Um Ambiente Paralelo/Distribuído para execução de Software de Controle

Luís Fernando Friedrich, Dr.

Departamento de Informática e Estatística
Universidade Federal de Santa Catarina - UFSC
Caixa Postal 476, 88010-970 Florianópolis, SC
lff@inf.ufsc.br

Resumo

O desenvolvimento de sistemas de controle representa hoje um dos maiores problemas na criação de indústrias de manufatura flexível. Uma boa parcela desta dificuldade pode ser atribuída aos altos custos envolvidos no desenvolvimento e manutenção do software de controle. Grande parte dos problemas encontrados no desenvolvimento de software para sistemas de controle decorrem da falta de um modelo adequado para tratar as necessidades envolvidas neste tipo de sistema. Por exemplo, os modelos adotados incluem uma abordagem hierárquica seqüencial e centralizada para o processo de desenvolvimento do sistema. Por outro lado, os ambientes de execução do software de controle para chão-de-fábrica (Shop Floor Control) são na maioria não distribuídos (*single-processor)*, fazendo com que os modelos não possam ser plenamente representados.

Neste trabalho apresenta-se um modelo de controle e um ambiente de execução distribuídos para o desenvolvimento/execução de software de controle. O modelo de controle é baseado em *objetos* (representando os componentes do sistema de controle), que definem a estrutura do software de controle, e no comportamento de cada um destes objetos que define a forma de interação entre os mesmos. O ambiente de execução é baseado em um *Multicomputador* onde os nós executam o comportamento dos objetos. O ambiente suporta os mecanismos básicos usados para expressar o paralelismo dos componentes de um sistema de controle e para permitir a interação dos componentes a partir da troca de mensagens.

1. Introdução

Um dos principais componentes das Células Flexíveis de Manufatura (**F**lexible **M**anufacturing **C**ell) é o controlador, ou sistema de controle da célula (SCC). Considerando que o objetivo das células de manufatura é a possibilidade de rápida adaptação às mudanças de mercado, o software de controle (que é o que realmente controla a célula) deve estar habilitado a receber as modificações necessárias para atender as constantes mudanças de requisitos.

Os modelos adotados incluem uma abordagem hierárquica seqüencial e centralizada para o processo de desenvolvimento do sistema [DUA93]. Mesmo que esta abordagem forneça uma ferramenta de representação útil para muitos sistemas, ela apresenta alguns problemas no tratamento das características dinâmicas de um sistema. Primeiro, o modelo centralizado é inadequado para representar atividades paralelas e simultâneas dos sistemas de chão-de-fábrica devido às restrições que o modelo impõe no que diz respeito à autonomia das entidades *controladas*. Segundo, visto que a centralização do controle exige a participação da entidade controladora em cada decisão, o sistema se torna menos flexível. Terceiro, a centralização do controle faz com que o sistema de controle seja mais dependente de aplicação e ajustado para configurações específicas do sistema, não fornecendo meios para uma rápida reconfiguração do sistema.

A adoção de uma estrutura distribuída para sistemas de controle de chão-de-fábrica permite a descrição das atividades de controle de uma forma genérica, a abstração necessária para os diferentes níveis e a possibilidade de reconfiguração necessária para representar as modificações no sistema.

Por outro lado, mesmo com a utilização de um modelo distribuído na especificação do sistema, a implementação do software assume uma forma diferente daquela do modelo desenvolvido. Isto é freqüentemente atribuído à utilização de ambientes que não suportam a implementação de software distribuído e paralelo.

As seções seguintes apresentam as necessidades básicas de um ambiente de execução paralela/distribuída, o modelo distribuído do software de controle e o ambiente de execução proposto.

2. Características de um ambiente paralelo

O processamento paralelo, no contexto de máquinas com memória distribuída, necessita de mecanismos através dos quais é possível expressar a execução paralela de um programa, como por exemplo: mecanismos para a expressão do paralelismo (particionamento), comunicação entre partes do programa e sincronização na execução destas.

2.1 Expressão do paralelismo

No caso da expressão do paralelismo é definida a unidade de paralelismo como sendo a menor parte de um programa capaz de ser executada em paralelo. Um dos conceitos mais básicos na expressão do paralelismo é o *processo*, definido como uma entidade totalmente independente que, executa algum código, possui dados e estado próprios e que podem ser criados implícita ou explicitamente[BAL89]. A criação

implícita esta associada a construções de linguagens de programação, enquanto que explicitamente os processos são criados através de uma função. Os processos em um ambiente paralelo podem ser criados de forma estática ou dinâmica. Na criação estática, é necessário conhecer a priori o número total de processos, sendo que os mesmos são definidos antes da execução e não existe alteração neste quadro até o programa terminar. A criação dinâmica permite que os processos possam ser criados a qualquer momento durante a execução do programa. A terminação dos processos é um fator importante, principalmente se existe uma hierarquia entre os mesmos, permitindo que o término de um processo cause a morte automática de todos os seus filhos ou o seu bloqueio a espera do término dos seus filhos.

A associação dos processos criados com os processadores do sistema é conseguida a partir da tarefa de *mapeamento*, que leva em conta o número de processadores disponíveis e a forma de distribuição dos processos objetivando o maior desempenho possível na execução de uma tarefa paralela. Isto é importante quando o número de processos é maior que o número de processadores. Este mapeamento pode ser feito pelo próprio sistema (suporte do ambiente paralelo) ou pelo programador.

2.2 Comunicação e Sincronização

A caracterização de um programa paralelo como sendo um *conjunto de processos que cooperam para a realização de uma tarefa*, define claramente a importância dos mecanismos de comunicação e sincronização num ambiente paralelo. A interação entre os processos pode ser realizada através de duas formas básicas: *memória compartilhada* ou *troca de mensagens*.

Na comunicação por memória compartilhada os processos utilizam uma mesma área de memória (comum aos processos comunicantes) para escrever e ler dados. Esta abordagem é mais adequada em máquinas paralelas com memória compartilhada e é necessário que mecanismos de sincronização [AND91] sejam utilizados para controlar o acesso a mesma área de memória.

Na abordagem de troca de mensagens os processos cooperam através do envio e recebimento de mensagens. Esta abordagem é mais adequada em máquinas paralelas com memória distribuída, onde a comunicação entre os processadores do sistema também se dá através da troca de mensagens. A sincronização dos processos, na abordagem de troca de mensagens, é obtida através da própria troca de mensagens, pois é possível a definição de uma política de sincronização para os processos receberem e enviarem suas mensagens. Estas políticas são definidas pelos modelos de comunicação por troca de mensagens existentes: síncrono, assíncrono e *rendezvous*. No modelo síncrono o processo emissor é bloqueado até que o receptor aceite a mensagem, enquanto que na troca assíncrona o processo emissor prossegue na sua execução mesmo que o processo receptor não tenha recebido a mensagem. O modelo *rendezvous* estabelece que os dois processos envolvidos na comunicação necessitam estar sincronizados para a mesma ser estabelecida.

Estes modelos envolvem apenas um processo emissor e um receptor definindo uma comunicação do tipo ponto-a-ponto. O modelo que estabelece a cooperação entre um conjunto de processos é conhecido como modelo de comunicação em grupo. Na comunicação em grupo permite-se aos processos o estabelecimento de comunicação do tipo um-para-todos, todos-para-um ou todos-para-todos. Neste caso é possível

identificar como vantagem uma maior facilidade no que diz respeito a expressão da comunicação entre múltiplos processos.

3. Modelo do Software de Controle

O modelo de controle estabelece os níveis de controle existentes assim como a estrutura do controlador e a forma de representação dos componentes.

3.1 A Arquitetura de Controle

A arquitetura de controle adotada neste trabalho leva em consideração a necessidade de informação global ao mesmo tempo que tenta manter a independência de desenvolvimento das entidades. A arquitetura de controle de chão-de-fábrica adotada prevê 2 níveis (Figura 1) : *equipamento* e *centros de trabalho*.

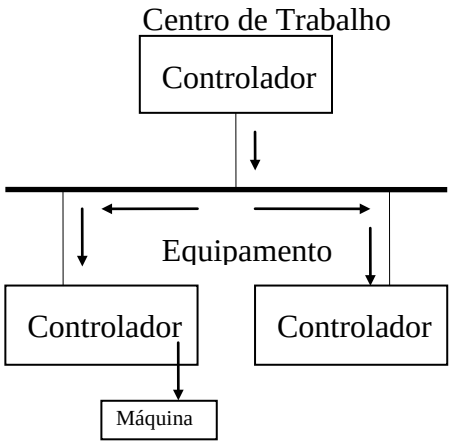

Figura 1. Níveis de Controle

O nível de equipamento tem como característica principal a representação lógica de uma máquina física, ou seja, uma entidade deste nível é composta pelo controlador do equipamento, o controlador da máquina, e a máquina propriamente dita. O controlador de equipamento está dividido em duas partes : 1) a parte genérica que funciona como interface entre a máquina que ele representa e o restante do chão-de-fábrica, e 2) a parte específica, que é responsável pelo controle da máquina real, através do controlador da máquina. O nível de equipamento pode ser definido como o conjunto de equipamentos $\mathbf{E = \{ e_1, e_2, ..., e_n \}}$, onde para cada $\mathbf{e_j \in E}$, $\mathbf{e_j = < Ce_j, M_j >}$ onde: $\mathbf{Ce_j}$ é um controlador de equipamento e $\mathbf{M_j}$ é uma máquina (dispositivo) física e seu controlador. O conjunto pode ser particionado em **{ EPM**, **EMM**, **ETM**, **EAM }** onde:

EPM = { $\mathbf{e_j}$ | $\mathbf{M_j}$ é um *equipamento de processamento de material* **}**,
EMM = { $\mathbf{e_j}$ | $\mathbf{M_j}$ é um *equipamento de movimentação de material* **}**,
ETM = { $\mathbf{e_j}$ | $\mathbf{M_j}$ é um *equipamento de transporte de material* **}**, e
EAM = { $\mathbf{e_j}$ | $\mathbf{M_j}$ é uma *equipamento de armazenamento de material* **}**.

O nível de centro de trabalho é composto por um ou mais equipamentos sob o controle do controlador de centro de trabalho. Neste nível, a ênfase é na sincronização dos controladores de equipamento. Usualmente, os centros de trabalho são definidos a partir do *layout* físico dos equipamentos.

Formalmente, $\mathbf{CT = \{ Ct_1, Ct_2, ..., Ct_n \}}$ é um conjunto de centros de trabalho onde para cada $\mathbf{Ct_i} \in \mathbf{CT}$, $\mathbf{Ct_i = < Cct_i, E_i, Pe/s_i >}$ onde:

$\mathbf{Cct_i}$ é o controlador do centro de trabalho,
$\mathbf{E_i = EPM_i \cup EMM_i \cup ETM_i \cup EAM_i}$,
$\mathbf{Pe/s_i}$ é um ponto de entrada/saída de material.

A arquitetura prevê a ligação entre entidades do mesmo nível, por exemplo entre centros de trabalho e entre equipamentos de um centro de trabalho, promovendo assim um modelo distribuído de controle. Vantagens desta estrutura incluem : a definição dos componentes de forma independente da estrutura global do sistema, possibilidade de descrição das atividades paralelas e simultâneas e capacidade de reconfiguração a partir da definição das interações entre os componentes.

3.2 Estrutura do Controlador

O controlador, independentemente do nível de controle (equipamento, centro), tem habilidade para receber informações, tomar decisões e trocar informações com os outros controladores [CHO93]. A função de tomada de decisões corresponde as funções de *planejamento* e *escalonamento*, enquanto que o recebimento e geração de informações corresponde à função de *execução*. As saídas do planejamento servem como entrada do escalonamento, as saídas do escalonamento servem como entrada para a execução, e as saídas da execução gerenciam os equipamentos físicos do chão-de-fábrica. A Figura 2 mostra a estrutura de um controlador.

A função de execução é desenvolvida com base no sistema físico ou no seu modelo. As funções de planejamento e escalonamento não dependem apenas da função de execução mas também dos requisitos de produção. Desta forma, as necessidades do planejamento e do escalonamento mudam conforme os requisitos de produção.

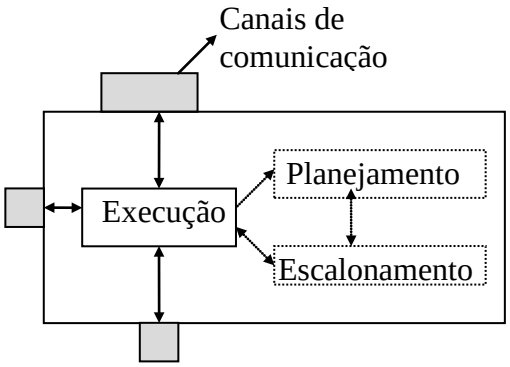

Figura 2. Estrutura de um Controlador

No caso da função de execução, as mudanças só acontecem quando da modificação da configuração da célula ou do centro de trabalho. A função de execução do controlador de centro de trabalho recebe as mensagens de controle, necessárias para processar as peças e gerenciar os recursos, e envia mensagens detalhadas para

a função de execução dos controladores de equipamento onde, para cada mensagem recebida, é especificada uma sequência de ações a serem tomadas. A função de execução dos controladores (centro de trabalho e equipamento), comunica-se com outros controladores e também com as funções de escalonamento e planejamento.

A função de execução pode ser detalhada em três sub-funções: 1) monitoração de mensagens de entrada, 2) execução de ações, e 3) geração de mensagens de saída. A sub-função de monitoração recebe mensagens dos outros controladores e as identifica para posterior execução de ações. Por exemplo, uma ação para um controlador de máquina pode ser a carga/descarga de peças, processamento, etc. A sub-função de geração de mensagens de saída envia mensagens (controle e informação) resultantes da execução de ações para os outros controladores. Desta forma, o controlador é um dispositivo reativo que responde a mensagens recebidas de controladores externos, a partir da interpretação das informações e da tomada das ações determinadas.

3.3 Representação dos componentes

A representação dos componentes do sistema leva em consideração a estrutura e o comportamento dinâmico dos mesmos. A estrutura é caracterizada pelos atributos (estáticos e dinâmicos) e pelas operações individuais. O comportamento define como cada subsistema executa suas operações.

O modelo funcional (estrutura) descreve os componentes individualmente, através da especificação dos seus atributos, operações e relacionamentos. Um componente é definido por:

C = { Cl, At, Op, Re } , onde :

- **Cl** é a classe do componente,
- **At** = $At_e \cup At_d$ é o conjunto de atributos do componente, onde :
  At_e é o conjunto de atributos estáticos, características que não mudam seus valores durante a operação, e
  At_d é o conjunto de atributos dinâmicos que mudam seus valores durante a operação (estado do componente).
- **Op** = $Op_e \cup Op_p \cup Op_s$ é o conjunto de operações do componente, onde :
  Op_e é o conjunto de operações que tratam eventos de entrada que recebem comandos ou requisições de outros componentes,
  Op_p é o conjunto de operações de processamento que executam tarefas de controle, e
  Op_s é o conjunto de operações que tratam eventos de saída através da comunicação com outros componentes.
- **Re** é o conjunto de outros componentes (recursos) que interagem com o componente sendo descrito.

O comportamento dos componentes é representado com o uso das *redes de comportamento* [FRI96]. A rede de comportamento de um componente é formada por um conjunto de estados e um conjunto de atividades. Cada estado representa uma condição particular do componente, e cada atividade representa a realização de alguma operação no ciclo de vida do componente. A idéia principal das redes de comportamento é inspirada nas redes de Petri [PET81].

Cada componente é representado por uma rede autônoma que troca mensagens com os outros componentes do sistema através dos *canais de Entrada/Saída*. Um canal de entrada recebe informações de fora do componente, enquanto que um canal de saída envia informações para fora do componente. O conjunto de canais de E/S do componente define a sua interface. A comunicação entre componentes é estabelecida pela ligação do canal de saída do enviante ao canal de entrada do receptor.

Uma rede de comportamento é um grafo dirigido **Rc = (E, C, A)**, onde:

- **E** é um conjunto finito de condições (lugares internos).
- **C** é um conjunto de canais externos, $C = C_e \cup C_s$ onde
  C_e é o conjunto de canais de entrada, e
  C_s é o conjunto de canais de saída
- **A** é o conjunto de atividades.

A cada atividade em **A** corresponde um par (E_e, E_s), onde E_e é o conjunto de estados de entrada e E_s é um conjunto de estados de saída da atividade. O conjunto E_e é um subconjunto não vazio de $C_e \cup E$, com no máximo um elemento de C_e. Se $E_e \cap C_e \neq \varnothing$ a atividade é dita global ou externa, caso contrário é chamada local ou interna. O conjunto E_s é um subconjunto de $C_s \cup E$. Para cada atividade em **A** está associada uma seqüência de ações. A seqüência de ações determina como são afetados os estados de saída correspondentes à atividade. A Figura 3 mostra a representação de um componente da classe Máquina_CN.

Neste caso o componente pertence a classe Máquina_CN, possui os atributos maq_status, que representa o estado da máquina (atributo dinâmico) , e maq_id, que representa o identificador da máquina (atributo estático).

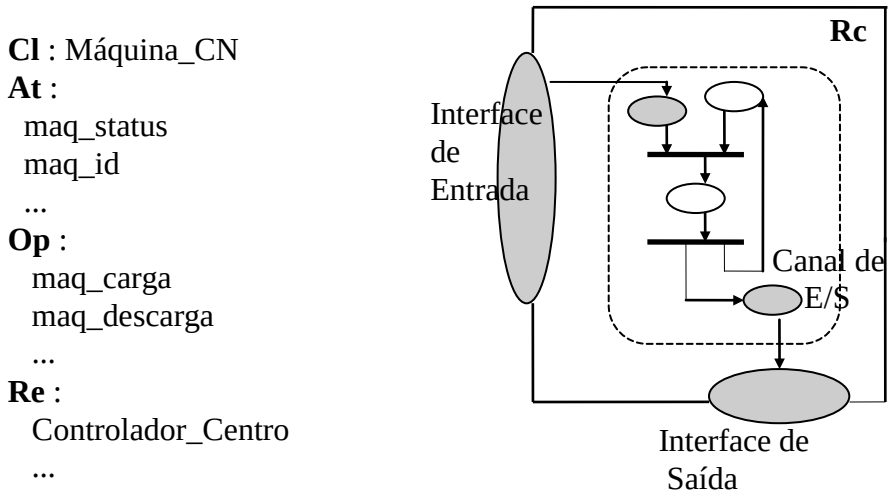

Figura 3. Representação de um componente

Entre as operações do componente estão as operações de carga de peças (maq_carga) e descarga de peças (maq_descarga). O componente descrito tem como um de seus relacionados o componente Controlador_centro com o qual relaciona-se através da troca de mensagens. Em seguida é apresentado a **Rc** do componente, ou seja, qual o comportamento que o componente executa. A Interface de entrada é composta por todas operações do componente que são consideradas de entrada, ou

seja, requisições de outros componentes. A interação com outros componentes é representada na interface de saída.

4. O ambiente de execução

Considerando o modelo do software de controle o suporte para implementação do mesmo tem como meta principal fornecer uma visão do sistema, onde os componentes do sistema são processos ativos que estão habilitados a se comunicar a partir de facilidades criadas para permitir a comunicação entre processos, suportando assim os conceitos estabelecidos no modelo, tais como: componentes e canais de comunicação.

A Figura 4., mostra como o software de controle é visto no ambiente de execução. Os componentes do sistema de controle (equipamentos, centros de trabalho) são processos, representados por círculos, que se comunicam entre si através da troca de mensagens, representado pelas setas. Cada um dos componentes tem um comportamento definido pela Rede de Comportamento associada ao componente. A comunicação se dá pelos canais de comunicação.

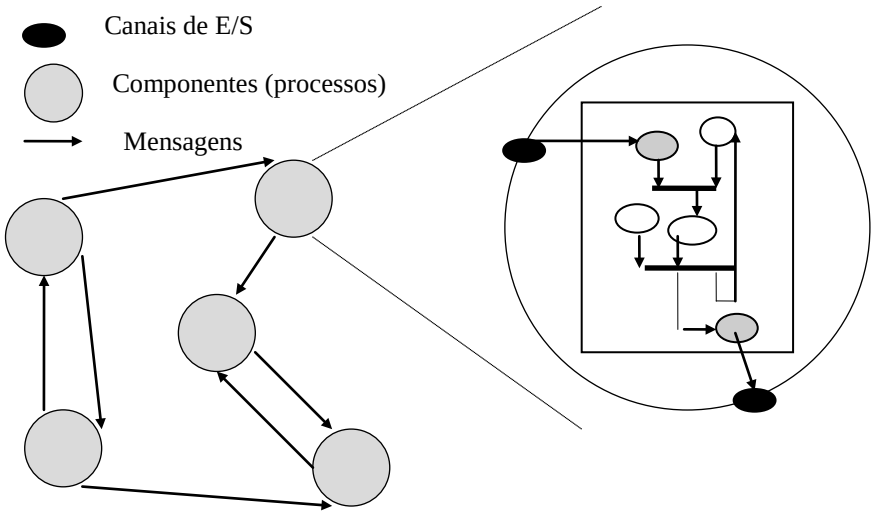

Figure 4. Visão do sistema no ambiente de implementação

Como os principais objetivos desta proposta são o de explorar a capacidade do processamento paralelo na execução de software de controle e com isto utilizar uma nova tecnologia para o desenvolvimento de software de controle, a plataforma escolhida para a implementação do sistema apresenta características de uma máquina paralela do tipo multicomputador. A máquina multicomputador é composta por processadores (PC 486), com memória local, interconectados por uma rede de interconexão dinâmica [COR93], como mostra a Figura 5.

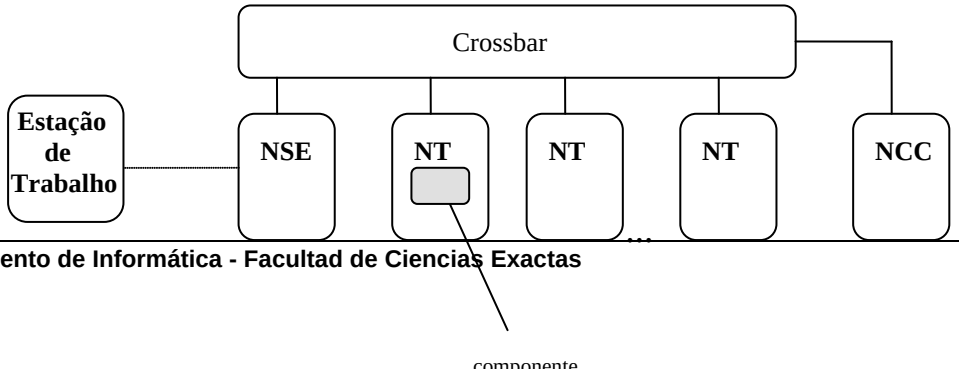

Figura 5. Ambiente de Execução

Cada processador ou nó de trabalho (**NT**) contém: o processo executor do componente (comportamento) que está sendo executado, um conjunto de serviços locais e um núcleo que suporta as primitivas do ambiente paralelo. Dois processadores têm funções específicas: o nó de controle de comunicação ou conexões (**NCC**) que é responsável pela gerência dos nós e estabelecimento das conexões entre os mesmos; e o nó servidor externo (**NSE**) que permite a comunicação com a estação de trabalho que fornece serviços do sistema de arquivos para os processos. Neste caso, a estação de trabalho funciona como interface entre o usuário e o sistema de arquivos.

4.1 As primitivas do ambiente de execução

O ambiente de execução proposto estabelece primitivas pelo menos duas abstrações que são consideradas fundamentais para ambientes de processamento paralelo: *processos* e *mensagens* [SCH94]. Estas primitivas ficam no componente biblioteca de cada um dos nós do multicomputador. As primitivas de gerência de processos, reponsáveis pela criação e remoção de processos, são:

- ***CreateProcess*** *(ntid)* Cria um processo no nó de trabalho *ntid*
- ***CreateAnyProcess****(ntid)* Cria um processo num nó de trabalho qualquer
- ***RemoveProcess****()* Remove um processo

Estas primitivas permitem que os processos executores do comportamento dos componentes possam ser criados e removidos de forma dinâmica. Por exemplo, um processo do tipo *centro de trabalho* cria um processo *equipamento* para cada um dos seus componentes e recebe os identificadores dos nós de trabalho(*ntid*) onde os mesmos foram alocados.

A comunicação entre os processos no modelo do software de controle acontece através dos canais de comunicação que são estabelecidos pelas redes de comportamento. Cada componente tem pelo menos um canal de comunicação através do qual recebe as mensagens que estabelecem a sua reação e outros canais de comunicação através dos quais envia suas requisições de serviço. Por exemplo, um controlador de *centro de trabalho* cuja configuração inclui um controlador de *equipamento de movimentação* (**EMM**) e dois controladores de *equipamento processador* (**EPM**), tem um canal de comunicação através do qual receberá as mensagens dos outros componentes (equipamentos, centros) e as mensagens de saída serão enviadas para os canais dos componentes com os quais o Centro interage. As primitivas usadas pelos processos para comunicar-se através dos canais

são orientadas ao modelo cliente-servidor e estão divididas em dois conjuntos: os serviços de acesso aos canais e serviços de comunicação por canais.

O primeiro grupo provê as primitivas que implementam serviços de acesso aos canais, as primitivas são:

- ***Register****(canal)* : requisita para o servidor de canais de comunicação o atendimento de um canal de comunicação, cujo identificador é especificado em *canal*.
- ***Remove****(canal)* : requisita a remoção do atendimento pelo canal de comunicação *canal* para o servidor de canal de comunicação.
- ***Get_channel****()* : requisita ao servidor de canais de comunicação o identificador de um dos processos que atende em *canal*.

O segundo conjunto fornece os serviços para a transmissão de mensagens utilizando os canais de comunicação:

- ***SendRec*** *(canal, requisição, tam_req, resposta, tam_res)* : envia ao canal de comunicação designado por *canal*, uma mensagem e espera pela resposta. O processo que está enviando a mensagem fica esperando que o processo responsável pelo atendimento do canal receba a mensagem e envie a resposta.
- ***Receive*** *(requisição,tam_req)* : recebe uma mensagem do canal de comunicação ao qual esta registrado. O processo permanece bloqueado até que um processo qualquer envie uma mensagem para o canal especificado.
- ***Reply*** *(Pid,resposta, tam_res)* : envia resposta a um serviço requisitado por algum processo (*Pid*) através da primitiva *sendrec*.

4.2 As primitivas de suporte (kernel) e os servidores

Além disto, existe um conjunto de primitivas que fornece o suporte básico para a execução do modelo paralelo. Conforme a plataforma descrita a conexão entre os diferentes processadores (nós) é conseguida a partir do servidor de comunicações que executa no nó . A comunicação com o nó **NCC** é feita a partir da utilização de uma função que envia uma mensagem e espera resposta. Esta função está assim definida :

- ***BS_SendRec*** *(msg)* Envia uma mensagem para o **NCC** e aguarda resposta

O recebimento de uma mensagem deste tipo faz com que o **NCC** atenda a requisição contida em *msg* e envie uma resposta. Para isto o **NCC** possui internamente duas outras funções *BS_ReceiveAny* e *BS_Send*. Estas funções são executadas através da requisição dos serviços do controlador de conexões. O **NCC** também é responsável pelos serviços de suporte as conexões e a alocação de nós processadores **,** que são utilizados na criação e comunicação entre processos. Os serviços fornecidos são :

- ***Allocate*** *(ntid)* Aloca um nó específico *ntid*.
- ***AllocateAny****()* Aloca um nó qualquer.
- ***Deallocate*** *()* Libera o nó corrente.
- ***Connect*** *(nid)* Pede conexão entre o nó *nid*.
- ***Connect_any****()* Pede uma conexão qualquer nó.

- ***Disconnect()*** Pede desconexão.

Por exemplo, a execução da primitiva ***CreateProcess*** *(ntid)* faz com que o processo criador requisite o serviço ***Allocate*** *(ntid)* para o servidor de conexões no nó **NCC** . A execução deste serviço implica na verificação da tabela de nós livres a procura de um nó que possa receber um processo para executar. Como resposta o servidor de conexões retorna para o processo (nó) criador o identificador do nó alocado.

O suporte de implementação para o conceito de canais de comunicação necessita, além das primitivas apresentadas na seção 4.1, uma estrutura de dados que permita o gerenciamento das comunicações. Este gerenciamento é responsabilidade do servidor de canais (**NCC**) o qual mantém a estrutura de dados, mostrada na Figura 6, que define os canais de comunicação. A principal função do servidor de canais é manter o mapeamento entre as portas e a localização dos servidores que as atendem. Cada elemento da lista contém o identificador do processo (nó) que está atendendo no canal desejado.

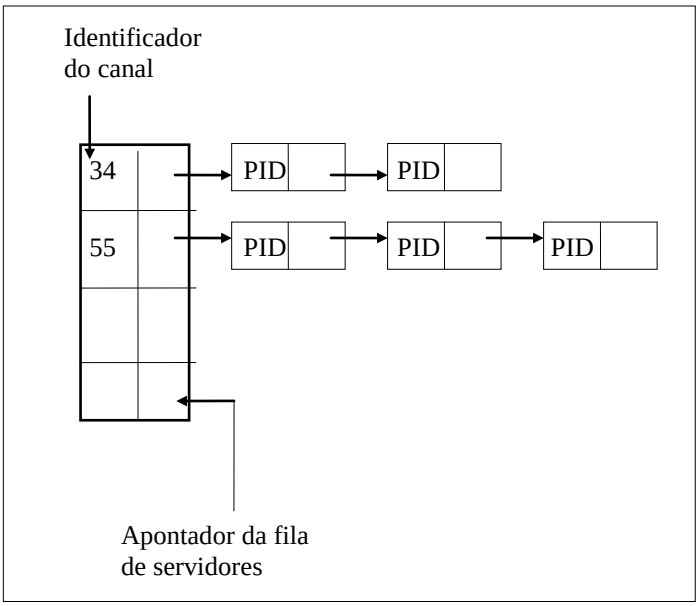

Figura 6. Estrutura de dados do Servidor de canais

A execução de um ***SendRec*** implica na requisição do identificador do processo que está servindo o canal, primitiva ***Get_channel***, depois no pedido de conexão com o nó responsável pelo canal, primitiva ***Connect*** *(nid)*, e finalmente o envio da mensagem.

Quando da criação de processos no ambiente paralelo é importante colocar de que forma o código a ser executado é associado ao processo criado. Neste caso define-se uma camada a mais no gerenciamento de processos para que o mecanismo adotado seja o mesmo utilizado em sistemas Unix, ou seja, *fork* para criação de um processo a semelhança do pai e *exec* para associação de um novo código executável.
No que diz respeito aos serviços não suportados pelo ambiente de execução é definido um servidor específico, *servidor externo* (**NSE**) que atende todas as requisições dos processos. Na arquitetura considerada, o *servidor externo* comunica-se com a *estação de trabalho* para executar as requisições.

Conclusão

Neste trabalho foi apresentado um modelo para o desenvolvimento de software de controle baseado no paradigma de troca de mensagens e um ambiente de execução, paralelo, que suporte o modelo utilizado. O paradigma de troca de mensagens, pela sua flexibilidade, é adequado ao modelo de software de controle adotado permitindo a elaboração de programas que expressam de forma eficiente os requisitos do mesmo. O ambiente paralelo de execução é composto por um conjunto de primitivas de suporte para criação dinâmica de processos, canais de comunicação e para a sincronização entre processos. O suporte básico de execução do modelo tem como plataforma um multicomputador com rede de interconexão dinâmica, onde cada processador executa um processo e a comunicação entre eles é feita por troca de mensagens. O ambiente de execução atende, inicialmente, as necessidades básicas do modelo de software de controle adotado. Entretanto, é possível expandir o ambiente de execução para que o mesmo suporte outros conceitos que poderiam ser úteis para a execução do software de controle (por exemplo, *multi-threads*). O ambiente paralelo de execução foi implementado em um simulador da plataforma (*multicomputador*) utilizada, possibilitando futuras investigações com respeito a desempenho e capacidade de expressar o paralelismo desejado. As primeiras experiências realizadas mostram que o desenvolvimento do software de controle, a partir do modelo de controle adotado, apresenta características como: modularidade (equipes diferentes podem desenvolver em paralelo componentes diferentes), reusabilidade (os modelos de componentes desenvolvidos podem ser utilizados para diversas aplicações com poucas modificações) e flexibilidade (basta mudar a ligação dos componentes para se obter um comportamento global diferente). Por outro lado a utilização de um ambiente paralelo de execução permite que o *gap* que existe entre o modelo projetado e a execução do software de controle propriamente dita, seja pelo menos grandemente diminuído. Espera-se, num futuro próximo, a conclusão de um protótipo da plataforma multicomputador com até 8 processadores, possibilitando assim uma verificação mais *real* das vantagens e desvantagens da utilização de um ambiente paralelo para execução de software de controle.

Referências Bibliográficas

[AND91] ANDREWS, G. R. Concurrent Programming: Principles and Practice. Redwood City: Benjamin/Cummings Publishing Company, 1991.

[BAL89] BAL, H.E.; STEINER, J.G.; TANEMBAUM, A.S. Programming Languages for Distributed Computing Systems. ACM Computing Surveys, New York, v.21(3), n.1, p.261-322, Set. 1989.

[COR93] CORSO, Thadeu B. Um Ambiente de Programação Paralela para um Multicomputador. UFSC-CTC-INE, Florianópolis ,SC. Novembro, 1993.

[CHO93] CHO, Hyuenbo. Na Intelligent Controller for Computer Integrated Manufacturing. PHD Thesis, Texas A&M University, December 1993.

[DUA93] DUAN, Niu. Extended Moore Machine Network Model for Discrete Event Control of Flexible Manufacturing Systems. P.H.D. Thesis, The Pennsylvania State University, December 1993.

[FRI96] FRIEDRICH, Luis F. Uma Abordagem Distribuída no Desenvolvimento e Implementação do Software de Controle de Chão-de-Fábrica em Sistemas de Manufatura Celular. Tese de Doutorado. Universidade Federal de Santa Catarina. Fevereiro, 1996.

[PET81] PETERSON, J.L. Petri net: theory and modelling of systems. Englewood Cliffs, Prentice-Hall, 1981. 287p.

[SCH94] SCHRÖDER-PREIKSCHAT, Wolfgang. The Logical Design of Parallel Operating Systems. Prentice-Hall, 1994. 370p.