UM NÚCLEO DE SISTEMA OPERACIONAL DISTRIBUÍDO PARA APLICAÇÕES EM TEMPO REAL

J. Fraga, J. M. Farines, L. Nacamura e V. Mazzola

Laboratório de Controle e Microinformática
Departamento de Engenharia Elétrica
Universidade Federal de Santa Catarina
Caixa Postal 476 - Florianópolis - SC

Resumo:

Este artigo descreve um núcleo de sistema operacional distribuído para aplicações em tempo real. O mecanismo foi especificado de maneira a permitir que as comunicações sejam uniformes e independentes da distribuição das tarefas no sistema. Este núcleo foi desenvolvido para servir como suporte de um sistema de programação distribuída para aplicações em tempo real. Sendo assim, oferece operações que facilitam a implantação e a gestão de configurações de sistemas distribuídos.

1. Introdução

Considerando a evolução atual em controle de processos e sistemas em tempo real, nos defrontamos com a crescente utilização de redes de computadores. São inúmeras as vantagens, apontadas na literatura, destes sistemas em relação a sistemas mais clássicos. A maior disponibilidade de recursos, o desempenho e as facilidades para o crescimento incremental são fatores normalmente citados.

As tendências observadas conduzem às necessidades de uma estrutura básica para o gerenciamento e a integração dos recursos distribuídos na rede, e de ferramentas adequadas para o desenvolvimento, validação e testes da implementação de aplicações. Estas necessidades foram determinantes para a nossa motivação no projeto de um ambiente de programação distribuída para aplicações em tempo real. Tais ambientes têm sido recentemente objeto de um esforço considerável /Leite85/, /Halsall83/, /Bressan85/, /Kramer83/, /Lopes85/.

O ambiente de programação distribuída proposto segue a abordagem "Programming in the Small - Programming in the Large" /DeRemer76/, onde um programa é construído a partir de um conjunto de componentes menores (módulos), desenvolvidos e compilados separadamente. Nestas condições, uma "configuração" de uma aplicação envolve a definição dos módulos que a compõem, a atribuição destes componentes às estações do sistema distribuído e do estabelecimento das ligações entre estes módulos. Exemplos notórios da utilização desta abordagem são os sistemas de programação MESA /Lampson80/ e CONIC /Kramer83/.

A separação na construção dos componentes de um programa ("Programming in the Small") daquela do próprio programa ("Programming in the Large") permite grande flexibilidade no desenvolvimento, implantação e gerenciamento de aplicações em tempo real: um componente pode participar de diferentes configurações, uma configuração pode conter várias instâncias de um componente, uma configuração pode ser modificada "on-line" (configuração dinâ-

mica). Em /Kramer85/ são discutidos os tipos possíveis de configuração na utilização desta metodologia para construção de programas distribuídos.

Outra necessidade reconhecida de um sistema distribuído é a existência de um suporte para tempo de execução. Este suporte, normalmente chamado de "kernel" de sistema operacional distribuído, é o responsável pela implantação do ambiente multitarefas que executa a aplicação distribuída. Consideramos que suas funções e atributos devem satisfazer os requisitos sugeridos no modelo de arquitetura de sistemas distribuídos definido em /Watson81/.

Neste sentido, no Laboratório de Controle e Microinformática (LCMI/EEL) da UFSC, estão sendo realizados estudos e trabalhos sobre: linguagem de programação e definição de testes para os componentes, linguagem e facilidades de sistema operacional necessárias para a configuração de um sistema e ainda de ferramentas para a validação destas configurações.

O objetivo desta publicação é a apresentação do kernel distribuído para sistemas em tempo real que se encontra em fase de desenvolvimento no LCMI. Os itens subsequentes descrevem o sistema considerado e as funções e atributos do kernel necessários para um ambiente de desenvolvimento e validação de programas distribuídos de aplicações em tempo real.

2. Sistema Considerado

O sistema distribuído é formado por um conjunto de estações (computadores pessoais) que se comunicam via uma rede local. Estas estações são divididas em estações de execução da aplicação e uma estação posto de trabalho.

A estação posto de trabalho deve conter facilidades para desenvolvimento, validação e testes iniciais dos componentes dos programas, o configurador, além de facilidades de sistema operacional e funções do kernel que permitam o gerenciamento local e a interação com o resto do sistema. As estações de execução apresentam funções para implantação do ambiente de execução multitarefas distribuído (funções do kernel) e facilidades de sistema operacional que permitem operações a nível local de carregamento e criação de estruturas em tempo de execução, de modo a suportar a instalação e possíveis modificações de configurações.

3. Suporte em Tempo de Execução

Entre as funções principais do kernel distribuído estão incluídas:

- comunicação e sincronização de tarefas;
- gerenciamento de tarefas;
- gerenciamento dinâmico de memória;
- manipulação de interrupções e E/S;
- operações utilizadas por facilidades de gerenciamento local na implantação de uma configuração.

Estas funções são objeto dos ítens subsequentes.

3.1.1. Comunicação e sincronização

A arquitetura de um sistema distribuído está centrada sobre os mecanismos de comunicação e sincronização entre tarefas (IPC) /Watson81/. Um bem definido modelo de IPC (embutido em um cuidadoso conjunto de construtos da linguagem de programação de componentes) fornece uma disciplina para o desenvolvimento de programas concorrentes e distribuídos que sejam eficientes e bem estruturados.

O ambiente de execução fornecido pelo kernel tem a comunicação e a sincronização uniformes e independentes da distribuição das tarefas no sistema. Dos dois modelos básicos existentes para comunicação e sincronismo: "troca de mensagens" e "chamada de procedimento", utilizamos o primeiro na implementação das primitivas de comunicação do kernel. O modelo troca de mensagens é bastante comum em sistemas de tempo real. Estes sistemas, principalmente em controle de processos (SDCD), são constituídos por um número relativamente pequeno e estático de tarefas em tempo de execução, coincidindo em parte com as características apontadas em /Lauer79/ para o modelo troca de mensagens.

Embora a dicotomia representada pelos dois modelos seja apresentada como falsa em /Lauer79/, no contexto de linguagens de programação, fazendo uso do modelo troca de mensagens, a interação entre tarefas é via chamadas de procedimentos SENDs e RECEIVEs. Neste caso, temos dois construtos de linguagem não relacionados sintaticamente, o que pode permitir a construção de comunicações não estruturadas, propensas a erros de interações. No modelo chamada de procedimento, a interação se dá através de um simples construto de linguagem: uma chamada de procedimento. A dificuldade, neste caso, é manter, em sistemas distribuídos, a propriedade que toda chamada de procedimento deve sempre retornar uma resposta.

3.1.1. Primitivas de comunicação

Transferência de dados e sincronismo estão fortemente ligados em primitivas de comunicação. A forma com que se dá o acoplamento sincronismo/comunicação determina diferenças nas operações de envio e recepção. Três tipos básicos de "envio" são normalmente identificados na literatura /Liskov79/, /Scott83/, /Rashid81/:

- SENDNB (no-wait send) - operação (SEND não bloqueante) em que a tarefa emissora não é bloqueada, após o envio da mensagem;

- SENDB - nesta operação (SEND bloqueante), a tarefa emissora é colocada em estado de espera até a recepção da mensagem;

- SENDWAIT - nesta operação, identificada como "envio com resposta", a tarefa emissora deve esperar a mensagem-resposta da tarefa receptora.

O "envio com resposta" se aproxima do modelo "chamada de procedimento". As outras opções de envio simplesmente transferem mensagens. No nosso sistema, as três operações citadas acima são implementadas pelo procedimento:

SEND(porto de saída, mensagem, resposta, timeout);

Onde:

porto de saída - referência local ao descritor de porto (item 3.1.2);

mensagem, resposta - endereços de buffers no espaço de endereços da tarefa emissora, usados respectivamente no envio da mensagem e na recepção da resposta (caso SENDWAIT);

timeout - especifica valores de timeout para os envios com espera (SENDB e SENDWAIT). Timeout = 0 define envio SENDNB;

As operações de recepção são normalmente divididas em receives bloqueantes (RECEIVEB) e não bloqueantes (RECEIVE), conforme a tarefa receptora é ou não colocada em estado de espera pelo envio. O uso mais generalizado é dos receives bloqueantes.

Alguns sistemas introduzem operações de recepção em grupo de portos: os "receives condicionais" e "receives seletivos". Estas operações permitem uma forma de "polling" no grupo de portos e também um certo grau de concorrência na recepção. O sistema ACCENT /Rashid81/ apresenta uma primitiva receive que opera sobre grupo de portos. Nesta primitiva a tarefa é bloqueada à espera da primeira mensagem que chegue no grupo de portos.

No executivo do CONIC é definido um conjunto de rotinas que associadas a condições lógicas permite a implantação do construto SELECT, definido em ADA /Brender81/. A utilização do SELECT na recepção forma então os receives seletivos.

As recepções bloqueantes, não bloqueantes e seletivas são suportadas por uma única primitiva. Esta primitiva é na realidade um receive seletivo, onde o receive único, seja ele bloqueante ou não, é transformado em um receive seletivo de apenas uma alternativa de recepção. Esta solução é a mesma apresentada no sistema CONIC. A primitiva de recepção é suportada pelos dois procedimentos subsequentes:

INICIA RECEPÇÃO (guarda, porto de entrada, mensagem);

Onde:

guarda - variavel booleana que indica se o respectivo porto de entrada está ou não habilitado;

porto de entrada - nome local que referencia o porto sobre o qual o procedimento deve operar;

mensagem - endereço do espaço de dados da tarefa receptora indicando o buffer para recepção da mensagem.

Este procedimento verifica o estado do guarda para cada alternativa dentro do construto SELECT. Se o guarda for verdadeiro, o endereço mensagem é registrado no descritor do porto de entrada. Em caso negativo, verifica o guarda da próxima alternativa. Os descritores dos portos habilitados são

enfileirados de forma a facilitar a verificação da existência de mensagem em cada porto.

RECEIVE (timeout);

Onde:

Timeout - tempo máximo que a tarefa pode ficar bloqueada à espera de mensagem. Para o receive não bloqueante utiliza-se timeout igual a zero. Utilizando-se Timeout = -1, a tarefa ficará bloqueada até a chegada da mensagem;

Este procedimento percorre a fila formatada no procedimento anterior, executando a recepção no primeiro porto que contém mensagem.

As comunicações entre tarefas podem ser colocadas em termos de comando/resposta (modelo cliente-servidor /Watson81/). Consideramos então que tarefas necessitam de "respostas" e que este tipo de requisito combina com a argumentação apresentada em /Saltzer81/ (end-to-end arguments) ou seja, a "resposta" serve como um reconhecimento fim-a-fim.

Neste caso, as comunicações entre tarefas levam à definição dos chamados pares "envio/resposta" /Liskov83/. Os pares "envio/resposta" podem ser definidos como "síncronos" quando o emissor é bloqueado até a chegada da resposta, e "assíncronos" quando o envio de uma mensagem e a recepção de uma resposta estão desacoplados. Neste último, a resposta é esperada mais tarde na tarefa emissora, como parte da semântica do mecanismo de comunicação ou da construção de comunicações bem estruturadas.

O sistema de programação distribuída ARGUS /Liskov83/ utiliza pares "envio/resposta" síncronos (característica do modelo "chamada de procedimento"). Em /Lauer79/, no modelo "troca de mensagens", é apresentado o envio e a resposta desacoplados. O desacoplamento não deve remover a necessidade da resposta em comunicações bem estruturadas.

Tomando como exemplo o sistema CONIC, adotamos a primitiva REPLY:

REPLY (porto de entrada, mensagem);

Onde:

porto de entrada - nome local que referencia o porto para o qual a resposta é enviada;

mensagem - endereço do buffer mensagem da tarefa emissora utilizado no envio do porto.

O REPLY compõe junto com a operação SENDWAIT um par "envio/resposta" síncrono (SENDWAIT/RECEIVEB... REPLY).

Pares "envio/resposta" assíncronos podem ser formados no nosso sistema a partir de combinações de SENDNBs e SENDBs e RECEIVEs. Com pares assíncronos podemos montar comunicações do tipo "pipeline", onde envios são executados sucessivamente, um atrás do outro e as respostas se sucedem da mesma forma. O modelo "pipeline" se aproxima dos chamados protocolos "con-

tínuos" enquanto o "envio/resposta" síncrono corresponde a um protocolo "pára-e-espera" ("stop-and-wait"). O uso de SENDB e SENDNB é determinado pela necessidade ou não de que o dado seja recebido. Em aplicações onde a perda de informações não e significativa diante da necessidade de desempenho, o SENDNB é o mais indicado.

3.1.2. Portos

A noção de "porto" é classicamente introduzida no modelo "troca de mensagens" como solução ao endereçamento e tratamento das filas de mensagens associadas às tarefas. Consideramos "porto", no projeto, uma estrutura de dados mantida pelo kernel (espaço de dados do kernel). É constituído de um descritor no qual são mantidos registros de informações relacionadas com a comunicação e sincronização da tarefa proprietária do porto. Eventualmente, em determinados tipos de interações, filas de mensagens são associadas ao porto (item 3.1.3).

A necessidade de separação da construção de um programa distribuído (configuração) daquela de seus componentes determinou certas imposições sobre a definição de porto:

- As interfaces dos componentes (módulos) no modelo "troca de mensagens" são definidas em termos de portos. A independência necessária de um módulo (sem referências externas) é mantida criando as noções de portos de entrada e portos de saída. Módulos (tarefas do módulo) enviam mensagens através de portos de saídas e as recebem de portos de entrada. Portos para comunicação intermódulo fazem parte das interfaces destes módulos. Portos de saída de um módulo são ligados a porto de entrada de outro módulo, na configuração de um programa;

- As características de sistemas estáticos das aplicações em controle de processos, determina que portos sejam criados estaticamente, em tempo de configuração. Portos como estruturas criadas estaticamente facilitam também a separação entre programação de componente e configuração de sistema.

Estas considerações determinam a necessidade de operações criar/destruir portos e ligar/desligar portos (tanto inter como intra-módulos[1]). Estas operações suportadas pelo kernel são disponíveis às facilidades de sistema operacional que participam da implantação das configurações e gestão local.

3.1.3. Mensagens

Certas considerações devem ser feitas sobre as técnicas de "bufferização" (ligadas ao controle de fluxo) e à estrutura de mensagens na implementação de mecanismos de comunicação. A manutenção de "buffers" tem associado o problema de alocação de recursos. Duas abordagens são normalmente apresentadas para a "bufferização":

---

1) Definimos módulo de maneira similar aos conceitos de "guardian" em ARGUS /Liskov83/ e de "task module" nas primeiras versões do CONIC /Kramer83/. Neste caso, módulo é constituído de uma coleção de tarefas e dados (programa concorrente). Módulos são objetos de tempo de configuração, enquanto portos e tarefas são objetos de tempo de execução.

- "buffers" utilizados nas mensagens fazem parte do espaço de endereço das tarefas. Neste caso, uma mensagem é declarada da mesma forma que variáveis locais. A conclusão do envio de uma mensagem é a cópia de uma variável no espaço do emissor para outro do espaço do receptor;

- "buffers" de mensagens são objetos do kernel organizados em filas associadas a portos. A alocação destes "buffers" pode ser estática, ligada à própria criação do porto, ou ainda dinâmica segundo as necessidades.

No primeiro caso, a passagem direta do espaço de memória do emissor para o do receptor torna desnecessário o uso de cópias intermediárias, aumentando o desempenho. Esta técnica é característica de pares "envio/resposta" síncronos. Interações que envolvem pares assíncronos com o envio não bloqueante, podem necessitar "buffers" intermediários e nestas circunstâncias a segunda abordagem é a considerada.

Estas duas técnicas são empregadas no nosso projeto. Utilizamos alocação dinâmica para o caso dos "buffers" intermediários. Embora haja perda de desempenho, a alocação dinâmica permite a otimização do uso da memória. A exemplo de outros sistemas, utilizamos também mensagens e portos com tipos definidos ("typed"), o que além de melhorar a confiabilidade das comunicações (verificação de incompatibilidade de tipos em tempo de compilação e de configuração), facilita a alocação dinâmica dos "buffers".

3.1.4. Tipos de Endereçamento

As comunicações são constituídas a partir de "associações" de dois ou mais portos que possibilitam a troca de mensagens. Estas associações caracterizam caminhos uni- ou bi-direcionais de dados e formam também diferentes tipos de endereçamento:

a. Associação "um-para-um"

A partir das operações de envio e de recepção introduzidas no item 3.1.1, podemos construir diferentes associações envolvendo um porto de saída "ligado" a um porto de entrada. A distinção entre estas associações está ligada às particularidades das operações utilizadas:

- Os casos SENDNB/RECEIVEB e SENDB/RECEIVEB determinam associações unidirecionais onde, para o envio não bloqueante, existe uma necessidade provável de memorização intermediária da mensagem;

- A associação definida por SENDWAIT/RECEIVEB ... REPLY é caracterizada pela bi-direcional, constituindo assim um par envio/resposta síncrono.

b. Associações "um-para-vários" e "vários-para-um"

Associações "um-para-vários" (endereçamento multi-destino, difusão seletiva, etc...) e associações "vários-para-um" também podem ser obtidas a partir destas operações de envio e de recepção citadas acima. A dificuldade nestas composições está em acomodar a semântica dos envios bloqueantes a este tipo de endereçamento. O não enquadramento com estes dois tipos de endereçamento determinou a não utilização da operação SENDB.

3.1.5. Comunicação remota

O mecanismo de comunicação, como apresentado anteriormente, é definido em termos de comunicações entre tarefas na mesma estação. A extensão do IPC, de forma transparente sobre a rede, é feita classicamente usando tarefas "servidoras de rede" /Rashid81/, /Ananda84/, /Kramer86/. Os requisitos colocados a um servidor de rede são o fornecimento de alguma forma de comunicação entre estações e que a sua existência possa acomodar as semânticas das primitivas de IPC, (definidas localmente) na extensão a nível de rede.

O servidor de rede que definimos "encapsula" a execução dos comandos da interface de transporte da rede local utilizada. Em relação ao escalonamento, a exemplo de tarefas que encapsulam drivers de E/S, é atribuída a esta tarefa a prioridade sistema (item 3.3). O servidor deve preparar o formato da mensagem aceito pelo serviço de transporte. A passagem da mensagem se da usando uma memória comum entre o servidor de rede e a entidade de transporte.

O kernel esta sendo desenvolvido provisoriamente em uma rede local comercial. O protocolo de transporte é a conexão e apresenta facilidades de controle de erro, como, por exemplo, retransmissão de pacotes. Estas características são tidas como discutíveis em se tratando de tempo real, devido às restrições impostas no desempenho da rede e ainda considerando que os níveis superiores (kernel e aplicação) apresentam funções de controle de erro e de fluxo, o que representa duplicação de esforços. Nos sistemas ditos embutidos ou integrados e em sistemas onde o desempenho é essencial, o uso mais generalizado é o de serviços sem conexão ou datagramas para o envio da mensagem através da rede (serviço de transporte praticamente inexistente).

O servidor de rede tem um porto de entrada e um de saída para comunicações com tarefas locais. O envio (nisto se inclui o REPLY) sobre um porto local para um porto remoto é transformado no envio ao porto de entrada do servidor de rede da estação local. Informações de ligação no descritor do porto da tarefa local identificam o porto de recepção remoto. Na recepção, o uso de pares síncronos (SENDWAIT/RECEIVEB... REPLY) na transferência da mensagem do servidor de rede para a tarefa local determina o bloqueio do servidor de rede numa comunicação, limitando a utilização da rede por outras comunicações. A criação de várias instânciações do servidor de rede em uma estação evita este problema (solução adotada no executivo do sistema CONIC). Adotamos uma solução mais simples que esta em termos de gerenciamento de recursos. Optamos pelo uso de SENDNB nos envios do servidor de rede para as tarefas locais. As semânticas das primitivas de comunicação e dos tipos de endereçamentos definidos nos ítens anteriores são mantidas nas considerações feitas para o envio remoto e na recepção da rede.

3.2. Gerenciamento de Tarefas

As tarefas são representadas por estruturas de dados classicamente conhecidas por Descritores de Tarefa. Os descritores são objetos mantidos pelo kernel e utilizados para o registro de informações referentes às tarefas associadas.

São definidos quatro estados que poderão ser assumidos pelas tarefas

durante a operação do sistema: Ativa, Pronta, Em Espera e Suspensa, onde as Filas de Prontas e Espera são organizadas segundo critérios de prioridade e tempo de espera (timeout), respectivamente. O escalonamento segue uma estratégia do tipo "preempty" por prioridade, que é a mais adequada a sistemas em tempo real. O sistema oferece dezesseis níveis de prioridade assim definidos: níveis 0 a 3 - prioridade Sistema; níveis 4 a 7 - prioridade Usuário Alta; níveis 8 a 11 - prioridade Usuário Média; níveis 12 a 15 - prioridade Usuário Baixa.

3.3. Tratamento de Interrupções e Operações de E/S

Drivers de E/S são encapsulados por tarefas do sistema, permitindo a interface da estrutura E/S com o modelo troca de mensagens. No nosso projeto, mantemos os drivers do sistema operacional original dos microcomputadores utilizados. De maneira idêntica, as interrupções são atendidas por tarefas do sistema, ativadas por um manipulador de interrupções no momento da sua ocorrência. As tarefas de interrupção e os drivers de E/S são declaradas com prioridade Sistema, a fim de evitar o preesvaziamento por outras tarefas.

3.4. Temporização

O kernel suporta uma função de temporização para tratamento da Fila de Espera do sistema. Esta função é ativada por relógio de tempo real e o seu objetivo é testar o timeout definido numa operação de comunicação. O mecanismo de timeout funciona sobre bases locais, sendo que a comunicação remota utiliza o conhecimento a priori dos atrasos do IPC inter estação.

3.5. Gerenciamento de Memória

Em função da necessidade de criação ou destruição de tarefas durante a operação do sistema, o kernel deve suprir funções de gerenciamento de memória. Estas funções possibilitam a alocação e devolução de segmentos da memória disponível na estação.

As funções de gerenciamento de Memoria são ativadas a partir das primitivas de criação, destruição, carga e descarga de módulo da linguagem de configuração. O kernel suporta também funções de alocação/dealocação de buffers intermediários para as mensagens (item 3.1.3).

Para os objetos de tamanho fixo, o problema é resolvido com o uso de variáveis dinâmicas (procedimentos NEW e DISPOSE do PASCAL). A solicitação de blocos de tamanho variável é atendida por uma lista de blocos livres. A estratégia de alocação utilizada é a first-fit /Peterson84/, pelo melhor desempenho algorítmico. Para este caso, mantemos uma tarefa de baixa prioridade responsável pela junção de blocos contíguos devolvidos ao bolo de memória da estação.

4. Conclusão

A integração dos recursos distribuídos numa rede, de forma a que estes cooperem na execução de uma aplicação é uma das características necessárias para os sistemas distribuídos utilizados em aplicações em tempo real. Estes sistemas são ditos "embutidos" devido à forma hierárquica de controle a que está submetida a aplicação. Embora muitos dos padrões definidos para a

transferência de dados em redes de computadores possam ser utilizados, a abordagem de tais sistemas é distinta daquela dos sistemas "abertos" introduzidos nos modelos de referência OSI/ISO /Day83/ e MAP /Crowder85/ em se tratando de autonomia /Sloman86/.

Alguns dos atributos dos sistemas embutidos se aproximam de requisitos sugeridos no modelo proposto em /Watson81/ Dentro desta perspectiva, discutimos então o kernel de sistema operacional distribuído. O IPC apresentado pelo suporte de tempo de execução visa comunicações uniformes e independentes da distribuição das tarefas no sistema. As estruturas de tempo de execução e operações do kernel foram conformadas de maneira a facilitar a implantação de configurações de programas de sistemas distribuídos a partir de um posto de trabalho.

O tratamento de exceçõoes é feito a nível de aplicação. Esta abordagem simplifica o kernel, além de permitir um maior grau de flexibilidade ao usuário. A definição de construtos de linguagem que facilitem a concepção de "handlers" de exceções está sendo objeto de estudo pelo grupo do LCMI que trata com a linguagem de programação dos componentes.

O kernel está destinado para a utilização em uma rede para controle de processos com as camadas física, enlace e mais ainda o serviço de transporte desenvolvidos segundo padrões MAP /Crowder85/. Apesar das considerações do item 3.1.5, manteremos o protocolo classe 4 para o serviço de transporte. O kernel se encontra em fase final de implementação e esta sendo implementado em MS-Pascal.

5. Referências Bibliográficas

/Ananda84/ A.A.Ananda, B.W.Marsden.A network operating system for microcomputers. Computer Communication, vol. 7, No. 2, april 1984.

/Brender81/ R.F.Brender, I.R.Nassi. What is Ada?. Computer, pp. 17-24, june 1981.

/Bressan85/ G.Bressan. Ambiente de programação distribuída: Definição, análise e síntese. Tese de Doutor em Ciências. Escola Politécnica USP, 1985.

/Crowder85/ R.Crowder. The MAP Specification. Control Engineering, october 1985.

/Day83/ J.D.Day, H.Zimmermann. The OSI reference model. Proceedings of IEEE, vol. 71, No. 12, december 1983.

/DeRemer76/ F.DeRemer, H.H.Kron. Programming-in-the-Large versus Programming-in-the-Small. IEEE Trans. on Software Engineering, Vol. SE-2, No. 2, june 1976.

/Halsall83/ F.Halsall, R.L.Grimsdale, G.C.Shoja, J.E.Lambert. Development environment for the design and test of applications software for a distributed multiprocessor computer system. IEE Proceedings, vol. 130, Pt. E, No. 1, january 1983.

/Kramer83/ J.Kramer, J.Magee, M.Sloman, A.Lister.CONIC: an integrated

approach to distributed computer control systems. IEE Proceedings, vol. 130, Pt. E, No. 1, january 1983.

/Kramer85/ J.Kramer, J.Magee. Dynamic configuration for distributed systems. IEEE Trans. on Soft. Eng., april 1985.

/Lampson80/ B.W.Lampson, D.D.Redell. Experience with processes and monitors in Mesa. Com. of the ACM, vol.23, No.2, feb 1980.

/Lauer79/ H.C.Lauer, R.M.Needham. On duality of operating system structures. Operating System Review, No. 13, april 1979.

/Leite85/ J.C.B.Leite, O.G.Loques, M.A.C.Pacheco, M.H. Szarcman. Sistema distribuído para suporte de aplicações em tempo real, I Simp. em Sist. de Comput. Tolerantes a Falhas, INPE, SP, Setembro 1985.

/Liskov79/ B.Liskov. Primitives for distributed computing. Proceedings of VII ACM SIGOPS Symp. on Operating Systems Principles. pp. 33-42, december 1979.

/Liskov83/ B.Liskov, M.Herlihy. Issues in process and communications structure for distributed programs. III Symp. IEEE on realiability in distributed software and database systems, 1983.

/Lopes85/ A.D.Lopes, J.M.A.Coello. Um ambiente para o projeto e implementação de software para sistemas distribuídos em tempo real. II Cong. Nacional de Automação Industrial - CONAI. São Paulo, SP, Novembro 1985.

/Peterson84/ J.L.Peterson. Operating systems concepts. Addison-Wesley Publishing Co., USA, 1984.

/Rashid81/ R.F.Rashid, G.G.Robertson. Accent: a communication oriented network operating system Kernel. VIII Symp. on Oper. Sist. Principles, Californie, USA, december, 1981.

/Scott83/ M.L.Scott. Message vs. remote procedures. Sigplan Notices, vol. 5, may 1983.

/Sloman86/ M.Sloman, J.Kramer, J.Magee, K.Twidle. Flexible com munication structure for distributed embedded systems. IEE Proceedings, vol. 133, No. 4, july 1986.

/Watson81/ R.W.Watson. Distributed system architecture model. Lecture Notes in Computer Science 105, Springer - Verlag, 1981.

approach to distributed computer control systems. IEE Proceedings, vol. 130, Pt. E, No. 1, january 1983.

/Kramer85/ J.Kramer, J.Magee. Dynamic configuration for distributed systems. IEEE Trans. on Soft. Eng., april 1985.

/Lampson80/ B.W.Lampson, D.D.Redell. Experience with processes and monitors in Mesa. Com. of the ACM, vol.23, No.2, feb 1980.

/Lauer79/ H.C.Lauer, R.M.Needham. On duality of operating system structures. Operating System Review, No. 13, april 1979.

/Leite85/ J.C.B.Leite, O.C.Loques, M.A.C.Pacheco, M.H. Szarcman. Sistema distribuído para suporte de aplicações em tempo real, I Simp. em Sist. de Comput. Tolerantes a Falhas, INPE, SP, Setembro 1985.

/Liskov79/ B.Liskov. Primitives for distributed computing. Proceedings of VII ACM SIGOPS Symp. on Operating Systems Principles. pp. 33-42, december 1979.

/Liskov83/ B.Liskov, M.Herlihy. Issues in process and communications structure for distributed programs. III Symp. IEEE on realiability in distributed software and database systems, 1983.

/Lopes85/ A.D.Lopes, J.M.A.Coello. Um ambiente para o projeto e implementação de software para sistemas distribuídos em tempo real. II Cong. Nacional de Automação Industrial - CONAI. São Paulo, SP, Novembro 1985.

/Peterson84/ J.L.Peterson. Operating systems concepts. Addison-Wesley Publishing Co., USA, 1984.

/Rashid81/ R.F.Rashid, G.G.Robertson. Accent: a communication oriented network operating system Kernel. VIII Symp. on Oper. Sist. Principles, Californie, USA, december, 1981.

/Scott83/ M.L.Scott. Message vs. remote procedures. Sigplan Notices, vol. 5, may 1983.

/Sloman86/ M.Sloman, J.Kramer, J.Magee, K.Twidle. Flexible com munication structure for distributed embedded systems. IEE Proceedings, vol. 133, No. 4, july 1986.

/Watson81/ R.W.Watson. Distributed system architecture model. Lecture Notes in Computer Science 105, Springer - Verlag, 1981.