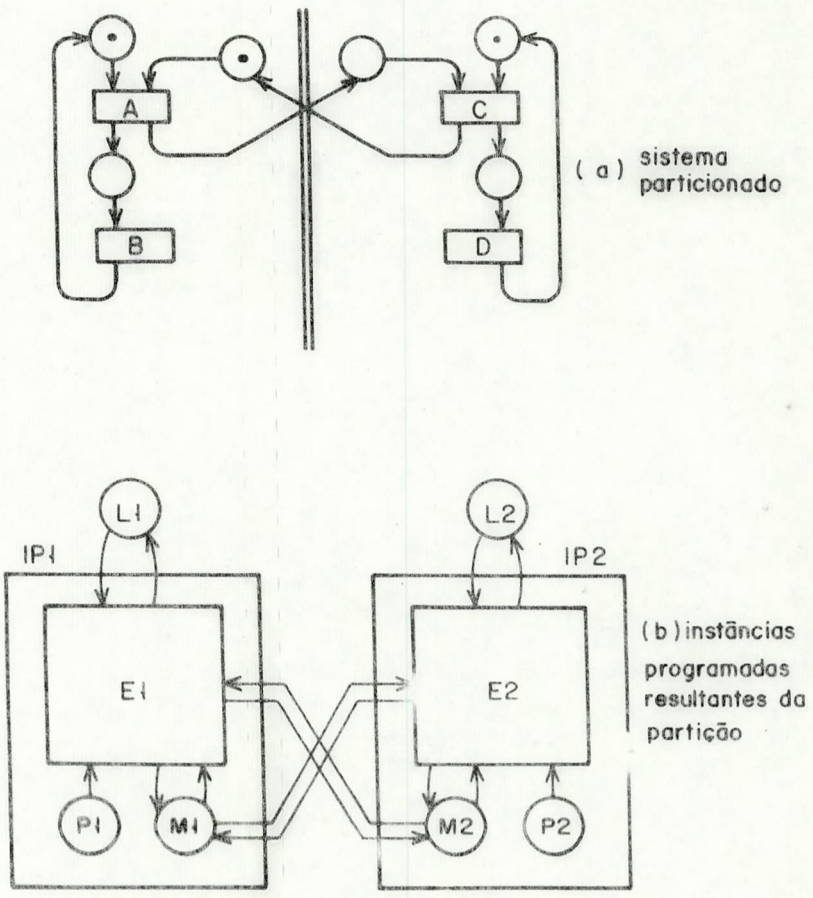


Figura 13

Imagine-se agora que se deseja implementar este sistema. Se temos um multiprocessador, IP1 e IP2 podem ser implementadas em dois processadores diferentes (ver relações entre instâncias programadas e processadores em [WEN 79]), e assim a Fig. 13b é um modelo bastante aproximado da situação real, a menos dos mecanismos de sincronização efetivamente utilizados. Se temos um sistema monoprocessador multiprogramado, a Fig. 13b pode ser um modelo adequado para um certo nível de abstração, no caso o nível no qual o programador enxerga o sistema (mais uma vez, a menos do mecanismo de sincronização). Imagine-se que sejam utilizados semáforos [DIJ 68] para a implementação da comunicação entre processos no monoprocessador. O programador enxerga uma máquina virtual onde as operações P e V são primitivas. Na máquina onde esta máquina virtual está implementada a execução das operações P e V deve significar a passagem do controle para uma terceira instância programada a ser introduzida no sistema com o objetivo de sequencializar as atividades. Esta instância pode ser denominada "administrador de semáforos". A Fig. 14 mostra então a estrutura de causalidade e as instâncias programadas desta máquina.

A máquina da Fig. 14b possui 3 instâncias programadas, o que supõe a existência de 3 unidades de execução. Numa realização direta [WEN 80] de um sistema especificado como uma rede de instâncias (programadas ou não), existe um mapeamento 1:1 entre as instâncias da estrutura lógica (i.e., do modelo) e as instâncias da estrutura física. No caso da Fig. 14b uma realização direta significaria uma máquina real contendo 3 processadores, em cada um dos quais seria implementada uma instância programada. Numa realização indireta [WEN 80] de um sistema especificado como uma rede de instâncias, utiliza-se o princípio da multiplexação: uma instância da estrutura física faz o papel de diferentes instâncias da estrutura lógica em diferentes instantes de tempo. A Fig. 15 mostra um sistema de recuperação de informações em um banco de dados [WEN 83]. A Fig. 15a mostra o sistema tal como visto pelos usuários. Cada um enxerga uma ins-

tância programada de recuperação de informações com acesso exclusivo ao banco de dados. Cada instância possui uma área

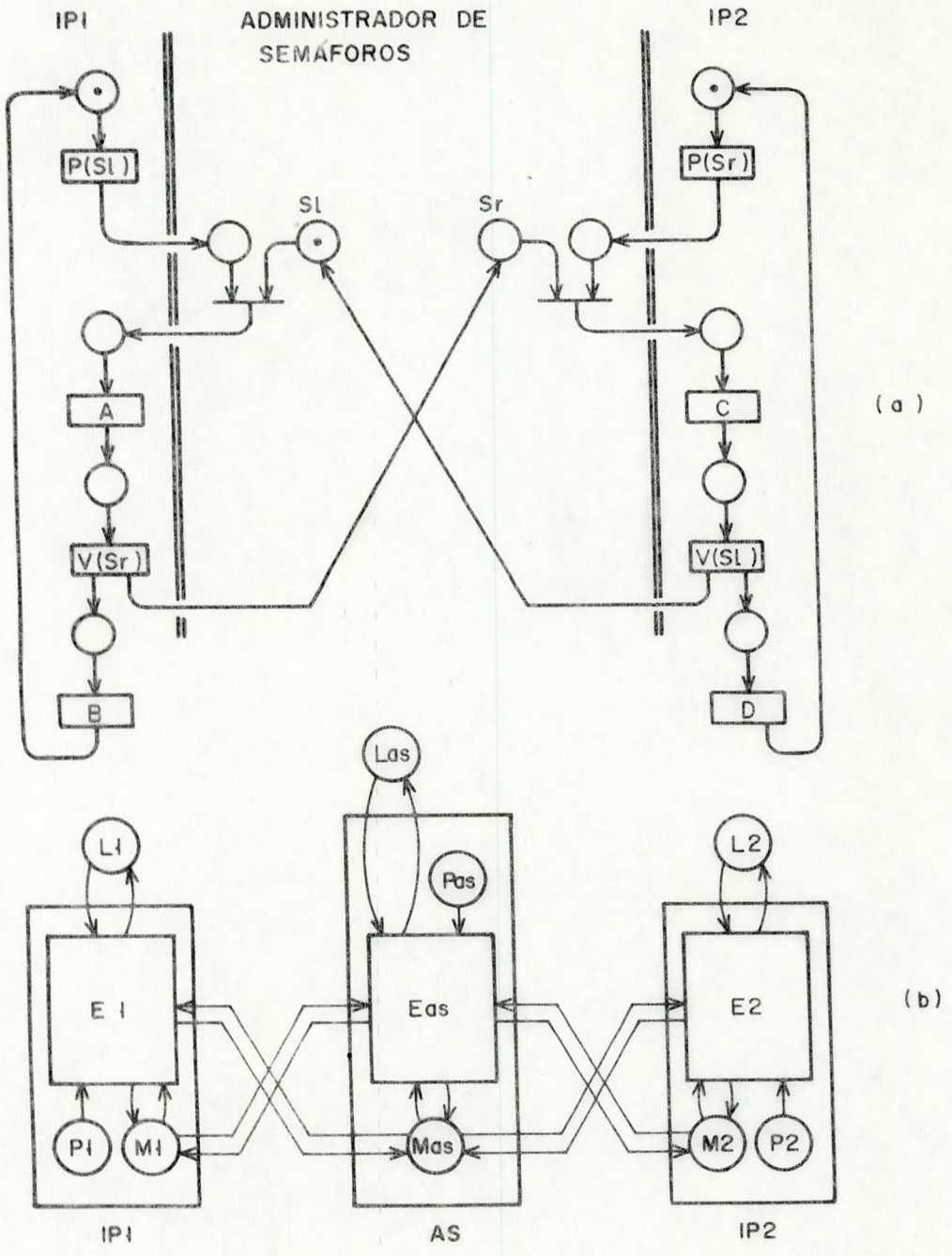


Figura 14

própria de trabalho L_i e um canal de comunicação K_i com o usuário. Tal sistema não é economicamente viável com uma rea

lização direta deste modelo, devido à existência de **n** unidades de execução. Na Fig 15b o sistema é realizado com uma única instância recuperadora de informações, que é multiplexada entre os usuários. Esta instância executa um programa P_R, idêntico para todos os usuários, e que portanto não precisa ser alterado nunca. As memórias para a marcação M_R e para a área de trabalho L_R são multiplexadas entre todas as instâncias de recuperação, por ser o seu conteúdo específico para cada uma destas. Introduz-se então uma instância de multiplexação. Esta instância recebe os pedidos de informação dos usuários e tem acesso às memórias M e L de todos os usuários. Ela decide dentre os pedidos qual será atendido a seguir (função de "dispatch"), traz para as memórias M_R e L_R o conteúdo das memórias M_i e L_i do usuário a ser atendido e passa o pedido (através de um canal genérico K_R) e o controle à instância genérica de recuperação.

O sistema poderia agora ser implementado quase diretamente a partir do modelo da Fig. 15b. A instância genérica de recuperação seria implementada com um processador para seu programa P. A instância de multiplexação é não-programada e seria implementada diretamente em hardware. O modelo da Fig. 15b no entanto ainda não seria diretamente realizável por não considerar os problemas de entrada/saída, especialmente o acesso à memória secundária.

Em [WEN 80] é mostrada uma outra estrutura para este sistema, na qual a instância de multiplexação é dividida em 2 instâncias. Uma é implementável em hardware e responsável pela entrada/saída (atendimento dos pedidos dos usuários e tratamento dos pedidos de acesso à memória secundária), sendo comparável ao sistema de interrupção de um processador ("hardware dispatch"). A outra é uma instância programada, responsável pela multiplexação da instância de recuperação de informações de acordo com os pedidos dos usuários. ("software dispatch"). Como o sistema passa a ter duas instâncias programadas, uma multiplexação adicional é neces

sária se ele deve ser implementado em um monoprocessador.

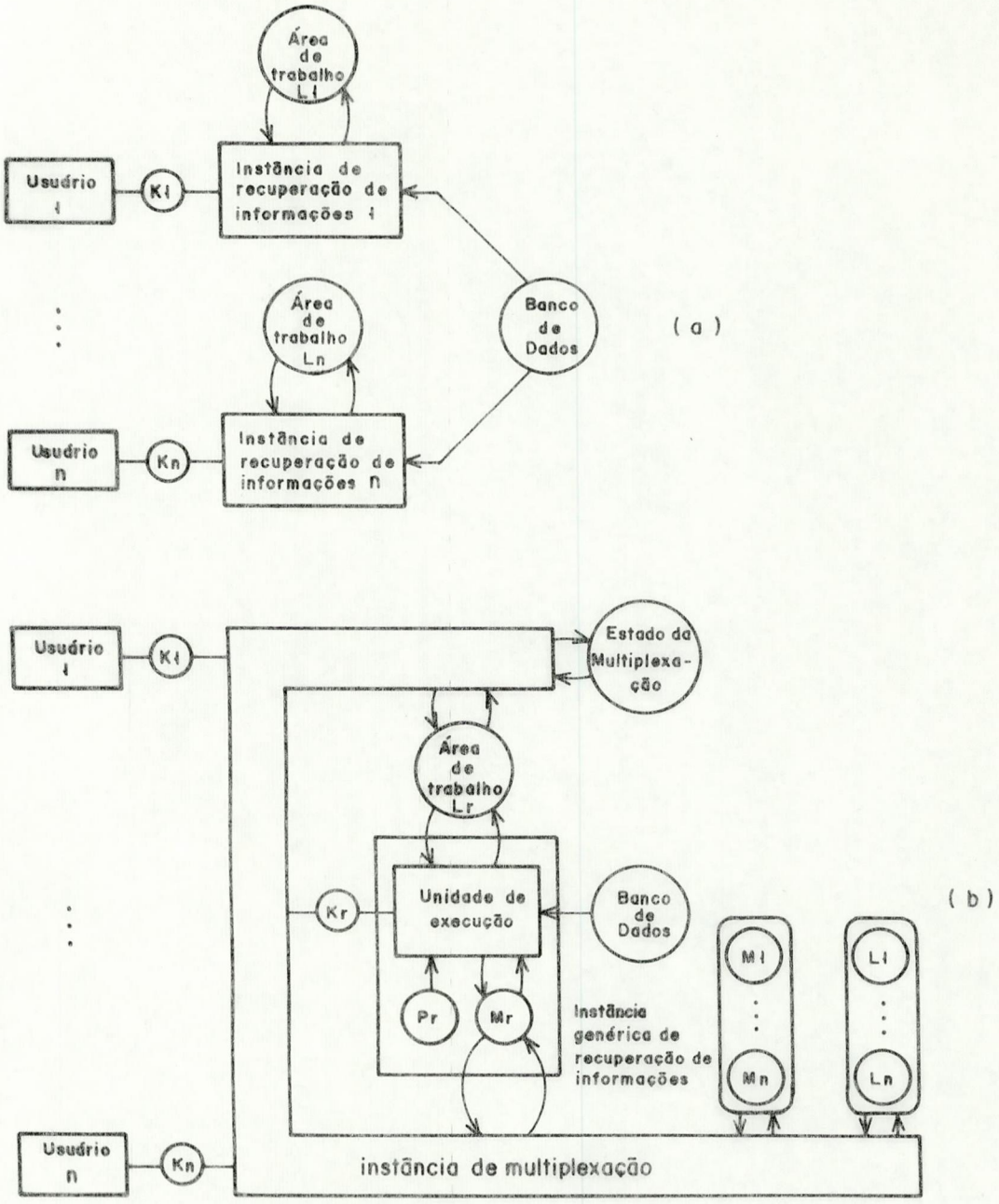


Figura 15

Uma última palavra deve ser dita a respeito da comunicação por marcas entre instâncias programadas. Esta comunicação implica em atividades concorrentes que devem ser sincronizadas. Um mecanismo de sincronização deve existir para implementar esta comunicação. Como neste trabalho utiliza-se redes de Petri para a descrição de estruturas de causalidade, pode-se modelar este mecanismo pelo acesso de uma instância à memória de marcação M de outra instância. Em um sistema real existiria um canal de comunicação entre as unidades de execução das 2 instâncias, que implementaria algum mecanismo de sincronização conhecido. A memória M passaria a ter a função única de armazenar o estado atual de execução do programa da instância (corresponderia ao contador de programa caso o programa fosse sequencial).

REFERÊNCIAS BIBLIOGRÁFICAS

[WEN 79] WENDT, Siegfried. The programmed action module: an element for system modelling. Digital Processes, 5(1979) 3-4.

[WEN 80] WENDT, Siegfried. On the partitioning of computing systems into communicating agencies. In: G. ZIMMERMANN (ed.) GI-NTG Fachtagung Struktur und Betrieb von Rechensystemen, Kiel (F.R.G.), March 19-21, 1980. Berlin, Springer Verlag, 1980. p.194-204.

[WEN 82] WENDT, Siegfried. Einführung in die Begriffswelt allgemeiner Netzsysteme. Regelungstechnik, 30. Jahrgang, Heft 1, 1982.

[WEN 83] WENDT, Siegfried. Grundlegende systemtechnische Begriffe und Modelle der Informationstechnik. Universität Kaiserslautern, Fachbereich Elektrotechnik. Bericht 83 B-14, Jan. 1983.

[DIJ 68] DIJKSTRA, E. W. Cooperating sequential processes. In: F. GENUYS (ed.) Programming Languages. Academic Press, London, 1968.

[PET 76] PETRI, Carl Adam. Kommunikationsdisziplinen. ISF-76-1, Gesellschaft für Mathematik und Datenverarbeitung. Bonn, 1976.