MODELAGEM E SIMULAÇÃO DISTRIBUÍDA DE SISTEMA PRODUTIVO BASEADOS EM REDE DE PETRI

Fabrício Junqueira[*]
fabri@usp.br

Paulo Eigi Miyagi[*]
pemiyagi@usp.br

[*]Escola Politécnica da USP
Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos
CEP 05508-030 São Paulo, SP

RESUMO

Em função da capacidade computacional instalada e da estrutura dispersa de alguns sistemas produtivos, existe interesse em soluções de modelagem e simulação distribuída destes sistemas. Isso é considerado como um recurso fundamental para o projeto, implementação e melhoria de desempenho desses sistemas produtivos. A idéia é que por meio de simulações com o uso de computadores fisicamente dispersos, mas integrados através de uma rede de comunicação, pode-se avaliar o comportamento de sistemas em concepção e melhorar resultados de plantas existentes. Este trabalho propõe assim, um procedimento para a modelagem de sistemas produtivos em ambientes distribuídos. Esse procedimento foi aplicado com sucesso a alguns estudos de caso onde sua eficácia foi confirmada. Este trabalho envolve ainda a proposta de um algoritmo para o gerenciamento da simulação distribuída. Com o método de modelagem e o algoritmo de gerenciamento tem-se os principais elementos para a implementação prática de um simulador de sistemas produtivos dispersos.

PALAVRAS-CHAVE: sistema produtivo disperso, simulação distribuída, rede de Petri.

ABSTRACT

Based on the existing computational capability and disperse infrastructure of some productive systems, there are interest for distributed modeling and simulation of these systems. These techniques are considered fundamental for design, implementation and performance improvement of productive systems. The approach is the simulation through the use of computers physically dispersed but integrated via a communication network to evaluate the behavior of systems still in conception level and also to improve the performance of existing plants. This work proposes a procedure for modeling of productive systems in distributed environment. This procedure was applied to case studies to confirm it effectiveness. The work includes an algorithm for the management of the distributed simulation. With the modeling method and the management algorithm we have the main elements for the practical implementation of the disperse productive system simulator.

KEYWORDS: disperse productive system, distributed simulation, Petri net.

1 INTRODUÇÃO

Algumas empresas do setor de manufatura vêm se estabelecendo de maneira distribuída e dispersa devido à globalização do mercado e a necessidade de atender demandas locais envolvendo inclusive fatores sócio-culturais. Eles aproveitam o crescimento e a capilaridade das redes de comunicação e da tecnologia da informação. Estas empresas deixam de ser consideradas como uma entidades isoladas, mas sim como partes de um consórcio de empresas cooperativas (Shi, Gregory, 1998) (Zhang, et al., 2006) (Shi et al., 2002), e este

Artigo submetido em 28/04/2008 (Id:872)
Revisado em 03/12/2008
Aceito sob recomendação do Editor Associado Prof. José Reinaldo Silva

novo padrão de relacionamento pode ser identificado como um novo paradigma de produção, chamado de rede de manufatura dispersa (Zhan et al., 2003) (Cheng, Cheng, 2006).

Uma rede de manufatura dispersa consiste de instalações físicas que estão geograficamente dispersas, mas precisam se comunicar e trabalhar cooperativamente trocando uma grande quantidade de informações e dados entre suas próprias fábricas, empresas terceirizadas e fornecedores. O projeto e operação deste tipo de sistema produtivo requerem uma abordagem de projeto distribuído, em que equipes geograficamente distribuídas em diferentes locais físicos colaboraram para a especificação do sistema e avaliação dos processos produtivos.

A evolução da tecnologia da informação e sua aplicação nos processos produtivos permitiram a integração de sistemas heterogêneos (Sanz, Alonso, 2001) (Sanz et al., 2003). Como um exemplo, um sistema supervisório industrial interage com um conjunto heterogêneo de *hardware* e *software* (por exemplo: estações de trabalho, unidades remotas, controladores programáveis, etc), a fim de monitorar e controlar um processo industrial. A integração de sistemas heterogêneos aumenta a complexidade na tarefa de analisar o sistema e requer o desenvolvimento de novas soluções. Entre elas, o uso da simulação distribuída merece atenção especial. A simulação distribuída trata da execução de programas computacionais em equipamentos geograficamente dispersos conectados através de uma rede de comunicação, o que pode ser visto como um tipo de supercomputador virtual (Fujimoto, 1999) (Banks, 2000) (Karatza, Theodoropoulos, 2006) (McLean, Riddick, 2001).

Neste contexto, o objetivo deste trabalho é apresentar uma nova abordagem de modelagem de sistemas própria para a simulação distribuída e, como esta simulação pode ser, de fato, implementada num ambiente disperso. Esta abordagem de modelagem baseia-se no conceito de orientação a objeto e rede de Petri associado a um procedimento de refinamento progressivo. E, para que os modelos sejam efetivamente integrados e simulados concomitantemente com outros modelos em um ambiente geograficamente distribuído, um algoritmo de gerenciamento da comunicação é também introduzido.

Na seção 2, conceitos básicos adotados neste trabalho são discutidos. A seção 3 apresenta o procedimento para a modelagem hierárquica de sistemas produtivos. A seção 4 apresenta um exemplo em que o procedimento é aplicado à modelagem de um sistema de transporte de material. Na seção 5 é apresentado um algoritmo para gerenciamento da comunicação para a simulação distribuída e dados de alguns experimentos são citados na seção 6. A seção 7 apresenta as considerações finais sobre este trabalho.

2 CONCEITOS BÁSICOS

Algumas razões para distribuir a simulação entre vários computadores são:

- Redução do tempo de execução de experimentos através da divisão do modelo de simulação entre os processadores;
- Possibilidade de trabalhar com modelos mais detalhados de cada processo já que a dimensão do modelo total não é mais um problema;
- Possibilidade de trabalhar com informações mais precisas de cada parte (cada planta) do sistema que podem ser modelados e simulados localmente sem obrigatoriamente necessitar de informações adicionais sobre esses modelos específicos de cada uma das partes envolvidas; e
- Aumento do grau de tolerância à falha, isto é, se um processador falhar, outros processadores podem continuar a simulação.

Preocupados com a complexidade desse tipo de simulação, alguns pesquisadores enfatizam o emprego de modelos reaproveitáveis baseados em componentes (Fujimoto, 1999) (Karatza, Theodoropoulos, 2006) (Kachitvichyanukul, 2001). Um componente pode ser selecionado a partir de um repositório e ser usado sozinho, ou ser combinado com outros para gerar um novo componente.

Um problema inerente da simulação distribuída é a partição do modelo entre os processadores, e algumas propostas utilizam o conhecimento prévio do sistema modelado para otimizar a simulação (Nevison, 1990). No entanto, a otimização da simulação torna-se inviável quando se trabalha com técnicas genéricas de modelagem e número indefinido de processadores. Neste caso, a técnica de modelagem não restringe o tipo de sistema produtivo em estudo. É, portanto, impossível a utilização de estratégias de otimização baseadas no conhecimento prévio sobre o sistema, bem como o número de processadores a ser utilizado na simulação.

Por outro lado, os modelos baseados em sistemas a eventos discretos (SED) são intensamente utilizados para descrever, analisar e controlar processos em sistemas produtivos. Estes sistemas são caracterizados por estados discretos e ocorrência de eventos instantâneos, que regem a sua dinâmica. A evolução destes sistemas é baseada em regras que definem as condições para a ocorrência de eventos, bem como o novo estado alcançado após um evento (Cassandras, Strickland, 1992). Neste contexto, a rede de Petri é uma técnica de modelagem gráfica e matemática que descreve com clareza

interações entre processos que caracterizam um SED (Murata, 1989) (Villani et al., 2007) e, explorando isto, ele tem sido utilizado intensamente para modelar sistemas dinâmicos em uma ampla gama de áreas (Davrazos, Koussoulas, 2007) (Miyagi, Riascos, 2006) (Tolba et al., 2005).

Os pesquisadores também apontam a modelagem hierarquizada como uma forma de lidar com sistemas de grande porte com grande número de interações entre seus elementos (Kachitvichyanukul, 2001) (Daum, Sargent, 1999, 2002) (Carullo et al., 2003) (Gomes, Barros, 2005). O modelador pode dividir este sistema em subsistemas (e conseqüentemente submodelos), que são relativamente mais fáceis de serem trabalhados. Os modelos podem ser gerados em diferentes níveis de abstração, ajudando a verificação e validação dos processos.

No domínio da simulação distribuída com rede de Petri, duas soluções são comumente consideradas: computadores paralelos e computadores em rede. No primeiro caso, o esforço de computação para simular um modelo de grande porte é distribuído em um ambiente multi-processado (Fujimoto, 2003) (Perumalla et al., 2005) (Nicol, Roy, 1991) (Chiola, Ferscha, 1993) (Kumar, Kohli, 1997) (Beraldi, Nigro, 1999). Contudo, a partição do sistema e a distribuição de processamento é transparente para os usuários. Do ponto de vista do usuário, a simulação é realizada como um sistema não-distribuído. Os principais inconvenientes desta solução são custos relativamente elevados e uma abordagem de modelo centralizadora. Alguns pesquisadores propõem a criação e manutenção de uma lista global de eventos que deverão ocorrer (Chiola, Ferscha, 1993) (Djemame et al., 1998). A lista é por sua vez classificada e dividida entre os vários processadores. Cada processador, em seguida, gerencia a sua própria lista local. A partição do modelo pode resultar em conflitos, como quando um estado é a pré ou pós-condição de dois ou mais eventos alocados em diferentes processadores. Neste caso, os processadores trocam um conjunto de mensagens para resolver o conflito preservando a causalidade entre os eventos. Entretanto, com os avanços da informática e de tecnologias de rede de comunicação, a segunda solução tem se tornado cada vez mais atraente (Nketsa, Valette, 2001). Uma das suas principais vantagens é a possibilidade de explorar a capacidade em geral ociosa de computadores dispersos geograficamente. Esta é a solução adotada no presente trabalho.

Um dos principais problemas da simulação distribuída com rede de Petri é como gerenciar a simulação de forma a garantir a consistência e coerência dos resultados. A solução proposta na literatura disponível pode ser agrupada em duas classes: a abordagem conservadora e a otimista. A abordagem conservadora evita erros de causalidade através do bloqueio da ocorrência de eventos que são considerados como inseguros, isto é, eventos que podem levar a uma situação onde as restrições de causalidade são violadas (Beraldi, Nigro (1999). Como resultado, todos os nós (computadores ou processadores) devem compartilhar o mesmo *clock* de simulação. Por outro lado, a abordagem otimista permite um certo grau de ocorrência de eventos inseguros e cada nó possui um *clock* virtual local. Quando as restrições de causalidade são violadas, a simulação deve reverter as últimas operações realizadas até que um estado seguro seja alcançado (o que é chamado de *rollback*). A simulação então recomeça a partir deste estado. Umas das implicações desta abordagem é que cada nó deve manter um registro de suas últimas operações. Para gerenciar o tamanho do registro, o *clock* virtual global indica o tempo do último estado seguro e limita a memória usada pelo algoritmo. Essa abordagem é normalmente utilizada na computação paralela com memória compartilhada, que explora o paralelismo inerente aos modelos quando comparado com o protocolo conservativo. No entanto, grande parte do esforço computacional é perdida devido às operações de retorno até o ultimo estado seguro (*rollback*).

Na simulação distribuída, quando uma rede de computadores é empregada, resulta-se no não compartilhamento de memória. Além disso, a possibilidade de operações de *rollback* associada com as várias mensagens que são normalmente trocadas entre computadores pode resultar na sobrecarga do tráfego da rede.

3 PROCEDIMENTO PARA A MODELAGEM

O modo pelo qual o sistema produtivo é modelado é baseado tanto nas suas características como a sua complexidade, bem como em fatores pessoais como experiência da equipe de projeto e no nível de abstração desejado. Em qualquer caso, a equipe de projeto deve ser capaz de visualizar o sistema produtivo (ou as principais partes em estudo) como um todo, as partes que o compõe, seu comportamento e a relação entre as partes (suas interfaces).O procedimento de modelagem é então organizado em passos, que são discutidos a seguir.

- Passo 1 – Definição do problema e delimitação do escopo do sistema produtivo

O modelador deve delimitar o âmbito do sistema produtivo em estudo, ou seja, aquilo que os departamentos, equipamentos (ferramentas) e pessoas (ambos considerados como recursos do sistema produtivo) envolvidos e, quais características e processos a serem modelados e analisados.

- Passo 2 – Refinamento sucessivo visando a identificação dos elementos básicos que compõe o sistema e seus relacionamentos

A abordagem *top-down* é adotada nessa etapa. O uso de técnicas de modelagem como PFS (*Production Flow Schema*) auxiliam na tarefa de construção do modelo do sistema produtivo (Hasegawa et al., 1999) (Miyagi et al., 2000). O PFS é um tipo de rede de Petri composto de atividades, elementos distribuídos e arcos. O PFS é propício para descrever a relação estrutural entre as principais partes do sistema. Este é o modelo conceitual aplicado na fase inicial da modelagem do sistema que é gradativamente traduzido em modelos em rede de Petri (este texto considera a classe específica de rede de Petri lugar/transição temporizada), que representa os detalhes e o comportamento dinâmico das atividades. Técnicas de simplificação também são aplicadas neste processo. No final desta fase, um conjunto de processos e elementos básicos que constituem o sistema produtivo é identificado, assim como o relacionamento entre eles, isto é, suas interfaces e o formato das mensagens trocadas entre eles.

- Passo 3 – Modelagem dos elementos básicos utilizando redes de Petri

Nesta etapa, as funcionalidades dos elementos básicos são modeladas com auxílio da rede de Petri. Cada modelo é chamado "classe" (Fig. 1a). Similar às linguagens de programação orientadas a objeto, a classe descreve um conjunto de objetos que compartilham os mesmos atributos, operações, relacionamentos e semântica. O modelo de cada elemento básico pode ser analisado isoladamente, facilitando a validação antes de sua utilização para compor modelos de elementos mais complexos.

- Passo 4 – Definição dos objetos

Cada classe definida no passo 3 é usada como elemento básico para gerar um ou mais objetos. A abordagem *bottom-up* é, então, utilizada.

- Passo 5 – Geração de componentes

Uma vez que os objetos foram definidos, eles podem ser combinados para formar um componente mais complexo. Esta fase tem três sub etapas:

1. Encapsular os objetos em componentes;
2. Conectar as interfaces dos objetos; e
3. Mapear as interfaces dos objetos remanescentes como interface de componente.

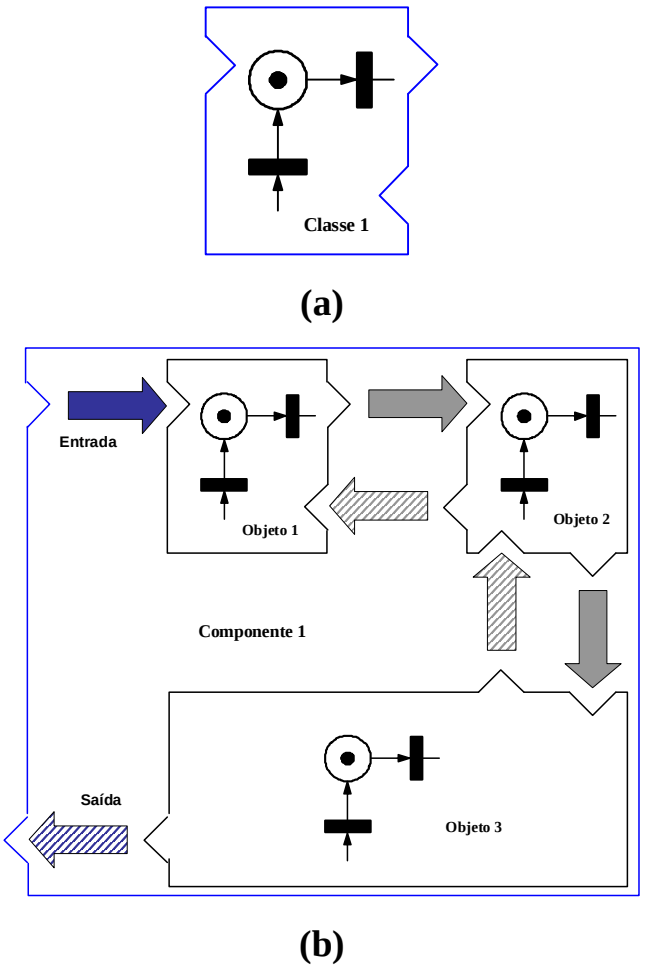
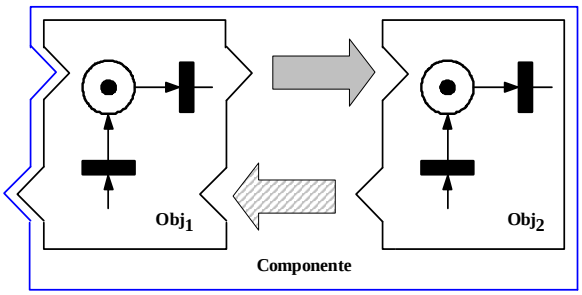


Figura 1: (a) classe implementada em rede de Petri; (b) componente constituído por três objetos.

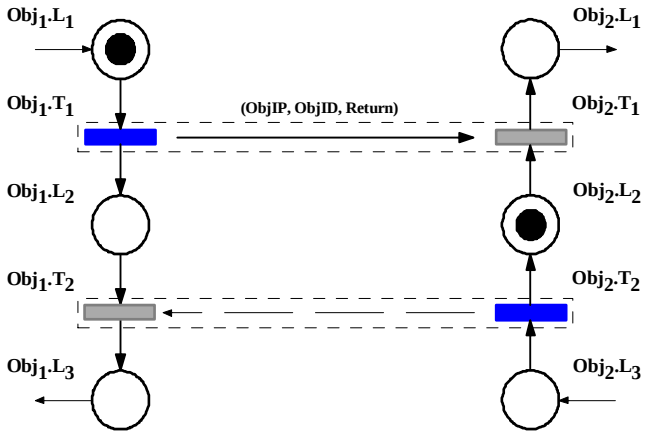
O processo de criação de componentes começa usando os objetos definidos no passo 4. Objetos que compartilham algumas características em comum, ou precisam trabalhar conjuntamente para a execução de uma tarefa, são agrupados (i) formando um componente (Fig. 1b). A seguir, (ii) as interfaces dos objetos são definidas e conectadas (setas cinzas na Fig. 1b). No presente procedimento, as interfaces são modeladas como transições[1] e a relação entre os modelos são realizadas através da fusão de transições (Fig. 2) (Gomes, Barros, 2005) (Sibertin-Blanc, 1993).

A chamada de método deve obedecer a seguinte regra: uma vez que um objeto faz a chamada de um método a um segundo objeto, ele deve esperar a resposta, não importa quanto tempo isso leve. Se um segundo objeto estiver executando a chamada de um método a um terceiro objeto no mesmo instante, ele irá adicionar o pedido a uma lista de pedidos e

[1] Neste texto, termos específicos da rede de Petri estão em Arial Narrow.



(a)



(b)

Figura 2: Exemplo de interface de um objeto: (a) representação esquemática; (b) Representação de uma rede de Petri com fusão de transições.

executa este assim que possível. Um exemplo é ilustrado na Fig 3. Na Fig. 3a, o objeto 3 envia uma chamada de método ao objeto 2, mas o objeto 2 ainda está executando o método solicitado pelo método 1. Na seqüência, Fig. 3b ilustra os três objetos após a resposta do objeto 2 ao objeto 1. Na Fig. 3c, o método do objeto 2 está agora disponível e, na Fig. 3d, o objeto 2 está executando o método do objeto 3.

No modelo em rede de Petri, esta regra implica que as transições que representam a chamada de método não podem estar em conflito com outras transições. Esta segunda regra também determina que as transições associadas à chamada de método são sempre instantâneas.

Para concluir o modelo de componente, é necessário (iii) mapear as interfaces dos objetos remanescentes como interface de componente. As setas azuis (sólida e tracejada) na Fig. 1b são um exemplo deste mapeamento.

- Passo 6 – Geração do aplicativo

Para gerar um aplicativo, dois ou mais componentes são agrupados e suas interfaces são conectadas (Fig. 4). Este passo é similar ao anterior. A diferença é que o aplicativo não possui interface externa. Em outras palavras, fazendo uma analogia com elementos de *software*, um componente semi-acabado não executa nada e pode ser usado em diferentes contextos, enquanto um aplicativo possui todos os elementos necessários para trabalhar sozinho e possui um propósito bem definido.

4 EXEMPLO

Para ilustrar a aplicação do processo, um exemplo prático é apresentado focando os aspectos mais relevantes do procedimento.

- Passo 1 – Definição do problema e delimitação do escopo do sistema produtivo

Considera-se que quatro estações de trabalho compõem um sistema de transporte de material. Cada estação produz ou consome produtos. A estação A fornece produtos para a estação B, e a estação C fornece produtos para a estação D. Um transportador, com capacidade unitária é usado para o transporte de produtos como mostrado na Fig. 5a.

Cada estação possui o seu tempo de processamento. As estações A e C entregam o produto (e as estações B e D recebem o novo produto) quando elas terminam suas tarefas, isto é, após os respectivos tempos de processamento. A tarefa para o carregamento de produtos do transportador para a estação e vice versa não é considerada neste exemplo, isto é, entende-se que ela não consome tempo relevante. No entanto, o tempo de transferência, isto é, de movimentação do transportador é modelado sendo que ele permanecerá parado até a conclusão dessa transferência.

- Passo 2 - Refinamento sucessivo visando a identificação dos elementos básicos que compõem o sistema e seus relacionamentos

Baseado na descrição do sistema, o PFS é usado para modelar as principais atividades, assim como detalhar os seus relacionamentos. A Fig. 5b representa o PFS do sistema mostrado na Fig. 5a. Este grafo representa a movimentação circular do transportador e a passagem do transportador através das estações. Algumas atividades deste modelo podem ser destacadas: (1) [Transporte entre duas estações]; (2) [Parada na estação] onde as mercadorias são (des)carregadas; (3) [Transportador]; e (4) [Estação]. Além disso, as setas azuis representam as informações trocadas entre [Parada na estação] e [Estação], e as setas cinzas, as informações entre [Parada na estação] e [Transportador].

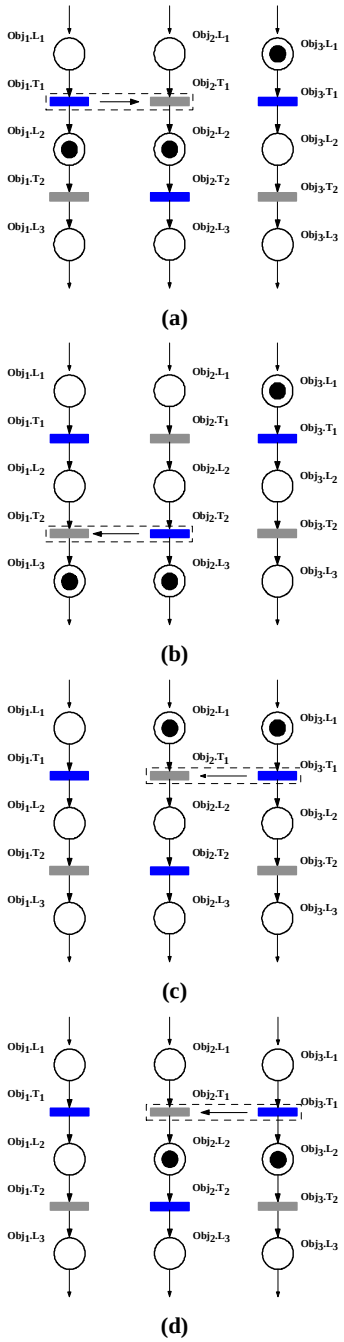


Figura 3: Duas chamadas de métodos concorrentes: (a) o objeto 2 está executando o método requisitado pelo objeto 1, através da fusão de transições $Obj_1 \cdot T_1$ e $Obj_2 \cdot T_1$, enquanto o objeto 3 está esperando pela disponibilidade do objeto 2; (b) o objeto 2 responde a chamada de método através da fusão de transições $Obj_2 \cdot T_2$ e $Obj_1 \cdot T_2$; (c) o método do objeto 2 ($Obj_2 \cdot T_1$) está agora disponível; e (d) o objeto 3 solicita o método fornecido pelo objeto 2 através da fusão de transições $Obj_3 \cdot T_1$ e $Obj_2 \cdot T_1$.

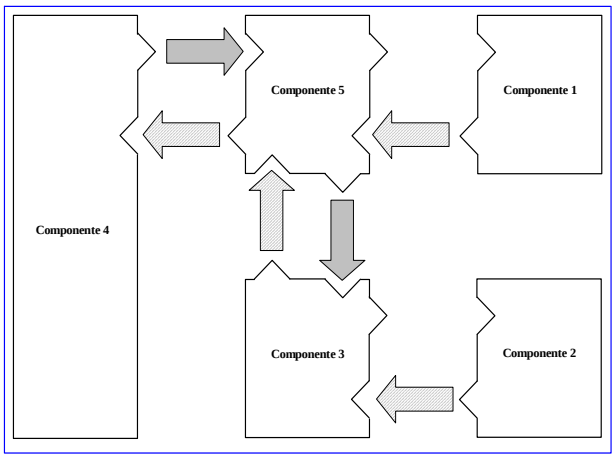


Figura 4: Um aplicativo composto por dois ou mais componentes.

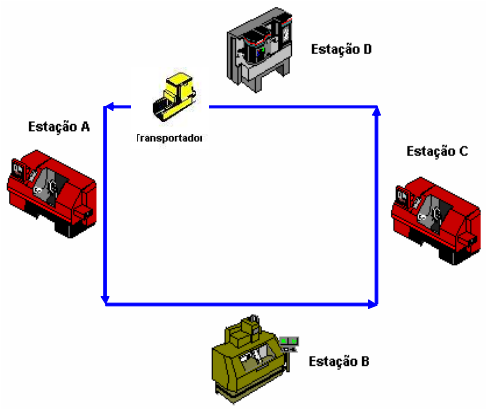


Figura 5: (a) movimentação circular do transportador, passando através das estações; (b) sistema de transporte de material modelado em PFS.

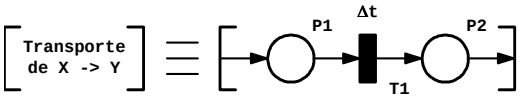


Figura 6: Modelo em rede de Petri da atividade [Transporte entre duas estações (de X a Y)].

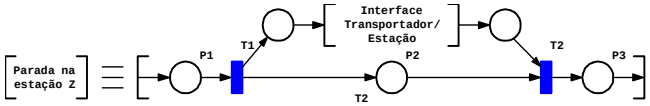


Figura 7: Modelo em rede de Petri da atividade [Parada na estação Z].

- Passo 3 – Modelagem dos elementos básicos em rede de Petri

Na Fig.6 tem-se a atividade [Transporte entre duas estações] modelada em rede de Petri onde as estações são genericamente identificadas por X (partida) e Y (chegada). O lugar P1 é a pré-condição desta operação. A transição T1 considera o tempo necessário para ir de X a Y. O lugar P2 é a pós-condição desta operação.

A Fig. 7 apresenta a atividade de parada em cada estação (genericamente chamada de estação Z). O lugar P1 assinala a chegada do transportador à estação. O inicio da operação é representado pela transição T1. O lugar P2 representa o (des) carregamento do produto. T2 indica o fim da operação e o lugar P3 representa o estado final da operação.

A atividade [Interface Transportador/Estação] é detalhada na Fig. 8. O lugar P1 representa o estado inicial, P2 a operação e P3 o término da operação de (des) carregamento. As transições T1 e T2 são, respectivamente, o inicio e o fim da operação de (des) carregamento.

A Fig. 9 apresenta o modelo em rede de Petri da atividade [Estação Z]. As transições T1 e T2 são os modelos das interface desta atividade. T1 representa o início da operação de (des) carregamento, enquanto que T2 representa o seu término. A transição T3 representa a duração da operação de (des) carregamento. O lugar P1 representa a espera da estação pelo término da operação de (des) carregamento. Os lugares P2 e P3 são, respectivamente, o início e o término do processamento dos produtos nas respectivas estações.

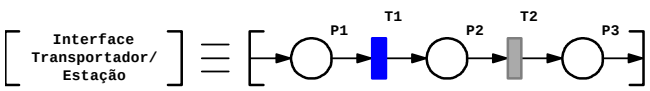


Figura 8: Modelo em rede de Petri da atividade [Interface Transportador/Estação].

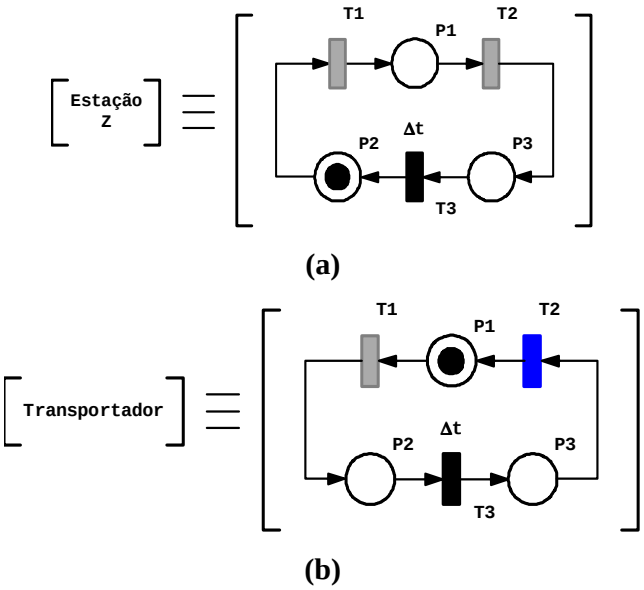


Figura 9: (a) Modelo em rede de Petri da atividade [Estação Z]; (b) Modelo em rede de Petri da atividade [Transportador].

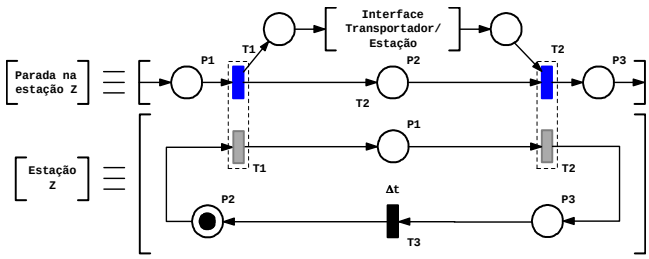


Figura 10: A relação entre as atividades [Parada na estação Z] e [Estação Z] através da fusão de transição.

O modelo em rede de Petri da atividade [Transportador] é mostrado na Fig. 9b. O lugar P1 representa o estado de movimentação do [Transportador], enquanto que P2 e P3 são respectivamente os estados inicial e final das operações de (des)carregamento. A transição T1 representa a parada do transportador e o início da operação de (des)carregamento. T3 representa a duração da operação de (des)carregamento, e T2 o fim da operação. T1 e T2 são também interfaces do modelo da atividade [Transportador].

A relação entre os modelos [Parada na estação Z] e [Estação Z] são modelados através da fusão de transições (Fig. 10). As transições T1 de ambos os modelos são unidas, trabalhando como uma só. O mesmo acontece com T2.

A Fig. 11 mostra a fusão de transição entre [Transportador] e [Interface Transportador/Estação]. As transições T1, de ambos os modelos, se comportam como uma só. O mesmo ocorre com a transição T2.

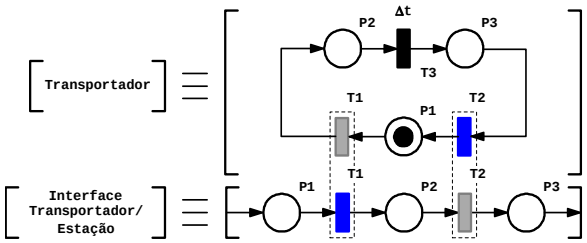


Figura 11: Relação entre as atividades [Transportador] e [Interface Transportador/Estação] através da fusão de transições.

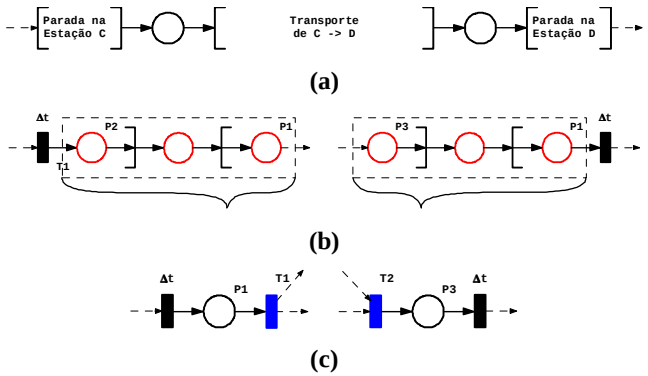


Figura 12: Aplicação de técnicas de redução do modelo.

Analisando a Fig. 5b, 6 e 7, uma simplificação pode ser aplicada na parte do modelo detalhado na Fig. 12a. Neste caso, o lugar P2 do modelo [Parada na Estação C] e P1 do modelo [Transporte de C para D] (Fig. 12b), por exemplo, podem ser unidos, resultando no lugar P1 do modelo da Fig. 12c.

A técnica de simplificação é aplicada ao modelo da Fig. 5b, que resulta no modelo [Malha de movimentação] da Fig. 13. Outra simplificação adotada neste exemplo é o uso de somente um par de transições (T13 e T14) para interagir com o modelo [Transportador]. Portanto, os conflitos por recursos são evidenciados no modelo e a geração de componentes (Passo 4) é simplificada.

- Passo 4 – Definição dos objetos

Cada classe definida no passo 3 é usada como modelo para gerar um ou mais objetos. A abordagem *bottom-up* é então adotada. As Figs. 14a, 14b e 14c mostram, respectivamente, o objeto "Estação A" (B, C e D) baseado na classe estação, objeto "Transportador" baseado na classe transportador e objeto "Malha de movimentação" baseado na classe de mesmo nome.

- Passo 5 – Geração dos componentes

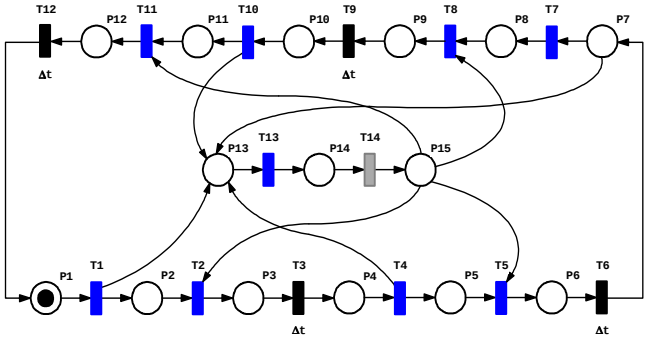


Figura 13: Modelo funcional do modelo da Fig. 5b em rede de Petri.

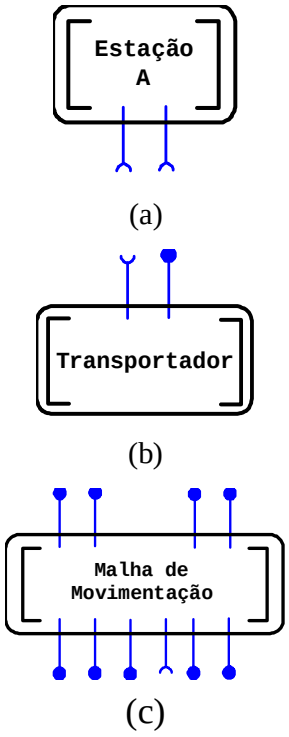


Figura 14: (a), (b) e (c) são objetos baseados nas classes definidas no passo 3.

Esses objetos são agora agrupados para comporem o componente da célula de manufatura (Fig. 15). Utilizou-se a notação de interfaces de entrada e saída do diagrama de componentes da UML para ilustrar as interface e os fluxos de informações entre os objetos do componente Célula de Manufatura (Figs. 14 e 15).

- Passo 6 – Geração do aplicativo

Para este exemplo simplificado, o modelo de aplicativo é o mesmo do componente obtido no passo anterior.

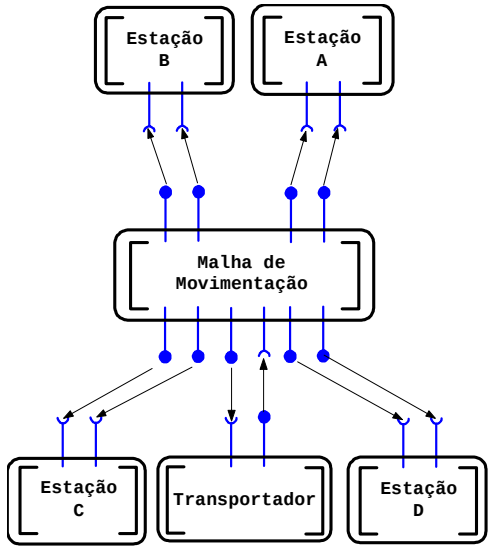


Figura 15: Componente da célula de manufatura.

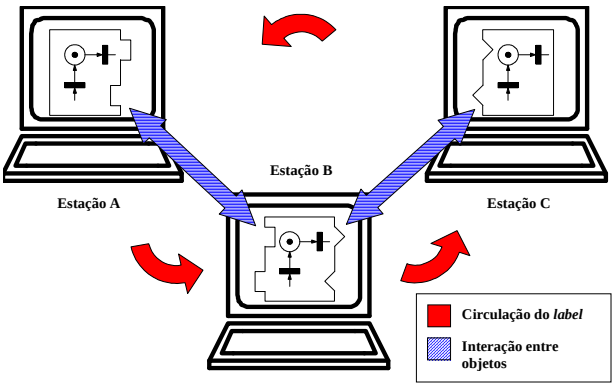


Figura 16: Comunicação entre modelos.

5 ALGORITMO DE COMUNICAÇÃO PARA O GERENCIAMENTO DA SIMULAÇÃO DISTRIBUÍDA

O algoritmo de gerenciamento da comunicação proposto para gerenciar a simulação distribuída é baseado no protocolo *label*[2] *ring* (Göhring, Kauffels, 1994). No entanto, no contexto deste trabalho, as estações não estão necessariamente conectadas através de uma topologia física em anel, isto é, implementa-se, de fato, um anel lógico. O algoritmo não interfere no protocolo de comunicação usado na rede de comunicação (como por exemplo, o protocolo TCP/IP). Ele é executado num nível superior. Cada estação conhece a identidade da próxima estação e um *label* é enviado através das estações em uma ordem pré-definida (Fig. 16).

[2]Utiliza-se aqui termo *label* ao invés de *token* para evitar confusão com o elemento *token* usado em rede de Petri.

O gerenciamento da simulação é distribuído entre as estações. O *label* é utilizado para sincronizar a evolução do tempo de simulação em todas as estações, de acordo com a abordagem conservadora. O *label* é composto por cinco diferentes campos que podem ser modificados por qualquer uma das estações do anel, a saber:

- Campo de identificação da estação (varLabelId) – este campo indica a última estação a alterar os valores dos demais campos do *label*.
- Campo de tempo futuro (varLabelTF) – este campo contém o tempo de simulação requisitado pela estação indicada no campo de identificação da estação.
- Campo de status (varLabelStatus) – este campo indica o *status* atual da estação indicada no campo de identificação da estação. A Tabela 1 apresenta seus possíveis valores.
- Campo de instrução (varLabelInstrucao) – este campo envia instruções para todas as estações, como por exemplo iniciar, pausar e parar a simulação.
- Campo de erro (varLabelErro) – este campo é usado para gerenciar erros de simulação, como quando uma estação perde a conexão por problemas de comunicação.

Os campos do *label* são todos zerados quando a simulação começa. A estação mestre seleciona o campo de instrução (varLabelInstruction) para iniciar e enviar o *label* pelo anel.

Tabela 1: Possiveis valores para o campo de status.

Valor do campo status	Significado
0	— (nenhuma estação está usando o label)
1	A estação está verificando o *status* corrente de outras estações.
2	A estação está enviando uma ordem para que as demais estações atualizem seus tempos de simulação com base no campo de tempo futuro.
3	A estação entrou em *deadlock*.

As 9 regras descritas na Tabela 2 gerenciam a simulação através da rede de modelos.

O modelo da Fig. 17 é usado como exemplo para ilustrar o funcionamento do algoritmo. Apesar das simplificações do

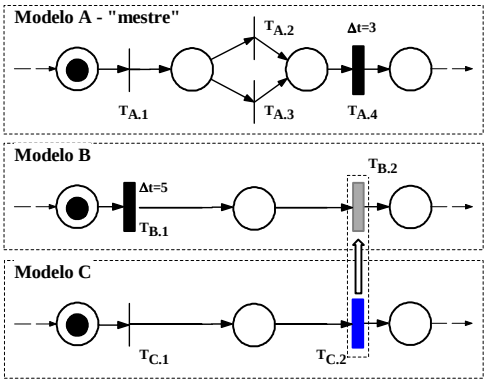


Figura 17: Exemplo do modelo em rede de Petri.

caso, ele mostra alguns dos principais pontos do algoritmo como a passagem do *label* e os métodos de chamada. No exemplo, o sistema a ser modelado é composto por três objetos (A, B e C), simulado em 3 diferentes estações.

O comportamento dos principais eventos é ilustrado no diagrama de seqüência em UML (Booch et al., 1999) da Fig. 18. É importante observar que as atividades preliminares para compor o anel não são aqui mostradas, estas devem também ser executadas pelo servidor. Além disso, intervalos de tempo para o envio e o recebimento do *label*, ou a execução das sub-rotinas dos disparos das transições são definidas aleatoriamente. Como esses intervalos de tempo dependem da capacidade dos computadores e da disponibilidade dos meios de comunicação, eles podem variar de caso para caso.

A simulação inicia quando o servidor envia o *label* através do anel com o campo Instrução = Start. Após receber o *label*, cada estação pode disparar suas transições instantâneas, que neste caso são o $t_{A.1}$ e $t_{A.2}$ (ou $t_{A.3}$) para a estação A, e $t_{C.1}$ para a estação C. A transição $t_{C.2}$ é a chamada de método do objeto B, mas este método não está ainda disponível, assim o objeto C deve aguardar.

Na próxima etapa, o objeto B requisita que o *clock* da simulação seja atualizado para 5; no entanto, antes que ele receba o *label* de volta, o objeto A altera o campo do *label* com uma requisição de alteração do *clock* para 3. Quando o *label* percorre o anel, todas as estações já dispararam todas as suas transições instantâneas possíveis, então a estação A confirma a atualização para 3 (campo Status = 2). Após o disparo $t_{A.3}$, a estação A alcança o estado de *deadlock* e envia o *label* com o campo Status = 3. A estação B, que novamente requisita a atualização do *clock* para 5, substitui esta informação. Esta última informação não é substituída, logo o *clock* de simulação é colocado em 5. A estação B pode então dispara $t_{B.1}$ e responder a chamada de método, que resulta no disparo de $t_{B.2}$ e $t_{C.2}$.

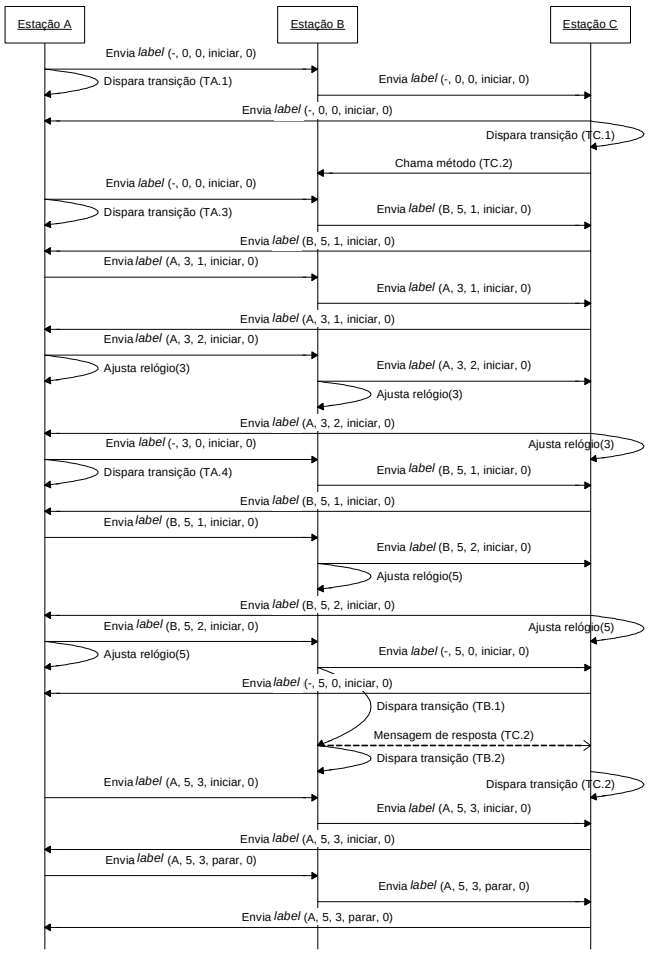


Figura 18: Exemplo do diagrama UML de seqüência.

Comparando a abordagem proposta com uma tradicional, o algoritmo conservador apresentado em (Fujimoto, 1999) troca mensagens (nulas ou não) para sincronizar cada *clock* local. Assim, se o problema de simulação possui N modelos interagindo entre si (caso mais crítico), cada modelo necessita enviar $N-1$ mensagens para determinar o tempo seguro para avançar o *clock* de simulação. Isto resulta em $N(N-1)$ mensagens trocadas. Considerando α a percentagem de mensagens nulas, a expressão resulta em $\alpha(N^2 - N)$ mensagens nulas trocadas para sincronizar o tempo de simulação.

No algoritmo proposto, cada modelo monitora a mensagem *label*. O número de mensagens *label* trocadas é definido como β. Assim, o simulador troca βN mensagens para sincronizar o tempo de simulação. Quando nenhum simulador altera os atributos do *label*, duas passagens de *labels* são necessárias (a primeira para consulta, e a segunda para atualizar o tempo de simulação), resultando em $2N$ mensagens trocadas.

Tabela 2: Regras para o gerenciamento da simulação

Linha de commando	Observações
`if (varLabelInstrucao != "stop") {`	
`  if (varLabelErro == "1") {`	// Se um erro for detectado, a simulação deve ser encerrada.
`    varLabelInstrucao = "stop";`	
`  }`	
`  else if (varLabelId == ) {`	// Nenhuma das estações está usando o *label*.
`    if (local object deadlock == 2) {`	// Regra 01 – o objeto/modelo local de simulação está em *deadlock* – ele não possui transições instantâneas ou temporizadas a serem disparadas. Ele precisa notificar aos demais objetos/modelos sobre o estado de *deadlock*.
`      varLabelId = local object identification;`	
`      varLabelStatus = "3";`	
`    }`	
`    else if (local object deadlock == 1) {`	// Regra 02 – o objeto/modelo local de simulação está em *deadlock* – ele não possui transições instantâneas a serem disparadas. Ele precisa consultar aos demais objetos/modelos antes de avançar o tempo de simulação do sistema como um todo.
`      varLabelId = local object identification;`	
`      varLabelStatus = "1";`	
`      varLabelTF = local object clock;`	
`    }`	
`  }`	
`  else if (varLabelId != local object identification) {`	// O *label* foi utilizado por outro objeto/modelo da simulação.
`    if (local object deadlock == 0) {`	// Regra 03 –o objeto/modelo local não está em *deadlock* e um outro modelo está consultando a rede sobre outros objetos/modelos em *deadlock*. O objeto/modelo local reinicia os campos varLabelId e varLabelStatus e copia o relógio de simulação local para o campo varLabelTF.
`      varLabelId = ;`	
`      varLabelStatus = "0";`	
`      varLabelTF = local object clock;`	
`    }`	
`    else if ((local object deadlock == 1) && (varLabelStatus == "1") && (varLabelTF > local object clock)) {`	// Regra 04 – o objeto/modelo local de simulação está em *deadlock* – ele não possui nenhuma transição instantânea ou temporizada a ser disparada. Ele recebe o *label*, que tem o varLabelTF maior que o *clock* local. O objeto/modelo local copia sua própria identificação para o campo varLabelId e seu *clock* local para o campo varLabelTF.
`      varLabelId = local object identification;`	
`      varLabelTF = local object clock;`	
`    }`	
`    else if ((local object deadlock == 1) && (varLabelStatus == "3")) {`	// Regra 05 – o objeto/modelo local de simulação está em *deadlock* – ele não possui transições instantâneas a serem disparadas.– e ele recebe o *label* informando que um objeto/modelo anterior está em deadlock. O objeto/modelo local copia seus valores para os campos varLabelId e varLabelFT e altera o varLabelStatus para "1" para consultar outros objetos/modelos antes de evoluir o seu *clock* local.
`      varLabelId = local object identification;`	
`      varLabelStatus = "1";`	
`      varLabelTF = local object clock;`	
`    }`	
`    else if (varLabelStatus == "2") {`	// Regra 06 – o objeto/modelo local recebe o *label* com a instrução para atualizar o seu *clock* de simulação local. Ele copia o valor do campo varLabelTF para o *clock* do objeto local e reseta a variável *deadlock* do objeto local para disparar suas transições.
`      local object clock = varLabelTF;`	
`      local object deadlock = 0;`	
`    }`	
`  }`	
`  else {`	// O *label* foi utilizado por um objeto/modelo local.

continua . . .

Tabela 2: Continuação da página anterior

`if (varLabelStatus == "3") {`	// Regra 07 – o objeto/modelo local, no estado de *deadlock* (sem transições instantâneas ou temporizadas), envia o *label* informando aos demais modelos sobre a sua condição, e recebe o *label* sem alterações. O objeto/modelo local escreve no campo erro do *label* para informar que todo o sistema está no estado de *deadlock* e toda a simulação deve ser encerrada.
`varLabelError = "1";`	
`}`	
`else if (varLabelStatus == "1") {`	// Regra 08 –o objeto/modelo local, no estado de *deadlock* (somente sem transições instantâneas), envia o *label* perguntando aos demais modelos sobre a possibilidade de avançar seus *clocks* locais e ele recebe o *label* de volta sem alterações. O objeto/modelo local altera o campo "varLabelStatus" para "2", e ele envia o *label* para os demais modelos que terão os seus *clocks* locais atualizados.
`varLabelStatus = "2";`	
`}`	
`else if (varLabelStatus == "2") {`	// Regra 09 – após os demais modelos terem os seus *clocks* locais atualizados, o *label* retorna ao objeto/modelo local. O objeto/modelo local atualiza o seu *clock* local e libera o *label*.
`varLabelId = ;`	
`varLabelStatus = "0";`	
`local object deadlock = 0;`	
`}`	
`}`	
`}`	

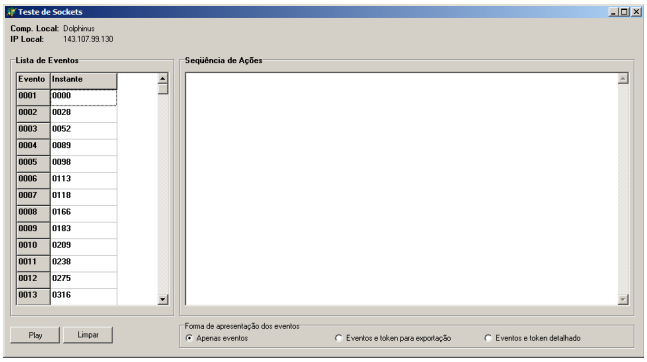


Figura 19: Interface do software de simulação.

Assim, se $\alpha(N^2\text{-}N) - \beta N \leq 0$, a abordagem tradicional utiliza menos a rede de comunicação trocando mensagens nulas para sincronizar o tempo dos simuladores, isto é, $N \leq (\beta/\alpha) + 1$.

6 AVALIAÇÃO DO ALGORITMO

O *software* de simulação foi implementado (Fig. 19) para avaliar o desempenho do algoritmo. Este simulador foi replicado em diferentes computadores, todos eles conectados a uma rede local de comunicação de10Mbps.

Existem algumas considerações sobre os experimentos realizados de simulação:

- Os simuladores, através do uso de *sockets*, formam um anel lógico entre si;
- Cada simulador executa uma consulta de eventos organizados em ordem ascendente;
- O evento do tempo ***i+1*** é o evento do tempo ***i*** adicionado de um número aleatório maior que zero;
- A consulta de eventos não possui dois eventos com o mesmo *time stamp* – isto foi adotado para analisar o algoritmo no caso mais crítico: quando apenas um evento é disparado em cada processo sincronizado;
- O processo de sincronização é registrado no campo de seqüência de ações ("Seqüência de Ações" na Fig. 19);
- O simulador deriva o tempo despendido pelo *label* para completar cada volta, desta vez não considera o tempo do algoritmo, apenas o tempo de circulação;
- O experimento usou computadores com processador Pentium IV e com no mínimo 512Mb de memória RAM.

Segue a descrição de algumas das experiências realizadas e os seus resultados.

6.1 Validação do algoritmo

Aqui, dois simuladores são usados, sendo que dez eventos foram programados para cada simulador (Fig. 20). Este exemplo ilustra um dos experimentos realizados para validar o algoritmo de comunicação para gerenciamento da simulação distribuída.

O resultado da simulação é apresentado na Tabela 3. A passagem do *label* inicia em 0 no primeiro simulador, pois ele dispara o processo de simulação. A regra 00 indica que nenhum dos simuladores está usando o *label* (o *label* está livre).

6.2 Influência do número de simuladores utilizados

Neste caso, 100 eventos foram programados para cada simulador. Este exemplo ilustra um dos experimentos realizados para analisar a influência do número de simuladores (N) nos seguintes parâmetros: (1) número de passagens do *label* para processar todos os eventos dos simuladores; (2) o tempo de circulação do *label*; (3) o número de passagens do *label* entre dois eventos consecutivos no simulador; e (4) o tempo real entre dois eventos consecutivos no simulador.

A influência do número de simuladores no (1) número de passagens do *label*, e (2) no tempo de circulação do *label* é apresentado na Tabela 4.

O resultado mostra que o número de passagens do *label* aumenta com o número de simuladores. É também observado que o tempo de circulação ficou próximo de 70ms e independente do número de simuladores.

A influência do número de simuladores no número de passagens do *label* entre dois eventos consecutivos no simulador é apresentado na Tabela 5. Nota-se que quanto maior o número total de eventos dos simuladores, menor é o número de passagens do *label* necessário para execução destes eventos (Fig. 21).

A influência do número de simuladores sobre o tempo real decorrido entre dois eventos consecutivos é apresentado na Tabela 6.

6.3 Influência da rede de comunicação sobre a simulação

Tem-se aqui um exemplo dos experimentos realizados para analisar a influência da rede de comunicação nos seguintes aspectos: (1) tempo de circulação do *label*; (2) o número de passagens do *label* entre dois eventos consecutivos no simulador; e (3) o tempo real decorrido entre dois eventos consecutivos no simulador.

(a) Lista dos Eventos do simulador 1 (Obj1)

(b) Lista dos eventos do simulador 2 (Obj2)

Figura 20: Lista dos eventos usados no algoritmo de validação.

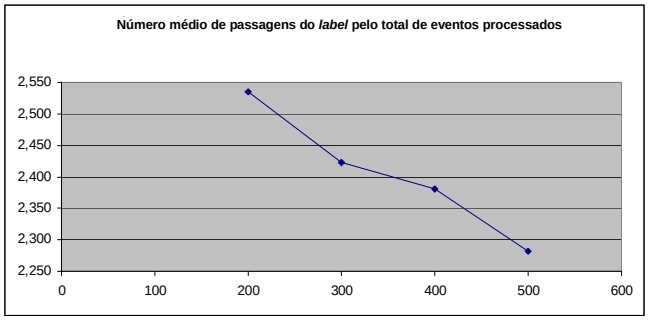


Figura 21: Influência do número de simuladores, ponderando-se pelo número total de eventos processados (eixo horizontal), em relação ao número de passagens do *label* entre dois eventos consecutivos considerando todos os eventos dos simuladores (eixo vertical).

Para este experimento, foram feitas as seguintes considerações:

Tabela 3: Resultado da simulação distribuída para dois simuladores (objetos).

Object 1				Object 2			
Passagem do label	**label recebido**	**Regra**	**Evento**	**Passagem do label**	**label recebido**	**Regra**	**Evento**
00				01	<< ><0000><0><1><0>>	00	
01	<< ><0000><0><1><0>>	00	0001	02	<< ><0000><0><1><0>>	00	0001
02	<< ><0000><0><1><0>>	02		03	<<Obj1><0017><1><1><0>>	04	
03	<<Obj2><0016><1><1><0>>	00		04	<<Obj2><0016><1><1><0>>	08	
04	<<Obj2><0016><2><1><0>>	06		05	<<Obj2><0016><2><1><0>>	09	0002
05	<< ><0016><0><1><0>>	02		06	<<Obj1><0017><1><1><0>>	00	
06	<<Obj1><0017><1><1><0>>	08		07	<<Obj1><0017><2><1><0>>	06	
07	<<Obj1><0017><2><1><0>>	09	0002	08	<< ><0017><0><1><0>>	02	
08	<<Obj2><0027><1><1><0>>	00		09	<<Obj2><0027><1><1><0>>	08	
09	<<Obj2><0027><2><1><0>>	06		10	<<Obj2><0027><2><1><0>>	09	0003
10	<< ><0027><0><1><0>>	02		11	<<Obj1><0042><1><1><0>>	00	
11	<<Obj1><0042><1><1><0>>	08		12	<<Obj1><0042><2><1><0>>	06	
12	<<Obj1><0042><2><1><0>>	09	0003	13	<< ><0042><0><1><0>>	02	
13	<<Obj2><0045><1><1><0>>	00		14	<<Obj2><0045><1><1><0>>	08	
14	<<Obj2><0045><2><1><0>>	06		15	<<Obj2><0045><2><1><0>>	09	0004
15	<< ><0045><0><1><0>>	02		16	<<Obj1><0078><1><1><0>>	04	
16	<<Obj2><0052><1><1><0>>	00		17	<<Obj2><0052><1><1><0>>	08	
17	<<Obj2><0052><2><1><0>>	06		18	<<Obj2><0052><2><1><0>>	09	0005
18	<< ><0052><0><1><0>>	02		19	<<Obj1><0078><1><1><0>>	00	
19	<<Obj1><0078><1><1><0>>	08		20	<<Obj1><0078><2><1><0>>	06	0006
20	<<Obj1><0078><2><1><0>>	09	0004	21	<< ><0078><0><1><0>>	02	
21	<<Obj2><0102><1><1><0>>	04		22	<<Obj1><0090><1><1><0>>	00	
22	<<Obj1><0090><1><1><0>>	08		23	<<Obj1><0090><2><1><0>>	06	
23	<<Obj1><0090><2><1><0>>	09	0005	24	<< ><0090><0><1><0>>	02	
24	<<Obj2><0102><1><1><0>>	00		25	<<Obj2><0102><1><1><0>>	08	
25	<<Obj2><0102><2><1><0>>	06		26	<<Obj2><0102><2><1><0>>	09	0007
26	<< ><0102><0><1><0>>	02		27	<<Obj1><0126><1><1><0>>	04	
27	<<Obj2><0106><1><1><0>>	00		28	<<Obj2><0106><1><1><0>>	08	
28	<<Obj2><0106><2><1><0>>	06		29	<<Obj2><0106><2><1><0>>	09	0008
29	<< ><0106><0><1><0>>	02		30	<<Obj1><0126><1><1><0>>	04	
30	<<Obj2><0122><1><1><0>>	00		31	<<Obj2><0122><1><1><0>>	08	
31	<<Obj2><0122><2><1><0>>	06		32	<<Obj2><0122><2><1><0>>	09	0009
32	<< ><0122><0><1><0>>	02		33	<<Obj1><0126><1><1><0>>	00	
33	<<Obj1><0126><1><1><0>>	08		34	<<Obj1><0126><2><1><0>>	06	
34	<<Obj1><0126><2><1><0>>	09	0006	35	<< ><0126><0><1><0>>	02	
35	<<Obj2><0152><1><1><0>>	04		36	<<Obj1><0144><1><1><0>>	00	
36	<<Obj1><0144><1><1><0>>	08		37	<<Obj1><0144><2><1><0>>	06	
37	<<Obj1><0144><2><1><0>>	09	0007	38	<< ><0144><0><1><0>>	02	
38	<<Obj2><0152><1><1><0>>	00		39	<<Obj2><0152><1><1><0>>	08	
39	<<Obj2><0152><2><1><0>>	06		40	<<Obj2><0152><2><1><0>>	09	0010
40	<< ><0152><0><1><0>>	02		41	<<Obj1><0161><1><1><0>>	00	
41	<<Obj1><0161><1><1><0>>	08		42	<<Obj1><0161><2><1><0>>	06	
42	<<Obj1><0161><2><1><0>>	09	0008	43	<< ><0161><0><1><0>>	01	
43	<<Obj2><0161><3><1><0>>	05		44	<<Obj1><0206><1><1><0>>	00	
44	<<Obj1><0206><1><1><0>>	08		45	<<Obj1><0206><2><1><0>>	06	
45	<<Obj1><0206><2><1><0>>	09	0009	46	<< ><0206><0><1><0>>	01	
46	<<Obj2><0206><3><1><0>>	05		47	<<Obj1><0234><1><1><0>>	00	

continua . . .

Tabela 3: Continuação da página anterior

Object 1				Object 2			
Passagem do label	**label recebido**	**Regra**	**Evento**	**Passagem do label**	**label recebido**	**Regra**	**Evento**
47	<<Obj1><0234><1><1><0>>	08		48	<<Obj1><0234><2><1><0>>	06	
48	<<Obj1><0234><2><1><0>>	09	0010	49	<< ><0234><0><1><0>>	01	
49	<<Obj2><0234><3><1><0>>	00		50	<<Obj2><0234><3><1><0>>	07	
50	<<Obj2><0234><3><1><1>>	00					

Tabela 4: Influência do número de simuladores na passagem e no tempo de circulação do *label*.

		Tempo (em segundos)					
N	Passagem do label	Min	Max	Mediana	Moda	Média	Desvio Padrão
2	0507	0,000	1,035	0,070	0,070	0,072	0,043
3	0727	0,000	0,365	0,070	0,070	0,085	0,022
4	0952	0,000	0,095	0,070	0,070	0,070	0,004
5	1141	0,000	0,090	0,070	0,070	0,070	0,004

Tabela 5: Influência do número de simuladores sobre o número de passagens do *label* entre dois eventos consecutivos no simulador.

		Número da passagem do *label* entre dois eventos consecutivos no simulador					
N	Passagens do *label*	Min	Max	Mediana	Moda	Média	Desvio Padrão
2	0507	0.000	11.000	5.000	5.000	4.970	2.372
3	0727	0.000	22.000	5.000	5.000	7.140	4.584
4	0952	0.000	23.000	8.000	5.000	9.090	4.866
5	1141	0.000	27.000	10.000	5.000	10.980	6.754

Tabela 6: Influência do número de simuladores sobre o tempo real decorrido entre dois eventos consecutivos.

		Tempo (em segundos)					
N	Passagens do *label*	Min	Max	Mediana	Moda	Média	Desvio Padrão.
2	0507	0.000	1.175	0.350	0.350	0.357	0.186
3	0727	0.000	1.890	0.490	0.420	0.599	0.405
4	0952	0.000	1.605	0.560	0.350	0.635	0.342
5	1141	0.000	1.890	0.698	0.210	0.767	0.473

- Uso de dois simuladores;
- 100 eventos programados para cada simulador;
- Um dos experimentos usou uma rede de comunicação comercial para conectar o computador da universidade com uma outra distante 470km;
- Outro experimento usou a Intranet, onde um *hub* de 10Mbps e um *switch* de 100Mbps foram testados como pontes entre os dois simuladores.

A influência da rede sobre a circulação do *label* é apresentado na Tabela 7. É observado que na rede de comunicação

Tabela 7: Influência da rede de comunicação sobre o tempo de circulação do *label*.

		Tempo (em segundos)					
Rede	Passagem do *label*	Min	Max	Mediana	Moda	Média	Desvio Padrão
comercial	0514	0.000	12.595	0.160	0.135	0.300	0.631
10Mbps	0507	0.000	1.035	0.070	0.070	0.072	0.043
100Mbps	0520	0.000	0.110	0.070	0.070	0.081	0.025

Tabela 8: Influência dos eventos instantâneos sobre o tempo de circulação do *label*.

		Tempo (em segundos)					
%	Passagens do *label*	Min	Max	Mediana	Moda	Média	Desvio Padrão
10	0414	0.020	0.405	0.105	0.105	0.097	0.034
30	0314	0.015	0.230	0.075	0.070	0.086	0.020
50	0211	0.025	0.390	0.100	0.105	0.092	0.029
70	0107	0.040	0.120	0.070	0.070	0.071	0.009
90	0022	0.045	0.130	0.105	0.105	0.090	0.025

Tabela 9: Influência dos eventos instantâneos sobre o número de passagens do *label* para executar dois eventos consecutivos no simulador.

		Tempo (em segundos)					
%	Passagens do *label*	Min	Max	Mediana	Moda	Média	Desvio Padrão
10	0414	0	14	3	3	4.110	2.463
30	0314	0	11	3	0	3.120	2.610
50	0211	0	11	0	0	2.030	2.601
70	0107	0	8	0	0	1.035	1.935
90	0022	0	3	0	0	0.180	0.656

Tabela 10: Influência dos eventos instantâneos sobre o tempo real entre dois eventos consecutivos executados no simulador.

		Tempo (em segundos)					
%	Passagens do *label*	Min	Max	Mediana	Moda	Média	Desvio Padrão
10	0414	0.000	1.365	0.350	0.000	0.398	0.259
30	0314	0.000	0.945	0.250	0.000	0.268	0.227
50	0211	0.000	1.015	0.000	0.000	0.186	0.244
70	0107	0.000	0.615	0.000	0.000	0.072	0.137
90	0022	0.000	0.330	0.000	0.000	0.014	0.057

dedicada, a circulação do *label* é 3 vezes mais rápida que em uma rede de comunicação comercial (e não dedicada). Além disso, é observado que a rede de comunicação não influenciou o número de passagens de *label*. A pequena diferença resulta do diferente conjunto de eventos aleatórios em cada execução.

Este experimento mostra que a simulação distribuída tem melhor desempenho numa rede de comunicação dedicada ou numa de baixo tráfego.

6.4 Influência de eventos instantâneos no desempenho da simulação

Apresenta-se agora um exemplo dos experimentos realizados para se determinar a influência de eventos instantâneos no desempenho da simulação onde, utilizou-se dois simuladores com 100 eventos programados para cada um.

A porcentagem de eventos instantâneos não influencia o tempo de circulação do *label* como mostrado na Tabela 8. Porém, eventos instantâneos influenciam muito o número de passagens do *label* entre dois eventos consecutivos como mostrado na Tabela 9. O número de passagens do *label* reduz com o aumento do número de eventos instantâneos.

A influência da porcentagem de eventos instantâneos sobre o tempo real entre dois eventos consecutivos é mostrado na Tabela 10. A média reduz com o aumento do número dos eventos instantâneos, pois há mais eventos em execução ao mesmo tempo na simulação.

7 COMENTÁRIOS FINAIS

Como observado em alguns artigos (Daum, Sargent, 2002) (Kachitvichyanukul, 2001) (Banks, 2000), a pesquisa na área de modelagem e simulação é necessária, especialmente no desenvolvimento de novas técnicas de modelagem, assim como o reuso de modelos.

O procedimento de modelagem proposto torna possível o entendimento e o detalhamento do sistema produtivo de modo progressivo, através de refinamentos sucessivos. A abordagem adotada permite uma melhor caracterização dos elementos do sistema e do relacionamento entre eles. Um modelo relativamente simples pode representar elementos que possuem características comuns, garantindo a sua reusabilidade, isto é, uma biblioteca de modelos pode ser implementada. Assim, as propriedades e funcionalidades de cada elemento podem ser verificadas e, a partir da composição desses elementos, elementos mais complexos, como sub-sistemas, podem ser criados, e as suas funcionalidades, validadas. Desta forma, o modelo total do sistema pode ser obtido através de um procedimento de composições sucessivas.

Além disso, o procedimento proposto é próprio para simulação distribuída, uma vez que os modelos resultantes e suas interações são explicitamente especificados. Assim, a simulação distribuída pode ser efetivamente realizada.

Sobre a proposta do algoritmo de simulação, suas principais características são a abordagem conservativa e a sincronização de modelos baseada em *label-ring* para o gerenciamento do tempo de simulação.

É importante salientar que a consideração de diferentes intervalos de tempo para a execução das subrotinas de disparo de transições e para a comunicação entre estações pode levar a diferentes valores dos campos do *label* e a seqüências diferentes de eventos. Isto também pode eventualmente resultar em diferentes marcações (estados) para os modelos em rede de Petri. Não obstante, é importante destacar que qualquer cenário possível é válido a partir do ponto de vista do formalismo da rede de Petri (a marcação inicial pode levar a diferentes marcações através de seqüências diferentes de disparo de transições).

Este artigo sintetiza e organiza alguns resultados preliminares apresentados em (Junqueira, Miyagi, 2006) (Junqueira et al., 2005) onde além da avaliação dos revisores, novas contribuições foram consideradas.

AGRADECIMENTOS

Os autores agradecem o apoio parcial do CNPq, CAPES e FAPESP para o desenvolvimento deste trabalho.

REFERÊNCIAS

Banks, J. (2000) Simulation in the future. *Proc. of the Winter Simulation Conference*, pp. 1568-1576.

Beraldi, R. and L. Nigro (1999) Distributed simulation of timed Petri nets: a modular approach using actors and time warp. *IEEE Concurrency*, 1999, **7**, 52-62.

Booch, G., J. Rumbaugh, and I. Jacobson (1999) *The Unified Modeling Language User Guide*. Addison Wesley Longman, Inc.

Carullo, L., A. Furfaro, L. Nigro and F. Pupo, (2003) Modelling and simulation of complex systems using TPN Designer. *Simulation Modelling Practice and Theory*, **11**, 503-532.

Cassandras, C.G. and S.G. Strickland (1992) *Sample Path Properties of Timed Discrete Event Systems in Discrete Event Dynamic Systems*. IEEE Press, New York.

Cheng, J. and L. Cheng (2006) Dispersed networked manufacturing mode and its application in China. *Proc. of*

IEEE Intern. Conf. on Industrial Informatics, pp. 1291-1294.

Chiola, G. and A. Ferscha (1993) A distributed discrete event simulation framework for timed Petri net models, *Technical Report Series of the Austrian Center for Parallel Computation*, ACPC/TR, pp. 93-21.

Daum, T. and R.G. Sargent (1999) Scaling, hierarchical modeling, and reuse in an object-oriented modeling and simulation system. *Proc. of the Winter Simulation Conference*, pp. 1470-1477.

Daum, T.S. and R.G. Sargent (2002) A Web-ready HiMASS: facilitating collaborative, reusable, and distributed modeling and execution of simulation models with XML. *Proc. of Winter Simulation Conference*, vol. 1, pp. 634-640.

Davrazos, G. and N.T. Koussoulas (2007) Modeling and stability analysis of state-switched hybrid systems via differential Petri nets. *Simulation Modelling Practice and Theory*, **15**, 879-893.

Djemame, K., D.C. Gilles, L.M. Mackenzie, and M. Bettaz (1998) Performance comparison of high-level algebraic nets distributed simulation protocols. *J. of Systems Architecture*, **44**, 457-472.

Fujimoto, R. (2003) Parallel discrete event simulation. *Communications of the ACM*, **33**, 30-53.

Fujimoto, R.M. (1999) Parallel and distributed simulation. *Proc. of the Winter Simulation Conference*, pp. 122-131.

Göhring, H-G. and F-J. Kauffels (1994) *Token ring: Principles, Perspectives and Strategies*. Addison-Wesley Pub. Co.

Gomes, L. and J.P. Barros (2005) Structuring and composability issues in Petri nets modeling. *IEEE Transactions on Industrial Informatics*, **1**, 112-123.

Hasegawa, K., P.E. Miyagi, D.J. Santos Filho, K. Takahashi, L. Ma, and M. Sugisawa (1999) On resource arc for Petri net modeling of complex resource sharing system. *J. of Intelligent and Robotic Systems*, **26**, 423-437.

Junqueira F. and P.E. Miyagi (2006) A new method for the hierarchical modeling of productive systems. *Proc. of BASYS2006 IFIP Intern. Conf. on Information Technology for Balanced Automation System in Manufacture and Services*, pp. 479-488.

Junqueira, F., E. Villani and P.E. Miyagi (2005) A platform for distributed modeling and simulation of productive systems based on Petri nets and object-oriented paradigm. *Proc. of ETFA IEEE Intern. Conf. on Emerging Technologies and factory Automation*, Catania, pp. 907-914.

Kachitvichyanukul, V. (2001) Simulation environment for the new millennium. *Proc. of the Winter Simulation Conference*, pp. 541-547.

Karatza, H.D. and G.K. Theodoropoulos (2006) Distributed systems simulation. *Simulation Modelling Practice and Theory*, **14**, 677-678.

Kumar, D. and A. Kohli (1997) Faster simulation of timed Petri nets via distributed simulation. *Proc. of the 21st Intern. Computer Software and Applications Conf.*, pp. 149-152.

McLean, C. and F. Riddick (2001) Integrating distributed manufacturing simulations. *Proc. of IEEE Intern. Conf. on Systems, Man, and Cybernetics*, vol. 2, pp. 1294-1298.

Miyagi, P.E. and L.A.M. Riascos (2006) Modeling and analysis of fault-tolerant systems for machining operations based on Petri nets. *Control Engineering Practice*, **14**, 397-408.

Miyagi, P.E., D.J. Santos Filho, and W.M. Arata (2000) Design of deadlock avoidance compensators for anthropocentric production systems. *Proc. of BASYS'2000 IEEE/IFIP Intern. Conf. on Information Technology for Balanced Automation Systems in Production and Transportation*, pp. 287-294.

Murata, T. (1989) Petri nets - properties, analysis and applications. *Proceedings of the IEEE*, **77**, 541-580.

Nevison, C. (1990) Parallel simulation of manufacturing systems: structural factors. *Proc. of 22nd SCS Multiconference on Distributed Simulation*, vol. 1, pp. 17-19.

Nicol, D.M. and S. Roy (1991) Parallel simulation of timed Petri-nets. *Proc. of the Winter Simulation Conference*, pp. 574-583.

Nketsa, A. and R. Valette (2001) Rapid and modular prototyping-based Petri nets and distributed simulation for manufacturing systems. *Applied Mathematics and Computation*, **120**, 265-278.

Perumalla, K.S., R.M. Fujimoto, P.J. Thakare, S. Pande, H. Karimabadi, Y. Omelchenko, and J. Driscoll (2005) Performance prediction of large-scale parallel discrete event models of physical systems. *Proc. of Winter Simulation Conference*.

Sanz, R. and M. Alonso (2001) CORBA for control systems. *Annual Reviews in Control*, **25**, 169-181.

Sanz, R., S. Galan, M. Rodriguez, C. Garcia, R. Chincilla, and A. Yela (2003) An experiment in distributed objects for real-time process control. *Proc. of ETFA'03 IEEE Intern. Conf. on Emerging Technologies and Factory Automation*, vol. 2, pp. 664-668.

Shi, Y., and M. Gregory (1998) International manufacturing networks – to develop global competitive capabilities. *Journal of Operations Management*, **16**, 195-214.

Shi, Y., D. Fleet, and M. Gregory (2002) Understanding and conceptualising the global manufacturing virtual network. *Proc. of IEMC'02 IEEE Intern. Conf. on Engineering Management*, vol. 1, pp.119-124.

Sibertin-Blanc, C. (1993) A client-server protocol for the composition of Petri nets. *Proc. of the* 14^{th} *Intern. Conf. on Application and Theory of Petri Nets*, pp. 377-396.

Tolba, C., D. Lefebvre, P. Thomas and A. El Moudni (2005) Continuous and timed Petri nets for the macroscopic and microscopic traffic flow modeling. *Simulation Modelling Practice and Theory*, **13**, 407-436.

Villani, E., P.E. Miyagi, and R. Valette (2007) *Modelling and Analysis of Hybrid Supervisory Systems - A Petri Net Approach*, Springer, London.

Zhan, H.F. et al. (2003) A web-based collaborative product design platform for dispersed network manufacturing. *Journal of Materials Processing Technology*, **138**, 600–604.

Zhang, Y., M.Gregory, and Y. Shi (2006) Foundations of global engineering networks: essential characteristics of effective engineering networks. Proc. *IEEE Intern. Conf. on Management of Innovation and Technology*, vol. 2, pp.1113-1117.