Universidade Estadual de Campinas
FACULDADE DE ENGENHARIA ELÉTRICA
DEPARTAMENTO DE TELEMÁTICA

UNICAMP

Usando Objetos Ativos para a Unificação da Rede de Gerência de Telecomunicações com a Rede Inteligente

Kleber Xavier Sampaio de Souza

27 de Fevereiro de 1996

Banca examinadora:

Dr. Ivanil Sebastião Bonatti
Dr. Marcos Bafutto
Dr. Eleri Cardozo
Dr. Moacir Pedroso Júnior
Dr. Shusaburo Motoyama

Suplentes:

Dr. Walter da Cunha Borelli
Dr. Edson Moschim

Dissertação apresentada à Faculdade de Engenharia Elétrica da UNICAMP, como requisito parcial para obtenção do título de Doutor em Engenharia Elétrica

Este exemplar corresponde à redação final da tese defendida por Kleber Xavier Sampaio de Souza e aprovada pela Comissão Julgadora em 27 02 96

Orientador

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA - BAE - UNICAMP

So89u

Souza, Kleber Xavier Sampaio de
Usando objetos ativos para a unificação da gerência de telecomunicações com a rede inteligente / Kleber Xavier Sampaio de Souza. – Campinas, SP:[s.n.], 1996.

Orientador: Ivanil Sebastião Bonatti.
Tese (doutorado) - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica.

1. Rede digital de serviços integrados. 2. Processamento paralelo (Computadores). 3. *Plataforma distribuída. I. Bonatti, Ivanil Sebastião. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica. III. Título.

Dedico este trabalho à minha mãe,
por seu amor e dedicação,
e ao meu pai, cuja alegria de ser
tanto nos faz falta.

Agradecimentos

Agradeço ao meu orientador, o professor Ivanil Sebastião Bonatti, pela confiança, total liberdade de escolha do tema desta tese e orientação exemplar. Trabalhar ao seu lado foi sempre uma tarefa agradável, e plena de aprendizado.

Agradeço aos meus amigos e colegas da Embrapa, pelo companheirismo e confiança demostrada na aprovação do plano de trabalho que deu origem a esta tese.

Agradeço aos professores Jean-Pierre Claudé e Nazim Agoulmine, do Laboratório PRiSM da Université de Versailles Saint-Quentin en Yvelines, pela acolhida, orientação e inserção desta pesquisa no Projeto RACE2.

Agradeço à Embrapa e ao CNPq pelo financiamento desta pesquisa, e aos laboratórios LabDT e PRiSM pela cessão das instalações computacionais;

Agradeço os professores Marcos Baffuto, Walter da Cunha Borelli, Eleri Cardozo, Moacir Pedroso Júnior, Edson Moschim e Shusaburo Motoyama, por haverem aceito fazer parte do júri examinador desta tese.

Resumo

Neste trabalho, a Plataforma Distribuída de Agregados de Atores é proposta para a implementação de serviços de gerência na Rede de Gerência de Telecomunicações (TMN), em um ambiente de rede de alta velocidade, como o Modo de Transferência Assíncrono [1] (ATM). As principais contribuições desta pesquisa são: a análise dos requisitos necessários à gerência de redes de alta velocidade, e a proposição de um modelo baseado em filas com prioridades: Extensão deste modelo para aplicação à Rede TMN, anexando mecanismos de transparência necessários à integração dos vários componentes funcionais da arquitetura TMN; Demonstração da adequação do modelo estendido à implementação de serviços em redes inteligentes (IN), confirmando a tendência de integração entre TMN e IN; Proposição da aplicação da Teoria de Atores na implementação do modelo proposto, mediante extensão da teoria básica incluindo: filas de eventos prioritários, conceito de Agregados (para *Transparência de Localização*) e mecanismo de *Transparência de Acesso*, usando a linguagem ASN.1; e a construção de um núcleo de suporte à implementação de Agregados de Atores, utilizando facilidades de *"threads" de controle*, ambiente ISODE (ISO Development Environment) e a linguagem de programação C++.

[1] "Asynchronous Transfer Mode"

Abstract

In this work, the Distributed Actor Aggregates Platform is proposed to implement services in the Telecommunications Management Network (TMN) in high speed network environment, such as Asynchronous Transfer Mode (ATM). The main contributions of this thesis are: the requirements analysis of management in high speed networks, and the proposition of a model based on priority queues; Extension of the model for application to TMN Networks, adding transparency mechanisms necessary to integrate TMN functional components; Demonstration of the adequacy of the extended model to implement Intelligent Network (IN) services, confirming the tendency of the unification of TMN and IN; Proposition of the application of Actor Theory in the implementation of the proposed model, by extending the basic theory to include: priority event queues, the Aggregates concept (for *Location Transparency*) and a mechanism of *Access Transparency*, using ASN.1 language; and the construction of a supporting kernel for the implementation of Actor Aggregates, using control threads facilities, the ISODE package and the C++ programming language.

Conteúdo

Lista de Figuras

Lista de Tabelas

Notação

Conjuntos

$\emptyset$	Conjunto vazio.
ω	Conjunto dos números naturais.
$P\omega$	Conjunto de potência dos números naturais.
$\mathbf{N}$	Conjunto dos números naturais.
$\mathbf{R}$	Conjunto dos números reais.
$\mathbf{Z}$	Conjunto dos números inteiros.
$A \bigcup B$	União de A e B.
$A \bigcap B$	Intersecção de A e B.
$x \in A$	x é um elemento de A.
$A - \{x\}$	Remoção do elemento x do conjunto A.
$A \subseteq B$	A é um subconjunto impróprio do conjunto B.
$A \supseteq B$	B é um superconjunto impróprio do conjunto B.
A^c	Fechamento do subconjunto A no espaço D.
$\bigcup X$	União de todos os subconjuntos da classe X.
$\bigcap X$	Intersecção de todos os subconjuntos da classe X.
$\#(A)$	Cardinalidade do conjunto A.
$A \uplus B$	União disjunta dos conjuntos A e B.

Álgebra

$x \leq y$	x é inferior ou igual a y de acordo com o ordenamento $\leq$.
$\langle A, \leq \rangle$	A é um conjunto parcialmente ordenado de acordo com o ordenamento $\leq$.
$\bigvee A$	Menor limitante superior de A em D.
$\bigwedge A$	Maior limitante inferior de A em D.
$\sqcup A$	Menor limitante superior de A em $\overline{D}$.
$\sqcap A$	Maior limitante inferior de A em $\overline{D}$.

Domínios

$\perp$	Valor mínimo de um domínio, ou valor indefinido.
$\langle \overline{D}, \sqsubseteq \rangle$	Completação do domínio $\langle D, \leq \rangle$.
$\mathcal{D}$	Domínio do diagrama de eventos em atores.
$\langle P[D], \sqsubseteq \rangle$	Domínio de potência do domínio D.

Funções

$f : A \rightarrow B$	Função de A em B.
$\{f^i(\perp) \mid i \in \omega\}$	Trajetória de sucessivas aproximações para o ponto-fixo, onde cada f^i é uma função parcial recursiva.

Semântica

$\xrightarrow{atv}$	Ordenamento de ativação.
$\xrightarrow{rec}$	Ordenamento de chegada.
$\longrightarrow$	Ordenamento combinado.
$\mathcal{B}$	Conjunto de comportamentos de um ator.
$\mathcal{A}(e)$	Ativador do evento e.
$rec(a)$	Função que retorna o próximo $i \in \mathbf{N} \mid \langle a, i \rangle \notin E$.
$\langle a, m, e \rangle$	Evento pendente, destinado ao ator a, contendo mensagem m e ativado pelo evento e.
$\lfloor \beta_i, e \rfloor$	Novo ator criado com comportamento β_i durante o processamento do evento e.
$\mathcal{S}(P)$	Significado semântico de P.
$\rho(\epsilon)$	Função que avalia o significado da expressão primitiva ϵ na linguagem hospedeira C++.
σ	Função que avalia o ambiente local.
$\sigma[\![a/x]\!]$	Representa o ambiente local que difere de σ apenas pelo valor a que é atribuído a x.

Capítulo 1

Introdução

O advento de novas tecnologias de transmissão e comutação para redes de alta velocidade, como o Modo de Transferência Assíncrono (ATM)[1] por exemplo, e a implantação de novos serviços de telecomunicações nas áreas de multimídia, entretenimento, vídeo-conferência, trabalho cooperativo e rede virtual privada[2], entre outros, tem causado um grande impacto em áreas como: provimento de serviços[3], gerência de rede e operação de serviços.

Para responder à necessidade de implantação de serviços com maior rapidez, menor custo e de forma diferenciada, de acordo com as necessidades de mercado, a ITU [4], elaborou as normas Q.12xx que estabelecem princípios para a **Arquitetura de Rede Inteligente (IN)**[5]. O nome Rede Inteligente lhe foi dado pelo fato de que os serviços nesta arquitetura são descritos como uma composição de fragmentos de seqüências lógicas de operações, chamadas **blocos construtores independentes de serviços**[6]. O exemplo mais simples de uma tal seqüência é o procedimento de encaminhamento de chamadas existente em qualquer central telefônica convencional, pois consiste num conjunto de passos que são executados a cada chamada.

A principal mudança da IN em relação ao processamento de chamadas convencional foi a separação entre o processamento básico de chamadas e os chamados *serviços inteligentes*, ou seja, os serviços que requerem um conjunto de procedimentos especializados para o seu tratamento. Esta separação permite que os serviços sejam implementados mais rapidamente porque não requerem uma mudança no software das centrais a cada novo serviço introduzido: são apenas criados *programas de lógica de serviço*, que são chamados quando uma requisição é detectada.

No que se refere à *gerência da rede de telecomunicações*, para lidar com a crescente complexidade dos equipamentos e também com a sua diversidade criou-se a **Rede de Gerência de Telecomunicações (TMN)**[7]. Como se trata de uma rede de gerência, a TMN deve se preocupar em fornecer uma infra-estrutura adequada a tarefas como [48]: planejamento, construção, instalação, manutenção, exploração e administração das redes e serviços de telecomunicações.

[1] "Asynchronous Transfer Mode."

[2] "Virtual Private Network."

[3] Serviço: qualquer suporte prestado pela rede de telecomunicações a seus usuários, como telefonia, transmissão de fax, etc.

[4] "International Telecommunications Union."

[5] "Intelligent Network."

[6] "Service independent building blocks."

[7] "Telecommunications Management Network"

O setor de padronização de telecomunicações da ITU, o ITU-T, separa em planos distintos o provimento e/ou operação de serviços e a gerência da rede [41, 42] como um todo. Também foram definidos padrões distintos para cada finalidade: a Arquitetura IN responde pela tarefa de implementação e operação, enquanto que a TMN, pela gerência de rede. Entretanto, a própria evolução de longo prazo prevista para as redes inteligentes prevê a integração com outros padrões como ODP (Open Distributed Processing) e TMN para a concepção de uma arquitetura aberta resultante da junção de computação, informação e tecnologia telefônica [40].

Como resultado desta integração, as redes inteligentes seriam aperfeiçoadas com a incorporação de avanços em áreas como:

processamento distribuído : processamento de informações cujos componentes discretos podem estar em locais diferentes, ou onde a comunicação entre componentes podem sofrer atraso ou falha [50]. Na nomenclatura OSI, o processamento distribuído é dito aberto quando está de acordo com o padrão ODP [49, 50, 51, 52]. O *processamento aberto distribuído* fornece a interoperabilidade necessária à operação em ambientes heterogêneos;

modelagem orientada a objetos : utilização de objetos para modelar entidades, que são *coisas* abstratas ou concretas pertencentes ao universo de discurso. A utilização desta técnica de modelagem significaria a migração de um modelo funcional hierárquico (no qual está baseada a arquitetura IN) para um paradigma que permite especificações modulares, encapsulamento e abstração de dados [26];

tecnologia da informação : fornecimento dos meios necessários a uma empresa para que esta possa integrar, compartilhar, processar e distribuir grandes quantidades de informação de modo a satisfazer clientes internos e externos;

processamento cooperativo : especialização do processamento entre dois sistemas, no qual dados e funcionalidade de rede funcionam coesivamente [40]. Esta coesão é conseguida através de um controle operacional projetado para coordenar, administrar e utilizar a *inteligência* distribuída na rede;

gerência de serviços e redes : tratamento uniforme de objetos tão diferentes quanto serviços, blocos construtores de serviços, serviços de transmissão, circuitos virtuais, conexões e chamadas. Devido à vasta gama de possibilidades de novos serviços, a arquitetura IN deve fornecer os meios de programar (em paralelo com a seqüência lógica de processamento) a seqüência lógica de gerência.

A junção TMN-IN traz, em contrapartida, alguns desafios no que diz respeito à implementação de aplicativos de operação de controle e de gerência. Os aplicativos destinados ao controle devem ter alta disponibilidade, isto é, devem fornecer o maior grau de paralelismo possível para permitir uma utilização plena dos recursos do sistema. Além disto, alguns destes aplicativos têm um tempo máximo de execução preestabelecido, possuindo portanto, restrições consideradas *temporais fortes*[8], como por exemplo, os algoritmos de controle de chamada que devem encontrar uma solução dentro do intervalo entre sucessivas solicitações.

[8] "Hard real-time."

Aplicativos destinados à gerência não possuem tais restrições, pois esta atividade é considerada como de longo prazo. A gerência é realizada através da observação do funcionamento do sistema, estabelecendo metas de longo-prazo a serem implantadas neste.

Para programar em paralelo a **seqüência lógica de processamento** e a **seqüência lógica de gerência**, conforme sugerido no item **gerência de serviços e redes**, deve-se lançar mão de um mesmo paradigma computacional para descrever as duas atividades. Entretanto, a planta existente de "software" destinado à gerência é inadequada para este propósito: do ponto de vista de gestão, agentes responsáveis pela monitorização do estado de um nó não são capazes de reagir prontamente a alguns eventos (como uma solicitação de serviço, por exemplo)[9], deixando outros, menos importantes, para uma etapa posterior, simplesmente porque tratam todas as requisições de acordo com a disciplina FIFO ("First-in, First-out").

Observando-se a forma com a qual alguns destes aplicativos foram construídos, nota-se que a raiz do problema está na escolha de um paradigma seqüencial ao invés de concorrente[10]. Estes aplicativos foram implementados em sua maioria em linguagem C ou C++, sobre sistema operacional UNIX[11] ou similar. A solução proposta neste trabalho é radicalmente diferente: modela-se o sistema em termos de *Atores*, que é um paradigma onde a concorrência é a regra e que, como conseqüência, para conseguir algo que funcione seqüencialmente é necessário um esforço considerável de modelamento e programação.

Os **Atores**[12] [19, 3, 56, 37] são objetos ativos que se comunicam através de troca de mensagens. A *Teoria de Atores* surgiu no final da década de 70, como proposta do Grupo de Semântica de Passagem de Mensagens[13] do MIT[14], resultado da evolução de idéias provenientes de experimentos realizados com a linguagem *Simula* [29] para a simulação seqüencial de sistemas concorrentes, e posteriormente realizados com a linguagem *Smalltalk* [26]. Esta teoria estabelece os Atores como elementos primitivos de qualquer sistema concorrente.

Neste trabalho, elabora-se um **Núcleo de Suporte à Implementação de Agregados de Atores**, acessível através de um conjunto de primitivas incorporadas à linguagem *C++*, tendo como base a Teoria de Atores. A linguagem C++ é orientada a objetos e compatível com a linguagem *C* em quase sua totalidade. Como a maior parte do "software" disponível de gerência de redes foi escrito em *C*, a utilização de *C++* permite o reuso do "software" já produzido tanto em *C*, quanto em *C++*, introduzindo concorrência apenas onde ela é necessária.

A maior parte da literatura que se utiliza do Modelo de Atores o faz para construir aplicações na área de *Inteligência Artificial Distribuída* [37, 56, 28]. Isto não acontece por acaso. O modelo possui características que o tornam especialmente atrativo para esta área, como por exemplo o conceito de **continuação**[15] [3]. A *continuação* é a explicitação de quem receberá a resposta a uma dada requisição feita na própria mensagem de requisição, ou seja, a resposta não irá necessariamente para quem solicitou o serviço, mas para quem o solicitante pediu que fosse mandada.

---

[9] Nestes eventos, não estão sendo consideradas as ações em tempo real realizadas com o intuito de controle de fluxo.

[10] O termo concorrência é utilizado no sentido de paralelismo em potencial, como em [10].

[11] UNIX é uma marca registrada de UNIX Systems Laboratories Inc.

[12] "Actors."

[13] "Message-Passing Semantics Group."

[14] Massachusets Institute of Technology

[15] "Continuation."

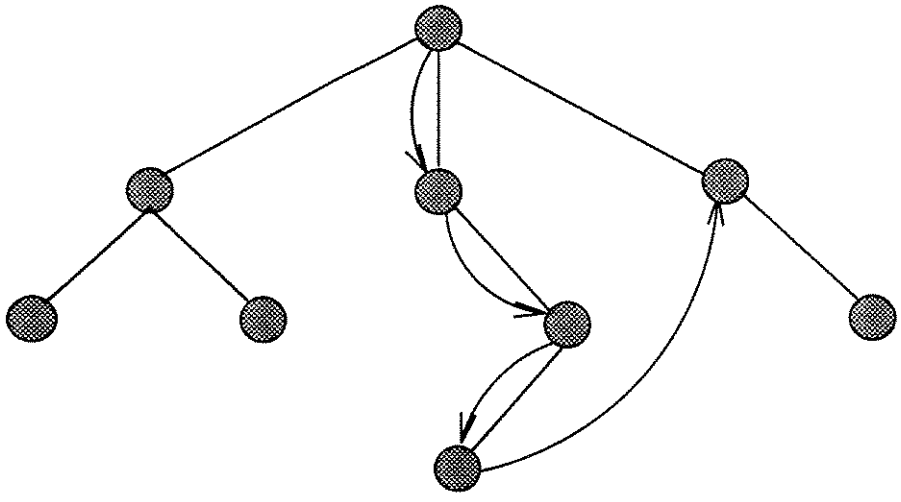


Figura 1.1 Continuações em uma árvore de busca

É importante notar que o conceito de *continuação* torna totalmente desnecessária a utilização de pilhas de execução existente em linguagens procedurais como *C*, *C++*, *Pascal*, *Fortran*, etc. Nestas linguagens, a cada nova chamada de função, o *ponto de retorno* é colocado em uma pilha chamada *pilha de execução.* Este ponto indica para onde a execução deve prosseguir quando uma função terminar. Utilizando-se *continuações*, a pilha deixa de existir, pois um Ator não *retorna* no sentido clássico, ele simplesmente envia a sua resposta para o *parâmetro de continuação* e deixa de existir.

O conceito de *continuação* tem uma implicação prática importante quando se tenta manter o *foco de atenção do computador* em um determinado objetivo, como quando se programa algoritmos de busca. Nestes programas, a busca em profundidade[16] que é a mais fácil de programar nem sempre é a melhor ordem de busca [8]. Neste caso, a *continuação* permite que se retorne para um ponto qualquer diferente do nó predecessor (ver Figura 1.1).

Entretanto, devido à heterogeneidade que caracteriza a maioria dos sistemas de gerência ou controle de redes, apenas concorrência não é suficiente: é preciso dotar o sistema de **Mecanismos de Transparência de Distribuição.** A função destes mecanismos é de retirar da aplicação algumas operações que lhe são necessárias, mas que podem ser realizadas automaticamente pela infraestrutura. Exemplificando, se uma entidade necessita trocar mensagens com uma outra, ela precisa antes de tudo obter o endereço de destino. Este endereço pode estar diretamente gravado na aplicação ou em algum arquivo de configuração, mas é preciso modificar o código da aplicação ou este arquivo de configuração a cada novo endereço de destino da aplicação. Ao prover um mecanisno de *Transparência de Localização*, a infraestrura localiza o destino automaticamente, através de um padrão adotado para o sistema distribuído.

[16] "Depth-first search."

1.1 Organização da Tese

A redação desta tese está organizada da seguinte forma: No Capítulo 2 são apresentados os Modelos OSI e Internet (TCP/IP) de gerência de sistemas, o padrão adotado para a Rede de Gerência de Teleconunicações (TMN) e o padrão de arquitetura de Redes Inteligentes (IN). Estes modelos são utilizados no levantamento dos requisitos necessários à construção de uma plataforma única de gerência e controle. No Capítulo 3 são apresentados alguns paradigmas de programação concorrente, mecanismos de transparência para sistemas heterogêneos e o projeto da *Plataforma de Suporte à Implementação de Agregados de Atores.* No Capítulo 4 é formalizada a semântica denotacional utilizada no modelo de atores, particularizando-a de acordo com o comportamento da classe C++ *ator* definida na plataforma[17]. No Capítulo 5 é descrita a implementação da plataforma e a sua aplicação no desenvolvimento de um *Servidor Concorrente de MIB*[18] e, finalmente, no Capítulo 6 são apresentadas as conclusões, onde destacam-se as principais contribuições que são, resumidamente:

- Análise dos requisitos necessários à gerência de redes de alta velocidade, e a proposição de um modelo baseado em filas com prioridades;
- Extensão deste modelo para aplicação à Rede TMN, anexando mecanismos de transparência necessários à integração dos vários componentes funcionais da arquitetura TMN;
- Demonstração da adequação do modelo estendido à implementação de serviços em redes inteligentes, confirmando a tendência de integração entre TMN e IN;
- Proposição da aplicação da Teoria de Atores na implementação do modelo proposto, mediante extensão da teoria básica incluindo: filas de eventos prioritários, conceito de Agregados (para *Transparência de Localização*) e mecanismo de *Transparência de Acesso*, usando a linguagem ASN.1;
- Construção de um núcleo de suporte à implementação de Agregados de Atores utilizando facilidades de *"threads" de controle*, ambiente ISODE (ISO Development Environment) e a linguagem de programação C++.

[17] Sugere-se uma primeira leitura desta tese omitindo o Capítulo 4.
[18] "Management Information Base".

Capítulo 2

O Problema

Os padrões adotados para a gerência de redes OSI e Internet são os mais utilizados na construção de aplicativos de gerência. No caso do padrão OSI, a sua importância é ainda mais ressaltada devido à sua adoção como base para o padrão para gerência de redes de telecomunicações TMN. O modelo TMN ampliou muitos dos conceitos de gerência OSI, adaptando-o a funcionar nos vários planos em que são divididas as redes TMN.

Neste capítulo são apresentados os modelos OSI-MF *(OSI Management Framework)*, INMF *(Internet Network Management Framework)*, TMN, e finalmente, a arquitetura de redes inteligentes IN. Considerando-se os modelos apresentados, os requisitos necessários construção de uma plataforma única de gerência e controle são discutidos.

2.1 Introdução

Embora cada rede possa consistir de uma tecnologia completamente diferente (Ethernet, ponto-a-ponto proprietária, X.25, etc), cada uma com suas próprias regras de transmissão, todos os *computadores* conectados àquelas podem lhes ter uma visão comum através da aplicação da *abstração de interconexão* [69]. Esta é conseguida através do uso de um protocolo comum e de algoritmos, fazendo com que até a mais complexa topologia, composta das mais variadas tecnologias possa ser encarada como uma simples conexão ponto-a-ponto sobre uma rede física homogênea.

Considerando-se a atividade de gerência, a abstração proporcionada pelo uso de um protocolo (Protocolo de Gerência) permite-se que se concentre apenas nos aspectos relevantes a esta atividade. Tanto no Modelo de Referência OSI (Open Systems Interconnection) para interconexão de sistemas abertos da ISO (International Standards Organization), quanto na Estrutura Internet de Gerência foram criados protocolos específicos, e são os seus modelos delineadores os assuntos das Seções 2.3 e 2.5.

A atividade de gerência é executada segundo uma *Política de Gerência*, que especifica regras básicas a serem usadas para guiar as suas decisões. Esta política pode ser utilizada para especificar [74] uma *estratégia* (i.e. um plano do que será feito para atender os objetivos) e uma *tática* (i.e. um plano de como os elementos desta estratégia serão executados). Dentre as funções que podem requerer estratégia e tática, em se tratando de redes de computadores, estão:

- configuração de aplicações;

- controle de dispositivos;
- monitoramento de estado, erros e desempenho;
- otimização de desempenho;
- detecção, localização e correção de falhas;
- estabelecimento, manutenção e controle de segurança;
- contabilização de uso.

Para realizar as *funções de gerência* listadas acima, a solução adotada tanto na Estrutura de Gerência OSI, quanto na Internet, foi a criação de protocolos de gerência e de modelos de representação da informação que satisfizessem as necessidades de todas estas funções.

Os modelos analisados neste capítulo possuem uma grande diversidade entre si, o que torna a sua compreensão bastante árdua, principalmente quando se deseja extrair seus pontos em comum. Resolveu-se, então, apresentá-los seguindo-se três aspectos principais:

Arquitetura Funcional : Descreve os componentes envolvidos e define a interação entre eles;

Modelo de Informação : Descreve a estruturação lógica da informação;

Modelo de Comunicação : Define a maneira pela qual a informação é transferida entre os componentes, explicitando os protocolos, a codificação dos dados e as interfaces.

2.2 Introdução à Interconexão de Sistemas Abertos (OSI)

Na presente seção será introduzida uma notação OSI básica, que será necessária a uma melhor compreenção do Modelo OSI de gerência.

2.2.1 Modelos, Convenções e Notação

O Modelo de Referência OSI é inerentemente abstrato. Portanto, não especifica:

- ligações com linguagens de programação;
- ligações com sistemas operacionais;
- interfaces relacionadas a aplicações;
- interfaces com usuário.

O modelo descreve o comportamento externo de um sistema, independente de sua construção interna. Do ponto de vista de comunicação, OSI descreve apenas o quê percorre o meio de transmissão, e não como deve ser a arquitetura dos computadores para exibir tal comportamento. Além disso, que o modelo OSI não é uma arquitetura de rede, porque ele não especifica os serviços exatos e protocolos que serão usados em cada uma das camadas que compõem o modelo. Ele apenas descreve o que cada camada deve fazer [79].

2.2.2 As Sete Camadas

O modelo OSI divide a atividade de comunicação de computadores em sete camadas. A estruturação em camadas permite que uma tarefa complexa, como a interconexão de *sistemas abertos*, seja vista como uma sucessão de camadas, cada uma abordando um aspecto particular da ligação em rede [25]. Na nomenclatura OSI, um **sistema aberto** representa a parte de um sistema computacional que implementa padrões OSI para realizar a troca de informações com outros sistemas. Isto significa que a parte do sistema que executa atividades específicas sem que haja troca de informações não é considerada como sistema aberto.

Descritas de forma sucinta, estas camadas são [69]:

Camada Física: responsável pela interface eletromecânica com o meio de comunicação, é nesta camada que ocorre a transmissão de "bits" entre os sistemas;

Camada de Enlace: responsável pela transmissão, enquadramento e controle de erros sobre um único enlace de comunicações;

Camada de Rede: diferentemente das duas camadas anteriores, que operam ponto-a-ponto, é na camada de rede que ocorre o concatenamento de sub-redes. As **sub-redes** são redes físicas completas até a camada de rede que são interligadas por **sistemas intermediários**. A transferência de dados na rede é realizada por esta camada, independente do meio que compõe as sub-redes, ou da topologia destas. Nesta camada é definido um esquema de endereçamento global para os sistemas abertos;

Camada de Transporte: responsável pela confiabilidade e multiplexação da transferência de dados através da rede ao nível requerido pela aplicação. Esta camada é implementada somente nas pontas da conexão, ou seja, os sistemas intermediários não precisam implementá-la;

Camada de Sessão: responsável pela adição de mecanismos de controle (sinais) à transferência de dados, e de coordenação das atividades entre aplicações. Estes sinais são interpretados pelas camadas superiores para controlar e estruturar os dados trocados por elas, como por exemplo, para delimitar o início e o término de uma atividade de processamento, dar pausa em uma transmissão, ou indagar sobre o progresso da atividade ;

Camada de Apresentação: é comum que aplicações troquem, em sua comunicação, estruturas de dados complexas, que muitas vezes são dependentes em sua representação interna da arquitetura de cada máquina, compiladores utilizados ou linguagens de programação. Para eliminar esta dependência, uma solução encontrada foi a adoção de uma notação para expressar sintaticamente estas estruturas, denominada **Notação Abstrata de Sintaxe Um (ASN.1)**[1] [39].

ASN.1 descreve as estruturas de dados, mas não a sua representação para efeito de transferência, ou seja, como transformá-la em seqüência de bits. Para este fim, foi desenvolvido mais um padrão: as **Regras de Codificação Básicas (BER)**[2].

[1] "Abstract Syntax Notation One."
[2] "Basic Encoding Rules."

Camada de Aplicação: responsável pela gerência das comunicações entre aplicações, esta camada apóia-se sobre as outras para fornecer serviços aos usuários fins, como por exemplo Terminais Virtuais, Processamento de Transações, Serviço de Diretório, etc.

Os padrões OSI são agrupados em pares <serviço, protocolo> [79]. Os serviços definem *o que* a camada fornece, e os protocolos detalham *como* a camada fornece o serviço.

2.2.3 Serviço

Um **serviço** é um conjunto de primitivas (operações) que uma camada fornece à camada superior, não determinando nada acerca da implementação destas operações. Serviços estão relacionados com a interface entre duas camadas, com a inferior sendo provedora deste, e a superior, a usuária. Além disto, os serviços oferecidos podem ser *orientados a conexão*, ou *sem conexão*:

Serviços Orientados a Conexão: modelados a partir do sistema telefônico. Para utilizá-los, estabelece-se uma conexão, usa-se a conexão e termina-se a conexão. Os dados enviados através desta chegam ao destino na mesma ordem de envio;

Serviços Sem Conexão: modelado com base no sistema postal, onde cada mensagem carrega o endereço completo do destinatário. Cada mensagem pode ser enviada por um caminho distinto, não se garantindo a chegada na mesma ordem de envio.

Os elementos ativos em cada camada (N) são chamados **entidades**. Estas podem ser de "software" (como um processo), ou de "hardware" (como um microprocessador inteligente), e seu objetivo é fornecer serviços à camada imediatamente superior $(N+1)$. Para ter acesso a um destes serviços, uma entidade da camada $(N+1)$ envia uma **unidade de dados de interface (IDU)**[3] a um endereço denominado **ponto de acesso a serviço (SAP)**[4], que se localiza na camada (N), o (N)-SAP, conforme a Figura 2.1.

2.2.4 Protocolo

Um Protocolo [79] é um conjunto de regras que governam o formato e o significado de quadros, pacotes, ou mensagens trocadas por entidades pares (uma em cada ponta do enlace) dentro de uma camada. Estas entidades, as quais mudam de estado de acordo com a interação ocorrida nos pontos de acesso, utilizam protocolos na implementação de sua definição de serviço, tornando-se, como conseqüência, bastante complicadas. Isto fez com que se tenha desenvolvido muita pesquisa na busca de técnicas formais matemáticas para a especificação e validação de protocolos, como Máquinas de Estados Finitos, Máquinas de Estados Estendidos e Redes de Petri, entre outras.

2.2.5 Abstração de Dados

A separação entre serviço e protocolo traz para o Modelo OSI o importante conceito de *Abstração de Dados* [58], que permite a construção de entidades individuais com pouco conhecimento de outras, evitando, assim, efeitos colaterais causados por interações indesejáveis. Abstração de Dados significa que o comportamento de uma entidade é descrito em alto nível, sem recorrer à sua implementação real.

[3] "Interface Data Unit".
[4] "Service Access Point".

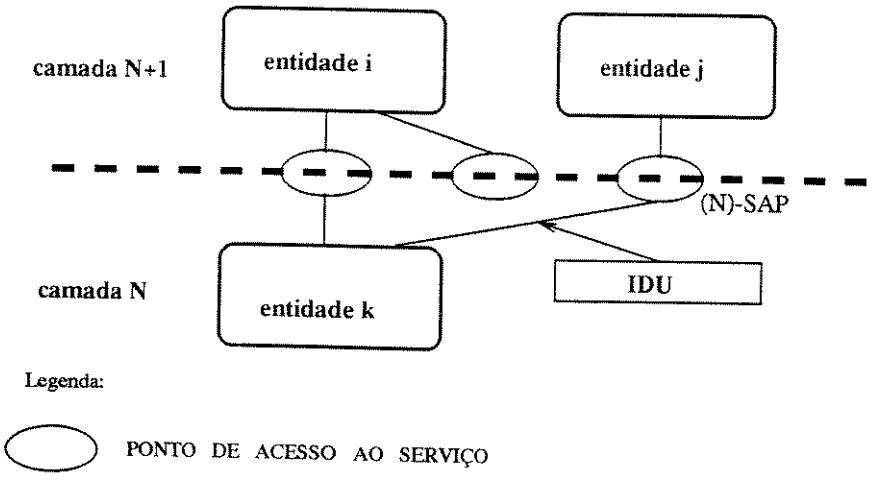


Figura 2.1 Pontos de acesso a serviço

2.2.6 Estruturação da Camada de Aplicação

A maior parte dos componentes de uma aplicação de gerência estão localizados na camada de aplicação. Por este motivo, uma apresentação mais detalhada da estrutura desta camada é necessária para uma melhor compreensão do Modelo OSI de Gerência.

Na camada de aplicação, as **Entidades de Aplicação** são a parte de um **processo de aplicação** sujeita a padronização, ou seja, são a parte do *processo de aplicação* que vai realizar a troca de informações com outros sistemas abertos. As *entidades de aplicação* por sua vez, são compostas de um ou mais **Objetos de Serviço de Aplicação (ASO)**[5] (ver Figura 2.2). Estes objetos são agrupamentos de um ou mais *Elementos de Serviço de Aplicação (ASEs)*[6], juntamente com uma *função de controle*, podendo ainda conter outros ASOs.

O objetivo da **função de controle** é o controle global do diálogo e a gerência de associações entre aplicações. A função de controle fornece controle de seqüenciamento, concatenação de *Unidades de Dados de Protocolo de Aplicação (APDUs)*, mapeamento entre serviços, etc.

Os **Elementos de Serviço de Aplicação** são os componentes básicos de cada entidade de aplicação. Cada ASE é definida por um um serviço e um protocolo OSI específico. Exemplos de ASEs são citados a seguir:

ACSE: *(Elemento de Serviço de Controle de Associações*[7]*)* é responsável pelo estabelecimento de *associações* entre aplicações. Uma **associação** é um conceito usado para representar uma conexão ao nível da aplicação. O conceito de associação foi necessário porque uma conexão é um serviço fornecido por uma camada a outras que estão acima dela, o que não se verifica no caso da camada de aplicação, que é a última. No estabelecimento de uma associação, uma aplicação precisa informar a outra acerca da sua composição em termos de ASEs, para que elas possam trocar as mesmas APDUs. Esta composição é chamada de **contexto de aplicação**;

[5] "Application Service Objects."
[6] "Application Service Elements".
[7] " Application Control Service Element."

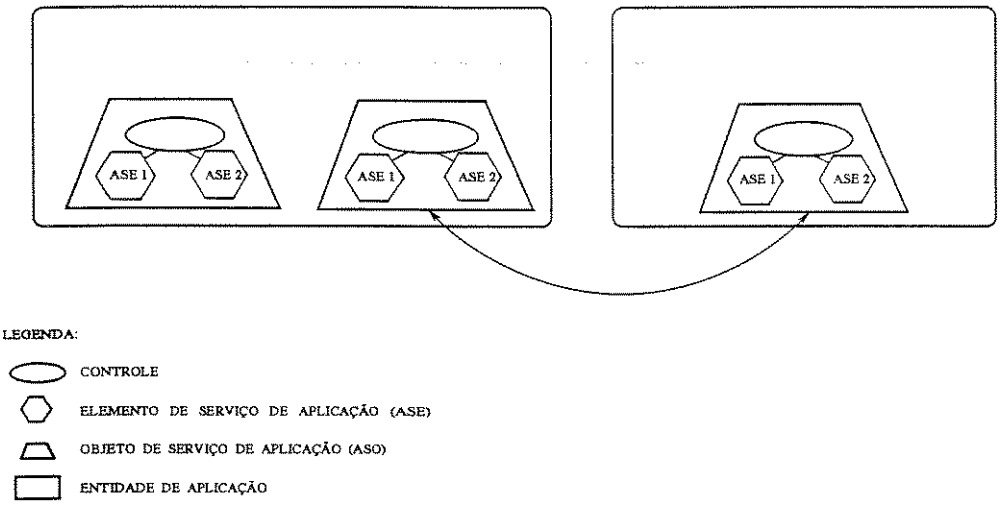


Figura 2.2 Estrutura da camada de aplicação

RTSE: *(Elemento de Serviço de Transferência Confiável*[8]*)* é um serviço que garante a entrega de dados entre aplicações de forma confiável;

ROSE: *(Elemento de Serviço de Operações Remotas*[9]*)* fornece um serviço equivalente ao RPC existente na Internet, ou seja, permite a chamada de procedimentos remotos;

FTAM: *(Gerência, Acesso e Transferência de Arquivos*[10]*)*;

CMISE: *(Elemento de Serviço Comum de Informação de Gerência*[11]*)* estabelece a base fundamental sobre a qual a atividade de gerência entre sistemas é realizada.

Endereçamento na Camada de Aplicação

Uma camada oferece serviços à camada superior através de seus elementos ativos, que são as *entidades*, através dos seus SAPs. Estes são, portanto, os elementos aos quais se deve direcionar um pedido de serviço, ou seja, a unidade de endereçamento nas camadas.

Dado que os SAPs identificam um ponto de acesso ao serviço de uma entidade, cada SAP deve ter correspondência biunívoca com as entidades da camada superior, mas dois SAPs podem ser acessados por uma mesma entidade da camada inferior, caso ilustrado na Figura 2.1.

Um SAP particularmente importante é o NSAP[12], o SAP da camada de rede, pois ele é o identificador de um nó na rede. A partir do NSAP, adicionando os SAPs das entidades das camadas superiores, tem-se o endereço completo de uma *entidade de aplicação*, o **endereço de apresentação.**

[8] " Reliable Transfer Service Element."
[9] "Remote Operations Service Element."
[10] " File Transfer Access and Management".
[11] "Common Management Information Service Element"
[12] "Network Service Access Point". Não confundir com o (N)-SAP, que é um SAP da camada (N).

Os SAPs das camadas superiores à de rede recebem um nome especial: **seletor.** Uma concatenação dos seletores de transporte, sessão e apresentação forma o *endereço de apresentação* de uma entidade na camada de aplicação[13].

2.3 Modelo OSI de Gerência

Dentro do Modelo de Referência OSI, três categorias de gerência são identificadas [22]:

Gerência de Aplicações, que está relacionada com processos de aplicação OSI;

Gerência de Sistemas, que está ligada à gerência de recursos OSI e seu "status" em todas as camadas do modelo;

Gerência de Camadas, na qual são identificados dois aspectos:

- Ativação e controle de erros na camada, que é implementado pelo protocolo OSI ao qual ele se aplica;
- Sub-conjunto da gerência de sistemas relevante à camada (N), cujos protocolos residem dentro da camada de aplicação, sendo utilizados pelas *entidades de aplicação de gerência de sistemas -(SMAE).*

Apresenta-se, a seguir, a **Estrutura de Gerência de Sistemas OSI (OSI-MF)**[14] (recomendações série X.700 [44]) de acordo com três aspectos: a Arquitetura Funcional, o Modelo de Informação e o de Comunicação.

2.3.1 Arquitetura Funcional

A Estrutura OSI de Gerência de Sistemas baseia-se na interação entre gerentes e agentes através do protocolo CMIP, conforme ilustra a Figura 2.3, onde um sistema pode ser gerente de um domínio e agente de um outro. Este modelo utiliza um esquema de organização de informações com estrutura padrão **X.500**, que é o usado no Sistema de Diretório[15].

Um **Agente** é uma aplicação de gerência cuja finalidade é supervisionar a rede, enviar informações sobre eventos ocorridos nos objetos gerenciados sob sua responsabilidade e responder a comandos referentes a pedidos de informação ou ações sobre estes objetos. A informação que o agente tem dos objetos é armazenada em uma *Base de Informações de Gerência (MIB).*

Um **Gerente**, por sua vez, possui a capacidade de processar e interpretar [25] os dados obtidos nos agentes em nome do administrador de rede. O gerente pode pedir a modificação de valores dos dados armazenados nos agentes, bem como a remoção remota de recursos.

A interação entre gerente e agente é feita através de sondagem[16] realizada pelo gerente, ou através da notificação[17] resultante de uma iniciativa do próprio agente, relatando uma mudança no estado de algum dos objetos da MIB.

[13] Note-se que o último seletor é o da camada de apresentação, pois esta é a última fronteira entre camadas na pilha OSI.

[14] "OSI Management Framework"

[15] aplicação projetada para servir de base de armazenamento para outras aplicações OSI, como **X.400 MHS** (Sistema para Tratamento de Mensagens), **X.700** (Sistema de Gerência) e EDI (Intercâmbio Eletrônico de Dados).

[16] "Polling."

[17] "Event report."

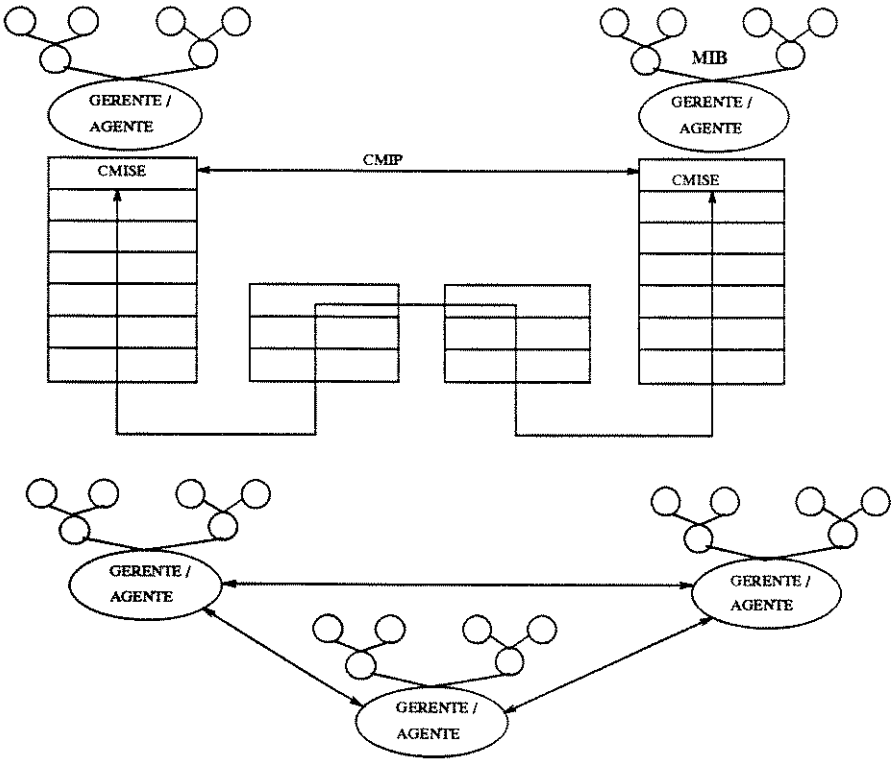


Figura 2.3 Estrutura OSI de gerência

Base de Informação de Gerência (MIB)

Para obter independência na representação dos recursos gerenciados, utilizou-se a *técnica de modelagem por objetos* como base para a descrição de um recurso lógico ou físico do sistema. Exemplos de recursos lógicos são conexões ou entidades de camada, enquanto um modem representa um recurso físico. A cada um destes recursos corresponde um objeto composto de uma série de atributos e operações. As operações podem modificar os atributos diretamente ou determinar uma ação a ser realizada sobre o objeto, como a ativação de um enlace, por exemplo.

Os objetos gerenciados são agrupados em uma base *conceitual*, chamada **base de informações de gerência (MIB)**. Ela é dita *conceitual* para evitar a obrigatoriedade de sua implementação física como um banco de dados orientado a objetos. De fato, o que se observa com freqüência é a implementação da MIB como um conjunto de objetos armazenados em memória.

Através da coleta de dados dos dispositivos representados na MIB e da definição de limiares para a geração de eventos, é possível analisar falhas da rede, avaliar o seu desempenho e obter estatísticas para planejamentos futuros.

2.3.2 Modelo de Informação

A informação manipulada por uma entidade da camada (N) pode ser decomposta em três classes [33]:

Informação de Protocolo que são todas as variáveis diretamente relacionadas à operação da camada (N), tais como as variáveis de estado e as de contexto;

Informação de Relatório são as relacionadas com a história da entidade, como estatísticas de tráfego, contabilização e falhas;

Informação de Ambiente que descreve o contexto em que se insere a entidade, tais como endereços, tabelas de roteamento e características de serviços da camada $(N-1)$.

Unificando estas três classes, adotou-se um **serviço comum de informação de gerência (CMIS)**[18], que permite: o *monitoramento*, que obtém informação para gerência, o *controle*, que atua sobre os dispositivos, e o *relatório* no qual os dispositivos relatam a ocorrência de eventos anormais. Contudo, para que se realize uma operação de gerência, é necessário que se conheça a forma de referenciar cada um dos objetos gerenciados. Foram definidos, então, três formas de estruturação entre objetos: a *hierarquia de herança*, a *hierarquia de nomes*[19] e a *hierarquia de registro*.

Hierarquia de Herança

Esta forma de hierarquia é simplesmente a relação existente entre as várias classes de objetos. Trata-se da estruturação entre as classes existente em linguagens orientadas a objetos, como C++, Smalltalk e Objective-C, entre outras. Os objetos pertencentes a uma mesma classe possuem mesmo comportamento, atributos e operações. As classes derivadas especializam o comportamento das superiores adicionando-lhes atributos ou operações.

[18] "Common Management Information Service"
[19] "Naming hierarchy".

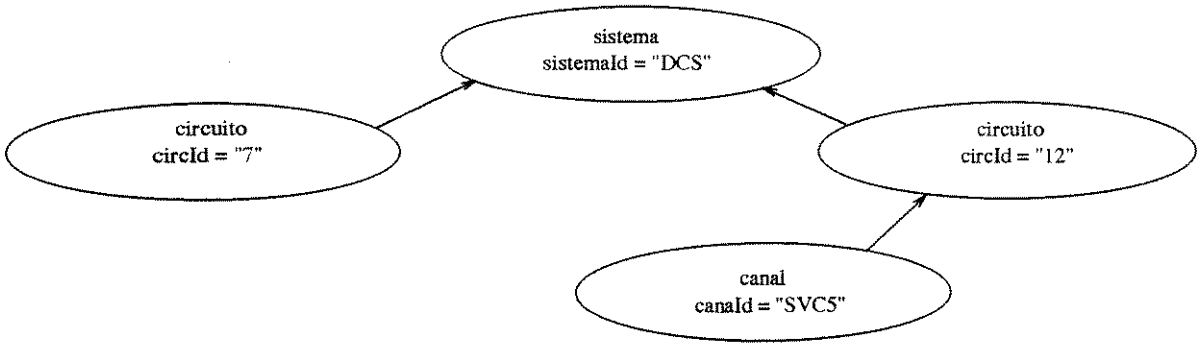


Figura 2.4 Relação de inclusão identificando objetos

Relação de Inclusão e Hierarquia de Nomes

Na **relação de inclusão**, quando um objeto que contém um outro deixa de existir, o mesmo acontece com o que estava contido.

A **hierarquia de nomes** aplica a *relação de inclusão* para identificar, de forma não ambígua, um objeto dentro de um dado contexto. Esta é a forma de organização de objetos utilizada no padrão de diretório X.500. Neste padrão, uma estrutura de árvore invertida é utilizada, tendo nos níveis superiores nomes de países, operadores, etc, e nos níveis próximos das folhas tem-se entidades de camadas, circuitos, canais virtuais, etc. Um objeto (subordinado) é identificado como sendo o nome do objeto que o contém (superior), juntamente com uma informação que identifica o objeto subordinado dentro do escopo do superior.

A Figura 2.4 mostra o objeto *sistema* composto de *circuitos* e estes, contendo *canais*. Os vários circuitos do sistema são identificados pelo atributo *circId*, e em de cada circuito, um canal é identificado pelo seu *canalId*.

Identificação dos Objetos Gerenciados

Dado que cada objeto pode ser identificado unicamente dentro de seu superior através de um atributo escolhido para tal, criaram-se os conceitos de *Nome Distinto Relativo*[20] *(RDN)* e *Nome Distinto*[21] *(DN)*. Nos RDN e DN, a **atribuição de valores a atributos**[22] é usada na identificação de um objeto[23]. A Tabela 2.1 mostra alguns dos RDN e DN dos objetos da Figura 2.4.

Delimitação de Escopo e Filtragem

O CMIS permite a realização de operações sobre um conjunto de objetos de uma só vez. Para determinar exatamente sobre quais objetos uma dada operação atuará são empregados dois conceitos: *delimitação de escopo* e *filtragem*. Como a informação é organizada em uma estrutura hierárquica, inicialmente escolhe-se um *objeto base* para ser tomado como referência. Este objeto

[20] "Relative Distinguished Name."

[21] "Distinguished Name."

[22] "Attribute value assertion."

[23] Note-se que esta identificação refere-se somente às instâncias dos objetos gerenciados, e não às suas respectivas classes.

Nome Distinto Relativo	Nome Distinto Local
sistemaId = "DCS"	{ }
circId = "12"	{ circId = "12" }
canalId = "SVC5"	{ circId = "12" , canalId = "SVC5" }

Tabela 2.1 Exemplos de RDNs e DNs locais

identificará o ponto mais alto da árvore no qual a operação é realizada. O **escopo** determina a profundidade máxima em relação ao objeto base na qual estão contidos os objetos de interesse, e o **filtro** identifica, usando uma expressão booleana, quais destes objetos possuem valores de atributos desejados.

A aplicação de escopo e filtragem, no exemplo da Figura 2.4, com uma delimitação de escopo em *um nível abaixo do objeto base "sistema"* e com um filtro de $(circId \geq 9)$, resultaria no circuito *circId = "12"*.

Hierarquia de Registro

Para que objetos de informação[24] possam ser identificados unicamente, criou-se a **hierarquia de registro**. Através desta hierarquia pode-se referenciar globalmente o formato de qualquer *tipo* que venha a ser transmitido pela rede, seja ele um objeto, um atributo de um objeto, uma notificação, uma operação ou um nome de objeto.

A forma de organização mais comum [25] é a disposição dos objetos em uma árvore invertida. A cada nó da árvore de registro ligado ao nó raiz está associada uma autoridade de registro, como ISO ou ITU, que administra a atribuição dos identificadores correspondentes à sua sub-árvore, conforme mostra a Figura 2.5.

Cada um dos nós da árvore de registro possui um **identificador de objeto**[25] **(OID)**. O OID é uma seqüência não ordenada de números que identifica unicamente um objeto de informação. A sintaxe para um OID é definida em ASN.1.

2.3.3 Modelo de Comunicação

De acordo com o OSI-MF, cada camada tem suas próprias funções de gerência, que são executadas pelo **Gerente de Camada**[26] (LM), conforme ilustrado na Figura 2.6. O gerente de camada serve para coordenar as atividades das entidades da camada e atua como uma janela para a gerência do sistema sobre ela. Estas extensões não implicam em nenhuma mudança nas especificações do protocolo da camada, ainda que requeiram drásticas mudanças na implementação deste [22].

Dado que não modificam as especificações dos protocolos das camadas, os LMs não podem interagir diretamente entre si, a menos que em sua especificação original o protocolo já preveja esta interação, como é o caso dos pacotes de registro e diagnóstico do protocolo X.25. Quando a interação direta não é possível, a sua coordenação e intermediação fica a cargo do **Processo**

[24] A noção de objeto de informação usada aqui é semelhante à noção de **tipo** discutida na Seção 2.5.2.
[25] "Object Identifier".
[26] "Layer Manager".

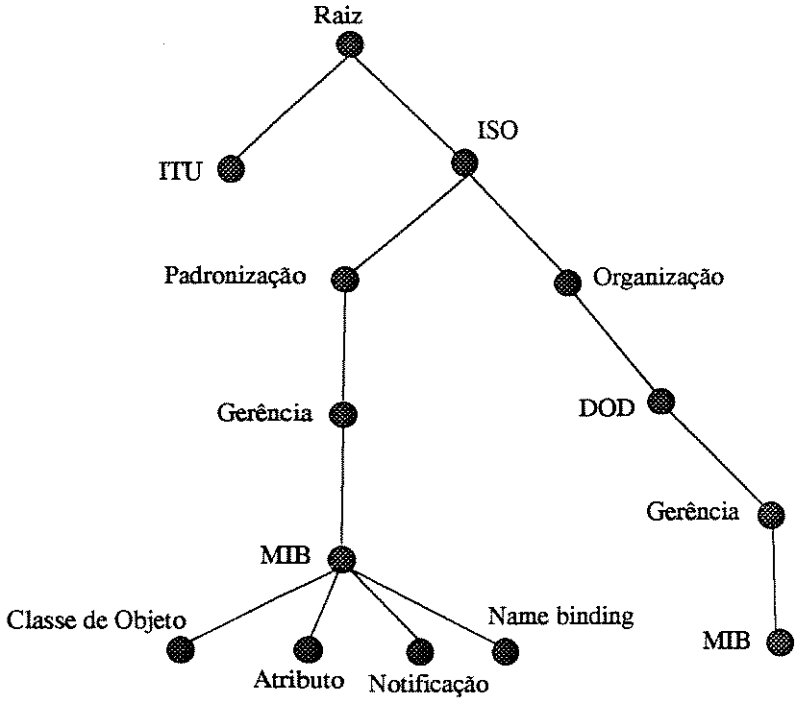


Figura 2.5 Hierarquia de registro

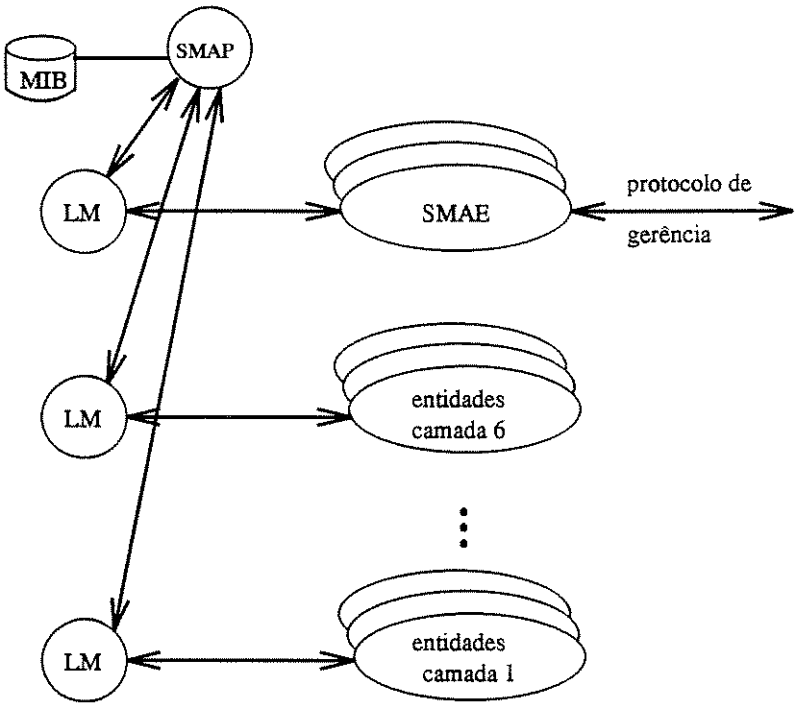


Figura 2.6 Modelo de gerência OSI

de Aplicação de Gerência de Sistemas (SMAP)[27] (ver Figura 2.6). Este processo está formalmente fora do ambiente OSI-MF, pois trata-se de um mapeamento interno entre a MIB e as entidades de camada, não havendo nenhuma troca de informação entre sistemas neste mapeamento.

O SMAP interage com os LMs e com as **Entidades de Aplicação de Gerência de Sistema (SMAE)** para executar a gerência do sistema. As SMAEs são entidades da camada de aplicação responsáveis pela comunicação com sistemas remotos. Para que um gerente de camada LM possa se comunicar com o seu correspondente em um sistema remoto, ele terá que encaminhar uma mensagem via SMAP/SMAE, o que pode se constituir em um problema quando se gerencia "gateways" ou "bridges", pois eles provavelmente não possuirão camadas superiores, como a de aplicação.

Protocolo

Associado ao serviço CMIS[28], que é um serviço, definiu-se o **Protocolo de Informação Comum de Gerência (CMIP)**[29], que define precisamente como ocorre a troca de informações entre aplicações de gerência.

O serviço CMIS é orientado à conexão. Portanto, para utilizá-lo deve-se:

a) Realizar uma conexão em nível de aplicação entre duas entidades de gerência;

b) Transmitir algumas das primitivas:

get retorna uma informação;

set manipula uma informação;

action executa algum comando imperativo (ex. "reboot" do dispositivo);

create cria uma instância de um objeto de gerência (ex. uma nova linha na tabela de roteamento);

delete deleta uma instância de um objeto de gerência;

event-report relata eventos extraordinários.

c) Terminar a conexão.

As primitivas **"get"**, **"set"**, **"action"**, **"create"** e **"delete"** são iniciadas pelo gerente que fica à espera de uma resposta por parte do agente. No caso de **"event-report"**, o gerente cria um objeto chamado **discriminador de envio de eventos**[30] contendo o seu endereço e um filtro. Quando um evento ocorre, o filtro é aplicado e, caso todas as condições sejam satisfeitas, o evento é enviado ao gerente.

[27] "Systems Management Application Process"
[28] Explicado no modelo de informação. Seção 2.3.2.
[29] "Common Management Information Protocol"
[30] "Event forwarding discriminator"

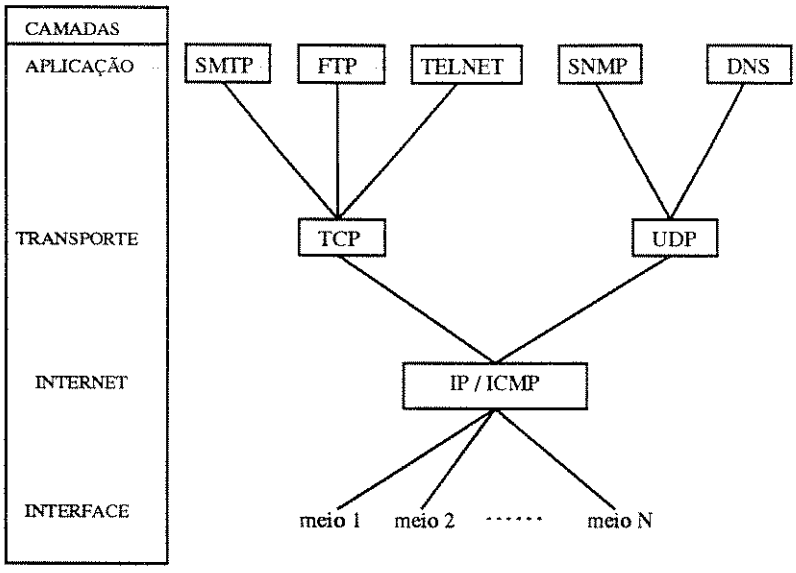


Figura 2.7 Hierarquia de protocolos Internet

2.4 A Família Internet de Protocolos

Quando se utiliza a Família Internet de Protocolos, a transparência desejável para a Gerência de Rede é conseguida com a utilização do **Protocolo Simples de Gerência de Redes**[31] **(SNMP)**. O SNMP pertence à Camada de Aplicação da **Arquitetura Internet de Gerência**[32] **(INMF)**, e se utiliza dos protocolos UDP e IP, que são de mais baixo nível, conforme ilustra a Figura 2.7.

2.4.1 Arquitetura e Filosofia

O Protocolo Internet [67] foi projetado para uso em sistemas de computadores interconectados por redes de comutação de pacotes, e fornece meios para a transmissão de blocos de dados (datagramas) de uma ou mais fontes, a um ou mais destinatários.

A família Internet oferece três conjuntos de serviços [21], os quais exibem a dependência que os níveis inferiores têm dos superiores:

- No nível mais baixo, um serviço de entrega de pacotes sem conexão fornece suporte aos demais serviços;
- No próximo nível, um serviço de transporte confiável fornece uma plataforma para a construção de aplicações que exigem integridade dos dados e seqüenciamento de pacotes;
- No nível de aplicação estão serviços que os usuários normalmente utilizam, tais como o FTP que serve para fazer transferência de arquivos.

[31] "Simple Network Management Protocol."
[32] "Internet Network Management Framework."

2.4.2 Camadas do Modelo

O Modelo Internet pode ser dividido em quatro camadas [69]:

Camada de Interface - responsável pela transmissão em uma única rede física denominada meio, como por exemplo, Ethernet, Token Ring, FDDI, X.25, etc. Corresponde, no Modelo OSI, às camadas física, de enlace, e parte da camada de rede;

Camada Internet - Fornece um nível comum de *serviço de entrega*, o qual é independente das potencialidades do meio; um mecanismo de *endereçamento global*; e um esquema de roteamento para transferir dados através da concatenação de redes físicas.

Camada de Transporte - responsável pela transferência de dados, com o nível de confiabilidade desejado pela aplicação. O serviço confiável entrega os pacotes em seqüência, com integridade garantida e é orientado a conexão (TCP), enquanto que o não confiável (UDP) é apenas um serviço de entrega com endereçamento para a camada de aplicação ("port");

Camada de Aplicação - descreve as tecnologias usadas para prover serviços a usuários finais. Devido ao número reduzido de camadas, muitas das tarefas realizadas pelas camadas superiores do modelo OSI são incorporadas à camada de aplicação no modelo Internet. Como conseqüência, conversão de dados entre computadores e sincronização de atividades entre aplicações, realizadas pelas camadas OSI de apresentação e sessão, respectivamente, ficam a cargo da camada de aplicação.

Na Figura 2.7 apresenta-se uma distribuição de serviços e protocolos existentes e a sua camada correspondente:

IP *(Internet Protocol)* - serviço de rede sem conexão, isto é, a ligação é feita pelo envio de pacotes;

ICMP *(Internet Control Message Protocol)* - permite a "gateways" ou computadores o envio de mensagens de erro ou de controle [21];

UDP *(User Datagram Protocol)* - serviço de transporte sem conexão, no qual o próprio usuário tem de fazer o controle de pacotes perdidos ou duplicados.

TCP *(Transmission Control Protocol)* - serviço de transporte orientado a conexão;

SMTP *(Simple Mail Transfer Protocol)* - fornece serviço do tipo armazena-e-envia[33] para mensagens textuais de correio eletrônico [69];

TELNET - oferece três tipos de serviços básicos [21]: define um terminal virtual qua fornece uma interface para sistemas remotos; inclui mecanismos que permitem a negociação de parâmetros entre cliente e servidor; e faz o tratamento de ambos os lados da conexão simetricamente, permitindo que qualquer destes seja um programa;

FTP *(File Transfer Protocol)* - fornece serviço de transferência de arquivos;

DNS *(Domain Name System)* - fornece o mapeamento entre nome de computadores e seus endereços de rede.

[33] "Store-and-forward."

2.4.3 Serviço de Entrega

O **serviço de entrega**, é sem conexão, com os dados de usuário sendo enviados através de datagramas, possuindo todos o endereço completo do destinatário. Este serviço possui as seguintes propriedades [21]:

- a camada internet não possui estado, ou seja, da perspectiva da camada, não existe relação entre os datagramas;
- o serviço é não confiável porque a entrega não é garantida;
- os pacotes de uma seqüência, enviados de uma máquina para outra, podem percorrer caminhos diferentes, podendo ser perdidos ou chegar fora de ordem;
- o serviço é chamado de melhor esforço, pois tentará ao máximo entregar os pacotes, tornando, desta forma, mais fácil a escolha de um meio de transmissão;

Endereçamento

O endereço Internet é uma quantidade de 32 bits (representada externamente na forma A.B.C.D, onde A,B,C e D são valores decimais de cada um dos quatro octetos) dividida em dois campos para tornar eficiente o roteamento dos pacotes:

Identificador da Rede - que se refere a uma rede física em particular;

Identificador do Hospedeiro (host) - que se refere a um dispositivo componente da rede. Um dispositivo conectado a mais de uma rede terá mais de um endereço;

Note-se que um endereço IP é um artefato lógico, ele não guarda relação com "hardware", meio de transmissão, nem qualquer outra conotação física.

Conforme a porção do endereço dedicada para a identificação do hospedeiro e da rede, são constituídas as chamadas classes de endereçamento [21]:

Classe A - são destinados a redes que têm mais de 65.535 hospedeiros. Dedicam 7 bits para o endereço da rede e 24 bits para a identificação do hospedeiro. Como só existem 7 bits para a rede, apenas 128 redes deste tipo podem existir no mundo (endereçado pela Internet);

Classe B - são destinados a redes de tamanho intermediário, isto é, que possuem entre 256 e 65.536 hospedeiros. Alocam 14 bits para o endereço de rede e 16 bits para o identificador do hospedeiro;

Classe C - para redes que têm menos que 256 hospedeiros. Destinam 22 bits para o endereçamento da rede e 8 bits para o hospedeiro.

A distribuição em classes permite uma melhor gerência dos endereços por parte de quem os distribui e controla, que é a **Autoridade de Atribuição de Números Internet**[34]**(IANA)** [69].

[34] "Internet Assigned Numbers Authority."

Roteamento

O termo *Roteamento* refere-se ao processo de escolha de um caminho através do qual são enviados os pacotes, sendo o *Roteador* o computador que faz esta escolha [21]. No roteamento, duas situações podem ocorrer:

i) Se o destinatário está na mesma rede física do remetente, este encapsula o dado em um datagrama e o envia diretamente ao destino;

ii) Caso a rede física não seja a mesma, seleciona-se um "gateway" para o qual o datagrama é enviado. Este, então, passa de um "gateway" para outro até que um deles o entregue diretamente ao destino. Os "gateways" são chamados sistemas intermediários na terminologia OSI.

O Protocolo de Mensagens de Controle Internet - (ICMP)

O protocolo Internet (IP) possui um mecanismo que permite às máquinas conectadas à rede informarem suas condições anormais ou de erro [21]. Este mecanismo é denominado **Protocolo de Mensagem de Controle Internet (ICMP)**[35] e é considerado parte integrante obrigatória de qualquer implementação do protocolo IP.

Embora a funcionalidade permitida pelo ICMP seja modesta, é possível a construção de ferramentas importantes no auxílio à gerência de redes [69]. Algumas destas ferramentas são:

Ping (Internet Packet Grouper) - Envia pacotes ICMP de requisição de eco a um endereço IP, e espera resposta. Isto ajuda na determinação da acessibilidade do referido endereço. O Ping também dá uma indicação da qualidade do canal através da apresentação do percentual de perda dos datagramas enviados em relação ao número de resposta obtidas.

Traceroute - Envia *pacotes de prova* a um endereço IP, e espera respostas dos vários hospedeiros pelos quais os pacotes passaram, permitindo que a rota seja traçada.

Gerência Passiva - utiliza monitores que são residentes ou conectados aos próprios dispositivos da rede, de forma a detectar anomalias no tráfego normal que flui pela rede. É importante notar, neste caso, que diferentemente do Ping e do Traceroute, nenhum tráfego adicional é gerado.

A funcionalidade oferecida pelo protocolo ICMP é equivalente à interação entre gerentes de camada (LM) da OSI-MF, pois trata-se de uma cooperação entre entidades existentes na camada IP.

As mensagens trocadas pelo protocolo são, entre outras:

destinatário inacessível : é enviada ao remetente quando um "gateway" não pode entregar um datagrama IP;

tempo excedido : é enviada ao remetente quando o campo *"time to live"* do datagrama IP chegou a zero;

[35] "Internet Control Message Protocol."

redirecione : é enviada ao remetente para informar que existe um "gateway" mais próximo do destinatário que o utilizado;

timestamp : informa o tempo de atraso entre dois hospedeiros, permitindo-lhes sincronizar os seus relógios;

2.4.4 Serviço de Transporte

Para o serviço de transporte existem dois protocolos disponíveis, dependendo da escolha do modo conectado ou sem conexão. No modo conectado usa-se o protocolo TCP e a comunicação é dita *confiável*: forma-se uma cadeia de "bytes" entre os dois processos que o utilizam. No modo sem conexão utiliza-se o protocolo UDP, e a comunicação é dita *não confiável*.

Endereçamento da Camada de Aplicação

Para realizar endereçamento no nível da camada de aplicação, os protocolos TCP e UDP utilizam o conceito de soquete ("socket"), identificando unicamente cada um dos processos residentes em um determinado hospedeiro, que está se comunicando ou à espera de mensagens. O soquete é composto do endereço IP do hospedeiro e de um "port", que é um inteiro de 16 bits atribuído unicamente, em cada máquina, aos processos que requerem identificação.

Existem alguns "ports" destinados especificamente para determinadas aplicações, que são denominados "ports" *bem conhecidos*, e atribuídos pela IANA. Como exemplo, tem-se o 161 que é atribuído à parte de uma implementação do SNMP, responsável pela espera de pedidos de informações, ou de quaisquer outras interações previstas neste protocolo.

2.5 Modelo de Gerência da Família Internet de Protocolos

O Modelo de Gerência de Redes Internet é muito próximo do definido para a gerência de redes no Modelo de Referência OSI. Ele foi desenvolvido a partir do Serviço de Informação Comum de Gerência (CMIS) e de seu protocolo associado, o CMIP. A própria definição dos objetos gerenciados é feita utilizando uma linguagem OSI, a ASN.1, e os objetos também são armazenados em uma MIB.

2.5.1 Arquitetura Funcional

Do ponto de vista funcional, o aspecto central é a interação entre gerentes e agentes, conforme mostra a Figura 2.8. Um agente somente pode exercer função de agente, o mesmo ocorrendo com o gerente. Isto torna o agente o mais simples possível, podendo ser implementado em qualquer dispositivo.

Como a diversidade dos dispositivos é grande, o modelo INMF terá tanto mais sucesso quanto menos carga impuser aos nós gerenciados, fazendo uma analogia com o IP, cujo sucesso se deve principalmente aos requisitos mínimos exigidos do nível de interface. Desta maneira, tal como no IP, a complexidade é deixada para as camadas superiores do protocolo de gerência.

Da mesma forma que na OSI-MF, os **agentes** são os depositários da MIB, que é a representação abstrata dos recursos que estão sob sua responsabilidade, respondendo comandos referentes

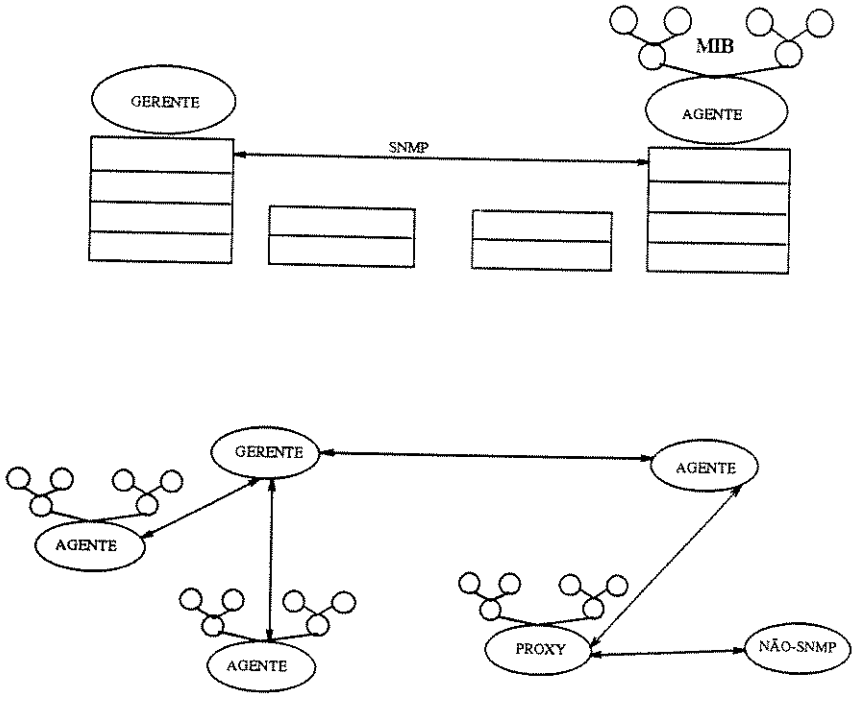


Figura 2.8 Estrutura de gerência Internet

a pedidos de informação sobre os objetos armazenados na MIB ou enviando informações relatando uma mudança de estado ocorrida em alguns destes objetos.

As operações de gerência são realizadas através da leitura e modificação dos dados armazenados na MIB. Este é um ponto em que os modelos OSI e Internet de gerência diferem. No caso do OSI, existem a leitura e a modificação de valores de dados da MIB bem como ações imperativas, como reinicializar um dispositivo ou tirá-lo de operação. No caso Internet, não existem ações imperativas, apenas as de leitura e modificação de valores. As ações imperativas são mapeadas em operações de mudança de valores. Exemplificando, ao se mudar o valor de um atributo que denota o estado de um dispositivo para *desligado*, o dispositivo sai automaticamente de operação.

O **gerente** é a aplicação que realiza todas as funções relativas à interpretação dos dados para executar a gerência propriamente dita.

A forma de interação entre gerente e agente é realizada por meio de sondagem, isto é, periodicamente o agente é indagado acerca dos objetos da MIB. Existe também o mecanismo de **trap**[36] através do qual o agente solicita ao gerente uma operação de sondagem. Esta forma de atuação, apesar de ser semelhante ao mecanismo de notificação, possui características diversas. Na notificação, o objeto que representa um dado recurso envia uma informação completa ao gerente que a tenha solicitado, enquanto que no *"trap"* o agente apenas solicita ao gerente uma sondagem para obter esta informação completa.

Domínios de Gerência

Administrativamente, os nós gerenciados são inseridos em **domínios de gerência**, que são regiões de atuação de um ou mais gerentes, permitindo que um agente determine como e por quem ele é gerenciado.

A partição de domínios de gerência é realizada através do conceito de *comunidades*[37]. As **comunidades** são um mecanismo de autenticação simples existente em cada agente, de forma a permitir a implementação de políticas de autorização. Cada agente responde a qualquer gerente pertencente às comunidades com as quais se identifica.

Quando as mensagens de gerência são trocadas, elas contêm duas partes: um *nome de comunidade*, com alguma informação adicional requerida para validar o originador como um membro da mesma; e dados, contendo uma das primitivas do serviço de gerência e operandos associados.

Uma vez autenticado o membro de uma comunidade, o nó gerenciado determina qual o nível de acesso permitido a este membro, selecionando os objetos que são visíveis ao nó de gerência. Cada objeto visível, pode ser acessado somente para leitura, ou disponível para leitura e escrita.

Gerência por Procuração

Para permitir a gerência de dispositivos simples, que não implementam nem a funcionalidade mínima requerida para um agente, foram criados os **agentes por procuração**[38]. Estes ficam responsáveis por elementos de rede sob seu domínio, construindo uma barreira entre estes e um ou mais gerentes.

---

[36] "Trap directed polling".

[37] "Communities."

[38] "Proxy agent".

O conceito de Gerência por Procuração foi introduzido para permitir que dispositivos da rede, que geralmente não implememtam o SNMP, tais como roteadores e pontes, possam ser gerenciados.

2.5.2 Modelo de Informação

Os serviços de informação de gerência fornecem três tipos de uso:

monitoramento: onde cada nó é visualizado através de variáveis, cuja leitura permite o monitoramento;

controle: a mudança nos valores das variáveis proporciona o controle sobre o nó;

relatório: habilita alguns dispositivos a relatarem eventos anormais;

No caso da Internet, os objetos gerenciados não são definidos em termos de *classes de objetos*, mas através de *tipos de objetos*. A noção de **tipo de objeto** usada no INMF é a mesma adotada no padrão ODP [50], na qual *tipo de* <*X*> é um predicado lógico caracterizando uma coleção de <X>s, ou seja, <X> é de um determinado tipo se o predicado correspondente àquele tipo se verifica para <X>. Desta forma, um tipo pode ser uma classe, um atributo, uma notificação, um "name binding", etc.

No INMF um tipo é uma macro definida usando a linguagem ASN.1 [25], e contém cinco itens: o identificador, a sua descrição em ASN.1, a definição da semântica, o modo de acesso (somente para leitura, leitura e escrita, somente escrita ou inacessível) e o status (obrigatório, opcional ou obsoleto).

Hierarquia de Registro

Na árvore da Figura 2.5, existe o registro tanto dos objetos de gerência da OSI-MF quanto os da INMF, registrados sob o nó *DOD*. Portanto, a hierarquia de registro adotada na INMF é a mesma adotada na OSI-MF, sendo ambas especificadas seguindo as regras definidas pela notação ASN.1 para OIDs.

Identificação dos Objetos Gerenciados

A identificação de instâncias na INMF difere substancialmente da OSI-MF. A organização da informação gerenciada é na forma de árvore, empregando um método de sufixação. A partir da árvore de OIDs da hierarquia de registro, adiciona-se um sufixo de acordo com as regras a seguir:

1. Somente instâncias de objetos folhas[39] podem ser identificadas;
2. Se o objeto não é uma coluna de tabela, o sufixo é 0, por exemplo, para o objeto *sysdescr*, a instância é *sysdescr*.0;

[39]Objetos que são localizados nas pontas da árvore transversalizada.

3. Se o objeto é uma coluna de uma tabela, sua descrição textual na MIB define como o sufixo é formado através da seleção de colunas necessárias para tornar único o sufixo de cada instância. Por exemplo, Uma tabela *ifTable*, que possui *ifIndex* como um valor de identificação das instâncias de suas colunas:

ifTable	
ifIndex	ifDescr
1	le0
2	lo0

Tem sua primeira instância identificada por *ifdescr.1*, cujo valor é *le0*.

2.5.3 Modelo de Comunicação

O sistema de gerência de redes é composto de [69]: vários *nós gerenciados*, cada um contendo um *agente*; ao menos uma *estação de gerência de rede* (NMS)[40]; e um protocolo de gerência de rede, o SNMP.

A interação entre os agentes e as entidades de protocolo é realizada através da *Instrumentação de Gerência*, conforme ilustrado na Figura 2.9. A Instrumentação de Gerência usa um mecanismo interno de comunicações, para modificar os valores das informações contidas na base de gerência.

Os agentes têm comunicação direta com a *Base de Informações de Gerência (MIB)*, e com a *Aplicação de Gerência* via protocolo de gerência. A comunicação entre os agentes e as demais entidades de protocolo é realizada através da instrumentação de gerência.

Operações do SNMP

As operações correspondentes às funcionalidades de monitoramento, controle e relatório no protocolo SNMP são:

- Operações para monitoramento:

  get - retorna o valor de uma informação específica de gerência;

  get-next - usado para a recuperação transversal da informação, dado que esta é organizada em forma de uma árvore[41]. Os identificadores das instâncias, que são OIDs seguidos de um sufixo, são utilizados para a varredura lexicográfica da MIB.

- Operação para controle:

  set - que manipula o valor de uma determinada informação, fazendo com que o sistema se adapte para que esta se torne verdadeira, como por exemplo, a mudança da variável "shutdown" de 0 para 1 faz com que o processo de "shutdown" seja iniciado naquele sistema.

[40] "Network Management System."
[41] Conforme visto na Seção 2.5.2.

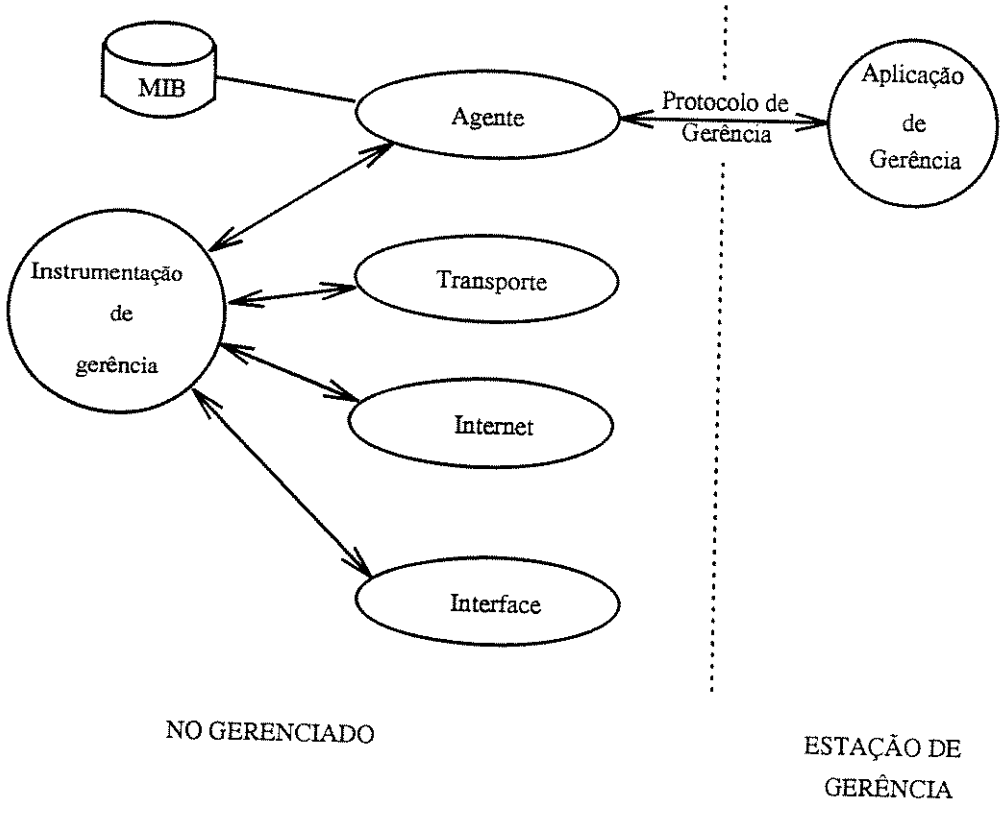


Figura 2.9 Interação entre entidades no modelo Internet de gerência

- Operação para relatório:

  trap - informa sobre a ocorrência de eventos extraordinários, de forma assíncrona.

As operações do protocolo SNMP são semelhantes às do seu correpondente na OSI-MF, o CMIP. Contudo, algumas diferenças são observadas:

a) O modo conectado é o indicado para o CMIP, enquanto que no SNMP, o preferido é o sem conexão (envio de datagramas). A argumentação utilizada para defendê-lo [69] é que a correção automática de erros, existente no modo conectado é, na maioria das vezes imprópria, quando se lida com a gerência de uma rede em colapso;

b) As operações *action*, *create* e *delete*, do CMIP, são substituídas por uma única no SNMP, o *set*. Neste caso argumenta-se [69] que não passa de uma discussão CISC x RISC (Complete Instruction Set Computers x Reduced Instruction Set Computers).

2.6 A Rede de Gerência de Telecomunicações

A **Rede de Gerência de Telecomunicações (TMN)**, cuja arquitetura segue a recomendação ITU-T M.3010 [48], é um sistema de computadores projetado para dar suporte às atividades de gerência associadas a redes de telecomunicações e engloba a Gerência de Sistemas OSI. Outro ponto importante na TMN é que ela prevê interações com outras arquiteturas através de adaptadores.

Para realizar a função de gerência propriamente dita, a TMN se utiliza dos mesmos mecanismos da OSI-MF (ver Figura 2.10). Cada um dos blocos funcionais usa a relação gerente-agente para realizar suas atividades. Conseqüentemente, continuam válidos o modelo de representação de objetos, os esquemas de nomeação de objetos e de escopo e filtragem, a hierarquia de registro, bem como o modelo de comunicação e o seu protocolo associado, o CMIP. Portanto, serão apresentados nesta seção apenas os pontos nos quais a TMN adicionou um maior nível de abstração.

2.6.1 Arquitetura Funcional

O ponto central na arquitetura TMN é a interação entre blocos funcionais de *Operações de Sistema (OSF)*[42] e de *Elemento de Rede (NEF)*[43], e entre os próprios OSFs entre si (Ver Figura 2.11). Os primeiros agem como gerentes com respeito ao segundos. Sempre que esta interação não é possível de forma direta, blocos funcionais de mediação (MF)[44] ou Adaptadores-Q são usados.

A função de cada um destes blocos é detalhada a seguir:

Operações do Sistema (OSF): realiza a gerência propriamente dita;

Elemento de Rede (NEF): que executa a função de agente e contém a MIB;

[42] "Operations Systems Function Block."
[43] "Network Element Function Block."
[44] "Mediation Function Block."

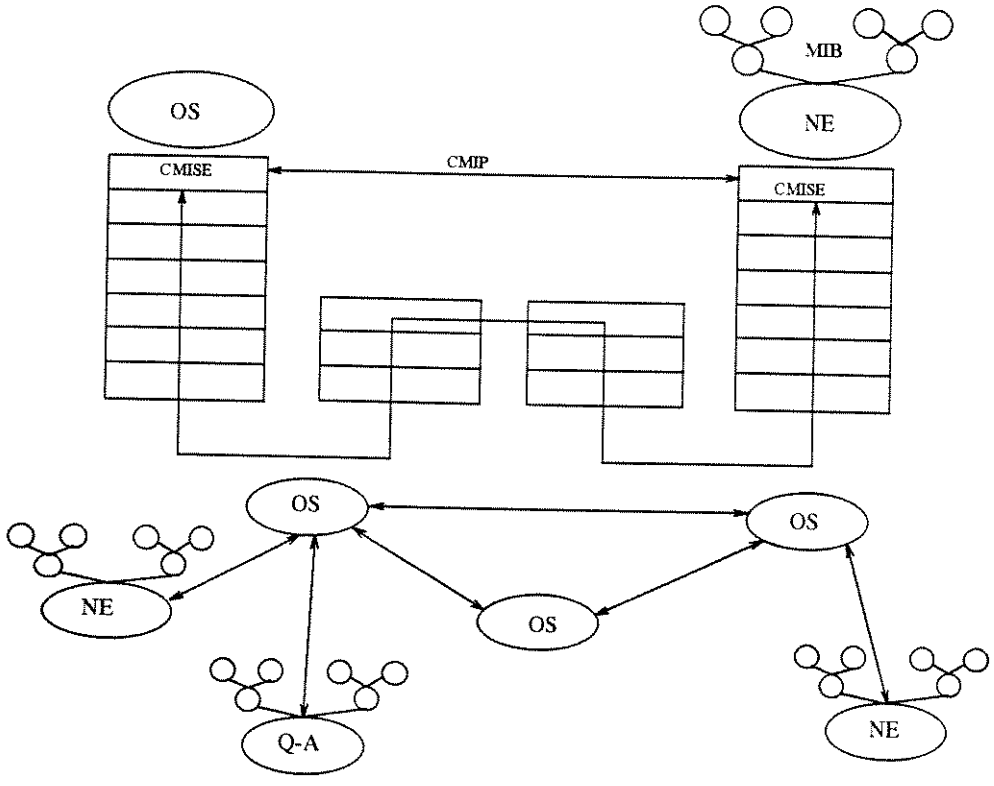
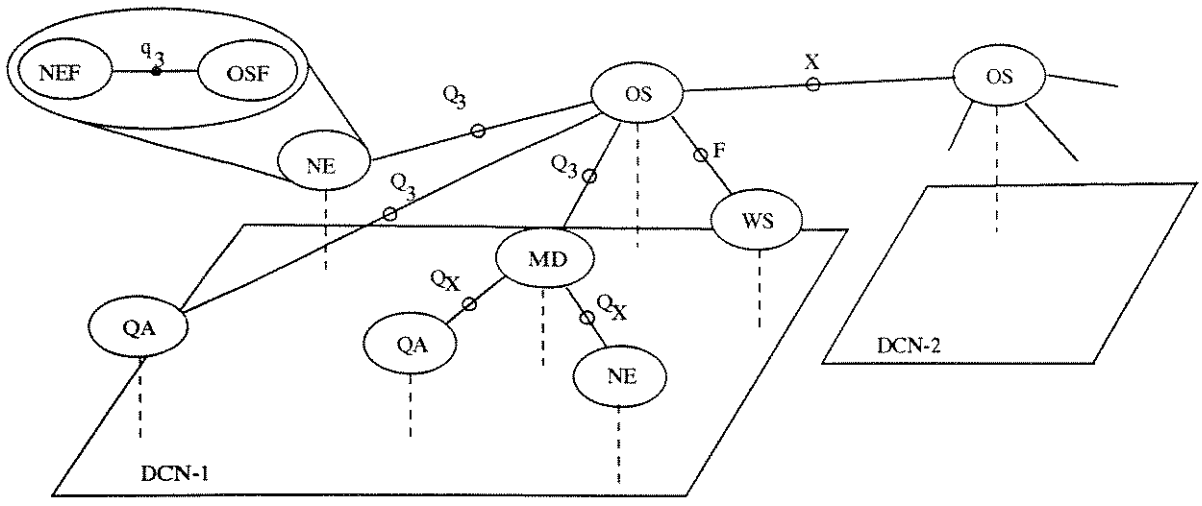


Figura 2.10 Os blocos TMN sob o ponto de vista da relação gerente-agente



Legenda:
OSF: FUNÇÃO OPERAÇÃO DE SISTEMA
NEF: FUNÇÃO ELEMENTO DE REDE
OS: OPERAÇÃO DO SISTEMA
Q_3, X AND F: TIPOS DE INTERFACES
DCN: REDE DE COMUNICAÇÃO DE DADOS
WS: FUNÇÃO ESTAÇÃO DE TRABALHO
QA: ADAPTADOR-Q
NE: ELEMENTO DE REDE
q_3: PONTO DE REFERÊNCIA

Figura 2.11 A arquitetura TMN

Adaptador-Q (QAF): usado quando a interface apresentada por um OSF ou NEF não é compatível com as adotadas no padrão TMN, sendo necessária uma conversão. Como exemplo, pode-se citar a interoperação entre um gerente OSI e um gerente Internet, na qual o adaptador terá de implementar ambos os modelos de gerência e fazer as conversões necessárias (ou possíveis);

Função de Mediação (MF): assegura que a informação tramitando entre blocos OSF e NEF (ou QAF) está de acordo com as expectativa de cada um destes blocos. Diferentemente do QAF, quando se emprega o MF a interface entre blocos está de acordo com o padrão TMN, permitindo a conexão de NEFs de diferentes complexidades ao mesmo OSF. As pilhas de protocolo não são necessariamente as mesmas, usando-se neste caso uma pilha de convergência. Isto é ilustrado pelas interfaces Q_x e Q_3 na Figura 2.11, onde Q_3 é a interface direta aos OSFs. A mediação pode consistir em [48]: conversão entre modelos de informação, como a correspondência entre representações distintas dos objetos; interoperação de protocolos, como entre protocolos orientados a conexão e sem conexão; manuseio de dados, como conversão e formatação; tomada de decisões, como re-roteamento de dados e análise de testes de circuitos; e armazenamento de dados, como os que contêm a configuração da rede.

Estação de Trabalho (WSF): [45] fornece os meios de interpretação da informação aos utilizadores da informação de gestão. A interface com o usuário não está compreendida neste

[45] "Workstation Function"

bloco, pois sendo interna ao sistema gerente, está fora do escopo delimitado pela TMN.

Para delinear a fronteira de serviço entre dois blocos funcionais, o conceito de **Ponto de Referência** foi introduzido. Estes foram então divididos em três classes[46]:

classe q: entre os blocos OSF, MF, NEF e QAF;

classe f: entre OSF/MF e WSF;

classe x: entre OSFs de duas TMNs, ou entre um OSF e o seu equivalente funcional na outra rede de gerência. Este ponto é particularmente importante quando se está interessado no ponto de interação entre duas administrações, como entre um provedor de serviço (VASP[47]) e o operador de uma rede pública (PNO[48]).

2.6.2 Modelo de Informação

Em seu modelo de informação, a TMN usa estruturação orientada a objetos, da mesma forma que a gerência de sistemas OSI, com objetos representando uma visão abstrata do recurso, isto é, considerando apenas a informação necessária ao seu gerenciamento. Um recurso pode ser representado por um ou mais objetos, e assim sendo, não existe necessariamente correspondência biunívoca.

A noção de *Conhecimento de Gerência Compartilhado* fornece ao gerente e ao agente uma visão comum da infomação intercambiada. Esta visão é estabelecida entre duas entidades da camada de aplicação[49] quando decidem acerca de seu contexto.

O conceito de **Arquitetura Lógica em Camadas (LLA)**[50] é a principal adição que o modelo TMN fornece à OSI-MF, no qual a arquitetura pode ser imaginada como uma superposição de camadas [48], sendo o escopo de cada camada mais abrangente do que o da camada inferior. Como conseqüência, ao invés de apenas uma visão plana da rede gerenciada resultante da interação entre domínios OSF, adquire-se uma visão espacial relativa ao nível de abstração.

Através da LLA, pode-se definir OSFs responsáveis pela gerência de rede, serviços e negócios, conforme ilustrado na Figura 2.12. Nesta figura, o Provedor de Serviço de Valor Agregado (VASP) fornece um serviço especializado, como por exemplo, Internet sobre os serviços de transmissão de dados da rede pública (PNO). Se o VASP não é também o provedor[51], existe uma interação entre o nível de serviço OSF de ambas as redes, seja durante o estabelecimento de uma nova instância de serviço ou no caso de eventuais problemas. O nível de serviço está relacionado a todas as transações de serviço, como o provimento ou encerramento de um serviço, contabilização, qualidade de serviço (QoS) e relato de falhas, entre outras.

[46] Existem outras classes de pontos de referência, mas não são relevantes para a apresentação da TMN neste trabalho.

[47] "Value Added Service Provider."

[48] "Public Network Operator."

[49] Neste caso, o gerente e o agente.

[50] "Logical Layered Architecture."

[51] Como é o caso ilustrado na figura.

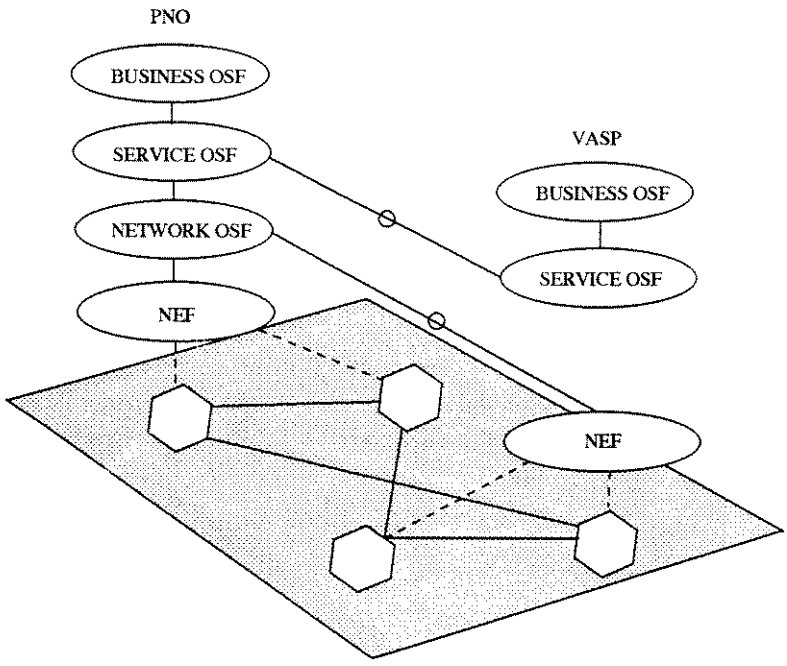


Figura 2.12 Relação entre um provedor de serviços de valor agregado (VASP) e o operador da rede pública (PNO)

2.6.3 Modelo de Comunicação

A realização física de um bloco constitutivo TMN é uma composição de um ou mais blocos funcionais. Por exemplo, um *Elemento de Rede* pode ser composto dos blocos NEF, MF, QAF e WSF, o que significa que todas as funcionalidades daqueles blocos podem estar presentes em um NE, embora o único obrigatório seja o NEF. Quando tal composição acontece, interfaces completas entre os blocos funcionais não são obrigatórias, mas o ponto de referência deve ser preservado, pois é ele que delimita a fronteira entre os blocos. Exemplificando, se um NE é composto de um NEF e um MF, o ponto q_x, e não a interface Q_x é preservada. A diferença entre um ponto de referência e uma interface é que na última, um canal usando um dos protocolos padrões adotados para aquela interface é estabelecido para ligar os blocos em questão[52].

A Figura 2.11 ilustra um NE composto das funções NEF e OSF, cuja troca de informação se dá pelo ponto de interação q_3, e a sua interação com o OS externo é realizada através de uma interface Q_3.

Interfaces Normalizadas

O par serviço/protocolo CMIS/CMIP para a troca de informações entre blocos funcionais determina apenas parte da camada 7 do modelo OSI, o suficiente para garantir a interoperabilidade [65]. Algumas funcionalidades podem não ser necessárias em todos os casos, como a transferência de

[52] Esta noção de interface é bastante diferente da usualmente adotada em modelagem por objetos, que não requer um canal com protocolos padrões para que uma interface seja definida.

arquivos, por exemplo.

Para a interface Q_3, recomenda-se que cada conjunto de funções de aplicação tendo as necessidades similares, utilizem escolhas de protocolos bem definidas para as camadas 4 a 7 do modelo OSI, havendo uma maior gama de escolha para as camadas 1 a 3 para permitir uma seleção que realize o transporte de dados de forma mais eficiente.

No caso da interface Q_x, há uma maior flexibilidade de escolha, permitindo que se substitua as camadas 4 a 6 por um mapeamento da informação, adaptando-a ao requerido na camada 7.

2.7 A Arquitetura de Rede Inteligente

Os modelos apresentados possuem características similares, como a existência de um modelo de organização da informação gerenciada, uma base de informações de gerência e protocolos que realizam a troca de informações entre entidades usuárias do serviço de informação de gerência.

Em contrapartida, a *Arquitetura de Rede Inteligente (IN*[53]*)* se preocupa muito mais com a implementação e controle de serviços, do que com a gerência da rede propriamente dita. Ao se concentrar no controle dos dispositivos, as atividades a serem realizadas pela IN requerem reações muito mais rápidas do que as existentes no plano de gerência, cujas atividades são normalmente de longo prazo, como planejamento e administração.

Entretanto, a previsão de evolução da IN advoga a sua integração com a rede TMN, unindo o provimento e controle dos serviços com a gerência da rede e serviços.

O termo **Rede Inteligente** é usado [40] para descrever um conceito arquitetural aplicável a todas as redes de telecomunicações para a operação e provimento de novos serviços. Suas principais características são o uso extensivo de técnicas de processamento de informação, a criação e implementação de novos serviços por meio do reuso de funções de rede modularizadas e o controle pelo assinante[54] e pelo usuário de alguns dos atributos específicos ao serviço.

A IN incluiu mecanismos que facilitam o fornecimento de serviços, independente da implementação destes ou da rede sobre a qual operam, tais como redes de comutação de pacotes, redes digitais de serviços integrados e redes de comutação de circuitos.

A rede IN inova na separação entre o serviço de **processamento básico de chamadas**[55] **(BCP).**" e os que requerem maior grau de *inteligência*, tradicionalmente unidos no software de controle das centrais telefônicas [7]. Esta união fazia com que o processo de introdução de novos serviços fosse dispendioso e de longa duração, pois uma nova versão deste software devia ser solicitada a cada fornecedor de equipamentos.

De acordo com o novo modelo de processamento de chamadas, o BCP possui *ganchos*[56] que permitem a sua interação com a implementação do serviço. Estes ganchos funcionam como uma válvula de escape para onde o processamento é dirigido após a detecção de que somente o BCP não é capaz de realizar a tarefa requerida na chamada.

---

[53] "Intelligent Network"

[54] O conceito de assinante e usuário é diferente do que se usa normalmente: o **assinante** é o contratante do serviço junto ao provedor e é responsável pelo pagamento de encargos junto a este e o **usuário** é quem tem acesso e efetivamente usa o serviço.

[55] "Basic Call Processing

[56] "Hooks."

2.7.1 Arquitetura Funcional

A IN é composta de quatro planos, cada um representando um refinamento maior que o precedente com respeito à implementação dos serviços:

Plano de serviço: no qual um serviço é visto como sendo composto de uma ou mais *características de serviço*[57]. A **característica de serviço** é o nível mais básico de decomposição de um serviço no Plano de Serviço. Este plano e o plano funcional global estão intimanente ligados à criação de novos serviços;

Plano funcional global (GFP): [58] neste plano, as *características de serviço* são decompostas em *blocos independentes construtores de serviço* SIB[59] que são agrupados usando-se a *lógica de serviço global (GSL)*[60] [43]. Esta lógica[61] funciona então como uma linguagem de composição de novos serviços, definindo o encadeamento, bifurcação e junção da seqüência de SIBs, desde o ponto em que a chamada deixa o SIB BCP, chamado de *ponto de início*[62] (POI), até o *ponto de retorno*[63] (POR), onde a chamada volta ao BCP (ver Figura 2.14). Como as SIB são independentes do serviço implementado, o único elemento neste plano que é dependente do serviço é a GSL;

Plano funcional distribuído (DFP): [64] os SIB são realizados como uma seqüência de *ações de entidade funcional*[65] *(FEA)* executadas em *entidades funcionais*[66] *(FE)* localizadas no DFP [45]. Uma FE é uma unidade de distribuição, pois é um grupo de funções em uma única localização que representa um subconjunto das funções necessárias para implementar um serviço. Das FEAs pode resultar uma troca de informações entre FEs, conforme mostrado na Figura 2.15.

As entidades funcionais são:

- *Agente de Controle de Chamadas*[67] *(CCAF)* que fornece acesso aos usuários, funcionando como interface entre estes e a função de controle de chamadas[68] (CCF). Ele interage com o usuário para estabalecer, manter, modificar e liberar uma chamada ou instância de serviço, também servindo como elo de comunicação entre o usuário e a CCF;
- *Função de Controle de Chamadas* é a função na rede que fornece controle e processamento de chamadas. Dentre as suas funções estão mecanismos de detecção de chamadas que requerem atividades de IN, encaminhando-as aos SSF[69];

[57] "Service features."
[58] "Global funcional plane."
[59] "Service independent building blocks."
[60] "Global Service Logic."
[61] Conjunto de sentenças condicionais e seqüenciais em alguma linguagem de programação.
[62] "Point of initiation."
[63] "Point of return."
[64] "Distributed functional plane."
[65] "Functional entity action."
[66] "Functional Entity."
[67] "Call control agent function."
[68] "Call control function."
[69] "Service switching function."

- *Função de Serviço de Dados*[70] *(SDF)* que contém dados da rede e do usuário para acesso em tempo-real;
- *Função de Controle de Serviços*[71] *(SCF)* é a função central no processamento de serviços. Ela comanda as SSF/CCFs, interage com as SDFs para obter informações referentes à implementação de um dado serviço e dados referentes ao usuário, e com as SRF para executar funções especiais, como o envio de mensagem gravada ou ativação de dispositivo de reconhecimento de voz;
- *Função de Comutação de Serviços*[72] *(SSF)* que estende a lógica do CCF permitindo-o interagir com a SCF. A SSF também gerencia a troca de sinalização entre CCF e SCF;
- *Função de Recurso Especializado*[73] *(SRF)* é a interface entre dispositivos como reconhecedores vocais, receptores de dígitos, respondedores, etc, e os SCF ou SSF;
- *Função de Gerência de Serviço*[74] *(SMF)* que gerencia o andamento dos serviços em operação. A SMF gerencia também a informação contida em SRF, SSF e CCF;
- *Função de Criação de Serviços*[75] *(SCEF)* que permite definição, desenvolvimento, testes e carregamento de novos serviços na SMF. Esta função gera a lógica do serviço, a lógica de gerência do serviço, *templates* de dados do serviço e informações de disparo do serviço;
- *Função de Acesso à Gerência de Serviços*[76] *(SMAF)* fornece a interface entre o gerente de serviços e a SMF.

Plano físico: as FEs são agrupadas em entidades físicas[77] (PE) pertencentes a este plano, e a troca de informações entre PEs é realizada através de protocolos de comunicação. O plano físico da IN é semelhante em sua concepção aos blocos físicos da arquitetura TMN. Na TMN um bloco físico também é composto de blocos funcionais, aqui chamados de entidades funcionais.

Modelo de Processamento de Chamadas e Serviços

O processamento de serviços e chamadas na IN compreende o processamento de chamadas e conexões nos SSF/CCF, a execução da lógica de serviço na SCF, e o uso de recursos de apoio como os fornecidos por SRFs e SDFs. A IN separa o processamento básico de chamadas do de serviço, mas para tal, precisou estabelecer limites nas funções de cada uma destas atividades e como a interação entre elas ocorre, ou seja, fornecer uma visão externa das funções SSF/CCF, SCF, SRF e SDF.

As principais entidades relacionadas ao modelo de execução de chamadas e serviços são CCAF, SSF/CCF e SCF. Os usuários interagem com o sistema via CCAFs, que recebem as solicitações

[70] "Service data function."
[71] "Service control function."
[72] "Service switching function."
[73] "Specialized resource function."
[74] "Service management function."
[75] "Service creation environment function."
[76] "Service management access function."
[77] "Physical entity."

de chamadas/serviços e as envia aos SSF/CCF para processamento. Caso a solicitação possa ser atendida apenas pelo serviço básico de chamadas, as SSF/CCF determinam a sua própria lógica e colocam-na em ação para atendê-lo. Caso contrário, a solicitação é encaminhada a uma SCF que, ao identificar o serviço, ativa *programas de processamento lógico de serviço (SLP)*[78] que são a forma compilada da lógica de serviço.

Para a definição de um novo serviço, é necessária a utilização de técnicas de modelagem que satisfaçam determinadas condições requeridas no modelo de processamento de chamadas e serviços. Estas condições são:

- Fornecer uma abstração de alto nível do processamento de chamadas independente de implementação de um determinado fornecedor. Através desta abstração, a interface entre SCF, SSF/CCF, CCAF, DAF e SRF pode ser especificada precisamente;
- Suportar a expressão de interações concorrentes entre entidades funcionais, como por exemplo mais de uma instância de SCF interagindo concorrentemente com um SSF/CCF em uma única chamada, para possibilitar à SSF gerenciar interações entre instâncias de lógica de serviço existentes em vários SCF ativos em uma única chamada.

  As implementações existentes até o presente momento são do tipo "single ended", ou seja, somente uma parte da chamada é independente (ortogonal) em nível de serviço e topologia com relação a outras partes da chamada. Outro requisito é o de ponto único de controle[79], ou seja, somente um SCF tem o controle em um determinado instante no tempo;
- Fornecer meios para a definição de mecanismos de disparo de serviço;
- Fornecer uma estrutura que assegure o seqüenciamento correto das funções dentro de uma SSF/CCF;

2.7.2 Modelo de Informação

A informação veiculada na IN corresponde aos dados de usuário, à lógica de implementação de serviços, aos dados resultantes de interações com dispositivos que possuem SRF e aos dados de gerência. Esta troca de informações é interpretada de acordo com o plano no qual se encontra:

Plano Funcional Global: Como neste plano um serviço é visto como uma seqüência de SIBs, as informações existentes são a GSL e as informações trocadas entre SIBs. A GSL está dividida em conjuntos, um para cada SF que compõe um novo serviço, definindo como SIBs são interligados, e qual é a informação trocada entre SIBs.

A informação veiculada entre SIBs é dividida em **dados de instância de chamada**[80] **(CID)**, que são parâmetros cujo valor muda a cada chamada, como o número do chamador, códigos de acesso, etc; e **dados de suporte ao serviço**, que são os parâmetros requeridos por um SIB, sejam eles fixos para todas as instâncias do serviço ou apontadores para dados CID;

[78] "Service Logic Processing Programs."
[79] "Single point of control."
[80] "Call instance data."

Plano Funcional Distribuído: cada SIB identificado no GFP é mapeado em uma ou mais FEs, e a GSL é fragmentada em conjuntos de **lógica de serviço distribuída**[81] **(DSL)**. É a DSL que comanda a troca de informações entre FEs e o próprio comportamento da FE de forma que esta consiga executar as funções requeridas no serviço. A principal parte da DSL está concentrada nos programas SLPs, que são carregados nas SCFs, conferindo a estas o papel central no controle da execução do serviço.

2.7.3 Modelo de Comunicação

O comportamento das **entidades físicas (PE)** depende de sua composição em termos de FEs [46]. Na composição, cada FE deve ser mapeada em uma única PE, mas uma determinada entidade física pode conter uma ou mais FEs. Instâncias duplicadas de uma FE podem ser mapeadas em entidades físicas diferentes, mas não em uma mesma PE.

A modelagem através do par <serviço, protocolo>, onde os serviços são oferecidos por entidades e os protocolos implementam as interfaces padrões no plano físico, traz para a arquitetura IN o conceito de *abstração de dados*, permitindo que entidades funcionais no Plano Funcional Distribuído possam ser realizadas no Plano Físico independendentemente de fornecedores de equipamentos ou dos serviços oferecidos.

As PEs usadas como base para a implememementação de uma IN genérica são:

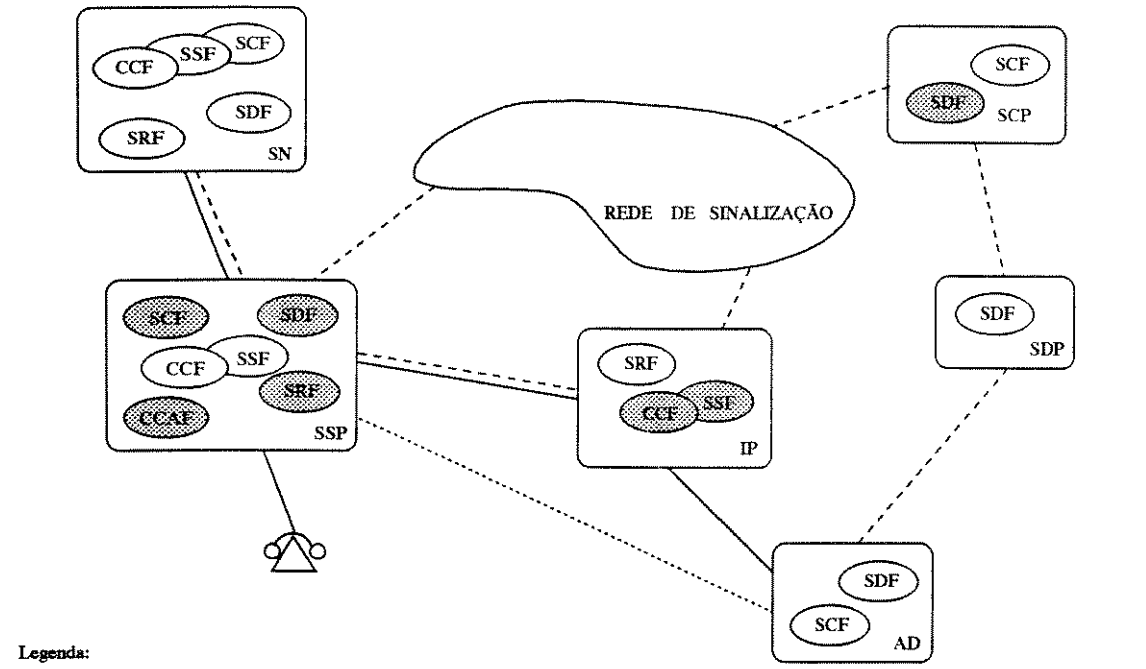
Ponto de comutação de serviços (SSP): [82] é o ponto de contato entre a rede de comutação comum e os serviços fornecidos pela IN, detectando requisições para estes serviços. Funcionalmente, um SSP contém obrigatoriamente as entidades CCF e SSF. Caso seja também uma central de comutação local, conterá um CCAF para dar acesso direto ao usuário. Também pode, opcionalmente, conter as entidades SCF, SDF e SRF (ver Figura 2.13);

Ponto de controle de serviços (SCP): [83] é o ponto onde estão localizados os programas SLPs usados para fornecer os serviços. A entidade que caracteriza este ponto é a SCF, mas opcionalmente o serviço de dados fornecido pela SDF pode ser implementado localmente. Neste caso, o acesso aos dados contidos na SDF será feito sem percorrer a rede de sinalização. A comunicação com IPs pode ser feita via rede de sinalização ou por intermediação de SSPs;

Ponto de serviço de dados (SDP): [84] contém os dados utilizados nos serviços. Funcionalmente, caracteriza-se pela entidade SDF. Os dados são acessados pelos SCPs, por outros SDPs, ou por SMPs no caso de operação de gerência;

Periférico Inteligente (IP): [85] fornece recursos adaptados às necessidades específicas dos serviços, como o reconhecimento de padrões de voz, síntese de voz, conversores de protocolo, entre outros. Caracteriza-se pela SRF, podendo conter opcionalmente SSF/CCF quando os recursos são disponibilizados para acesso externo.

[81] "Distributed service logic."
[82] "Service switching point."
[83] "Service control point."
[84] "Service data point".
[85] "Intelligent periferal."



Legenda:

Entidade funcional obrigatória — Transponte

Entidade funcional opcional ----- Sinalização ····· Transmissão em alta velocidade

ENTIDADES FUNCIONAIS

CCF - Função de controle de chamadas
CCAF - Função agente de controle de chamadas
SCF - Função de controle de serviços
SDF - Função de dados de serviço
SRF - Função de recurso especial
SSF - Função de comutação de serviços

ENTIDADES FÍSICAS

SSP - Ponto de comutação de serviços
SCP - Ponto de controle de serviços
SDP - Ponto de dados de serviço
IP - Periférico inteligente
AD - Adjunto
SN - Nó de serviço

Figura 2.13 Entidades físicas da rede inteligente (adaptado da norma Q.1205).

Durante a execução de um serviço sob o comando de um SCP, um usuário pode ser conectado aos recursos do IP mais próximo, ou seja o que está conectado ao SSP que originou o serviço, ou um IP conectado a um outro SSP;

Adjunto (AD): [86] possui a mesma funcionalidade de um SCP, mas está diretamente conectado a um SSP, por um meio de comunicação de alta velocidade, ao invés de estar conectado pela rede sinalização. As mensagens trocadas no nível de aplicação permanecem as mesmas nos dois casos;

Nó de serviço (SN): [87] pode controlar serviços e participar de interações flexíveis com o usuário. Comunica-se com outros SSPs via sinalização ponto-a-ponto e conexão de transporte. Funcionalmente, o SN contém as entidades SCF, SDF, SRF e SSF/CCF. O par SSF/CCF é tão fortemente acoplado ao SCF que torna-se inacessível a outros SCFs. O SN interage com usuários da mesma maneira que um IP, ou seja, um usuário pode ser conectado aos recursos do SN mais próximo do SSP que originou o serviço, ou a um SN conectado a um outro SSP;

Ponto de gerência de serviço (SMP): [88] executa o controle da gestão, provisão e desenvolvimento de serviços. Funcionalmente, o SMP contém a entidade SMF, e opcionalmente SCEF e SMAF;

Ponto de controle e comutação de serviço (SSCP): [89] reune as funcionalidades dos pontos SCP e SSP;

Ponto de criação de serviços (SCEP): [90] é onde os serviços são desenvolvidos e testados para posterior carregamento nos SMPs. Funcionalmente, o SCEP contém a entidade SCEF;

Ponto de acesso de gerência de serviço (SMAP): [91] é o ponto de contato entre o usuário e o SMP. Como entidade funcional, contém a SMAF.

Protocolos

A IN utiliza a rede de sinalização para realizar a troca de informações entre entidades funcionais contidas em diferentes entidades físicas. Para cada conjunto de *funcionalidades básicas*[92] são selecionados conjuntos de protocolos que satisfaçam às suas necessidades. Estes protocolos são expressos nas recomendações Q.12x5 e Q.12x8.

2.7.4 Exemplo de Serviço em Rede Inteligente

Para ilustrar como os componentes dos diversos planos que compõem a IN interagem entre si, o seguinte exemplo de um serviço fictício foi elaborado. Trata-se de um serviço de compras

[86] "Adjoint"
[87] "Service node."
[88] "Service management point."
[89] "Service switching control point."
[90] "Service creation environment point."
[91] "Service management access point"
[92] "Capability set."

SF 174		
Cadeia	SIBs	
	De	Para
A	BCP	Traduza
	Traduza	DigProd
	DigProd	ValidaProd
	ValidaProd	DadosTot
	DadosTot	BCP
B	DigProd	SolCart
	SolCart	ValCart
	ValCart	BCP

Tabela 2.2 Mapeamento de Característica de Serviço no Plano Funcional Global.

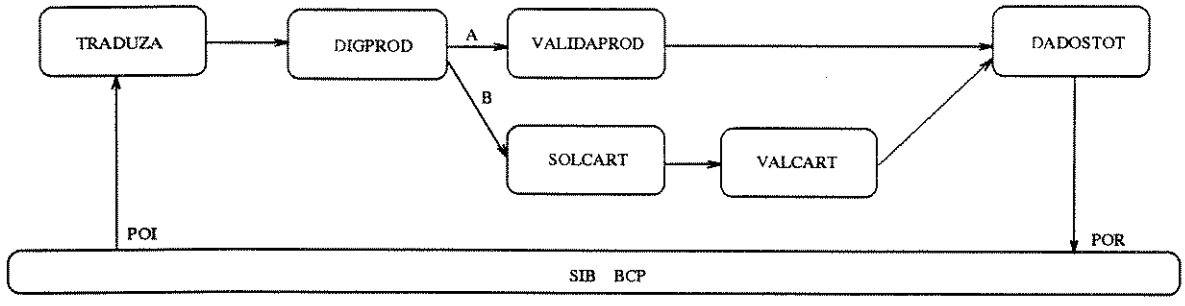


Figura 2.14 Encadeamento de Blocos de Serviço (SIB) segundo lógica contida na Característica de Serviço (SF).

à distância com cartão de crédito. Nele, a função da IN é verificar para onde encaminhar a chamada, de acordo com a localização do usuário e com o horário desta, validar o seu número de cartão de crédito junto a uma operadora de cartões, verificar a disposição do produto junto ao fornecedor em termos de código e quantidade desejados e enviar um conjunto de informações globais resultantes ao fornecedor.

O serviço de compras é composto de apenas uma *característica de serviço*, a SF 174, cujo mapeamento para o Plano Funcional Global em termos do seqüenciamento de SIBs é mostrado na Tabela 2.2 e representado graficamente na Figura 2.14[93].

No mapeamento entre o Plano Funcional Global e o Plano Funcional Distribuído, os SIBs são decompostos nas entidades funcionais SSF, SCF, SDF e SRF e troca de informações entre estas, conforme mostrado na Figura 2.15.

O funcionamento do serviço é descrito a seguir:

1. O usuário disca o código de acesso ao serviço (0800) seguido do número do fornecedor[94] (123456);

[93] A representação dos SIBs desta figura não satisfazem à padronização da recomendação Q.1218.
[94] Assinante na nomenclatura IN.

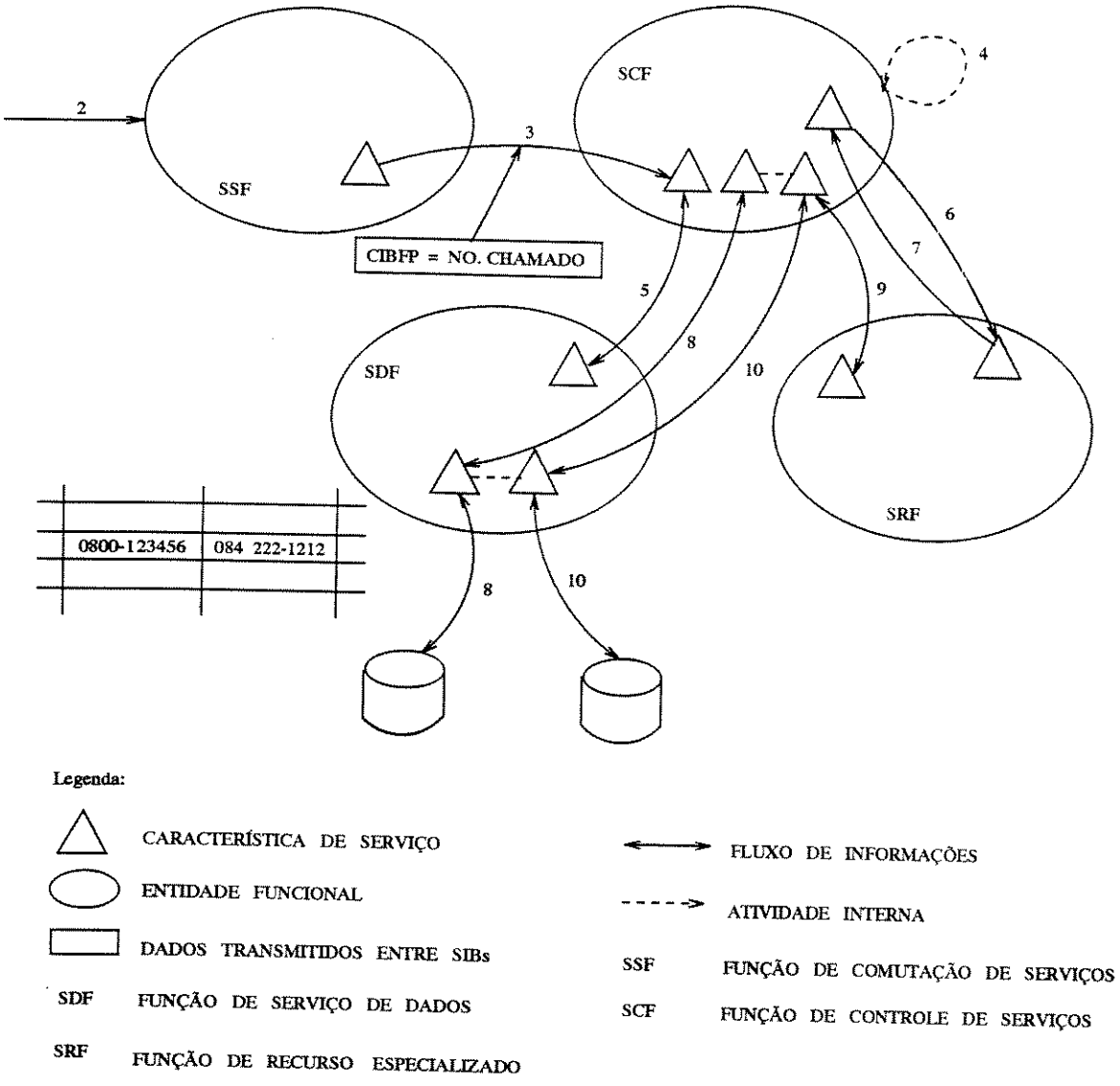


Figura 2.15 Mapeamento do serviço de compras por cartão de crédito no Plano Funcional Distribuído.

2. A chamada é encaminhada ao SSP mais próximo pela rede pública de comutação, ativando a SSF nele contida;

3. A SSF detecta que o serviço requer facilidades IN e sinaliza o evento a uma SCF, enviando-lhe os dados do usuário (chamador) e do fornecedor (chamado);

4. A SCF carrega a lógica do serviço solicitado, que é composta de uma única característica de serviço[95], a SF 174 (ver Tabela 2.2);

5. A SCF consulta a SDF traduzindo o número do fornecedor, obtendo o número de destino final da chamada, que é o (084)2221212;

6. A SRF é ativada para solicitar ao usuário o código do produto e a quantidade desejada. A solicitação é feita através do envio de uma mensagem gravada;

7. Os dados são convertidos e enviados à SCF;

8. Os dados do produto são validados no banco de dados do cliente[96];

9. Simultaneamente à etapa (8), os dados do cartão de crédito são solicitados ao usuário, juntamente com o reconhecimento vocal da senha;

10. Os dados do cartão são enviados com a senha vocal para validação no banco de dados da operadora de cartões de créditos;

11. Os dados completos são enviados ao fornecedor.

2.8 A unificação TMN-IN

Os modelos de gerência OSI, Internet e a Arquitetura TMN têm vários pontos em comum: todos são baseados em um serviço de informação de gerência que oferece primitivas para monitoramento, controle e envio de relatório de eventos e definem gerentes e agentes como usuários deste serviço.

O agente é o que possui severas restrições do ponto de vista de desempenho, pois deve receber ações provenientes dos gerentes, monitorar e controlar os dispositivos, enviar relatórios aos gerentes e ainda manter a consistência da MIB, cujos objetos devem apresentar um estado fiel dos dispositivos que representam. (Figura 2.16). Na arquitetura funcional da INMF, a função de agente é exclusiva, o mesmo ocorrendo com o gerente. A partição de domínios de gerência é realizada através do conceito de **comunidade**, definida como uma relação entre um agente e um ou mais gerentes.

Na OSI-MF, o ambiente de gerência é particionado em domínios correspondentes às funções de segurança, contabilidade, falhas, desempenho e configuração, ou seja, tem-se uma partição de domínio de acordo com o critério funcional. Dentro de cada domínio, uma aplicação de gerência pode assumir ora papel de gerente, ora de agente, mas sempre em associações de gerência distintas. A duplicidade de papéis implica em uma maior restrição no desempenho da aplicação.

[95] Neste plano, o Funcional Distribuído, não existem SFs. O que a SCF efetivamente carrega é o Programa de Processamento Lógico de Serviço, mas a idéia aqui é mostrar o mapeamento entre planos.

[96] A IN não armazena dados de assinantes, contudo ela fornece os meios para que estes dados sejam acessados indiretamente através da SDF.

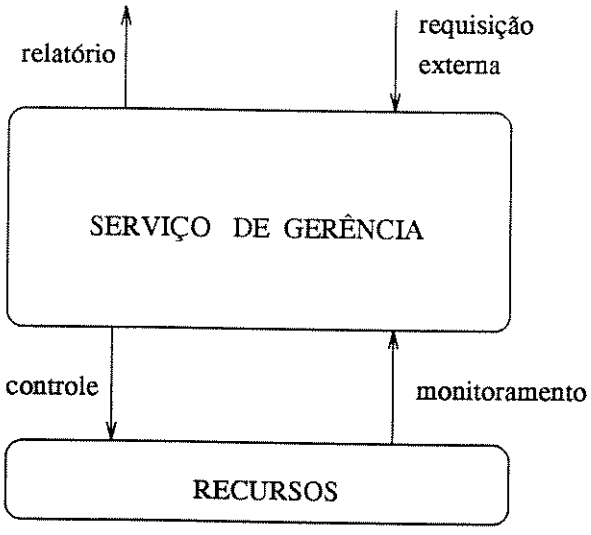


Figura 2.16 O serviço de gerência

Enquanto na OSI-MF e na INMF a partição de domínios é plana, esta adquire espacialidade na TMN através da LLA. Define-se um domínio OSF para cada uma das camadas da LLA, ou seja, para cada camada existe partição funcional. Esta OSF comanda hierarquicamente agentes e sub-domínios OSFs. Na Figura 2.12 pode-se visualizar uma interação entre OSFs de camada de serviço. O OSF de serviço do PNO comanda então o OSF de rede que, por sua vez, comanda o agente NEF. O OSF de rede realiza e atua como gerente com relação ao OSF de serviço e como gerente do NEF. Portanto, na TMN existe a mesma sobrecarga causada pela duplicidade de papéis existentes na OSI-MF.

Um outro aspecto que se deve considerar na TMN é a composição física de seus blocos constituintes, pois um NE pode ser composto de um gerente (OSF), um agente (NEF), um adaptador (QAF) e ainda realizar a função de estação de trabalho (WSF). Para melhorar o desempenho global do sistema, o ambiente de execução deve prover alguma forma de concorrência, para que quando um bloco funcional estiver realizando uma atividade de entrada ou saída, os demais blocos funcionais possam ocupar o processador que lhe estava alocado.

As configurações de pior e de melhor desempenho dos casos OSI e TMN são ilustradas na Figura 2.17. O objetivo é maximizar a utilização dos recursos do ambiente de execução, como processadores e dispositivos de E/S.

Mesmo estando separados do ponto de vista da infra-estrutura, estes blocos ainda são uma entidade única do ponto de vista de sistema aberto. Por este motivo não há necessidade de implementar toda a pilha OSI de sete camadas para fazer a comunicação entre o agente e o gerente. A arquitetura TMN requer que o ponto de referência seja preservado, ou seja, que a fronteira entre dois blocos funcionais esteja claramente delimitada, mas não requer a implementação da interface, o que implicaria no uso de protocolos. Portanto, nos dois casos, a comunicação pode ser feita com a simples troca de mensagem entre as entidades concorrentes.

Neste ponto é conveniente fazer a distinção entre distribuição lógica e física [9]: um sistema de software logicamente distribuído consiste em múltiplos processos que se comunicam através da troca explícita de mensagens. Em contraste, num sistema de software logicamente não-distribuído,

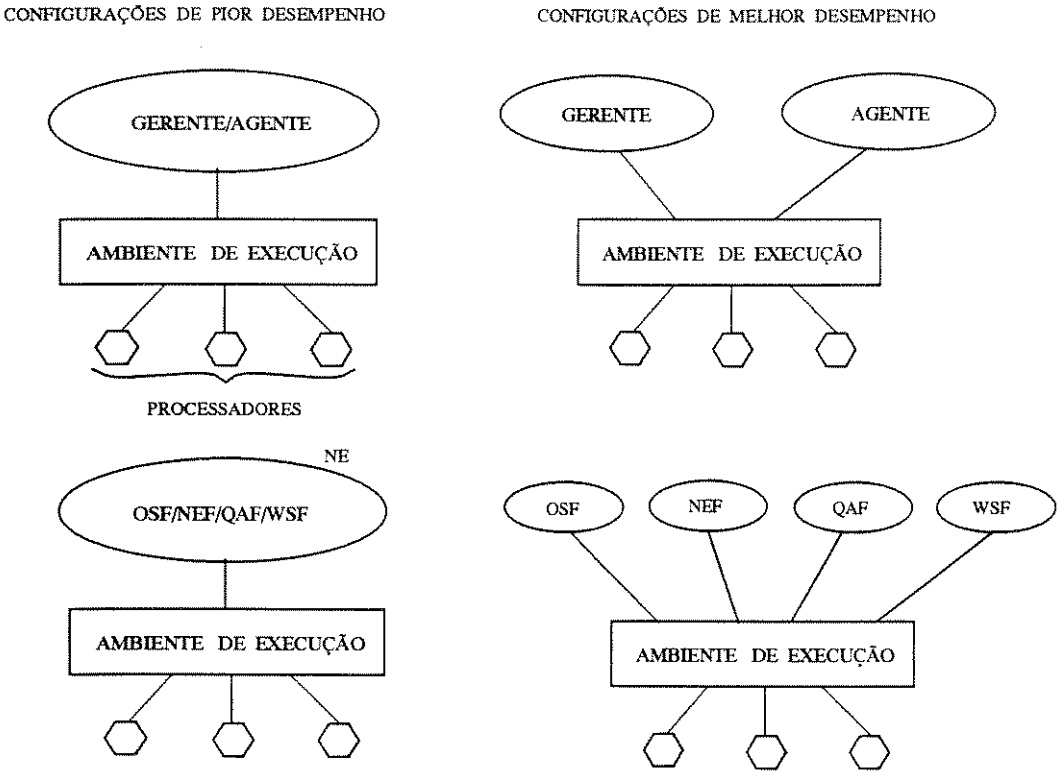


Figura 2.17 Configurações de melhor e pior desempenho

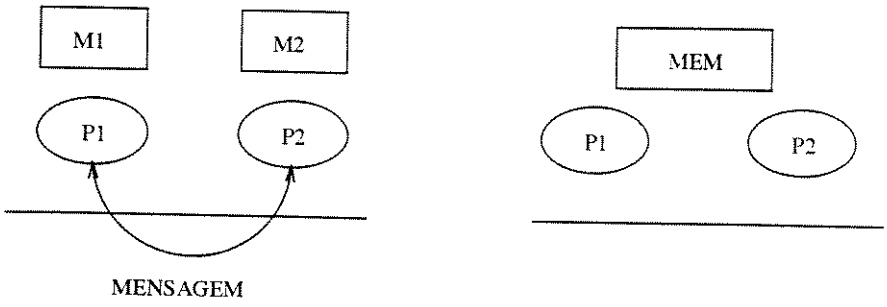


Figura 2.18 Arquiteturas fisicamente distribuída e com compartilhamento de memória

os processos se comunicam através do compartilhamento de memória.

De forma similar, a plataforma de "hardware" é fisicamente distribuída, caso não exista compartilhamento da memória entre os processadores que a compõe, em oposição às fisicamente não-distribuídas que possuem uma memória única, conforme a Figura 2.18.

Vale salientar que uma plataforma de "hardware" fisicamente distribuída não impede a implementação de um sistema logicamente não-distribuído, o mesmo ocorrendo na combinação de "hardware" fisicamente não-distribuído com sistema de software logicamente distribuído. De fato, as quatro combinações possíveis são todas viáveis.

Na arquitetura TMN como visto na Seção 2.6.3, um bloco constituído resulta da composição de vários blocos funcionais preservando o ponto de referência que delimita a fronteira de serviço fornecida por cada bloco funcional. Esta fronteira, ao identificar a informação que passa através dela, permite que a realização física dos blocos constituintes possa ser concretizada na forma de um sistema de software logicamente distribuído, dado que neste sistema a troca de informações se dá através de troca de mensagens e, portanto, pode ser precisamente identificada.

2.8.1 Introdução de Concorrência no Acesso à MIB

O desempenho do agente é muito importante quando se pretende unificar gestão e controle, pois as atividades de controle requerem um tempo de resposta muito mais curto. Em redes de alta velocidade, como ATM, por exemplo, o monitoramento das filas internas dos comutadores é seguido de perto, pois ao serem atingidos determinados níveis de ocupação das filas, tenta-se controlar a injeção de dados na **Interface usuário-rede**[97] **(UNI)**, advertindo-se um NE ou Q.A.

É provável que tal elemento seja implementado em "hardware" para que possa responder prontamente, antes que a situação se torne crítica. Na presença de uma situação de controle de congestão, o gerente do nó afetado pode decidir quais aplicativos que utilizam a conexão são os mais importantes, e instruir o sistema operacional para priorizá-los, garantindo a QoS destes, ou ainda decidir pelo descarte de mensagens.

A separação em entidades com processamento concorrente melhora o desempenho na OSI-MF e TMN, mas em nada auxilia a INMF. Para que isto ocorra, é preciso que se introduza concorrência não apenas entre as entidades, mas dentro da própria entidade. É conveniente então, uma análise de seu funcionamento interno. Uma representação interna do agente da Figura 2.16 é mostrada na Figura 2.19.

[97] User-Network Interface

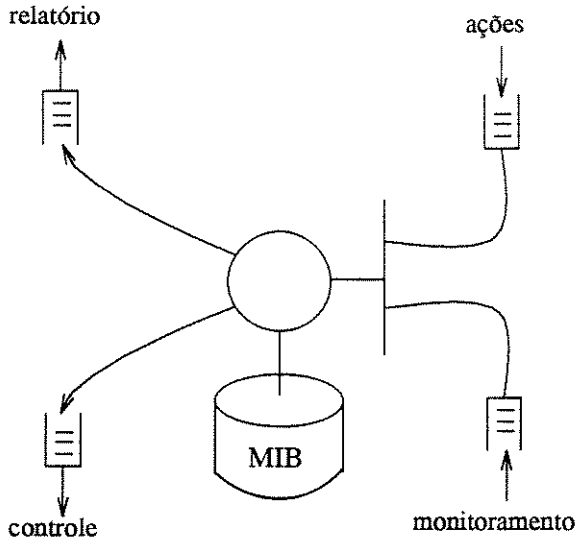


Figura 2.19 Representação interna do agente.

Qualquer um dos eventos existentes nas filas corresponde a uma ou mais operações em objetos MIB. Assim as atividades básicas do servidor são a execução de procedimentos e o envio de mensagens.

Seja um conjunto de s servidores dividindo a capacidade np de processamento do ambiente de execução, onde n é o número de processadores disponíveis e p é a capacidade de processamento de cada processador efetivamente fornecida pelo sistema operacional. Se esta capacidade de processamento é alocada aos servidores por uma estrutura de controle, como no caso de um servidor único, o máximo que se pode utilizar é p, enquanto que para múltiplos servidores é de aproximadamente np[98].

Quando há apenas um processador disponível, a estrutura de controle ainda consome uma parte da capacidade efetiva fornecida, mas esta parte é tanto menor quanto maior for a parcela de I/O realizada durante o processamento de cada serviço, podendo inclusive apresentar ganho de desempenho em relação à situação sem concorrência.

2.8.2 Classificação dos Dados da MIB

Um agente pode ser implementado como um sistema com múltiplos servidores concorrentes conforme ilustrado na Figura 2.20. Cada um dos servidores concorrentes existentes no agente executa um ou mais acessos à MIB, que passa então a ser vista como um conjunto de objetos, compartilhado pelos servidores.

A introdução de concorrência no acesso à MIB dá origem a duas questões: como manter a consistência da MIB, e têm todos os acessos a mesma prioridade de execução?

Para responder a estas questões, é necessário estudar a semântica dos dados que compõem a MIB. Grosso modo, os dados de gerência podem ser classificados em três tipos: dados de monitorização, dados estruturais e dados de controle. De fato, de acordo com o OSI-MF, um objeto pode

[98] Uma pequena parcela é consumida pelo controle.

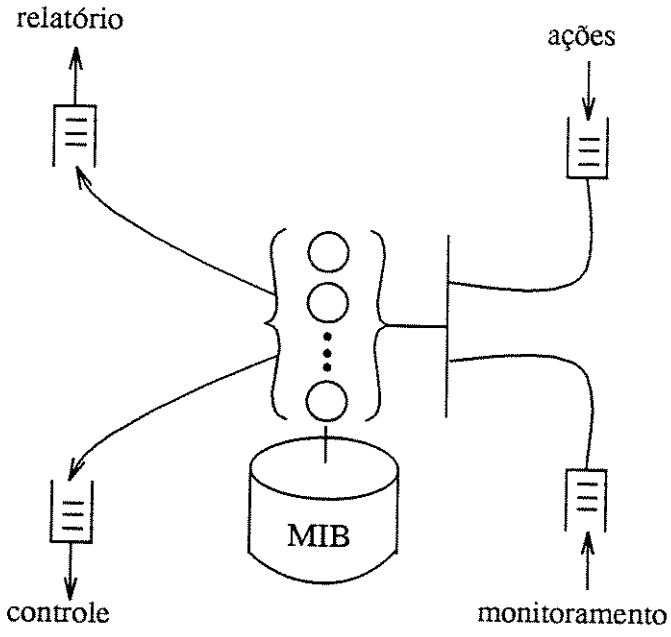


Figura 2.20 Sistema com múltiplos servidores.

ser definido como sendo composto deste três tipos de dados. Por exemplo: um objeto que representa um circuito pode ter um atributo NÚMERO_DE_CIRCUITO_VIRTUAL (dado estrutural), uma ação ACTIVATE (dado de controle) e uma notificação DESCONEXÃO_DE_CIRCUITO (dado de monitorização). Contudo, esta é uma visão lógica do objeto, e não a sua implementação, que é o aspecto que importa nestas questões:

Dado Sensorial: é o dado que é recebido pelo processo de monitoramento. Pode ser esporádico, como uma notificação de falha, ou periódico, como o resultado de uma operação de sondagem executada sobre algum dispositivo para avaliar o seu desempenho;

Dado de Controle: é o resultado de uma ação tomada pelo gerente de rede, ou por um processo automatizado de decisão após analisar a informação sensorial. Exemplificando, se um invasor é detectado em um determinado nó, uma ação pode ter sido programada para cortar os enlaces de comunicação daquele nó, ou simplesmente desativá-lo;

Dado Estrutural: é composto de informações que não se alteram freqüentemente, como a capacidade de um enlace, sua relação sinal-ruído, e o seu estado (ativo ou inativo).

A implementação do serviço de informação de gerência pelo agente define a classificação dos dados visando a implementação da MIB. Conseqüentemente, existe uma relação entre as primitivas dos protocolos de gerência e esta classificação. Os *dados sensoriais esporádicos* estão associados no CMIS/CMIP à criação de um discriminador de eventos, através da primitiva **create** e do envio de **event-aport** do agente para o gerente, sempre que o evento detectado satisfaz às condições de envio estabelecidas. No SNMP, esses dados geram um "trap" que é dirigido aos gerentes cadastrados como receptores de "traps" quando o sistema foi inicializado. Após a recepção do "trap", o gerente solicita dados precisos sobre o evento, enviando um pedido "get".

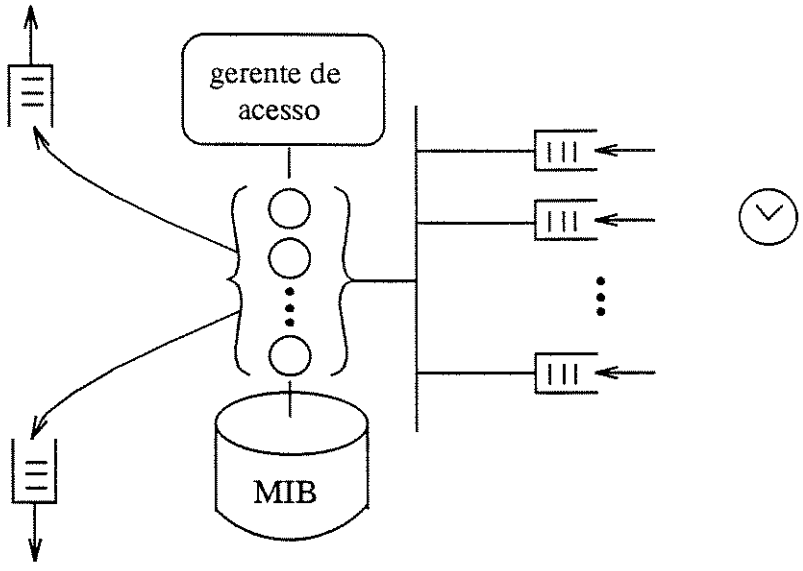


Figura 2.21 Eventos de sondagem disparados por um temporizador.

Os dados de controle estão associados às primitivas **action** e **set** nos protocolos CMIP e SNMP, respectivamente. Para os dados estruturais as operações são **get** e **set** nos dois protocolos.

A partir desta classificação, é possível verificar que existem diferentes graus de prioridade entre as atividades executadas na MIB. Por exemplo, um evento relacionado à atualização de dados de sondagem pode ser considerada de menor prioridade do que uma notificação de falha, pois no segundo tem-se uma situação concreta que necessita de uma ação, enquanto que no primeiro, o resultado provável é uma situação de regime.

Os dados sensoriais periódicos requerem a alocação de um servidor a cada período τ no qual devem repetir a sua atividade. Com a adoção do escalonamento em graus de prioridades, no qual os eventos de sondagem são tratados com precedência, pode-se modelar a periodicidade da sondagem através de mais uma fila, cujos eventos são disparados por um temporizador. (Ver Figura 2.21.)

A consistência da MIB pode ser mantida através de um gerenciador de acesso. Antes de realizar uma operação, um servidor solicita deste gerenciador o acesso ao objeto, e somente após a obtenção, a operação é concretizada.

2.8.3 Concorrência nas Unidades Funcionais da Rede Inteligente

O provimento de uma arquitetura única de gerência e controle para as INs implica na consideração de dois aspectos: gerência de serviços e redes, e processamento dos serviços. Para a gerência de redes e serviços são considerados os pontos de vista do assinante e do operador de rede. Do ponto de vista do assinante, deve-se fornecer uma interface para que este possa ativar, desativar, adaptar, obter relatórios de tarifação e monitorar os aspectos relativos à qualidade do serviço, segurança, atribuições, etc.

Ao operador de rede deve ser fornecido um ambiente, permitindo-o carregar as lógicas de serviço e de gerência de serviço nos elementos de rede apropriados, ativar estas lógicas, validar os

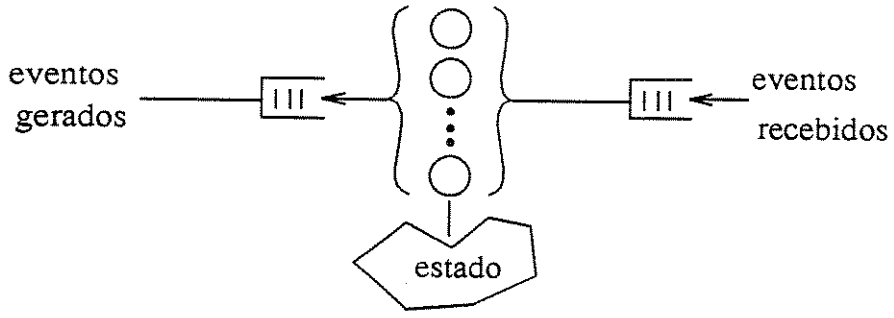


Figura 2.22 Modelo de funcionamento de uma FE.

dados de configuração modificados pelo usuário e realizar a gerência de rede propriamente dita. Tanto do ponto de vista do assinante, quanto do operador de rede, deve ser provida uma interface para a gerência de criação de serviços. Este ambiente pode ser construído utilizando-se o modelo TMN, através das camadas de serviço e rede da LLA.

A partir do Modelo de Processamento de Chamadas e Serviços, podem ser extraídos alguns requisitos que devem ser satisfeitos pelo ambiente no qual os serviços são executados:

- Os SLPs são ativados nos SCF mediante eventos de disparo proveniente dos ganchos existentes no processamento básico BCP. Para que a QoS não seja prejudicada, deve haver um grau de paralelismo neste processamento compatível com a demanda. Isto significa que os SCFs devem ativar várias FEAs concorrentemente, ou seja, que exista concorrência internamente num SCF e nas demais FEs;

- O agrupamento de entidades funcionais em uma mesma entidade física e a troca de informações entre estas entidades implicam na necessidade de um mecanismo de troca de informações interno à entidade física, sem ter de recorrer obrigatoriamente a um protocolo de comunicação. Isto é especialmente importante no caso de SNs, onde entidades SSF/CCF e SCF estão fortemente acopladas, fazendo com que as primeiras não sejam acessíveis de fora do SN;

- Deve haver suporte às interações concorrentes entre entidades funcionais, como entre vários SSFs e um SCF e vice-versa.

O primeiro destes requisitos leva a uma solução semelhante a da introdução de concorrência dentro do agente, no qual a MIB é partilhada pelos servidores. Neste caso, porém, os dados armazenados são os correspondentes aos serviços em processamento e os servidores, ao invés de executarem métodos em objetos da MIB, executam FEA nas entidades funcionais. O modelo resultante para a FE está ilustrado na Figura 2.22.

O comportamento de FEs pode ser modelado através de Máquinas de Estados Finitos [45], pois esta técnica permite: a abstração das funções de processamento de chamadas ou serviços de sua implementação; uma visão observável de uma entidade com relação a uma outra; a definição de fluxo de informações entre entidades e uma estrutura adequada para a garantia do seqüenciamento de funções em uma entidade.

O estado da entidade modelada através da máquina de estados finitos é análogo à MIB do agente, pois ele é acessado concorrentemente pelos servidores existentes nessa entidade. Como estão tratando instâncias de serviços, este estado é de fato um conjunto dos estados de cada instância. A cada novo evento, tendo como referência uma dada instância, se o evento é válido, uma transição entre estados é efetivada e um conjunto de ações é realizado.

O segundo requisito exige que os servidores, ao executar FEAs, possam também enviar mensagens externas à entidade, ou seja, direcionadas a outras entidades que estão localizadas na mesma PE.

O terceiro requisito é satisfeito estendendo-se o modelo de concorrência do SCF para uma entidade qualquer, tornando-a capaz de ativar múltiplas instâncias de FEAs concorrentemente, pois as mensagens enviadas ou recebidas pelos servidores executando estas FEAs resultam em troca de informações concorrentes entre entidades.

2.8.4 Requisitos à Unificação de Gerência e Controle em Redes de Alta Velocidade

Os requisitos identificados nas seções precedentes para a gerência de redes de alta velocidade de acordo com os modelos OSI-MF, INMF e TMN, juntamente com a gerência e controle de redes inteligentes, não consideram ações *fortes de tempo real*[99], como as realizadas com o intuito de controle de fluxo, roteamento, controle de admissão de chamadas, etc. O que se procura é a redução do tempo de processamento de requisições, de forma que as ações de gerência sejam realizadas o mais rapidamente possível[100], sem a existência de um prazo máximo[101]

Sumariando os requisitos identificados para a gerência de redes de alta velocidade, obtém-se:

1. Introdução de mecanismos de concorrência internos às entidades, sejam elas blocos funcionais TMN, agentes SNMP ou entidades funcionais IN. Com isto, evita-se a seriação das requisições que lhes são dirigidas;

2. Capacidade de estabelecer diferentes níveis de prioridades no processamento das atividades sendo processados concorrentemente;

3. Fornecimento de uma estrutura padrão de protocolos para que as entidades possam comunicar-se através da rede de comunicações;

4. Provimento de meios para que dados possam ser compartilhados pelas atividades concorrentes mencionadas e gerenciar o acesso a estes dados para manter sua consistência;

5. Disponibilização de mecanismos de comunicação para a troca de informação entre entidades localizadas em uma mesma máquina.

[99] "Hard real-time."
[100] "Soft real-time."
[101] "Deadline."

Capítulo 3

Projeto da Plataforma de Suporte

No Capítulo 2 foram estudados os modelos de gerência mais comumente aplicados a redes de computadores e as implicações do emprego destes a redes de alta velocidade, considerando-se a tendência de unificação entre gerência e controle em uma plataforma única. Esta tendência tem-se mostrado bastante forte, principalmente na implementação de serviços em Redes Inteligentes, que através do reuso de software na forma de **Características de Serviços** ou **blocos construtores independentes** e da separação das funções de comutação básicas das especialidades ao controle dos serviços, tende a disponibilizar os serviços em um intervalo de tempo cada vez menor.

Baseados nos modelos OSI-MF, INMF, TMN e IN, elaborou-se uma lista de requisitos a serem satisfeitos por um ambiente de suporte que pretenda fornecer um suporte para a unificação destes modelos.

Neste capítulo são avaliadas algumas alternativas de projeto para este ambiente de suporte, considerando-se o estado da arte em termos de paradigmas de computação, os avanços tecnológicos na área de sistemas operacionais e o impacto da adaptação da planta software já existente ao atendimento dos requisitos.

3.1 Software existente: refazer ou aproveitar?

As palavras chaves que podem ser extraídas dos requisitos são: concorrência, comunicação, prioridades, sincronismo e compartilhamento de dados. Para atender a estes requisitos, o ambiente de suporte pode ser concebido de duas formas: através do uso de uma linguagem de programação ou da elaboração de uma plataforma de suporte. A primeira alternativa implica na reconstrução de toda a base computacional existente em termos de software de gerência e controle nesta nova linguagem. Para se ter uma idéia do que isto representa, os ambientes ISODE e OSIMIS, que fornecem conjuntos de bilbiotecas e compiladores para a construção de aplicações OSI em ambiente Internet, e de aplicações de gerência OSI e Internet, respectivamente, representam um esforço de programação de centenas de milhares de linhas de código em C e C++. Uma vantagem da utilização de uma linguagem é que esta pode verificar, em tempo de compilação, inconsistências no programa.

A segunda alternativa apresenta maior flexibilidade, pois permite que aplicações distribuídas possam ser implementadas a partir de linguagens de programação comuns. Esta abordagem tem sido usada com sucesso tanto para a construção de aplicativos de gerência de redes [18], quanto

na resolução distribuída de problemas em Inteligência Artificial [15]. Esta foi a forma adotada para o ambiente de suporte, juntamente com a linguagem C++.

Concorrência, comunicação, sincronismo, prioridades e compartilhamento de dados são conceitos metalinguísticos subjacentes a linguagens de programação de sistemas concorrentes. Portanto, para ilustrar cada um deles, serão fornecidos exemplos de sua incorporação em algumas linguagens. Onde houver introdução da sintática e semântica associadas a estas linguagens, isto será feito da forma mais breve possível para não sobrecarregar o texto. Em particular, o compartilhamento de dados é uma forma de interação entre elementos de um sistema concorrente e o sincronismo entre estas entidades está intimamente relacionado com a sua forma de interação.

3.2 Concorrência

A plataforma de suporte deve prover algum meio para que o programador defina o grau de concorrência que deseja introduzir nas aplicações. Para tal, é preciso definir que *unidade de paralelismo* a plataforma fornece. A seguir, é discutida a utilização de processos, funções, sentenças e objetos, como unidades de paralelismo.

A associação de concorrência com linguagens procedurais[1] modernas, como C, Pascal, Fortran, APL, etc., traz à tona um problema estrutural comum a todas elas: *o gargalo de von Neumann*, uma expressão criada por John Backus em 1978.

O modelo mais simples de computador de von Neumann é composto de CPU, memória e um barramento que as interconecta. Um programa para esta arquitetura opera através da transferência de dados entre a CPU e a memória, através deste barramento. Como a memória contém tanto dados quanto programa em um espaço de endereçamento global, cria-se um gargalo, pois as instruções de programa modificam o conteúdo da memória durante a execução e a própria seqüência de execução também requer acesso à memória para obter a próxima instrução a ser executada.

O espaço de endereçamento global com programa e dados e a execução do programa através da modificação deste espaço associa a semântica do modelo ao conceito de *estado global*, cujas transações são realizadas por meio das instruções contidas no programa. Este estado global torna a análise semântica dos programas uma tarefa bastante difícil, tanto do ponto de vista denotacional quanto dedutivo[2].

Lingüisticamente o gargalo de von Neumann está associado ao operador de atribuição (:=), pois é ele que relaciona a atualização do estado, representado no lado esquerdo da expressão, com uma expressão contida no lado direito. Todas as demais estruturas lingüísticas existentes, como **laços**, **subscritos** e **desdobramentos** servem apenas para realizar um conjunto de operações baseadas em atribuição [6].

Duas abordagens em termos arquiteturais para acelerar o processamento em processadores "von Neumann" são os dutos de informações[3] e os processadores vetoriais [2]. Na primeira são utilizados processadores auxiliares que acessam a memória e decodificam as instruções sempre

[1] Linguagem em que o programador deve descrever tanto a estrutura da informação quanto os procedimentos de controle do programa [68].

[2] Na semântica denotacional o sistema é observado como uma função do estado inicial para o estado final, enquanto que na dedutiva, artefatos lógicos são usados para provar propriedades entre o programa [66].

[3] pipelines

à frente das unidades de execução, tentando otimizar a execução destas. Como ainda se deve manter a seqüência lógica das operações, a concorrência é explorada apenas em uma pequena janela da seqüência de controle.

Os processadores vetoriais realizam operações em vetores internos de uma só vez, evitando a sobrecarga imposta pelas operações repetitivas realizadas em cada um de seus elementos. Seu desempenho é limitado apenas pela largura de banda de memória dos processadores vetoriais e, evidentemente, do percentual do código que pode ser vetorizado, o que pode variar de dez a noventa por cento em aplicações científicas.

Uma terceira abordagem, ainda no modelo "von Neumann", é o compartilhamento de memória entre vários processadores. Neste modelo, os vários "threads" que executam em paralelo devem ser sincronizados para manter a consistência da memória, e o desempenho será tanto maior quanto menor for o grau de sincronismo necessário no programa.

Outra proposta para evitar a proliferação da dependência de dados criada é a programação funcional, na qual a exclusão de atribuições a valores armazenados permite a possibilidade de avaliação concorrente de expressões em um programa. Infelizmente, a programação funcional não satisfaz os casos onde o comportamento seja sensível à história do sistema. Para contorná-lo, a arquitetura "data-flow" inclui o conceito de retroalimentação do fluxo de dados em nós com comportamento funcional.

3.2.1 Processos Seqüenciais

Operacionalmente, um processo pode ser visto como uma máquina de estados no sentido de que ele reage a eventos em certos pontos de detecção denominados *entradas*, executa uma seqüência de operações em seu estado interno e pode produzir *saídas* como resposta aos eventos de entrada.

Ao definir um sistema como um conjunto de processos, estabelece-se que unidades devem ser executadas em paralelo, embora dentro de cada processo a execução seja seqüencial.

Exemplos de linguagens que utilizam processos são CCS (Calculus of Communicating Systems) [61], CSP (Communicating Sequential Processes) [38], LOTOS [12], Concurrent C [20] e Ada [64].

As linguagens CCS, CSP e LOTOS, usadas para descrever sistemas como processos concorrentes, fazem parte do grupo de técnicas formais de especificação das *álgebras de processos* [54]. Neste grupo, um sistema é descrito através do conjunto de transições entre estados do modelo construído para representá-lo. O estado permanece invisível ao exterior do processo, que pode apenas inferi-lo através das transições, sendo estas deferidas como uma relação ternária entre [14]: o estado *inicial* de um processo; uma seqüência descrevendo suas interações com o ambiente *durante* a sua execução; um estado possível do processo *após* estas interações.

Concurrent C engloba a linguagem C, provendo recursos de programação paralela tanto em sistemas com processadores fracamente acoplados, como em uma rede local, quanto em arquiteturas paralelas fortemente acopladas com compartilhamento de memória[4]. Um processo UNIX pode contar com um ou mais processos no conceito de C concorrente. Estes são ativados periodicamente por um escalonador pertencente ao ambiente de execução que realiza uma espécie de troca de contexto local.

A criação de processos pode ser feita implicitamente ou explicitamente. Na relação implícita,

[4] Multiprocessador na nomenclatura adotada em[9].

define-se um *tipo de processo* e, a partir deste, criam-se as instâncias pela declaração de variáveis daquele tipo. Na explícita, uma primitiva (**fork, create, cobegin,...**) é usada especificamente para este fim, criando um processo a cada invocação e, geralmente, permitindo a passagem de parâmetros para o novo processo.

Processos também podem ser sincronizados em pontos determinados do processamento, permitindo a um dado processo esperar o término de um outro em andamento, e do qual conheça o identificador. Esta espera é feita através de uma primitiva de sincronização, como o **join** ou **coend**.

O par **fork-join** é considerado [31] de baixo nível em termos de primitivas de linguagem de programação de sistemas distribuídos, enquanto que o par **cobegin-coend** é considerado de alto nível. Isto deve-se ao fato de que o uso de **fork-join** pode ser feito de forma indisciplinada dentro do programa tornando a sua visibilidade tão difícil quanto a utilização de "go-to's", enquanto que **cobegin-coend** abraçam um conjunto de declarações de processos a serem executados concorrentemente, formando, portanto, um par estruturado.

3.2.2 Funções com Avaliação Retardada de Parâmetros

Diferentemente dos processos seqüênciais que guardam um estado interno, modificado por transições internas ou externas, as funções não possuem estado associado, apenas transformam os valores presentes nas entradas. Isto faz com que não possuam *memória*, sendo, portanto, insensíveis à história do sistema.

Como se trata de funções que operam sobre valores, o termo **modelo aplicativo** é geralmente utilizado como sinônimo de **funcional**, e suas linguagens associadas são referenciadas como **linguagens aplicativas** [1]. Em termos históricos o conceito de avaliação aplicativa de expressões remonta à invenção do λ-cálculo em 1941 [17, 1], e linguagens como Lisp pura[5] cuja base é o λ-cálculo, são precursoras das linguagens aplicativas que culminaram no desenvolvimento da arquitetura "data-flow" de computadores.

Os processadores "data-flow" são computadores de programa armazenado, no qual o programa é uma representação de grafos de fluxo de dados [2]. Além de incorporar propriedades do modelo aplicativo como a ausência de efeitos colaterais herdados do λ-cálculo e da ausência de estado interno, a arquitetura "data-flow" exibe ainda [1]:

- Equivalência entre restrições de escalonamento de instruções e dependência dos dados, ou seja, toda a informação necessária para executar um programa está contida em seu grupo de fluxo de dados;

- Uma variável só pode aparecer uma vez do lado esquerdo de uma atribuição na área de programa na qual está ativa, ou seja, não são permitidas expressões do tipo $I = I + 1$ que são algebricamente incorretas;

- Notações incomuns para iterações causadas pelo item precedente. No caso da linguagem Id [1], cria-se uma variável com o mesmo nome do anterior e se lhe atribui o novo valor: $newJ \leftarrow J + 1$.

[5] As funções que causam efeitos colaterais, como RPLACA e RPLACD são eliminados [1].

O mecanismo usado para introduzir concorrência é chamado **avaliação retardada de expressões** [34], no qual define-se quais parâmetros devem ser avaliados em um dado instante e quais podem esperar o momento em que sejam necessários. A avaliação retardada é especialmente útil no processamento de listas, pois pode-se definir que apenas a cabeça da lista seja processada, postergando o restante desta. Conseqüentemente, listas virtualmente infinitas podem ser manipuladas desde que se tenha o cuidado de explorar apenas uma parte finita delas.

Funções com Retroalimentação

Dado que as funções não possuem estado interno, estas não são capazes de realizar operações que contenham a história anterior do sistema, como por exemplo, contar o número de partículas que chegam a um contador, pois este depende do número do último processamento. Para contornar este problema, criaram-se as retroalimentações. Através destas, um dado de saída anterior é colocado novamente como um dos parâmetros de entrada de uma função.

Exemplos de linguagem de processamento paralelo de funções são Id e VAL, adaptadas mais especificamente à arquitetura "data-flow" de computadores paralelos [1]. Um exemplo de linguagem aplicativa não ligada à arquitetura "data-flow" é FP,[6] para a qual se desenvolveu [6] uma *álgebra de programas*, cujas variáveis são programas funcionais e cujas operações são combinações destes resultando em **formas funcionais**. Através da álgebra de programas, é possível demonstrar-se que uma dada função possui o comportamento para o qual foi projetada.

3.2.3 Sentenças Paralelas

Das formas de paralelismo vistas até agora, esta é a forma de granularidade mais fina, pois dentro de um processo, podem existir várias funções e cada função é composta de sentenças que são paralelizáveis ou não. Sentenças paralelas são geralmente adequadas a arquiteturas com processadores vetoriais ou transputers, como Occam [59], onde a expressão de sentenças a serem seriadas é realizada pelo comando SEQ e as paralelizadas, pelo comando PAR.

Também existem laços paralelos, como [9]:

```
PAR i:=0 FOR n
    A[i] := A[i] + 1
```

Nestes laços, todas as iterações são realizadas em paralelo, caso existam n processadores disponíveis.

Outra forma de sentenças que podem ser executadas em paralelo são as existentes em programação lógica, pois pode-se processar tanto as cláusulas em paralelo (paralelismo "OR") quanto as metas destas (paralelismo "AND"). Exemplificando, para provar A a partir de $A : -B, C$ e $A : -E, F$, pode-se tentar provar E, F em paralelo com B, C, correspondendo ao primeiro caso, ou ainda desdobrar o processamento serial de E, F e B, C em quatro tarefas paralelas (paralelismo "AND").

Problemas surgem quando as metas de uma cláusula compartilham alguma variável e, por conseguinte, não podem ser avaliadas em paralelo. Neste caso, um dos métodos empregados de solução é deixar ao programador a tarefa de regular o acesso a esta variável. Este é o método usado

[6] Functional Program.

em Concurrent PROLOG [73], onde a notação $A : -B(X), C(X?)$ indica que B tem permissão de leitura e escrita, mas C pode apenas ler X.

3.2.4 Objetos

Nas linguagens seqüenciais orientadas a objetos tais como C++, por exemplo, um objeto é ativado ao receber uma mensagem, enquanto o objeto que a enviou permanece passivo, enquanto uma resposta não chega. Em qualquer instante, apenas um objeto detém o controle do processamento.

Para que paralelismo seja introduzido, o modelo de objetos seqüenciais pode ser modificado através de uma das seguintes formas [9]:

- Permitir a um objeto estar ativo mesmo sem haver recebido uma mensagem;
- Permitir a um objeto continuar a execução após o envio de respostas;
- Enviar mensagem a vários objetos de uma só vez;
- Permitir ao enviador prosseguir em paralelo com o recebedor.

Essas formas de introdução de paralelismo estão intimamente relacionadas com a atribuição de múltiplos "threads"[7] de controle aos objetos ou ao envio assíncrono de mensagens.

A introdução de "threads" diminui a distância, outrora maior, entre um sistema operacional e as linguagens que ele suporta. A razão para esta aproximação é prover suporte de baixo nível eficiente para abstrações de mais alto nível [16]. Como conseqüência deste processo, interfaces padrões estão sendo desenvolvidas para sistemas operacionais para evitar que a flexibilidade conseguida com a separação precedente seja perdida. Um exemplo disto é o padrão POSIX da IEEE.

A composição de uma linguagem de programação com um sistema de suporte dá origem a um **sistema de programação**. Este sistema é dito **baseado em objetos** [16] quando suporta todas as propriedades inerentes ao conceito de objetos, exceto herança, ou seja, é um sistema *orientado a objetos* sem herança. A disponibilidade de sistemas baseados em objetos é muito maior devido ao fato de que a introdução de herança no nível do sistema operacional é uma tarefa ainda pouco pesquisada [16].

Granularidade

A granularidade dos objetos que compõem um sistema de programação é determinada pelo tamanho relativo dos objetos, pela sobrecarga que impõem ao sistema para o seu gerenciamento, e pela quantidade de processamento que este executa a cada invocação [16].

Os sistemas com *grãos grandes* possuem objetos que geralmente possuem o seu próprio espaço de endereçamento, permitindo sua proteção pelo "hardware" contra interferências indesejáveis causadas por falhas em outros objetos. Os objetos de granularidade grande possuem a desvantagem de serem entidades pesadas, dado que a cada objeto é associado um espaço de endereçamento próprio. Dado que a proteção e o controle do sistema é realizado no objeto como um todo, o grau de concorrência que pode ser fornecido é restrito.

[7] Um "thread" é um fluxo de controle independente dentro de um processo.

O grau de concorrência pode ser aumentado reduzindo-se a granularidade dos objetos, fazendo-se com que vários *objetos de grão médio* residam no espaço de endereçamento de um objeto de grão grande, desde que se reduza o controle de acesso aos dados para manter a consistência global. O controle de acesso e o processamento concorrente acarretam no aumento da sobrecarga de controle imposta ao sistema.

Reduzindo-se ainda mais a granularidade, chega-se aos *objetos de grão fino*. Neste, o sistema é fragmentado em correspondentes aos tipos de dados providos pelas linguagens de programação, como inteiros e booleanos. A sobrecarga imposta pelo gerenciamento de tais objetos é elevada, pois transforma-se em invocação de método de qualquer operação realizada sobre um dado objeto. A contrapartida é que o ambiente resultante é completamente uniforme, pois todas as entidades de dado são objetos.

A granularidade dos objetos é um ponto de conflito entre os projetistas de sistema em que o conceito de objeto é utilizado. Enquanto em Smalltalk-80 tudo é considerado como objeto, sendo portanto de granularidade fina, C++ divide os dados em tipos básicos e objetos, deixando o programador decidir o que são objetos.

Objetos Passivos e Ativos

A atribuição de "threads" de controle aos objetos de forma permanente ou temporária permite a classificação dos objetos de um sistema como *ativos* ou *passivos*. Os objetos são ditos **passivos** quando detêm o "thread" de controle apenas durante a execução de um de seus métodos. Desta forma, o "thread" pode percorrer vários objetos diferentes para completar todas as operações requeridas em uma determinada atividade.

Em contrapartida, os **objetos ativos** possuem o seu próprio conjunto de "threads", os quais são criados, associados aos objetos e destruídos quando estes deixam de existir. Na execução de uma atividade, os objetos trocam requisições entre si na forma de mensagens. Como os "threads" não são transferidos nas invocações, o objeto receptor deve aceitar explicitamente a requisição caracterizando os objetos como um par cliente-servidor.

A associação de "threads" a objetos ativos permite duas variações a saber: a variação **estática** ocorre quando se dispõe de um número fixo de "threads" criado para cada objeto ativo, enquanto que na **dinâmica** os "threads" são criados à medida em que são necessários.

Exemplos de estruturação de objetos em sistemas baseados em objetos são: o Amoeba [62], cujos objetos são de granularidade grande e a variação estática do modelo de objetos ativos é utilizada; Argus [57] que suporta objetos de grão grande e médio, e a variação dinâmica do modelo de objetos ativos; e Emerald [16], cujos objetos passivos podem ser grandes, médios ou pequenos.

3.2.5 Atores

O surgimento da Teoria de Atores se deu a partir de experimentos realizados com as linguagens Simula e Smalltalk para a simulação seqüencial de sistemas concorrentes. Diferentemente do modelo de objetos, cujo centro são os dados do programa, nos atores o controle de execução é o ponto principal, procurando-se utilizar ao máximo a concorrência do ponto de vista da aplicação. Os atores permitem separar a concorrência lógica, existente no programa, do paralelismo real, fornecido pela infra-estrutura.

Semelhantemente ao modelo de objetos, os atores também permitem o **encapsulamento dos dados**. No primeiro, através do encapsulamento, a memória privada do objeto é manipulada apenas pelas operações que lhe são aplicadas. No segundo, o comportamento de um ator somente pode ser alterado por uma uma comunicação que lhe é enviada, conseqüentemente eles mantêm privacidade e segurança com respeito a sua estrutura interna. Desta forma, programas que incorporam atores, diferentemente de linguagens baseadas em lógica, são capazes de incorporar pontos de vista diferentes, e às vezes contraditórios, do sistema representando o mundo real que implementam [4].

No entanto, Atores não são classificados através de uma árvore de herança. Em linguagens orientadas a objetos, o conceito de herança agrupa os objetos em classes e meta-classes, permitindo-lhes o compartilhamento da informação e a produção de novas classes derivadas, especializando as entidades do mundo real que representam. Como a noção de classe não faz parte da Teoria de Atores, um ator pode transformar o seu comportamento sem sofrer restrições impostas pela condição de membro de uma classe.

Linguagens baseadas em atores evitam o comando de atribuição associado ao "gargalo" de von Neumann e ainda permitem capturar a história do sistema através do *comportamento de substituição*[8]. A substituição permite ainda a avaliação concorrente de expressões que não envolvem dependência entre os dados [3]. Uma vantagem da utilização de Atores para modelar *sistemas abertos* é que, da mesma forma que o modelo de objetos, atores são dinamicamente reconfiguráveis, e como conseqüência, são adequados à representação de sistemas em constante evolução[4].

Atores são agentes computacionais mais poderosos do que Processos Seqüenciais ou Funções com Avaliação Retardada de Parâmetros [3], pois é possível definir um sistema puramente funcional como um sistema de atores, e é possível especificar processos seqüenciais arbitrários através de um sistema conveniente de atores. Atores podem criar outros atores, enquanto funções não podem criar outras funções; e processos seqüenciais não podem criar outros processos, podem apenas ativá-los.

Comportamento de um Ator

O tipo de Ator utilizado nesta tese é de uma categoria especial chamada *Seriador primitivo* [19]. Um seriador primitivo consiste em um árbitro, uma fila e um processador[9] (ver Figura 3.1.). Quando uma mensagem é dirigida ao seriador, esta é submetida ao árbitro que a coloca em uma determinada ordem na fila para tratamento pelo processador. O árbitro deve ser confiável, ou seja, não deve perder nenhuma mensagem que chega até ele sem colocá-la na fila. O árbitro é o responsável pela eqüidade do processamento das mensagens que chegam. Quando o processador retira uma mensagem da fila, ele a trava e não aceita nenhuma outra mensagem até que o substituto que irá processar a mensagem seguinte seja ativado.

O endereço de um Ator (seu "mailbox") pode ser livremente comunicado a outros Atores, uma característica que possibilita a reconfiguração dinâmica do sistema e permite a sua extensão, dado que endereços de Atores recém-criados podem ser comunicados a outros Atores. O processamento de uma mensagem pode envolver:

[8] "Replacement behaviour".

[9] Um processador é a entidade que processa as mensagens na teoria de atores. Não corresponde, portanto, a um processador físico em si, mas na capacidade de processamento que o sistema provê ao ator.

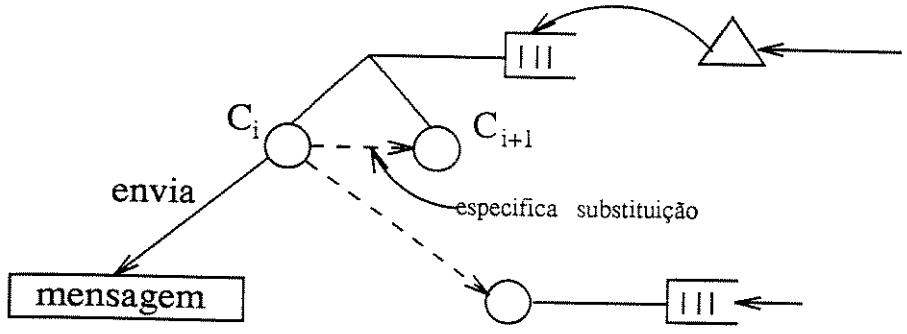


Figura 3.1 Comportamento de um ator.

- A tomada de algumas decisões locais, baseadas no seu comportamento atual;
- O envio de um número finito de mensagens a atores específicos dos quais o seriador atual conhece o endereço. Em particular, um Ator pode enviar uma mensagem a si mesmo;
- A criação de um número finito de novos seriadores primitivos, uma atividade semelhante à criação de novos processos. Inicialmente, o endereço dos novos seriadores pode ser conhecido apenas pelo criador e, possivelmente, pelo seriador criado. Contudo, este endereço pode ser posteriormente comunicado a outros atores;
- A especificação de uma *substituição*. Um Ator **deve** especificar um substituto que irá processar a próxima mensagem na fila. Tão logo a substituição seja efetuada, o processamento da próxima mensagem pode começar, ainda que outras ações decorrentes do processamento da mensagem precedente estejam sendo executadas. A substituição difere de uma mudança no estado local: a substituição C_{i+1} **não** altera as variáveis do estado anterior, e qualquer informação relevante para o comportamento C_{i+1} deve ser incluída em seus parâmetros.

Uma das formas de introdução de paralelismo em um sistema composto de objetos é o envio de mensagens a vários objetos de uma só vez em resposta a uma única mensagem. Esta forma de geração de atividades concorrentes é exatamente a utilizada em um sistema de atores. O processo de substituição pode assumir a estrutura de um duto de informações, permitindo a um Ator processar concorrentemente mais de uma mensagem. Em particular, se o comportamento de um Ator não depende da história anterior do sistema, a sua substituição pode ocorrer logo no início do tratamento de uma mensagem, antes mesmo de qualquer cômputo relativo a esta.

A construção de um novo processador causado pela operação de substituição é, simplesmente, uma hipótese conceitual para salvaguardar o modelo contra os detalhes de uma implementação em particular. Concorrência apenas significa potencial paralelismo. Algumas implementações podem julgar conveniente retardar a substituição até que o processador antigo possa ser destruído. Contudo, o atraso na construção de um substituto não é um requisito universal que tem de ser seguido como ocorre com um processador seqüencial.

Como o objetivo da Teoria de Atores é exatamente explorar ao máximo o paralelismo que o sistema pode oferecer, se existirem suficientemente recursos disponíveis, a execução em um sistema de atores pode ser acelerada simplesmente ao proceder com a próxima mensagem enfileirada a

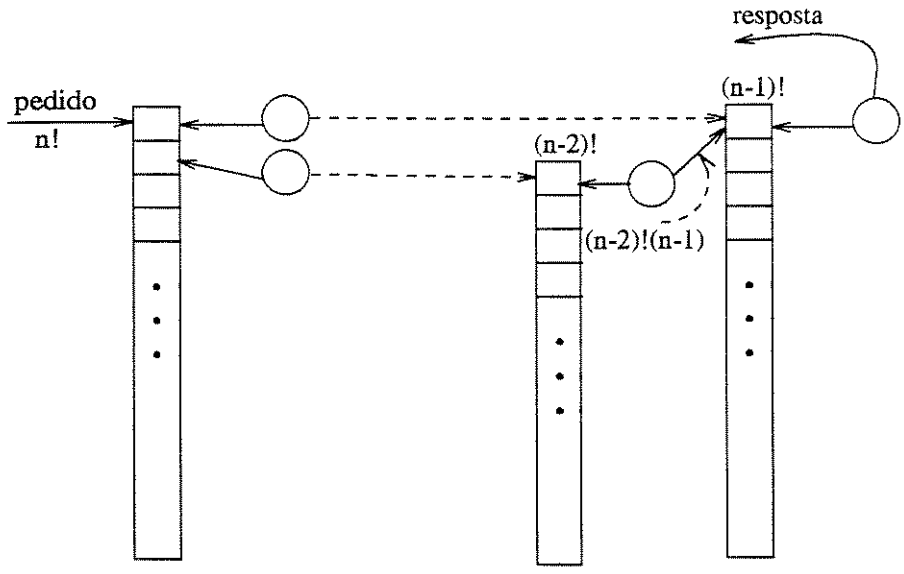


Figura 3.2 Processamento recursivo de fatorial.

determinação do comportamento do substituto e de seus parâmetros associados. Quando um segmento em particular do processamento de uma mensagem completa a sua execução, os recursos usados por este segmento são colocados imediatamente à disposição.

É importante que se observe que não é imprescindível que a substituição ocorra logo no início do processamento de uma mensagem para que a teoria de Atores explore a concorrência presente no sistema. Apenas a geração de várias mensagens em resposta a uma única e o processamento concorrente destas mensagens, ainda que por atores distintos entre si, são suficientes para explorar a concorrência presente na infra-estrutura. De fato, o seriador primitivo usado em [19], para estabelecer as bases da semântica das linguagens baseadas em Atores, somente começa o processamento da próxima mensagem após o da mensagem corrente.

Exemplo: Fatorial Recursivo

Para que se tenha uma idéia do grau de paralelismo que pode ser explorado em um sistema de Atores, o cálculo do fatorial de um número é ilustrado. Um fatorial recursivo é implementado em termos de um Ator que, ao receber um pedido de cálculo do fatorial de n:

1. a) Cria um cliente para esperar por uma mensagem contendo o fatorial de $n-1$ (ver Figura 3.2), após cuja chegada fará a multiplicação por n e enviará o resultado total ao originador do pedido de $n!$;

2. b) Envia a si mesmo uma mensagem solicitando o envio de $(n-1)!$ ao cliente recém-criado.

Em pseudo-código, os programas correspondentes ao ator fatorial e multiplicador podem ser representados por:

```
Fatorial(self) {
        accept <mensagem contendo um inteiro n e um cliente u >
        new-behaviour(Fatorial)
        if n = 0
              then send [1] to u
              else {
                    c = new-actor(Multiplicador, n, u)
                    send [n-1, c] to self
              }
}


Multiplicador(n,u) {
        accept <mensagem contendo um inteiro k>
        send [n*k] to u
}
```

Neste pseudo-código, os parâmetros representam as *posses*[10] de um ator no momento de sua criação. A variável `self` é o endereço que aponta para o próprio ator. Em uma implementação, este endereço pode ser enviado sob a forma de parâmetro ou obtido através de uma palavra reservada cujo valor é garantido pelo ambiente, a exemplo do que ocorre em C++ onde a palavra **this** sempre aponta para o endereço do objeto corrente.

Logo após o aceite de uma mensagem através da primitiva `accept` o ator `Fatorial` já está apto a aceitar um novo pedido, pois neste caso não há sensibilidade à história do sistema. Isto é expresso pela primitiva `new-behaviour`. O `Multiplicador` não possui tal primitiva porque sua tarefa será executada uma única vez, após a qual deixará de existir.

3.3 Forma de Paralelismo usada na Plataforma

Dentre todas as formas analisadas na Seção 3.2 como candidatas à unidade de concorrência a ser fornecida pela plataforma, selecionou-se a abstração de atores, baseando-se nos seguintes fatos:

- Esta teoria, inicialmente desenvolvida para a implementação de aplicações em inteligência artificial, representa um conjunto de idéias recentes, cuja aplicação em outros ramos do conhecimento permanece praticamente inexistente. Portanto, a verificação de uma adequação à gerência de redes de telecomunicações e, aos problemas resultantes de sua unificação com as redes inteligentes é bastante válida;

- A profusão de sistemas que possibilitam o gerenciamento de múltiplos "threads" de execução dentro de um mesmo espaço de endereçamento, e o fácil mapeamento entre "threads" e Atores favorecem uma implementação relativamente fácil da plataforma. Nos "threads", a criação, existência, destruição e sincronização é de tão baixo custo ao sistema operacional, quando comparados aos processos normais, que programadores podem usá-los para todas

[10] Acquaintances

as necessidades de concorrência. Tipicamente, milhares de "threads" podem ser criados em um único processo maior (no mesmo espaço de endereçamento):

- Os atores fornecem uma abstração de alto nível a ser usada na programação de sistemas concorrentes, com uma semântica precisamente estabelecida através de uma série de leis que, desenvolvidas para o modelo de Atores, foram posteriormente usadas como axiomas gerais a serem satisfeitos por todos os sistemas distribuídos [19, 4, 36];
- Usando-se Atores, pode-se implementar processos seqüenciais CSP e funções com avaliação retardada de parâmetros;
- Os atores foram desenvolvidos com o objetivo de explorar maciçamente o paralelismo disponível em um sistema, sem com isto se ficar preso a detalhes arquiteturais do "hardware", como o número de processadores existentes;
- A proximidade entre atores e objetos;
- O desenvolvimento do modelo de Atores tem servido de base para pesquisa dos princípios fundamentais de sistemas abertos distribuídos ultraconcorrentes, os quais estão adaptados às gerações futuras de "hardware" concorrente [37].

Não se pretende, nesta tese, construir uma linguagem baseada em Atores, e sim, introduzir as primitivas relativas à criação, envio de mensagens e substituição de comportamento de um ator em classe C++ de objetos. A função dos objetos desta classe é definir o comportamento básico de um Ator, em termos de seu mapeamento com "threads" de execução.

Também não será obrigatório que qualquer objeto definido no sistema seja um Ator, como advoga Lieberman [56]. Na linguagem ACT 1, qualquer objeto, mesmo inteiros e variáveis booleanas são Atores. Este modelo é tão purista quanto o modelo de objetos adotado em Smalltalk-80, na qual tudo é um objeto. A pureza no modelo traz a vantagem de uniformizar o tratamento das entidades que o compõem, mas implica em uma sobrecarga adicional ao sistema. Por exemplo, ao declarar que qualquer entidade é um Ator, deve-se criar o seu "mailbox" associado e controlar a fila de mensagens deste "mailbox". Fazer isto para todos os tipos de dados seria impraticável.

É por este motivo que existem os modelos híbridos. Em C++, por exemplo, dá-se uma maior atenção à eficiência do processamento, mesmo que com isto algumas propriedades de modelo puro se percam. Os tipos de dados são divididos em básicos, como inteiros, reais, etc., e objetos compostos destes tipos básicos.

Neste trabalho, adotou-se a mesma estratégia: é o programador quem define quais objetos são atores e quais são apenas objetos comuns C++. Com isto, ele estará determinando o grau de paralelismo máximo que deseja utilizar.

3.4 Comunicação e Sincronismo

A comunicação e o sincronismo são dois tipos de interação entre elementos de um sistema concorrente. A comunicação define a forma pela qual a informação é veiculada entre as entidades e o sincronismo é um ordenamento temporal do conjunto de eventos produzidos pelas entidades concorrentes. Estes conceitos estão relacionados porque algumas formas de comunicação sempre

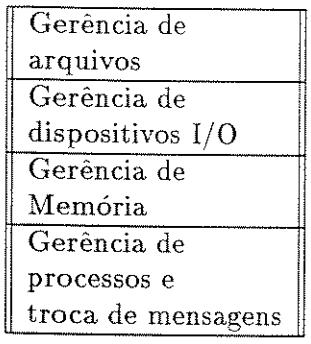


Figura 3.3 Gerenciamento de Memória .

requerem sincronização prévia. Em particular, dois processos que se comunicam por meio de troca de mensagens seguem um processo de sincronização definido pela semântica das primitivas de envio e recepção. Um exemplo é o *rendezvous* usado na linguagem ADA, cuja interação é totalmente síncrona.

O modo de comunicação entre entidades pode ser dividido em duas classes principais: o compartilhamento de dados e a troca de mensagens.

A realização física dos blocos constitutivos da arquitetura TMN pode ser feita na forma de um sistema de software logicamente distribuído, pois neste sistema a informação é intercambiada através de troca de mensagens. A introdução de concorrência para melhorar o desempenho dos agentes do INMF e OSI-MF, bem como nas entidades funcionais IN, tornaram necessário o compartilhamento de dados entre as diversas atividades que se desdobram em paralelo e a sua conseqüente sincronização para manter a consistência dos dados compartilhados.

As duas formas de comunicação acima são duais, dado que a troca de mensagens pode ser implementada através do compartilhamento sincronizado de dados e que uma região compartilhada de memória pode ser implementada usando-se troca de mensagens.

Alguns sistemas operacionais distribuídos como Accent, Mach e V-System [31] têm troca de mensagens entre processos, memória virtual e armazenamento de arquivos fortemente integrados. A integração entre memória virtual e mensagem entre processos permite a um processo manusear a região de memória de outro. Estes sistemas são exemplos de sistemas de "software" logicamente não-distribuídos, implementados sobre uma plataforma de "hardware" fisicamente distribuída. A distribuição física implica que o modo de comunicação básico seja a troca de mensagem através da rede, enquanto o compartilhamento de dados é uma abstração construída sobre esta forma básica. isto é possível colocando-se a troca de mensagens abaixo do gerenciamento de memória na estrutura do sistema operacional, conforme ilustrado na Figura 3.3.

A dualidade destas formas de interação não implicam, contudo, em que elas possuam sempre a mesma eficiência, e é por este motivo que as duas são freqüentemente exploradas em conjunto. Mesmo em sistemas operacionais em que existe comunicação entre processos por meio de troca de mensagens, como no Solaris, por exemplo, o espaço de endereçamento de um processo pode ser compartilhado por vários "threads" de controle.

3.4.1 Compartilhamento de Dados

No compartilhamento de dados é que vários processos podem ler e escrever em regiões de dados comuns a mais de um deles.

As principais razões para o provimento de compartilhamento de memória entre processos de usuário são [31]:

1. Redução da sobrecarga causada pela comunicação entre processos e transição de processo-a-processo. Isto é crucial em sistemas distribuídos, e também em multiprocessadores, nos quais o desempenho da comunicação é o fator mais importante. Memória compartilhada provê a forma de comunicação mais rápida quando os processos estão na mesma máquina;

2. Desenvolvimento de servidores que possam tratar requisições de vários clientes em paralelo, ao invés de sua seriação, ou da criação de um processo servidor por cliente;

3. Compensação de dispositivos lentos (discos, redes, terminais e impressoras), de modo que um programa eficiente possa realizar alguma outra tarefa proveitosa enquanto espera por um destes dispositivos;

4. Programação de ações concorrentes disparadas por usuários humanos, como por exemplo, na construção de um sistema com janelas;

5. Um conjunto de processos em espaço de endereçamento compartilhado é a forma mais natural de programar muitas aplicações;

6. Construção de sistemas mais facilmente extensíveis. Quando se utiliza troca de mensagens, cada novo processo deve possuir, de alguma forma, o endereço dos demais com os quais precisa interagir, o que não ocorre como compartilhamento de memória;

7. Definição precisa do estado do sistema, embora a falta de estruturação neste estado possa trazer efeitos colaterais;

8. Não favorece nenhuma arquitetura em particular de sistema distribuído.

As principais desvantagens do compartilhamento de memória são:

1. O processo de compartilhamento em si não possui nenhuma estruturação para prover abstração de dados ou ocultamento de informações. Um processo pode, por exemplo, determinar se um outro escreveu na região de memória que ele está prestes a ler, mas não pode proteger estes dados contra acesso indevido ou operações impróprias. Geralmente, a região compartilhada tem seu acesso disciplinado com a utilização de mecanismos de exclusão mútua, mas nenhum mecanismo é fornecido para proteger os dados contra acesso indevido.

   O encapsulamento dos dados através de programação orientada a objetos pode ajudar bastante neste processo ao garantir que as próprias operações que atuam sobre os dados solicitem a permissão de acesso, mas mesmo neste caso não se consegue proibir o acesso indevido;

2. Os processos devem estar próximos da localização dos dados que fazem acesso freqüentemente. Caso contrário, a latência do acesso torna-se um gargalo no sistema.

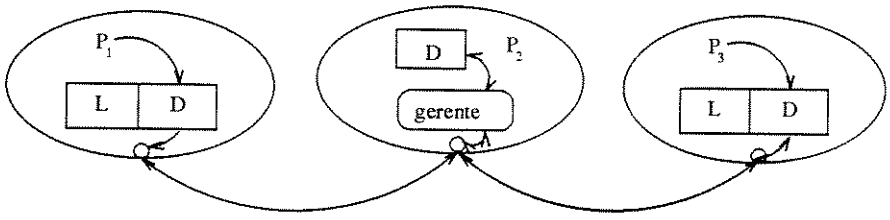


Figura 3.4 Memória compartilhada centralizada.

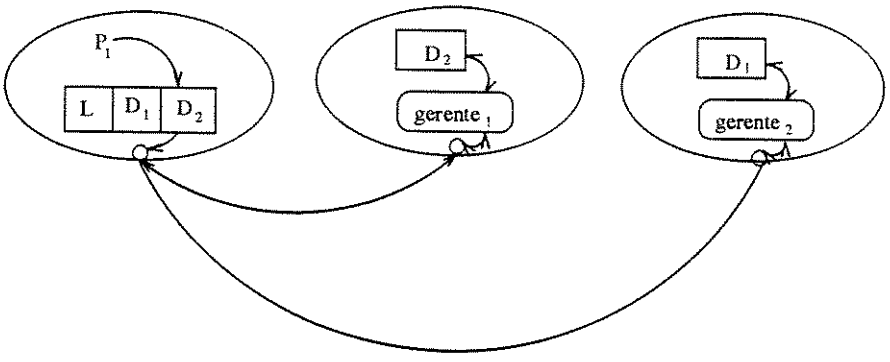


Figura 3.5 Memória compartilhada distribuída.

A memória compartilhada pode ser implementada na forma **centralizada** ou **distribuída**. A forma centralizada é de fácil implementação, bastando para tal realizá-la como o espaço de endereçamento de um processo, e dirigir a um gerente de acesso localizado neste processo mensagens solicitando porções desta. Neste caso, a consistência é garantida pois o gerente de acesso é o único a manipular a memória. Na Figura 3.4, o processo P_1 ao tentar fazer acesso à memória compartilhada centralizada D, causa o envio de uma mensagem ao processo P_2, onde está o gerente responsável por esta memória.

As principais desvantagens deste tipo de memória são a criação de gargalo no gerenciador de memória e a confiabilidade reduzida pelo uso de um gerenciador único. Algumas arquiteturas baseadas em multiprocessadores já provêem memória fisicamente compartilhada entre estes processadores, como é o caso do Solaris e Mach. Nestes sistemas, cada processo possui um espaço de endereçamento de memória que é compartilhado por múltiplos "threads" de execução. Como é fornecida em nível de sistema operacional, o controle de acesso, que envolve sincronização, é realizado muito mais facilmente, dado que a memória e o relógio são únicos [31].

A memória compartilhada distribuída pode ser implementada com ou sem replicações. Em qualquer dos casos a realização é por meio da divisão da memória a ser compartilhada em segmentos que são distribuídos entre vários gerentes de acesso. No caso sem replicação, cada segmento existe em apenas um local no sistema distribuído. Este é o caso ilustrado na Figura 3.5, na qual o processo P_1 usa dois segmentos de memória distribuídos localizados exclusivamente em P_2 e P_3.

O compartilhamento distribuído sem replicação possui as vantagens de garantia da assistência e relativa facilidade de implementação e do tratamento paralelo de requisições, dado que há um gerente para cada segmento. O seu principal problema está, como no caso da memória compartilhada centralizada, na baixa confiabilidade. Se um dos gerentes deixar de funcionar, o

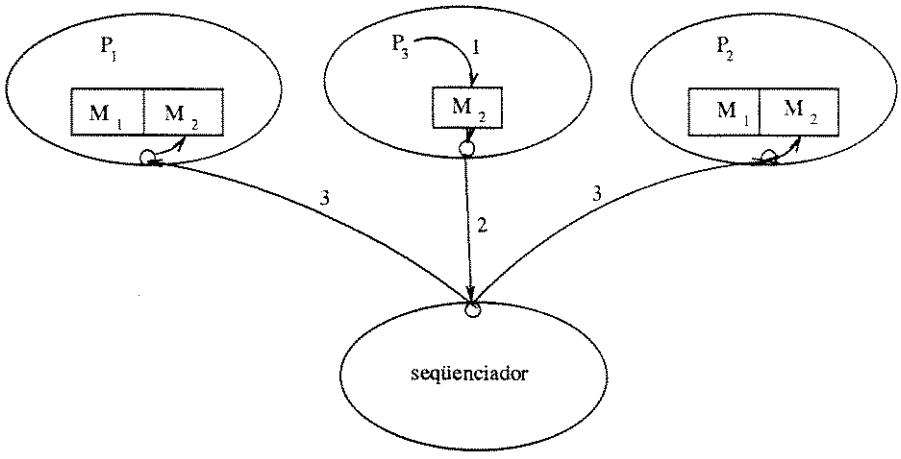


Figura 3.6 Acesso à memória controlado por seqüenciador global.

seu segmento de memória torna-se inacessível.

A memória distribuída com replicação possui na consistência dos dados o seu maior problema. Dois esquemas são possíveis: o com *múltiplos leitores e único escritor* (MRSW)[11] e o com *múltiplos escritores e múltiplos leitores* (MWMR)[12]. O MRSW possui as vantagens de ser tolerante a falhas, eficiente quando a taxa de leituras sobre gravações é alta e de paralelizar a leitura, mas a manutenção da assistência é complexa pois somente a cópia mestra pode ser atualizada e todas as outras devem ser invalidadas de antemão.

Quando múltiplos processos podem escrever na região compartilhada (MWMR), cria-se um problema conhecido como Problema de Atualização de Múltiplas Cópias para que sua consistência seja garantida. A classificação das soluções para este problema divide-as em Votadas e Não-votadas [31]. As soluções votadas são o resultado da negociação entre processos para chegar a um acordo sobre o ordenamento das atualizações a serem executadas sobre a região enquanto que as não-votadas são baseadas em processo central encarregado de fazer o seqüenciamento de todos os pedidos de alteração. No exemplo da Figura 3.6 o seqüenciador recebe o pedido de P_3 e encarrega-se de enviar a alteração para todos os processos que possuem a região afetada (no caso P_1 e P_2). Além de possuir as mesmas vantagens do MRSW, este método ainda permite que múltiplos processos possam escrever na região compartilhada. Suas principais desvantagens são o gargalo criado com o seqüenciador único e o retorno a um estado anterior diante da ocorrência de uma eventual diferença entre as versões.

3.4.2 Compartilhamento de Memória na Plataforma

A introdução de concorrência no acesso à MIB e no processamento de serviços nas entidades funcionais da Rede Inteligente implicam no compartilhamento de um estado interno destas entidades.

Uma abordagem possível é a transformação em atores de todos os objetos contidos neste estado. Como os atores só interagem por meio de troca de mensagens e encapsulam a informação

[11] "Multiple reader, single writer."
[12] "Multiple writer, multiple reader."

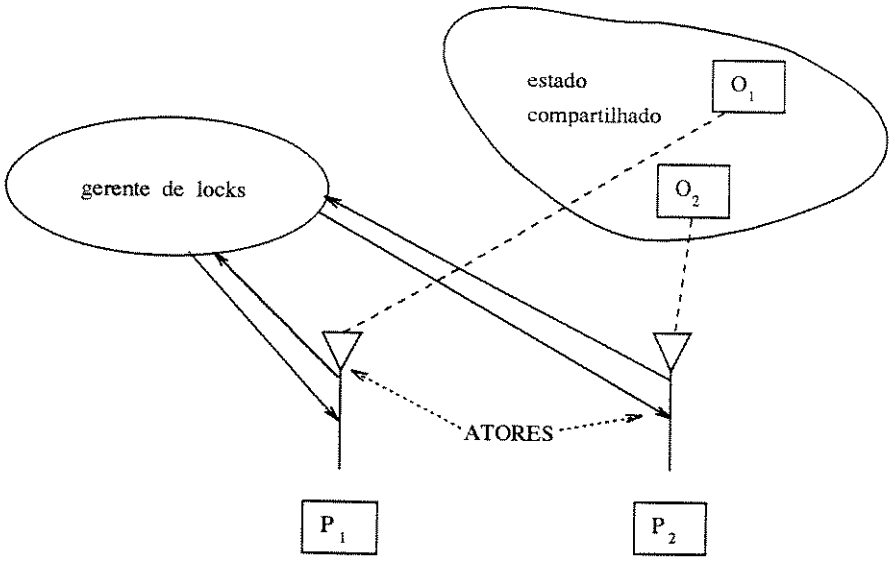


Figura 3.7 Acesso à memória compartilhada através do Gerente de Locks.

referente ao estado, esta transformação eliminaria qualquer necessidade de controle de consistência dados. Contudo, a abordagem de atores puros acarretaria uma sobrecarga aos processadores.

A solução adotada foi a criação de um gerente para este estado, cuja função é a de receber as requisições relativas à manipulação deste estado, e garantir o seqüenciamento das operações realizadas sobre cada objeto envolvido. A sua implementação se deu na forma de um Gerenciador de "Locks". Após a verificação de que nenhum outro Ator está fazendo acesso ao objeto, o gerente informa ao Ator que está interessado em operar sobre o objeto que este encontra-se livre e estabelece um "Lock" sobre o objeto, que somente será disponibilizado novamente quando o Ator que o solicitou enviar mensagem liberando-o.

Quando os Atores que fazem acesso a este estado compartilhado são implementados usando-se "threads" de execução sobre um mesmo espaço de endereçamento, (ver Seção 5) a troca de mensagens entre Atores é muito mais rápida, pois se dá através desta memória compartilhada, não envolvendo a rede de comunicação. Este compartilhamento também possibilita aos Atores que, uma vez adquirido o "Lock", cada um deles possa executar o método diretamente, resultando em verdadeira concorrência sobre o estado.

Na Figura 3.7, os atores A e B solicitam ao Gerente de Lock acesso aos objetos O_1 e O_2, respectivamente. Após a obtenção da permissão, eles podem ter o seu processamento atribuído aos processadores P_1 e P_2 e agir de modo totalmente paralelo sobre O_1 e O_2.

3.4.3 Referência de Tempo Global é Fisicamente Irrealizável

Nos sistemas distribuídos, o limite da velocidade com a qual a informação pode trafegar também inviabiliza a hipótese de realizabilidade de um relógio global, capaz de ordenar todos os eventos que ocorrem no sistema, isto é, num sistema distribuído, um relógio global único não é definível. Esta intuição foi inicialmente axiomatizada por Hewitt e Baker [36] e sua consistência com outras leis do processamento paralelo foi mostrada por Clinger [19].

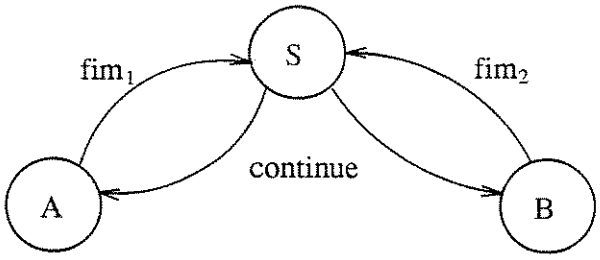


Figura 3.8 Sincronizador global.

Entretanto, Clinger mostrou em [19] que múltiplas janelas de referência são possíveis em sistemas que usam ordenamento de eventos de chegada ou ativação. No ordenamento de ativação, se e_1 ativa e_2, isto será verdadeiro para todos os observadores em todas as janelas de referência. O mesmo ocorre com a chegada de mensagens. Se m_1 chega antes de m_2 em um ator seriado. todos os observadores também perceberão a chegada desta forma, qualquer que seja a janela de referência. Para eventos não relacionados por este ordenamento, a percepção de cada observador será diferente acerca da ordem dos eventos.

A falta de uma referência global de tempo, contudo, não impede a existência de um sistema distribuído, cujo comportamento é tal que os elementos do sistema possam ser abstratamente construídos para agir sincronicamente [3]. A construção deste sincronismo é baseada no ordenamento de chegada e ativação.

Na Figura 3.8 os processos A e B são controlados pelo sincronizador S. Este sincronizador recebe as mensagens fim_1 e fim_2 (ordenamento de chegada) e, somente após as duas haverem chegado, envia uma mensagem "continue" aos processos A e B, (ordenamento de Ativação) que continuam as suas atividades.

O sincronismo construído desta forma cria um gargalo que pode ser extremamente ineficiente em um ambiente distribuído, pois cada processo deve esperar pelo outro, independentemente de suas velocidades relativas de processamento. É por este motivo que o paradigma de comunicação usado na construção da teoria de atores é o de troca assíncrona de mensagens. Sempre que for necessário, pode-se construir o comportamento síncrono a partir do assíncrono, como um caso particular.

3.4.4 Comunicação por Troca de Mensagens

Vários modelos de computação concorrente usam a troca de mensagens entre agentes independentes como forma de comunicação. Diferentemente do compartilhamento de dados, esta forma permite a cada um destes agentes manter a integridade da informação que ele encapsula contra efeitos colaterais causados por acesso indevido. O agente, ao receber uma mensagem, é livre para decidir se esta deve causar, ou não, uma alteração no seu estado interno.

Uma das principais decisões ao se projetar um sistema de troca de mensagens é a escolha acerca da sincronização ou não entre o remetente e o destinatário:

- Na forma **síncrona**, o enviador e o recebedor de uma mensagem devem estar prontos para se comunicar antes que a mensagem seja efetivamente transmitida, ou seja, o enviador é bloqueado até que o recebedor aceite a mensagem, quer isto ocorra de modo explícito, com o

próprio destinatário acusando a recepção, ou implícito, quando alguma entidade do sistema do destinatário acusa a recepção e garante a entrega;

- No caso **assíncrono**, o recebedor não precisa estar pronto a receber a mensagem no momento em que ela foi enviada. O enviador pode continuar o prosseguimento normal de suas atividades logo após o envio. A implementação em uma dada linguagem pode suspender o enviador até que a mensagem tenha sido copiada para o envio, mas este atraso reflete uma característica desta implementação, e não uma variação semântica.

Exemplos de troca de mensagens síncrona e assíncrona são o sistema telefônico e o sistema postal (Correio), respectivamente, embora, com relação ao sistema telefônico, isto não seja mais totalmente verdadeiro após a invenção da secretária eletrônica.

CSP e CCS assumem comunicação síncrona como forma de interação, enquanto no modelo de Atores e no de Funções com Avaliação Retardada de Parâmetros, adota-se a forma assíncrona.

A Comunicação Síncrona é um Caso Especial da Assíncrona

Devido à irrealizabilidade de uma referência de tempo global, a abstração de sincronismo só pode ser construída na dependência de uma janela de referência, através da qual o sistema é observado. Para construir tais janelas, usa-se o ordenamento causal definido pelas ordens de ativação e de chegada. O ordenamento total então obtido é uma ordem parcial, pois os eventos ocorrendo em agentes computacionais diferentes estão desordenados, a menos que se possa conectá-los direta ou indiretamente por uma ou mais ligações causais (ordenamento de ativação).

Para estabelecer uma conexão síncrona, o enviador deve, portanto, ordenar os eventos que afetam o seu estado interno, com os do sistema remoto, de forma a construir a janela de referência. O primeiro ato neste sentido é que para saber se o recebedor está livre para aceitar uma troca de mensagens, ele deve enviar uma mensagem ao próprio recebedor. Conseqüentemente, qualquer modelo de comunicação síncrona começa com um contato assíncrono.

Durante o decorrer da comunicação surge um outro problema que é o da diferença de velocidade entre o enviador e o recebedor. A solução óbvia de esperar a resposta a cada mensagem enviada não é aceitável devido ao tempo finito (porém, em muitos casos, elevado) de transmissão. O que se faz, então, é armazenar as mensagens a ser enviadas em alguma parte ("buffer") no sistema, de tal forma que o enviador possa realizar algum trabalho útil enquanto espera a transmissão, evitando a *espera ocupada*[13].

A conclusão é que, embora a sincronicidade possa ser construída através de um ordenamento temporal, a sua implementação utiliza-se da assincronicidade para aumentar-lhe a eficiência.

Outro problema com a comunicação síncrona [3] é a impossibilidade de uma entidade comunicar-se com ela mesma, como é o caso quando se define um procedimento recursivo. Dado que o recebedor da comunicação deve estar livre para aceitar a conexão, quando o enviador manda uma mensagem para si mesmo, este entra automaticamente em "deadlock".

3.4.5 Troca de Mensagens na Plataforma

A forma de comunicação usada em Atores é o envio de mensagens assíncronas ao seu "mailbox". Estas mensagens são submetidas ao árbitro e colocadas na fila de mensagens do Ator. Esta é,

[13] "busy waiting"

portanto, a forma básica de comunicação adotada na plataforma. Mesmo quando o compartilhamento de memória é usado entre atores localizados em uma mesma máquina, uma troca de mensagens assíncronas é realizada entre o ator desejando manipular um objeto e o gerenciador de "locks".

A utilização de "mailboxes" é uma forma indireta de nomeação de objetos, e em sua forma mais simples, um "mailbox" é apenas um nome global. Na forma como é usado na teoria de Atores, os "mailboxes" podem ser passados de um ator a outro como parte de uma mensagem. Isto permite que padrões altamente flexíveis de comunicação sejam expressos [9].

Nos sistemas abertos de Gerência OSI, o modo de interação é através de troca de mensagens após o estabelecimento de uma conexão. Assim, a comunicação entre atores localizados em entidades distintas OSI, como é o caso entre um OS e um NE, tem de ser apoiada em algum mecanismo de controle de conexão.

O mesmo ocorre nas Redes Inteligentes quando se utiliza a SCCP, como é o caso da IN brasileira [7]. A SCCP adiciona funções à Parte de Transferência de Mensagens, que se situa nas camadas OSI 1-3, de forma a habilitá-la a fornecer serviços de redes orientados a conexão e sem conexão [47].

Para evitar que cada ator que se comunica com outro tenha de gerenciar todos os aspectos relativos à conexão através da qual as mensagens entre eles é veiculada, introduziu-se um gerente de comunicação como parte da infra-estrutura da plataforma. Este gerente foi implementado usando-se o ambiente ISODE, é parte dos mecanismos de transparência fornecidos pela plataforma.

3.4.6 Tratamento de Prioridades

É importante que se introduza concorrência no acesso à MIB e no estado interno das entidades funcionais IN, e também é necessária a classificação das atividades concorrentes segundo uma escala de prioridades.

O Seriador Primitivo possui um árbitro que é o responsável pelo ordenamento das mensagens que lhe são dirigidas, isto é, pelo tratamento eqüitativo que o sistema fornece ao processamento das mensagens em trânsito.

Na semântica desenvolvida para a Teoria de Atores, possui como um de seus axiomas a hipótese de que o tempo de entrega de uma mensagem é finito. Na prática, isto implica na garantia de entrega de todas mensagens e, como conseqüência na eqüidade do árbitro.

Cada mensagem existente em um sistema de atores possui associada a si uma carga de processamento a ser executada pelo sistema e é devido a este fato que o envio assíncrono de uma ou mais gerado pelo processamento de uma mensagem aumenta a concorrência pelos recursos de processamento do sistema, visando a sua utilização ao máximo. Esta carga de processamento é chamada doravante de **tarefa**.

O tratamento de prioridades associadas às tarefas pode ser realizado de duas formas: globalmente ou somente no processamento. No tratamento global, as mensagens prioritárias são tratadas como tal tanto pelo sistema de entrega de mensagens[14] quanto durante o processamento, enquanto no segundo caso, a prioridade é considerada somente durante o processamento. Na plataforma proposta, o tratamento é feito apenas no processamento, pois:

[14] "Mail System"

a) a introdução de prioridade no sistema de entrega de mensagens implicaria na quebra da garantia de eqüidade e, como conseqüência, a hipótese de tempo de entrega finito não poderia ser assegurada;

b) Mesmo que localmente se pudesse influenciar o árbitro do sistema da entrega de mensagens a tratar as mensagens de acordo com seu grau de prioridade, isto seria muito difícil de se conseguir quando se tem de passar pelo sistema de comunicação externo. O sistema operacional teria de decidir entre mensagens com prioridades idênticas, mas provenientes de sistemas de atores diferentes, qual a prioridade de envio. Os protocolos de rede existentes não possuem prioridade no tratamento de mensagens em trânsito. Uma avaliação feita no sentido de estender o IP para aplicações de multimídia concluiu que no caso de prioridades simples, tão logo houvesse muitos fluxos de dados competindo pela prioridade mais elevada, cada um deles seria degradado [13].

Num sistema de atores bem dimensionado, a taxa de utilização da banda de processamento fornecida pelo sistema operacional deve ser inferior a um, considerando-se ρ_p taxa de utilização pela prioridade p, tem-se que:

$$\sum_p \rho_p < 1.$$

Quando bem dimensionado, o tempo de espera na fila para uma dada prioridade P é finito, pois o comprimento da fila é finito.

A Teoria de Atores não impõe nenhuma restrição ao processamento de uma tarefa, pois a ordem através da qual um ator processa as mensagens é dependente do comportamento dos atores individualmente. Existem classes de atores [3] que ignoram todas as mensagens, ou armazenam indefinidamente algumas delas. Este armazenamento feito internamente por um ator é distinto do armazenamento realizado pelo sistema de entrega de mensagens. Neste último, uma mensagem não pode ser armazenada indefinidamente. Conseqüentemente, a introdução de prioridades no processamento não altera em nada a teoria básica de atores.

Para regular o processamento das atividades, foi incorporado na plataforma proposta um escalonador, cuja função é a de supervisionar a execução em cada nível de prioridade. Um ator pode começar a processar a próxima mensagem da fila tão logo a sua *substituição* seja efetuada. Como as mensagens podem ter prioridades diferentes, o mesmo ator pode estar processando tarefas em vários níveis de prioridades.

Por exemplo, na Figura 3.9, o Ator A_n está processando a mensagem corrente com prioridade 2, e possui tarefas sendo processadas com prioridades P_i e P_n.

Para manter sua eqüidade, o árbitro coloca as mensagens serialmente na fila de cada um dos atores. Como tais mensagens ainda não foram recebidas pelo ator, colocá-las em uma posição relativa às já existentes de acordo com o seu nível de prioridade violaria esta eqüidade.

O escalonador possui também a incumbência de garantir a evolução das várias tarefas concorrentes, dividindo a capacidade de processamento efetiva fornecida pelo sistema operacional entre todas as tarefas pendentes. As regras seguidas pelo escalonador são:

1. Os servidores sempre estão alocados em maior prioridade;

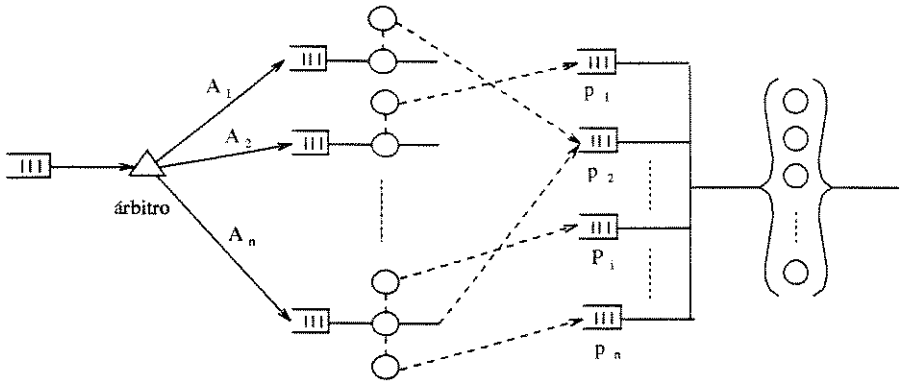


Figura 3.9 Prioridades na execução das mensagens.

2. Caso uma tarefa de menor prioridade esteja executando e chegue uma de maior prioridade, esta sofre preempção[15];

3. Uma tarefa que sofre preempção não precisa recomeçar, ou seja, se tinha tamanho Δt e executou durante $\Delta t'$ deverá executar apenas a parcela restante. $\Delta t'' = \Delta t - \Delta t'$

4. A tarefa que sofre preempção vai para o início da fila correspondente ao seu nível de prioridade;

5. Para que em um dado nível de prioridade uma tarefa não monopolize o recurso atribuído temporariamente àquele nível de prioridade, um sistema de "round-robin" é aplicado, isto é, se uma tarefa já usou o seu *quantum*, ela vai para o fim da fila correspondente.

Inversão de Prioridades

A inversão de prioridades ocorre quando a execução de uma tarefa de alta prioridade é bloqueada por uma de baixa prioridade que detém um recurso que a de alta prioridade necessita. Como ilustração, considere-se o exemplo da Figura 3.10, na qual a tarefa T_1 detém o recurso que T_2 deseja usar.

Simplesmente tomar o recurso de T_1 iria causar inconsistência. Então, a solução é T_2 esperar que T_1 o libere. O problema surge quando T_3, de prioridade média e que não necessita do recurso, causa uma preempção em T_1. Se T_2 pudesse tomar o recurso de T_1, T_3 não haveria começado. Esta é a situação que caracteriza a inversão de prioridade.

Na plataforma proposta, a inversão de prioridades pode ocorrer em duas situações:

a) Quando o processamento de uma tarefa de alta prioridade requer o acesso (lock) de um dos objetos da memória compartilhada e este objeto está sendo manipulado por uma tarefa de baixa prioridade;

[15] As ações de tempo real nos sistemas de comunicação não utilizam, de maneira geral, o modo preemptivo, seja ele "restart" ou "resume", devido ao tempo de processamento das operações e ao custo envolvido com a preempção. Contudo, as ações consideradas neste trabalho são de gerência, ou ações de controle com restrições fracas de tempo real (soft real-time).

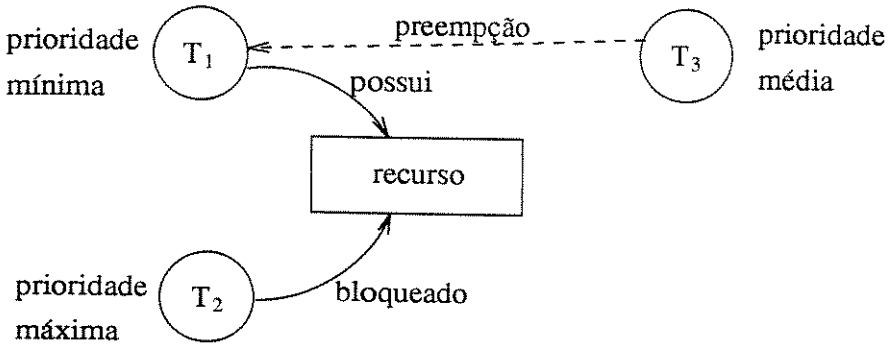


Figura 3.10 Inversão de prioridades na utilização de um recurso.

b) Quando uma mensagem de alta prioridade espera na fila de mensagens de um ator até que todas as que estão na sua frente e possuem prioridades inferiores sejam processadas.

Uma das formas de se resolver o problema de inversão de prioridades é através de *protocolos de herança de prioridades* [72]. A versão mais básica deste protocolo, tenta limitar a duração do bloqueio causado por inversão de prioridades, fazendo a prioridade da tarefa de maior prioridade ser propagada para todas as de menor prioridade que a estão bloqueando. Quando a tarefa de menor prioridade termina o uso do recurso que deu origem ao bloqueio, sua prioridade é revertida para o nível original.

A introdução de herança de prioridade determina a existência de dois níveis de prioridade para cada tarefa, a prioridade *global* e a *herdada*. A **prioridade global** é aquela que está associada à tarefa desde o momento de sua criação, enquanto a **prioridade herdada** é a que a tarefa obtém na herança de prioridade quando bloqueia tarefas de alta prioridade. A prioridade efetiva de *despacho* é a obtida somando o máximo entre estes dois atores. Uma tarefa T é executada no seu nível de prioridade global, a menos que adquira a posse de um recurso do qual uma outra de maior prioridade necessita. Quando isto ocorre, T herda o nível da tarefa que está bloqueando. Quando o recurso é liberado, T volta ao nível de prioridade global.

É importante observar que ao herdar o nível de prioridade de despacho, a tarefa poderá estar herdando uma prioridade previamente herdada. Exemplificando, sejam $P_1 > P_2 > P_3$ as prioridades de despacho das tarefas T_1, T_2 e T_3, respectivamente. Então, se T_3 bloqueia T_2 e T_2 bloqueia T_1, então T_3 herda a prioridade de T_1 via T_2. A relação de herança de prioridade é, então, transitiva. Esta transitividade é necessária para garantir que a duração da inversão seja limitada [53].

Na aplicação da herança de prioridades nos casos de inversão de prioridades (a) e (b), tanto o gerenciador de "locks" quanto o árbitro da fila de mensagens enviam mensagens ao escalonador advertindo-o da existência de inversão de prioridade. O escalonador, baseado nesta informação, aumenta o nível de prioridade das tarefas que estão causando a inversão. Quando a situação deixa de existir, uma nova mensagem é enviada ao escalonador para que ele reverta a situação.

No caso do gerenciador de "locks", as tarefas envolvidas na inversão estão efetivamente sendo processadas, enquanto no caso da fila de mensagens, trata-se de mensagens pendentes, com exceção da que está sendo processada antes da especificação do substituto. Para as mensagens pendentes que estão causando a inversão, duas soluções são possíveis:

1. O árbitro pode enviar recursivamente mensagens ao escalonador, até que a mensagem de alta prioridade comece a ser processada. Alguns problemas com esta solução são:

   - A espera pela liberação do recurso ocorrerá somente após todas as mensagens serem processadas de acordo com sua ordem de precedência na fila, o que pode ser após um tempo arbitrariamente longo;
   - Pode haver uma avalanche de mensagens de prioridade elevada decorrente de apenas uma mensagem de prioridade elevada localizada no fim da fila de dado ator.

   A vantagem desta abordagem é que o árbitro permanece eqüitativo, dado que todas as mensagens são entregues.

2. O árbitro altera a ordem de entrega de mensagens, ordenando-as segundo o seu grau de prioridade, e envia uma mensagem ao escalonador informando-o da inversão. As vantagens desta abordagem são:

   - A inversão é resolvida com muito maior rapidez, dado que as mensagens pendentes são contornadas;
   - Apenas duas mensagens são trocadas com o escalonador: uma para solicitar a herança de prioridade e outra para desfazê-la, ao invés $2n$ do caso anterior onde n é o número de mensagens pendentes com prioridade inferior.

   Sua desvantagem potencial é que a eqüidade do árbitro poderia ser violada, se um algorítmo adaptativo não fosse utilizado para mantê-la.

O principal objetivo na garantia da eqüidade do árbitro do sistema de entrega de mensagens é a satisfação da hipótese de tempo de entrega finito de uma mensagem. Um ponto importante a se notar é que se está influenciando o sistema de entrega apenas no final de todo o processo de tramitação de uma mensagem, que pode haver passado por sistemas de transmissão com tecnologias completamente diversas, e sofrido um atraso finito, mas arbitrariamente longo. Contudo, para atores pertencentes a um mesmo sistema, implementado sobre um conjunto de processadores fortemente acoplados, a comunicação se dá de forma bastante rápida, em relação à velocidade de processamento das tarefas e, portanto, mesmo localmente a abordagem do problema é válida.

O algoritmo adaptativo para tratar da inversão de prioridades, semelhantemente aos algorítimos usados no processamento de tarefas, usa duas prioridades associadas a cada mensagem: a sua prioridade intrínseca $Pi(m)$, ou seja, a real prioridade da mensagem no momento do seu envio e uma prioridade de entrega, $Pe(m)$. As mensagens são ordenadas de acordo com a regra a seguir:

1. O valor inicial da prioridade de entrega é igual ao da sua prioridade intrínseca;

2. A prioridade de entrega de uma dada mensagem é incrementada de uma unidade, a cada vez que uma outra lhe é considerada superior;

3. Ao chegar uma nova mensagem, o valor de sua prioridade intrínseca é comparada com os valores das prioridades de entrega das mensagens existentes. A sua ordem de entrega será então imediatamente após todas as mensagens cujas prioridades de entrega sejam maiores ou iguais a sua prioridade intrínseca.

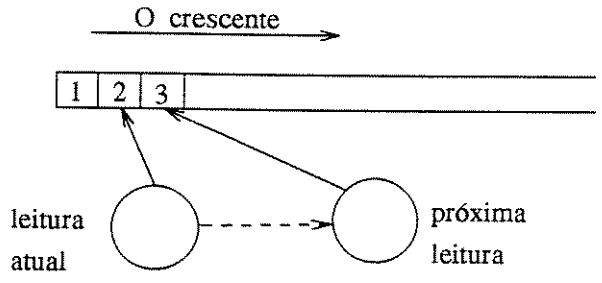


Figura 3.11 Retirada de mensagens da fila.

O restante desta seção será dedicado à prova que se a fila de mensagens de um ator é finita, então as regras acima implicam na entrega de todas as mensagens. As mensagens ordenadas pela função O estão dispostas de forma que os menores valores de O indicam as mensagens com maior prioridade de envio.

A recepção é modelada como um contador cujo valor é incrementado a cada nova mensagem retirada da fila por um ator, conforme ilustrado na Figura 3.11.

Teorema 1 (Árbitro Eqüitativo) *Sejam $O^k(m_i)$ e $Pe^k(m_i)$ a ordem de entrega e a prioridade de entrega, respectivamente, da mensagem m_i, após a chegada da k-ésima mensagem ao árbitro, e U_j^k o conjunto de índices das mensagens presentes na fila após a retirada da j-ésima mensagem e chegada da k-ésima mensagem na fila. Se o comprimento da fila de mensagens a serem lidas por um ator não cresce indefinidamente, e o número de níveis de prioridades das mensagens é finito, então o ordenamento das mensagens que chegam ao ator de acordo com a construção definida pelas funções O e Pe, a seguir, implica na entrega de todas as mensagens.*

inicialização:

$$\begin{aligned} O^1(m_1) &= 1 \\ Pe^1(m_1) &= Pi(m_1) \end{aligned}$$

retirada da fila: *seja i_{rec} o valor do contador de recepção, então a mensagem m_i a ser retirada é aquela com $O(m_i) = i_{rec}$ e:*

$$U^k_{i_{rec}} = U^k_{i_{rec}-1} - \{i\}$$

colocação na fila:

$$U^{k+1}_{i_{rec}} = U^k_{i_{rec}} \bigcup \{k+1\}$$

reordenamento das mensagens já existentes:

$$O^{k+1}(m_i) = \begin{cases} O^k(m_i), & \forall i \in U^k_{i_{rec}} \mid Pe^k(m_i) \geq Pi(m_{k+1}) \\ O^k(m_i) + 1 & \forall i \in U^k_{i_{rec}} \mid Pe^k(m_i) < Pi(m_{k+1}) \end{cases}$$

ordem da mensagem que acabou de chegar:

$$O^{k+1}(m_{k+1}) = \min_{j \in U^k_{i_{rec}}} \{O^k(m_j) \mid Pe^k(m_j) < Pi(m_{k+1})\}$$

atualização das prioridades de entrega das mensagens existentes:

$$Pe^{k+1}(m_i) = \begin{cases} Pe^k(m_i), & \forall i \in U^k_{i_{rec}} \mid Pe^k(m_i) \geq Pi(m_{k+1}) \\ Pe^k(m_i) + 1 & \forall i \in U^k_{i_{rec}} \mid Pe^k(m_i) < Pi(m_{k+1}) \end{cases}$$

prioridade de entrega da mensagem que acabou de chegar:

$$Pe^{k+1}(m_{k+1}) = Pi(m_{k+1})$$

Prova:

A progressão do contador de recepção é garantida pela hipótese de não crescimento para infinito da fila de mensagens a serem lidas por um ator. Como o contador é incrementado a cada mensagem retirada da fila, uma mensagem m_i será recebida com certeza se a sua ordem $O^j(m_i)$ permanecer invariável após um $j \geq j_{max}$. Portanto, a prova estará completa se sempre existir um j_{max} finito.

Observe-se que $O^{k+1}(m_i)$ somente é incrementado a partir de $O^k(m_i)$, se a prioridade da nova mensagem $Pi(m_{k+1})$ for maior do que a sua prioridade de entrega $Pe^k(m_i)$, permanecendo no mesmo valor caso contrário.

Supondo que m_i permaneça indefinidamente na fila, então $O(m_i)$ será incrementado indefinidamente, o mesmo ocorrendo com $Pe(m_i)$. Isto significa que sempre haverá uma nova mensagem com $Pi > Pe(m_i)$, o que é impossível pela hipótese de que o número de níveis de prioridade é finito. Portanto, sempre haverá um j_{max} tal que $Pej_{max}(m_i) \geq Pi(m_j)$ para $j > j_{max}$. Este j_{max} é finito dado que o número de níveis de prioridade também o é. □

3.5 Mecanismos de Transparência

O conceito de serviço é muito importante no projeto da plataforma, pois todas as entidades tratadas, sejam elas agentes OSI, agentes TCP/IP, blocos funcionais TMN, ou entidades funcionais IN têm como objetivo prestar um serviço, seja este de gerência ou um serviço qualquer implementado na Rede Inteligente.

Com a preocupação de delimitar a fronteira de um serviço, define-se neste trabalho o conceito de *agregado de atores*. Um **agregado de atores** é um conjunto de atores que compartilham memória entre si, ou seja, são os atores que fazem acesso concorrente à MIB, ou os que compartilham o estado global em uma entidade IN. Os atores assim agrupados estão fortemente acoplados, tornando a comunicação entre eles extremamente rápida. Este acoplamento na implementação é fundamental, dado que em alguns casos, como na entidade física SN da IN, as entidades funcionais estão tão fortemente acopladas que algumas delas se tornam invisíveis a outras entidades funcionais localizadas em outras entidades físicas.

Quando a comunicação é realizada dentro do agregado, não há problema algum de transparência a ser resolvido, pois todos os recursos são disponíveis localmente. Contudo, ao ultrapassar a fronteira do agregado surgem problemas relacionados à distribuição e heterogeneidade, tais como: representação dos dados em diferentes máquinas; localização do ator em uma máquina distante; e se os atores são fixos a uma dada máquina uma vez criados.

Para uniformizar os conceitos a serem usados na criação de padrões de sistemas de processamento distribuído, a ISO criou o *Modelo de Referência para Processamento Aberto Distribuído*. Este modelo divide os mecanismos de transparência em:

Transparência de Acesso: que permite a interoperação entre arquiteturas computacionais e linguagens de programação heterogêneas. O principal aspecto ligado à transparência de acesso é a diferente representação dos dados em um ambiente heterogêneo;

Transparência de Persistência: que esconde a alocação ou desalocação de recursos, favorecendo o seu compartilhamento. Por exemplo, pode-se tratar um objeto como se este estivesse sempre ativo, deixando a cargo do sistema a tarefa de ativá-lo logo que uma solicitação chegue;

Transparência de Localização: que esconde de um objeto a localização dos objetos com os quais ele interage;

Transparência de Relocação: que esconde de um objeto o fato de que as interfaces dos objetos com os quais ele interage sofreram migração;

Transparência de Migração: que esconde de um objeto o fato de que ele mudou de localização;

Transparência a Falhas: que ou evita a falha de um objeto, ou esconde dele mesmo o fato de que falhou;

Transparência de Transações: é o provimento de um processo automático de refinamento da especificação computacional de um objeto que não possui controle de transação, transformando-a em uma outra especificação contendo mecanismos de transação;

Transparência de Replicação: que mascara o uso de um grupo de objetos, fornecendo uma interface única comum a todos eles.

Destes mecanismos de transparência, os únicos incorporados na plataforma proposta foram os de acesso e localização. Isto no que diz respeito à troca de informações entre atores pertencentes a agregados diferentes, dado que os atores que estão dentro de um mesmo agregado compartilham a mesma representação dos dados e sabem da existência uns dos outros através da troca de mensagens e do conceito de *continuação*. Uma **continuação** é que a colocação dentro de uma mensagem do endereço para o qual uma eventual resposta deve ser encaminhada.

3.5.1 Transparência de Localização

Tecnicamente, nada impede que se estabeleça um identificador global para todos os atores dentro de um sistema, mas em um contexto no qual as interações são realizadas entre entidades que são implementadas como um agrupamento destes, esta identificação é desnecessária. Portanto, a unidade de identificação global dentro do sistema será um agrupamento de atores que fornecem um determinado serviço.

A incorporação de transparência de localização foi realizada através do uso do serviço de diretório **quipu**, que é parte integrante do ambiente ISODE. Neste diretório, as aplicações que desejam oferecer um serviço são registradas e as que desejam utilizá-lo executam uma consulta para obter o endereço. O padrão de endereçamento usado é bastante flexível, especificando se se trata de um endereço X.25, Internet, ISDN, etc.

Ao implementar um agregado, o projetista define um dos atores como sendo o responsável pelas comunicação através da fronteira do agregado, que doravante será referenciado como **recepcionista**. Este ator, então, registra o serviço que o agregado presta junto ao diretório, e estabelece qual o ator (se nao for o próprio) que recebe novas requisições dirigidas ao agregado.

Ao incorporar a definição de serviço, a plataforma adquire a propriedade de *abertura ao mundo exterior*[16], pois para que uma outra entidade se utilize de algum dos serviços implementados na plataforma, esta entidade não precisa também ter sido implementada no mesmo ambiente. Basta que se utilize do par <*serviço, protocolo*> que define o serviço. Como este par é padronizado em todos os ambientes de que trata este trabalho, esta é uma propriedade muito importante.

3.5.2 Transparência de Acesso

Quanto à transparência de acesso a questão não se resolve de maneira tão simples quanto a anterior. Somente um determinado ator tem a capacidade de especificar exatamente o conteúdo das mensagens que recebe. A infraestrutura não pode fazer isto sozinha, pois uma cadeia qualquer de bits pode significar um número de ponto flutuante, um vetor de inteiros, ou ainda uma cadeia de caracteres.

O ambiente ISODE fornece um conjunto de funções que implementam a representação canônica ASN.1, que foi projetada exatamente para tornar transparente a representação dos dados em diferentes ambientes. Esta notação, usada em conjunto com as regras BER padronizam a conversão dos dados, mas não evitam que a entidade realizando a comunicação tenha de saber quando esta conversão deve ser feita, pois as funções devem ser chamadas explicitamente.

Visando simplificar esta tarefa, foi desenvolvido no âmbito deste trabalho um compilador de interfaces usando-se o recurso de *funções virtuais*[17] e *classes abstratas*. Antes deste compilador, uma estrutura de dados XXX a ser transmitida era especificada em ASN.1, e posteriormente compilada, gerando uma estrutura de dados e um conjunto de funções *encode_XXX* e *decode_XXX*, e estas eram chamadas a cada nova estrutura recebida, fosse ela local, ou distante.

O novo compilador desenvolvido *lê* a estrutura gerada e define uma sub-classe da classe *interface* usada na aplicação. Como esta super-classe contém os métodos *encode* e *decode*, a função *accept* [18] do ator pode realizar a conversão automaticamente.

[16] "Openness."
[17] "Virtual functions."
[18] Esta é a função de recepção de mensagens por um ator. Ver Seção 3.2.5.

Capítulo 4

Sistemas de Atores e sua Semântica

Um **Sistema de Atores** é um conjunto de atores que interagem entre si para realizar uma tarefa comum. A Teoria de Atores foi desenvolvida pelo Grupo de Passagem de Mensagens do MIT como resultado da exploração de uma teoria básica para sistemas concorrentes, criando um conjunto de primitivas e algumas leis que regem o comportamento de tais sistemas [36].

Atores e mensagens são os dois únicos tipos de objetos dos sistemas de atores. O Sistema evolui através do envio de mensagens entre atores, os quais por sua vez enviam mais mensagens. Atores podem ser criados no decorrer do processamento, e seus nomes podem ser enviados em mensagens. Conseqüentemente, tipos abstratos de dados podem ser modelados com grande eficácia em atores, que recebem mensagens indicando as operações que devem efetuar sobre si mesmos [8].

O envio de mensagens assincrono e o seu processamento concorrente é a chave do modelo de atores na exploração do paralelismo que pode ser oferecido pela infraestrutura.

A Teoria de Atores é capaz de expressar outros paradigmas que podem ser usados para explorar a concorrência em sistemas distribuídos, como as funções com avaliação retardada de parâmetros baseada em λ-cálculo. Isto foi provado em [35] onde se mostrou que o modelo de λ-cálculo poderia ser imerso em uma subconjunto da teoria de atores, dado que o critério de continuidade ao qual todas as funções devem satisfazer é atendido por qualquer função que possa ser desenvolvida em um sistema de atores.

Em [8], são explorados vários problemas associados à utilização da Teoria de Atores como base para o projeto de sistemas computacionais, estabelecendo axiomas que devem ser satisfeitos por qualquer sistema de troca de mensagens fisicamente realizável.

Objetivando o desenvolvimento da Teoria de Atores na área da semântica denotacional, produziu-se em [19] um modelo matemático que satisfaz as leis de ordenamento postuladas na teoria. Mostrou-se através de um axioma de *realizabilidade em tempo global* que sempre é possível construir uma referência de tempo global, embora esta referência não seja única, e sim uma janela de referência associada a um observador.

Também mostrou-se em [19] que dos diagramas de eventos criados [32] para a *semântica comportamental*[1] acrescidos dos eventos pendentes criados em [8] formam um domínio incompleto. Sobre este domínio construiu-se um *domínio de potência* para obter-se uma *semântica de ponto fixo* para a Teoria de Atores. Este é o modelo semântico usado nesta tese e, por conseguinte, uma apresentação de seus principais postulados será realizada nas próximas seções.

[1] "Behavioral Semantics."

Um outro modelo operacional, para a semântica de linguagens baseadas em atores, foi desenvolvido em [3]. Neste modelo são utilizadas as transições realizadas sobre *configurações* para descrever o comportamento de um sistema de atores. As **configurações** funcionam como uma fotografia do sistema em um dado instante. Em [3], também foram definidos vários construtores de mais alto nível aumentando o poder de expressividade das linguagens baseadas em atores.

Antes da apresentação das principais leis que integram a teoria de atores e do modelo semântico que lhe foi associado, são discutidos alguns conceitos relativos a sistemas concorrentes, como indeterminismo e eqüidade, e incluídos alguns conceitos básicos necessários à compreensão do modelo semântico.

4.1 Eqüidade

O assunto *eqüidade* já foi abordado na Seção 3.4.6 para provar propriedades inerentes ao sistema de entrega de mensagens. A preocupação com a eqüidade surge exatamente em situações nas quais existe um árbitro que deve decidir acerca da próxima ação a ser executada dentre um conjunto de ações pendentes.

Considere-se, por exemplo, o acesso a disco requerido por vários processos concorrentes. Se estes forem servidos por ordem de chegada, garante-se que todas as requisições são, em algum momento satisfeitas, mas isto não pode implicar no movimento da cabeça de leitura para posições distantes umas das outras, o que tomaria muito tempo.

Por outro lado, se o acesso que gerar o menor movimento da cabeça de leitura for sempre escolhido, minimiza-se o tempo de resposta, mas o algoritmo é desigual, pois existe a possibilidade de um pedido nunca ser atendido.

A semântica clássica de domínios de potência baseia-se na evolução do sistema por meio de transições realizadas sobre o seu estado, e na definição do próximo estado como uma expansão do estado atual considerando-se todas as possibilidades geradas pelo não-determinismo. Isto implica na adoção de *pontos de escolha*[2] a cada geração de um novo estado. O ponto de escolha é representado pela consulta a um oráculo[3] que decide acerca do próximo evento não-determinístico gerado.

É exatamente esta consulta ao oráculo que causa problemas quando se tenta especificar uma estratégia eqüitativa, como a FIFO, por exemplo. Na Figura 4.1, o ator A_1 envia uma mensagem O ao ator A_3 e o ator A_2 envia uma seqüência infinita de mensagens todas contendo 1 a A_3. Sendo eqüitativo, em algum momento o árbitro deve entregar a mensagem contendo o O a A_3. Isto significa que a seqüência 1^ω é impossível.

Como na semântica baseada em domínios de potência, o oráculo é consultado a cada nova transição e todo o espectro de possibilidades é considerado, (ver Figura 4.2) a saída 1^ω pode ocorrer. Como conseqüência, nenhum árbitro eqüitativo pode ser representado usando-se a semântica convencional de domínios de potência. Este problema é conhecido como o *problema da combinação eqüitativa*[4].

---

[2] "Choice Points."

[3] Entidade externa ao processo que retorna amostragens independentes (sem memória) dentro de um espaço amostral limitado.

[4] "Fair Merge Problem".

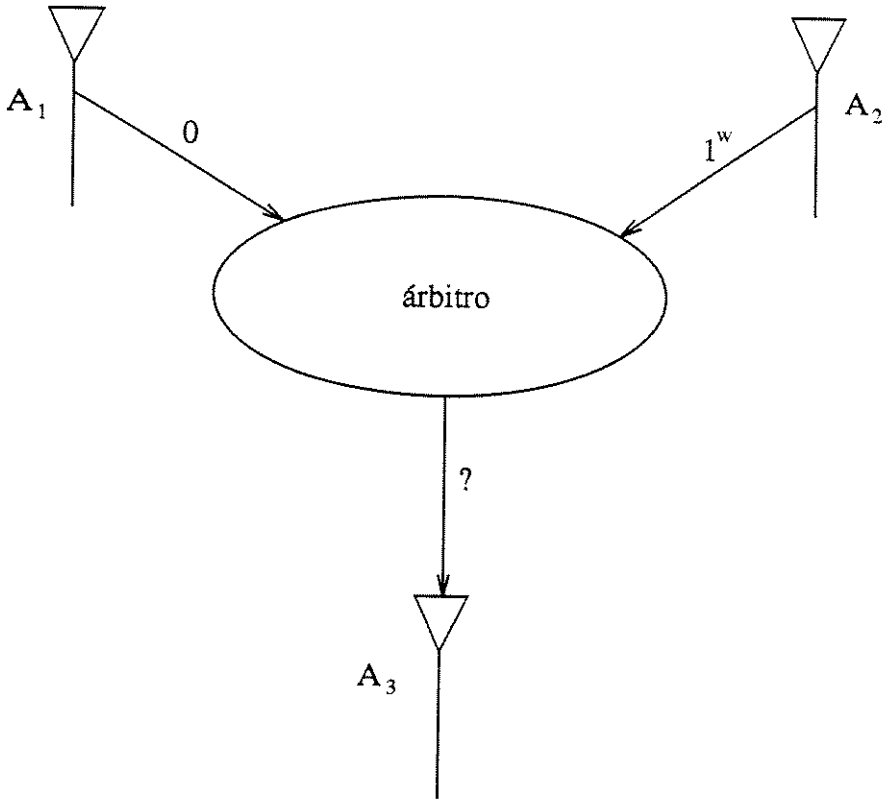


Figura 4.1 O árbitro eqüitativo entrega a mensagem O em algum momento.

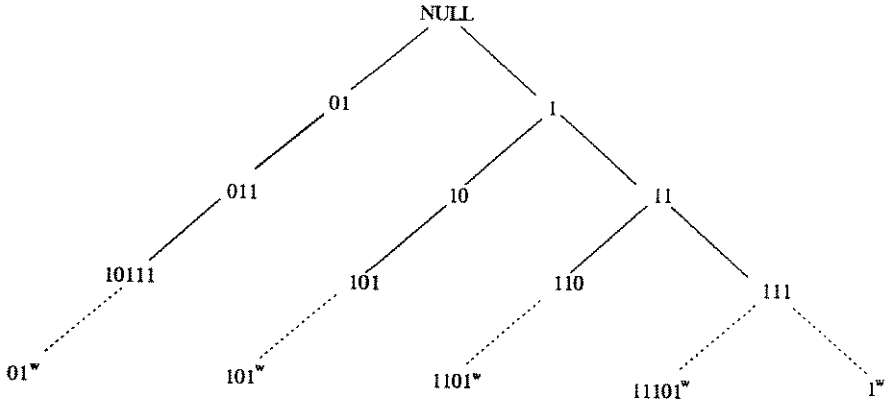


Figura 4.2 Árvore de escolha do árbitro.

Um exemplo de linguagem cuja semântica é baseada em domínios de potência é CSP. Nesta linguagem, o comando *select* pode ser usado para esperar não deterministicamente por uma mensagem proveniente de um grupo de processos. O *select* é composto de um ou mais comandos *guard*, que consistem em expressões booleanas conjugadas a uma espécie de requisição de comunicações [9]. O *select* bloqueia a execução até que todos os seus comandos *guard* falhem, ou algum deles seja verdadeiro. No primeiro caso, o comando inteiro é tornado sem efeito, enquanto no segundo, um dos *guard* é escolhido não-deterministicamente.

É exatamente neste último caso que a semântica de domínios influi, pois nenhuma hipótese pode ser feita acerca do *guard* selecionado. Uma execução repetidas vezes pode de fato escolher o mesmo *guard* sempre, ainda que haja outros habilitados. Como a eqüidade não faz parte da semântica da linguagem, nada se pode afirmar sobre o seu comportamento neste aspecto.

Algumas implementações podem ser razoavelmente eqüitativas, garantindo que um *guard* será escolhido em um número finito de iterações, ou dando chances iguais a todos os *guards*. Por outro lado, outras implementações podem adotar a avaliação dos *guards* sempre seqëncialmente e escolher o primeiro habilitado.

Como as linguagens baseadas no modelo de atores devem ser capazes de expressar eqüidade na escolha de eventos não-determinísticos, domínios de potência convencionais não são adequados como base para o desenvolvimento de seu modelo semântico. O que se fez, então, foi o desenvolvimento de um domínio de potência não convencional capaz de lidar com paralelismo eqüitativo.

4.2 Não-determinismo

A introdução de não-determinismo em linguagens concorrentes difere de escolha aleatória feita em pontos de escolha. Uma prova disto é que não se pode expressar eqüidade quando estes pontos são usados para expressar o não determinismo.

Em muitas linguagens, como CSP, por exemplo, a adoção de pontos de escolha para modelar não-determinismo ocorreu, porque estes reduzem o problema da elaboração de uma semântica capaz de lidar com não-determinismo ao problema da generalização da teoria de semântica existente para programas seqüenciais de forma a habilitá-la a tratar pontos de escolha [19]. Contudo, se esta teoria semântica produz resultados diferentes de teorias verdadeiramente baseadas em concorrência, então a idéia de pontos de escolha como modelo de concorrência deve ceder. Isto é o que realmente ocorre [19] quando se tenta reproduzir eqüidade.

O ponto principal entre ponto de escolha e eqüidade está na diferença entre em objetos recursivamente enumeráveis e *objetos finitamente geráveis* [80]. Os objetos recursivamente enumeráveis presumem eqüidade, isto é, qualquer objeto pertencente à família de objetos recursivamente enumeráveis deve ser gerado em algum momento. Os objetos finitamente geráveis são o resultado da introdução da consulta a um oráculo em determinados pontos de escolha.

A obrigatoriedade da geração em algum momento dos objetos recursivamente enumeráveis implica na existência de uma *memória* inerente ao processo de recursão que armazena quais objetos ainda não foram gerados [5]. Esta memória não é necessária ao oráculo, que pode considerar cada nova consulta como um evento independente.

[5] Ou o seu complementar, o conjunto de objetos gerados

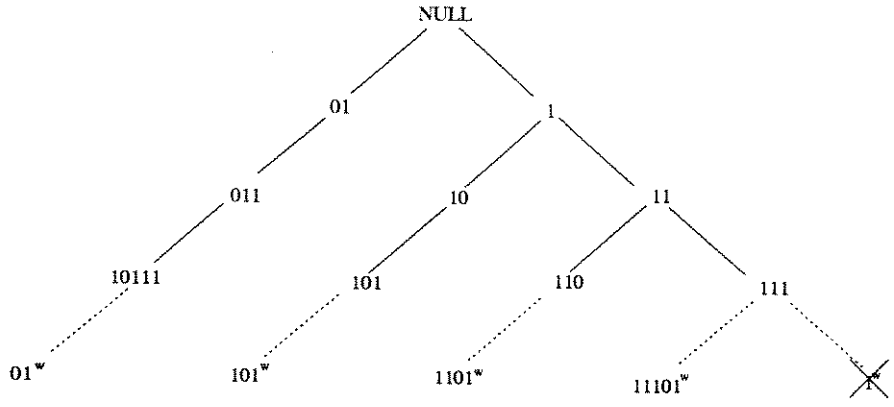


Figura 4.3 Árvore de escolha para não-determinismo ilimitado.

Os objetos recursivamente enumeráveis devem ser gerados obrigatoriamente, mas o instante em que isto acontece, embora finito, pode ser ilimitado. Este fato é conhecido como **não-determismo ilimitado** [19]. Observando-se novamente a árvore de escolha do árbitro, tem-se que para o não-determinismo ilimitado, a solução1^{ω} foi eliminada, conforme ilustra a Figura 4.3.

É importante notar que embora programar um árbitro eqüitativo em uma dada linguagem seja impossível, pode ainda ser possível programá-lo nesta linguagem de forma que o seu comportamento seja razoavelmente eqüitativo [19]. Na prática, implementações podem ser bastante eqüitativas. O fato de que o exame da semântica de uma linguagem de programações mostre que um árbitro eqüitativo não pode ser escrito na linguagem revela não uma deficiência da linguagem, mas da teoria corrente de semântica de linguagem de programação.

4.3 Construção de Uma Semântica Denotacional para Atores

A teoria das linguagens de programação e suas técnicas formais associadas tem suas raízes em áreas do conhecimento tão diversas quanto a linguística e a lógica formal.

A teoria denotacional das linguagens de programação sempre tenta encontrar objetos matemáticos para representar o que um programa faz [76, 77, 60]. Exemplos de objetos comumente usados são funções parciais, seqüenciais de estados e, no caso do modelo semântico de Atores, os diagramas de eventos de atores, que serão abordados posteriormente.

O principal problema a ser abordado no mapeamento de um programa à função que este realiza é o das *definições circulares* [11], que surgem, por exemplo, em procedimentos recursivos.

Então, parte do trabalho a ser realizado pela teoria semântica é garantir a existência de solução para tais **definições circulares** e determinar qual delas será a adotada quando existirem múltiplas soluções [11].

A existência da solução é abordada pelo Teorema de Ponto Fixo, a seguir:

Teorema 2 (Ponto Fixo [80]) *Seja t uma função total que mapeia código em código de funções parciais recursivas. Então, é sempre possível encontrar um inteiro p tal que $\Phi_p = \Phi_{t(p)}$. A função Φ_p é chamada de* ponto fixo de t.

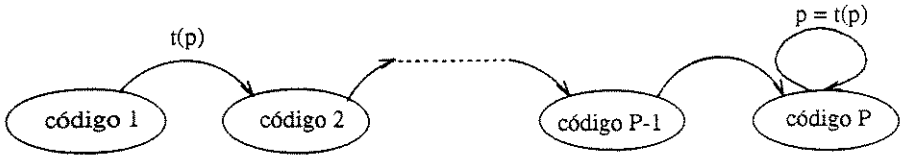


Figura 4.4 Busca pelo ponto fixo.

A idéia básica do ponto fixo é mostrada na Figura 4.4, onde o mapeamento entre códigos é realizado até que o valor p seja atingido. Quando isto ocorre, a função $t(p)$ retorna p como o próximo código e estabiliza.

Resolvido o problema da existência de uma solução, resta ainda determinar pelo menos uma delas e qual adotar no caso de multiplicidade. Isto é realizado através de *reticulados*, que são conjuntos nos quais os elementos possuem uma ordenação entre si. Quando aos reticulados é conjugada uma função monotônica, encontra-se o **menor ponto fixo**[6]. Este ponto corresponde ao menor limitante superior do espaço de aproximações resultante da aplicação da função parcial e, geralmente, é o ponto mais facilmente determinável.

Teorema 3 (Pontos Fixos em Reticulados [80].) *Sejam $\langle A, \leq \rangle$ um reticulado completo e $f : A \rightarrow A$ uma função monotônica. Se P é o conjunto de todos os pontos fixos de f, então o conjunto P é não-vazio, e o sistema $\langle P, \leq \rangle$ é um reticulado completo. Em particular, tem-se $P \subseteq \{x \in A \mid x \leq f(x)\}$ com:*

$$\bigvee P = \bigvee \{x \in A \mid x \leq f(x)\},$$

e $P \subseteq \{x \in A \mid x \geq f(x)\}$ com

$$\bigwedge P = \{x \in x \mid x \geq f(x)\}.$$

4.3.1 Pontos Fixos em Ordenamentos Parciais Completos

O Teorema 3 caracteriza o comportamento de uma função monotônica em um reticulado completo. Porém, para cômputos envolvendo objetos infinitos, é necessário mais do que a monotonicidade [80]: é preciso garantir a continuidade das funções e definir o seu comportamento na fronteira do domínio. Uma das abordagens para este problema é introduzir a noção de *reticulado contínuo* [70], que permite a construção recursiva do domínio ($D_{n+1} = [D_n \rightarrow D_n]$) em um processo conhecido como construção limite inversa e a sua imersão no domínio de potência dos números naturais $P\omega$ [71]. Outra abordagem é a utilização de ordenamentos parciais completos, ao invés de reticulados contínuos. Esta é a forma adotada em [19], e será mostrada a seguir.

Definição 1 (Função Contínua em ω) *Uma função f é contínua em ω se, e somente se é monotônica e preserva todos os menores limitantes superiores de seqüências contáveis crescentes, tal que se $\{x_i\}_{i \in \omega}$ é uma seqüência em D com $x_i \leq x_{i+1}$ para todo $i \in \omega$ então:*

$$f(\bigvee_{i \in \omega} x_i) = \vee'_{i \in \omega} f(x_i).$$

[6] "Least fixed-point."

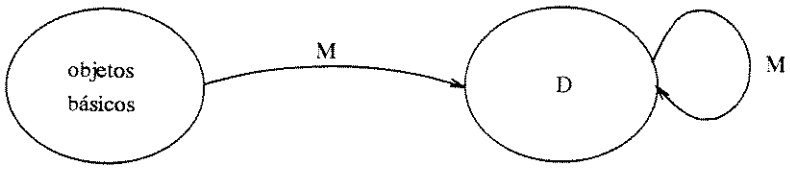


Figura 4.5 Definição do significado na Semântica Denotacional.

Dado que a avaliação de continuidade é realizada sobre conjuntos parcialmente ordenados, e não sobre reticulados, é importante observar que esta definição não requer que todas as seqüências crescentes tenham menores limitantes superiores, mas preserva todos os que existirem.

Definição 2 (Ordenamento Parcial em ω) *Um ordenamento parcial $\langle D, \leq \rangle$ é completo em ω se, e somente se cada conjunto dirigido contável tem menor limitante superior em D. Isto é, se $x_{i \in \omega} \in D$ com $x_i \leq x_{i+1}$, então $\bigvee_{i \in \omega} x_i$ existe.*

Tendo estabelecido as condições para a continuidade em ω, agora é possível garantir a existência do último ponto fixo.

Teorema 4 (Existência do Último Ponto Fixo [19]) *Suponha que $\langle D, \leq \rangle$ possui o menor elemento $\perp$ e é completo em ω. Então, qualquer função $f : D \rightarrow D$ contínua em ω tem um ponto fixo dado por $\bigvee_{i \in \omega} f^i(\perp)$, e este ponto fixo é o menor entre todos os pontos fixos de f.*

Conclui-se, assim, a base da semântica de ponto fixo. Para obter-se o significado M de um programa P, a primeira tarefa é definir a classe de objetos básicos cuja semântica será definida. A partir deste conjunto básico D, extrai-se recursivamente o significado de construções mais complexas (ver Figura 4.5). Nesta extração, a busca por um ponto fixo será necessária sempre que houver definições circulares.

4.3.2 Domínios de Potência

A base conceitual da semântica denotacional de pontos fixos não é suficiente para abordar o não-determinismo existente na teoria de Atores. Em linguagens que se utilizam de pontos de escolha para modelar não determinismo, é suficiente definir um *domínio de potência* a partir de um domínio completo que é usado como base semântica [19]. Na Figura 4.6, estão ilustradas as formas clássicas de construção do modelo semântico. a primeira delas é a realizada para o caso de execução seqüencial, onde são usados pontos fixos em um domínio básico completo. A segunda expande o modelo anterior para tratar não-determinismo com pontos escolha, e a terceira é a forma usada na construção de um modelo semântico denotacional para a Teoria de Atores.

Neste último caso, ao invés de se elaborar o domínio de potência a partir de um domínio completo, extende-se a construção de domínios de potência para domínios incompletos, e mostra-se que para tais domínios, o domínio de potência definido pela extensão é isomórfico ao definido convencionalmente a partir da completação em ω do domínio original [19]. Nesta subseção, serão enunciados os principais teoremas e definições que possibilitam a construção do domínio de potência a partir de um domínio incompleto. Como se trata de um desenvolvimento voltado para o Modelo de Atores, seguir-se-á de perto o desenvolvimento realizado em [19].

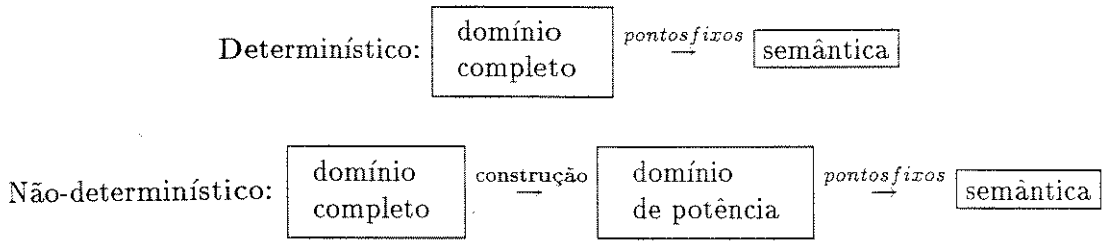


Figura 4.6 Formas clássicas de construção do modelo semântico denotacional.

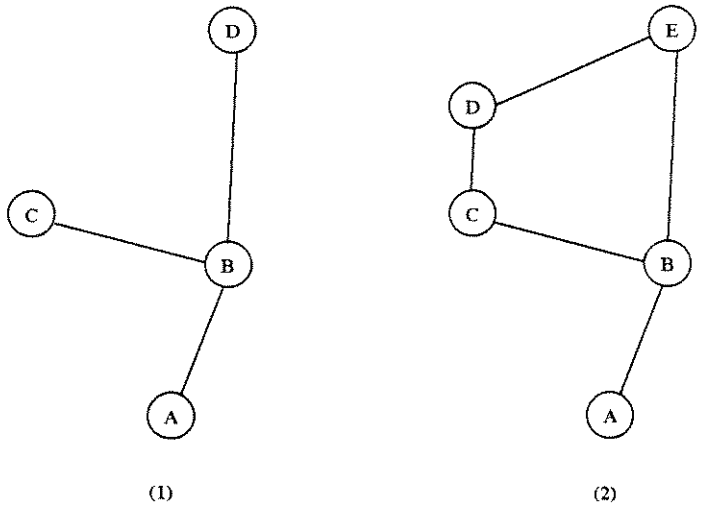


Figura 4.7 Elementos isolados em um conjunto parcialmente ordenado.

Um elemento $x \in D$ de um conjunto parcialmente ordenado $\langle D, \leq \rangle$ é **isolado** se, e somente se deve-se passar por ele para chegar a um elemento que lhe é superior através de um processo limite, ou seja, se sempre que $A \subseteq D$ é dirigido, existe $\bigvee A$ e $x \leq \bigvee A$, é garantida a existência de um $a \in A$ com $x \leq a$. Um exemplo de conjunto no qual todos os elementos são isolados é o conjunto dos números racionais no espaço dos números reais.

Na Figura 4.7, todos os elementos do conjunto (1) são isolados, enquanto no conjunto (2), apenas A e B o são, pois pode-se chegar a E sem passar por C e D.

Definição 3 (Fechamento) *O fechamento de $A \subseteq D$ é:*

$$A^c = \{d \in D \mid (\exists X \subseteq D, X \textit{ dirigido}, d = \bigvee X, \forall x \in X \exists a \in A \mid x \leq a)\}.$$

Definição 4 (Domínio [19]) *Um domínio é um conjunto parcialmente ordenado $\langle D, \leq \rangle$, tal que:*

1. *D possui o elemento mínimo $\perp$;*
2. *Qualquer elemento de D é um menor limitante superior de uma seqüência contável crescente de elementos isolados;*
3. *Os elementos isolados de D são contáveis.*

Esta definição de domínio não é padrão, pois normalmente se requer que D seja completo em ω, de forma que funções contínuas em ω tenham pontos fixos no mapeamento, $f : D \rightarrow D$. Contudo, o domínio de potência é definido a partir da completação de D.

Definição 5 (Compleição de um Domínio) *Seja $\langle D, \leq \rangle$ um domínio com elemento mínimo $\perp$. Sua completação é $\langle \overline{D}, \sqsubseteq \rangle$, onde:*

$$\overline{D} = \{A^c \mid \perp \in A \subseteq D, A \text{ dirigido}\}$$

e para todo $A, B \in \overline{D}$:

$$A \sqsubseteq B \Leftrightarrow A \subseteq B.$$

$\langle \overline{D}, \sqsubseteq \rangle$ é um ordenamento parcial, mas geralmente não é um reticulado.

O teorema a seguir define o menor limitante superior para um conjunto dirigido em termos de cada um dos elementos existentes na completação $\overline{D}$. Neste teorema, $\bigcup X$ é o conjunto formado pela união dos elementos da classe X.

Teorema 5 (Menor Limitante Superior em $\overline{D}$ [19]) *Se $X \subseteq \overline{D}$ é dirigido, então X tem menor limitante superior em $\overline{D}$ dado por:*

$$\bigsqcup X \triangleq (\bigcup X)^c.$$

O Teorema 5 permite determinar o menor ponto fixo no domínio semântico dos Atores.

A definição de domínio de potência a seguir completa a base semântica no que diz respeito ao não-determinismo. Para completar a definição da semântica denotacional para a Teoria de Atores, basta definir os elementos básicos necessários: quem será o domínio, qual a sua função de significado, qual o ordenamento utilizado para os elementos do domínio e como são definidos os pontos fixos.

Definição 6 (Domínio de Potência [19]) *Seja $\langle D, \leq \rangle$ um domínio com elemento mínimo $\perp$. O seu domínio de potência é $\langle P[D], \sqsubseteq \rangle$, onde:*

$$P[D] = \{A^c \mid \perp \in A \subseteq D\}$$

e para $A, B \in P[D]$,

$$A \sqsubseteq B \Leftrightarrow A \subseteq B.$$

O domínio de potência $\langle P[D], \sqsubseteq \rangle$ é um reticulado completo contínuo de base contável, ou seja, é um domínio completo em ω, no qual qualquer subconjunto X possui tanto o menor limitante superior $\bigsqcup X$ quanto o maior limitante inferior $\sqcap X$ [19].

4.3.3 Definição de um Domínio para Atores

A chegada de uma mensagem a um ator representa o evento que é tomado como base para a descrição do comportamento do sistema. Este é o único evento observável, e não existe o evento *envio de uma mensagem*. Uma outra suposição é que as mensagens são sempre entregues, mas

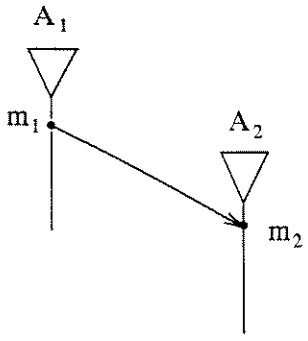


Figura 4.8 Ordenamento de ativação.

estão sujeitas a atrasos variáveis dependendo da rota que tomam em direção ao destino. Esta hipótese é denominada *atraso finito.*

Um ator, ao receber uma mensagem, pode tomar algumas decisões locais, criar novos atores e enviar um certo número de mensagens. Conseqüentemente, a chegada de uma mensagem m_2 em um ator A_2 está ligada ao evento que representa a chegada de uma mensagem m_1 ao ator de origem A_1, conforme ilustra a Figura 4.8. Esta ligação recebe o nome de **ordem de ativação**[7], representada por $m_1 \xrightarrow{atv} m_2$.

Como conseqüência da recepção de m_2, o ator A_2 pode enviar novas mensagens e assim sucessivamente, criando uma série de ativações em cadeia. Na Figura 4.9, m_1 ativa m_3, que por sua vez ativa m_4. Conseqüentemente, a ordem de ativação cria um ordenamento parcial, pois permite relacionar m_1 e m_4, mas não relaciona m_4 com m_2.

Algumas vezes, um evento não ativa novos eventos, apenas causa uma mudança local no ator que o recebe. Para este ator, a relação entre os eventos de acordo com a sua ordem de chegada é importante, pois ele pode influenciar eventos futuros.

Considere-se, por exemplo, um ator que representa uma conta bancária (ver Figura 4.10). Se uma mensagem de retirada chegar antes dos depósitos que iriam cobrí-la, isto causará um saldo negativo, o que pode não ser aceito. Em suma, a ordem de chegada de mensagens a um ator é importante devido à alteração que esta pode causar ao seu estado privado. Este ordenamento é denominado **ordenamento de chegada**[8], representado pr depósito $\xrightarrow{rec}$ retirada.

Ao juntar-se estes dois ordenamentos, obtém-se um terceiro ordenamento denominado **ordenamento combinado** [9], ilustrado na Figura 4.11, onde os eventos *depósito1* e *entrega do dinheiro* estão relacionados através deste ordenamento, enquanto o *depósito2* não possui nenhuma relação com a *entrega do dinheiro.* O ordenamento combinado de *depósito1* com *entrega do dinheiro* é representado por depósito $\longrightarrow$ entrega.

O ordenamento combinado é similar a ordenamento de correspondência de alguns outros modelos, mas a sua decomposição em ordenamentos de ativação e chegada foi realizada somente no Modelo de Atores. Leslie Lamport utilizou um *ordenamento causal* para resolver problemas de sincronização em sistemas distribuídos [55]. Para compensar a ausência do ordenamento de chegada, introduziram-se os conceitos de *múltiplos relógios físicos* e *múltiplos relógios lógicos.*

[7] "Activation ordering".
[8] "Arrival ordering".
[9] "Combined Ordering".

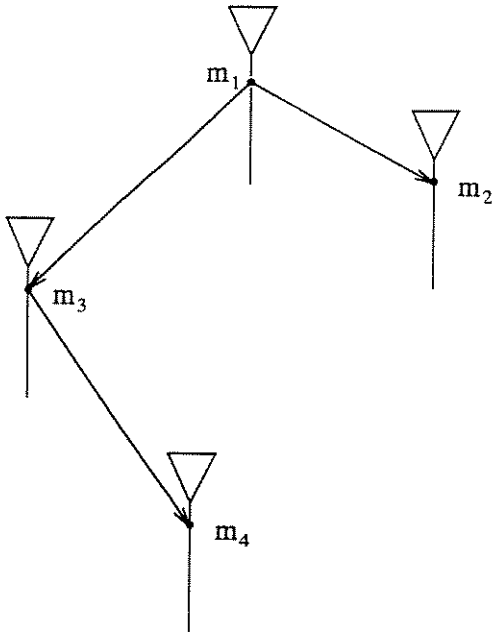


Figura 4.9 Ordenamento parcial de ativação.

Figura 4.10 Ordenamento de chegada.

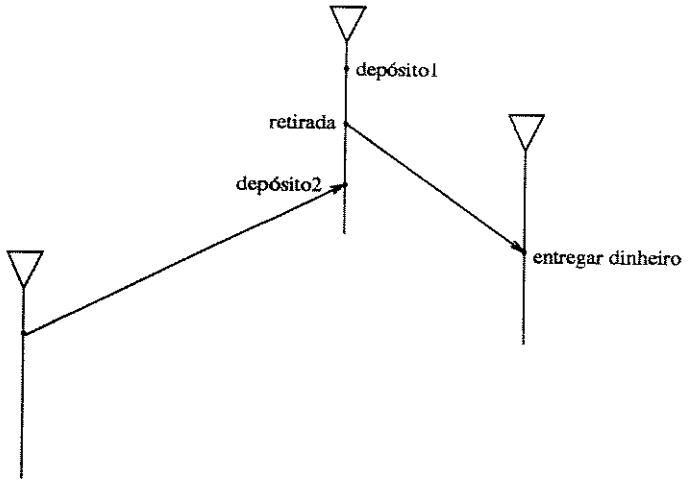


Figura 4.11 Ordenamento combinado.

Desta forma, o ordenamento causal[10] foi imerso em um esquema de ordenamento total. Desta maneira, as requisições de acesso à região crítica podem ser totalmente ordenadas e servidas em ordem FIFO.

A utilização de múltiplos relógios fornecem ao modelo de Lamport a garantia da realizabilidade física do ordenamento combinado. Esta garantia também é muito importante no modelo de atores, pois cada ator deve ter uma visão consistente dos eventos que ocorrem no sistema. É lógico que esta visão ocorrerá dentro de sua janela de referência, na qual eventos que não possuem ligação entre si através do ordenamento combinado podem ser observadas de maneiras diferentes por atores distintos.

Para garantir a realizabilidade do ordenamento combinado, algumas restrições foram impostas. Geralmente, a função que atribui seqüência aos eventos ordenando-os globalmente na janela de referência do ator é uma função inteira, como acontece com os relógios lógicos de Lamport, mas um conjunto mais geral de axiomas baseados no modelo de tempo global no conjunto dos números reais foi proposto em [5]

Em particular, os números inteiros também são usados para garantir a realizabilidade, mas o ponto de partida foi estabelecer uma função $g : E \rightarrow \mathbf{R}$, onde E é o conjunto de eventos e $\mathbf{R}$ o conjunto dos números reais, e posteriormente restringí-la de acordo com algumas regras [19]:

1. A causa precede o efeito, ou seja, g preserva o ordenamento de ativação. Então, se $e_1 \xrightarrow{atv} e_2$, $g(e_1) < g(e_2)$;
2. A ordem de chegada é preservada, ou seja, $e_1 \xrightarrow{rec} e_2 \Rightarrow g(e_1) < g(e_2$;
3. g preserva o ordenamento combinado, representado por $\longrightarrow$;
4. O ordenamento combinado é um ordenamento parcial irreflexivo. Esta é a **Lei da Causalidade Estrita**;
5. A imagem de g é um subconjunto não-negativo de $\mathbf{R}$.

Estas regras podem ser sumariadas no seguinte axioma:

Axioma 1 (Axioma Forte de Realizabilidade) *Existe um mapeamento biunívoco entre os eventos E e a função g no domínio dos números reais $\mathbf{R}$ não-negativos que preserva o ordenamento combinado $\longrightarrow$, tal que $g^{-1}(I)$ é finita para qualquer intervalo limitado I de $\mathbf{R}$. Equivalentemente, existe um mapeamento biunívoco $g : E \rightarrow \mathbf{N}$ que preserva o ordenamento combinado, onde $\mathbf{N}$ é o conjunto dos números naturais.*

Uma versão menos restritiva deste axioma é dada a seguir, onde não-negatividade do mapeamento foi eliminada. Aa razões para este abrandamento das restrições é permitir que sistemas em estado estacionário, como por exemplo, uma rede de computadores que tem estado em operação durante anos, possam ser expressos no modelo.

Axioma 2 (Axioma Fraco de Realizabilidade) *Existe um mapeamento biunívoco entre os eventos E e a função g no domínio dos números reais $\mathbf{R}$ que preserva o ordenamento combinado,*

[10] Que corresponde ao ordenamento de ativação no modelo de atores.

tal que $g^{-1}(I)$ é finita para qualquer intervalo limitado I de **R**. *Equivalentemente, existe um mapeamento biunívoco $g : E \rightarrow$ Z que preserva o ordenamento combinado, onde* **Z** *é o conjunto dos números inteiros.*

Embora a realizabilidade global no tempo seja uma necessidade para mostrar que o sistema é fisicamente factível, a sua utilização em provas sobre as propriedades do sistema não é prática. Isto se deve ao fato de que os axiomas de realizabilidade garantem a existência de uma solução, mas o problema é que pode haver multiplicidade da solução, ou seja, mais de uma possível linearização[11] consistente de eventos concorrrentes que não estão relacionados pelo ordenamento global.

Exemplificando, na Figura 4.9, a inversão no tempo pode ser feita de três maneiras diferentes: $[m_1, m_2, m_3, m_4]$, $[m_1, m_3, m_2, m_4]$ e $[m_1, m_3, m_4, m_2]$. Estas três formas não acrescentam nenhuma informação sobre o sistema e ainda escondem o fato de que m_2 não pode influenciar m_3 nem m_4 de forma alguma.

Teorema 6 *O Axioma Forte de Realizabilidade é equivalente à conjunção das seguintes leis:*

Lei da Causalidade Estrita: $\nexists e \in E \mid e \longrightarrow e$. *Nenhum evento pode ativar a si próprio;*

Lei da Contabilidade: *E é contável, isto é $E \subseteq (A \times N)$, tal que se $i \leq n$ e $\langle A, n \rangle \in E$, então $\langle a, i \rangle \in E$;*

Lei da Predecessão Finita: $\forall e_i \in E$, *o conjunto $\{e \mid e \longrightarrow e_i\}$ é finito.*

O Axioma Forte de Realizabilidade mapeia biunivocamente os eventos ocorridos no conjunto dos números naturais **N**. A Lei de Causalidade Estrita garante a consistência, do mapeamento, através da não-reflexividade. É interessante observar que a irreflexividade do ordenamento combinado não decorre da irreflexividade dos ordenamentos de chegada e ativação, como mostra o diagrama da Figura 4.12.

Nesta figura, existe a seqüência $e \longrightarrow e_1 \longrightarrow e_2 \longrightarrow e_3 \longrightarrow e$, ou seja, apesar de não haver reflexividade nos ordenamentos de chegada e ativação, no ordenamento combinado, $e \longrightarrow e$.

A Lei de Contabilidade garante que E pode ser mapeado em um conjunto contável, podendo ser ele o conjunto dos números naturais **N** ou o conjunto dos números internos **Z**, mas a Lei de Predecessão Finita garante que o mapeamento em **N** é realizável, pois não permite que se tenha de numerar os eventos que precederam um outro até $-\infty$, o que implicaria no conjunto **Z**.

A semântica desenvolvida em [19] para as construções metalingüísticas das linguagens baseadas em atores é uma formulação em termos de domínio de potência da *semântica comportamental* [32]. Nesta semântica, o significado de um programa é a especificação das atividades representadas formalmente por diagramas de eventos, como o da Figura 4.11.

O Diagrama Aumentado de Eventos em Atores

A evolução de um sistema de atores ocorre através do processamento de mensagens pendentes pelos atores existentes e da criação de novos atores. Esta criação, contudo, também está condicionada ao recebimento de mensagens, excetuando-se os atores iniciais ao processamento.

[11] Colocação em uma reta.

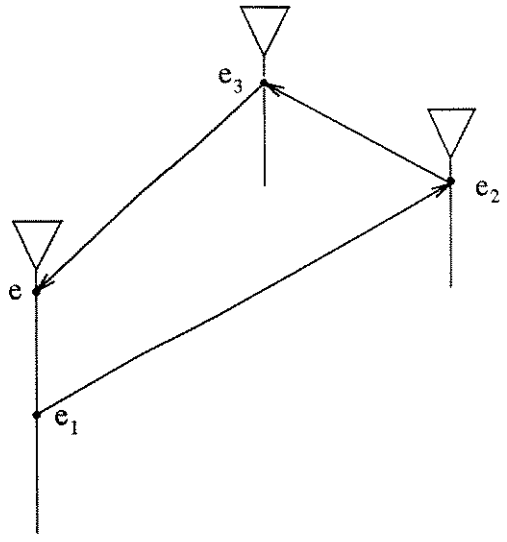


Figura 4.12 Reflexividade no ordenamento combinado.

Um ator é completamente determinado pelo seu nome e seu comportamento, que é o da atividade que ele vai executar ao receber uma mensagem. Ao receber esta mensagem, o ator pode mudar o seu estado privado, enviar um certo número de mensagens e especificar um novo comportamento substituto.

Um elemento do domínio do diagrama de eventos em atores é um diagrama de eventos, acrescido dos eventos pendentes. Por exemplo, no caso da cálculo do fatorial mencionado na Seção 3.2.5, um destes elementos poderia ter a configuração mostrada na Figura 4.13.

O comportamento do ator fatorial ao receber o pedido de fatorial de n é criar um novo ator, enviar o seu endereço, juntamente com o pedido de cálculo do fatorial de $(n-1)$, para si próprio. O evento pendente é no caso, o pedido do fatorial de 1, cuja resposta deverá ser enviada ao ator $mult_2$.

As condições impostas na Definição10, a seguir, visam manter a consistência de diagrama aumentado de evento em atores a cada evolução possível do sistema.

Definição 7 (Ordenamento de Chegada) *$\forall a \in A$, o ordenamento de chegada de a é definido em E por $\langle a,i\rangle \xrightarrow{rec}_a \langle a',j\rangle \Leftrightarrow a = a'$ e $i < j$;*

Definição 8 (Ordenamento Combinado) *Seja o ordenamento combinado — em E definido como o fechamento transitivo de : $\bigcup_{\underrightarrow{atv}}(\bigcup_{\underrightarrow{rec}} a \in A)$;*

Definição 9 (Função Alvo) *Seja $T : E \rightarrow A$ a função que define o ator alvo do evento E;*

Definição 10 (Diagrama de Eventos) *O conjunto de diagramas, aumentados de eventos em atores é o multi-conjunto $\mathcal{D}$ de estruturas $\langle E, M, \xrightarrow{atv}, P\rangle$, onde*

- *E é o conjunto de eventos realizados;*
- *M é a função que extrai (lê) a mensagem associada a um dado evento: $M : E \rightarrow M$;*

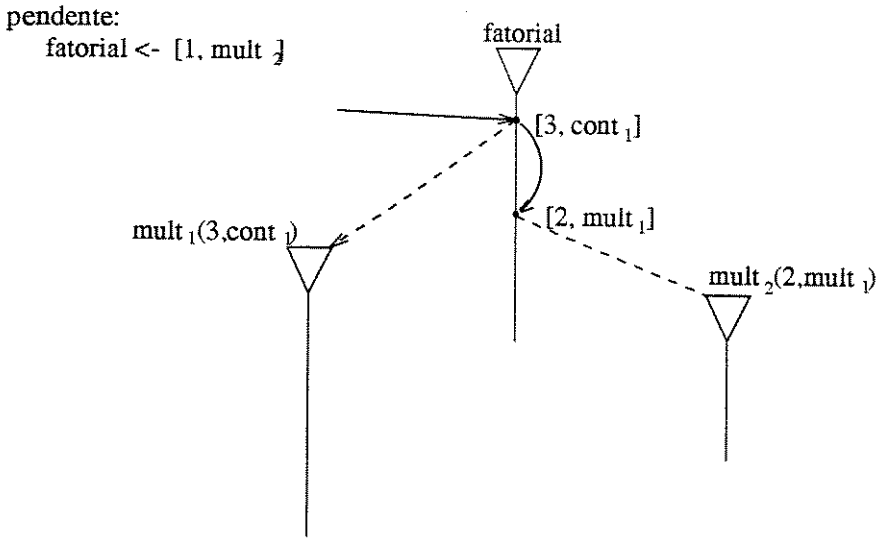


Figura 4.13 O domínio do diagrama de eventos e atores.

- $\xrightarrow{atv}$ *é um ordenamento parcial irreflexivo em* E*, tal que nenhum evento tem mais de um processador;*
- P *é o multi-conjunto de eventos pendentes.*

E neste multi-conjunto, as seguintes condições se verificam para um dado conjunto de atores A*:*

P **é Finito:** P *é um multi-conjunto com repetições finitas de elementos* $(A \times M) \times E$*, isto é,* P *é uma função* $P : ((A \times M) \times E) \rightarrow N$*, onde* N *é o conjunto dos números naturais. Esta lei serve para garantir que o conjunto de mensagens pode ser enumerado utilizando-se o conjunto dos números naturais. Fato que é importante para o árbitro:*

Axioma Forte: *as leis da Causalidade Estrita, Contabilidade e Predecessão Finita se verificam para os eventos do diagrama;*

Ordenamento: *os ordenamentos de chegada e combinado relacionam os eventos do diagrama;*

Atraso Finito:[12] *Se* $\#(E) \rightarrow \infty$*, então* $\#(P) \rightarrow \emptyset$*;*

Evento Inicial *ou* E *e* P *são ambos vazios ou existe um evento* e_0 *tal que* $\forall e \in E, e_0 = e$ *ou* $e_0 \xrightarrow{atv} e$*.*

É importante ressaltar que do ponto de vista do diagrama aumentado de eventos, a única hipótese requerida com relação ao processamento das mensagens pendentes é que o árbitro seja eqüitativo, ou seja, que todas as mensagens pendentes sejam em algum momento processadas. Conseqüentemente, não é necessário acrescentar nenhuma informação neste diagrama com respeito às prioridades associadas a estas mensagens, dado que o algorítimo de tratamento de mensagens pelo árbitro é eqüitativo.

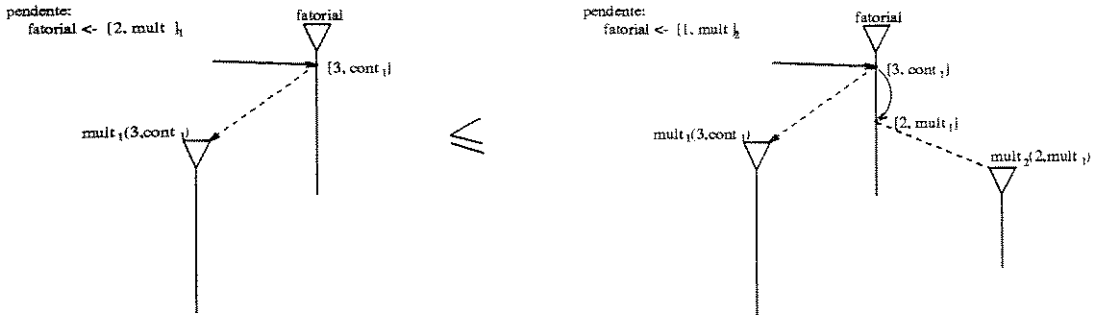


Figura 4.14 Exemplo de ordenamento histórico correto.

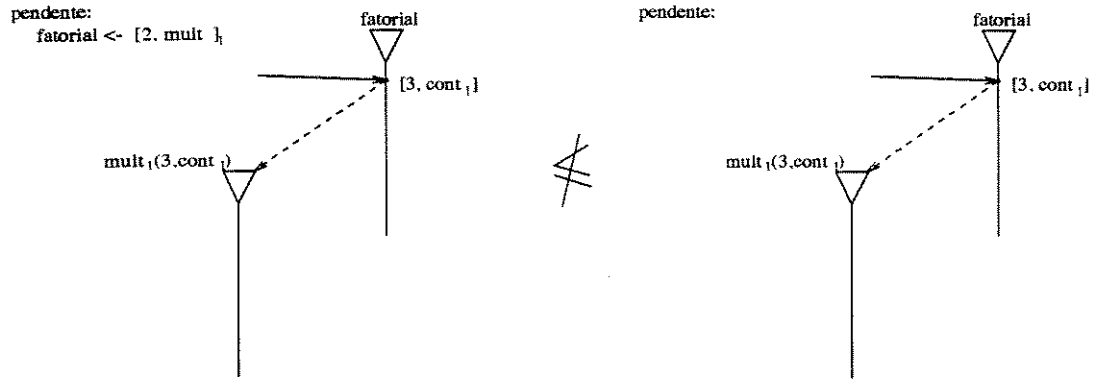


Figura 4.15 Exemplo de ordenamento histórico incorreto.

4.3.4 Ordenação dos Eventos do Domínio

Para que o Diagrama Aumentado de Eventos possa ser usado como domínio é necessário definir o ordenamento parcial que relaciona os elementos deste domínio. Dado que este diagrama representa uma "fotografia" do sistema em um dado instante, medido na janela temporal do observador, o comportamento do sistema pode ser especificado através da história resultante destas observações. O ordenamento parcial adotado é exatamente o que determina a história: $\forall x, y \in D, x \leq y$ significa que x é um dos estágios intermediários possíveis na trajetória do sistema para y, ou seja, y pode ser obtido a partir de x através da expansão de eventos pendentes.

A evolução histórica do sistema define um ordenamento parcial dado que: cada elemento representa uma possível história até ele mesmo (reflexividade); se x_1 precede x_2 e x_2 precede x_1, como se trata de um ordenamento temporal $x_1 = x_2$ (anti-simetria); e se $x_1, x_2 e x_3$, tais que x_1 precede x_2 e x_2 precede x_3, então x_1 precede x_3 (transitividade).

Tomando-se novamente o exemplo do cálculo de fatorial, ilustra-se na Figura 4.14, um exemplo de ordenamento correto, enquanto na Figura 4.15, um dos eventos pendentes do diagrama aumentado da direita desaparece sem ter sido realizado.

A formalização do ordenamento é realizada na definição a seguir:

Definição 11 (Ordenamento Histórico) *Sejam* $x = \langle E_x, M_x, \xrightarrow{atv}_x, P_x \rangle \in \mathcal{D}$ *e* $y = \langle E_y, M_y, \xrightarrow{atv}_y, P_y \rangle \in \mathcal{D}$. *$x$ é uma história inicial de y, denotado por $x \leq y$ se, e somente se*

a) $E_x \subseteq E_y$;

b) $\forall e \in E_x, M_x(e) = M_y(e)$;

c) $\forall e, e' \in E_x, e \xrightarrow{atv}_x e' \Leftrightarrow e \xrightarrow{atv}_y e'$.

d) Seja a função $\mathcal{A}(e)$ *o predecessor imediato de* e *no ordenamento de ativação,*[13] *então:*

$$P_x = \{\langle a, m, e\rangle \in P_y \mid e \in E_x\} \biguplus \{(T_y(e'), M_y(e'), e') \mid e' \in (E_y - Ex), e = \mathcal{A}(e')e, \in E_x\}.$$

Os três primeiros itens garantem que a história posterior seja consistente com a anterior, enquanto o último assegura que cada evento pendente em x ou continue pendente em y, ou transforme-se em um evento realizado.

Definição 12 (Domínio do Diagrama de Eventos em Atores) $(\mathcal{D}, \leq)$ *é o domínio do diagrama de eventos em atores.*

Os elementos isolados de $\mathcal{D}$ são aqueles com um número finito de eventos realizados. O menor elemento ($\perp$) representa a inexistência de eventos realizados ou pendentes, e o ordenamento $\leq$ garante que qualquer elemento de $\mathcal{D}$ é o menor limitante superior de uma seqüência contável[14] crescente de elementos. Portanto, $\mathcal{D}$ é um domínio de acordo com a Definição 4.

4.3.5 O Comportamento dos Atores e seu Significado Semântico

Definido o domínio sobre o qual a semântica é construída como sendo o diagrama de eventos aumentados, o próximo passo é definir o comportamento dos atores sobre este domínio, ou seja, como as várias ações executadas por um ator, como criação, definição de substituto e envio de mensagens, afetam o domínio.

Definição 13 (Comportamento) *O comportamento de um ator* a *é um elemento de* $\mathcal{B}$, *onde* $\mathcal{B}$ *é uma relação:*

$$\mathcal{B} = (\{a\} \times M \times E \longrightarrow G \longrightarrow (B \times (A \times M)^* \times B^*))$$

onde:

- $\{a\} \times M \times E$ *é o conjunto de mensagens pendentes cujo destino é* a*:*
- $(A \times M)^*$ *é uma seqüência de mensagens enviadas;*
- G *é o conjunto de novos atores criados por* a*;*
- B *é o conjunto de comportamentos que* a *pode assumir após a recepção da mensagem* $m \in M$*;*
- B^* *é conjunto de comportamentos dos novos atores criados.*

[13] $A \biguplus B$ a união disjunta dos conjuntos A e B.

[14] Por definição no diagrama aumentado de eventos, o conjunto E é contável.

Além disto, sejam $rec(a)$ *a função que retoma o próximo* $n \in \mathbb{N}$ *no ordenamento de chegada de* a, *tal que* $\langle a, n\rangle \notin E$, *e* $\beta(m) = (P, G, \delta)$ *o comportamento de um ator ao receber a mensagem* m, *onde* $\delta \in B$ *é o comportamento do substituto e:*

$$P = \{\pi_1, \pi_2 \ldots, \pi_k\},$$

$$G = \{\alpha_1, \alpha_2, \ldots, \alpha_k\}$$

onde:

1. *O evento atual é identificado por* $e = \langle a, rec(a)\rangle$;
2. *O evento e constará como ativador de todos os novos eventos pendentes gerados:*

$$\forall i(1 \leq i \leq k \Rightarrow \exists m_i \in M, \exists a_i \in A \mid \pi_i = \langle a_i, m_i, e\rangle);$$

3. *O evento e causou a criação dos novos atores:*

$$\forall i(1 \leq i \leq k' \Rightarrow \exists \beta_i \in B^*, \mid \alpha_i = \lfloor \beta_i, e \rfloor).$$

Significado Semântico

A linguagem escolhida para ser hospedeira da *extensão* de atores foi a linguagem C++, e os atores foram incluídos abstratamente como uma classe básica de objetos, da qual todas as classes de objetos cujo comportamento corresponde ao de um ator são derivadas. Esta classe, então, é a responsável pelo funcionamento dos atores conforme o significado que é dado a cada um de seus métodos.

Definição 14 (Expressões Sintáticas de Comportamento) *Seja a classe ator composta dos seguintes métodos:*

- **send**$[a, p, m]$: *envia a mensagem* m *com prioridade* p *ao ator* a, *onde* m *pode ser uma estrutura de dados ("struct");*
- **accept**$[m]$: *aceita uma mensagem para ser processada;*
- **new_behaviour**$[método, \epsilon_1, \ldots, \epsilon_i]$: *especifica o comportamento do substituto e os parâmetros que lhe serão enviados. A partir do ponto no programa onde o substituto* C_{i+1} *(ver Figura 3.1.) é definido, o processador corrente* C_i *não recebe mais mensagens, nem especifica uma nova substituição;*
- a_j = **new_actor**$[método, \epsilon_1, \ldots, \epsilon_i]$: *cria um novo ator, instância da classe associada à variável* a_j. *Seu primeiro comportamento é especificado no parâmetro* método, *que recebe como parâmetros as expressões* $\epsilon_1, \ldots, \epsilon_i$.

Definidas as expressões sintáticas, pode-se dar o significado semântico a cada uma delas, ou seja, especificar como cada uma delas afeta o domínio do diagrama aumentado de eventos. A semântica é determinada pela função de significado $\mathcal{S}$:

Definição 15 (Função de Significado) *Seja ρ a função que associa um significado às expressões primitivas do ambiente local σ, ou seja, é o resultado da avalição destas expressões de acordo com as regras da linguagem hospedeira, C++. A função de significado $\mathcal{S}$ associa a cada uma das expressões sintáticas de comportamento sujeitos ao ambiente σ a uma tripla representando as mensagens enviadas, os atores criados e o comportamento substituto. As mensagens precedidas de sinal negativo na posição referente às mensagens enviadas indica mensagem recebida, ou seja, retirada do conjunto de mensagens pendentes.*

A expressão $\sigma[\![a/x]\!]$ representa o ambiente local que difere de σ somente pelo fato de atribuir o valor primitivo a à variável x, e as variáveis primitivas ev e *self* referem-se ao evento corrente sendo processado, e ao identificador do ator cujo código contém o comando em questão. O significado semântico das expressões sintáticas **send**, **new_behaviour**, **accept** e **new_actor** é dado por:

$$\mathcal{S}(\textbf{send}[a,p,m])(\sigma[\![rec(self)/ev]\!]) = \{\langle \rho_\sigma(a), \rho_\sigma(m), ev\rangle, \emptyset, \emptyset\}$$

$$\mathcal{S}(\textbf{accept}[m])(\sigma[\![rec(self)/ev]\!]) = \{-\langle \rho_\sigma(self), \rho_\sigma(m), \mathcal{A}(ev)\rangle, \emptyset\emptyset\}$$

$$\mathcal{S}(\textbf{new\_behaviour}[\text{método}, \epsilon_1, \ldots, \epsilon_i]) = \{\emptyset, \emptyset, [\rho_\sigma(\text{método}), \rho_\sigma(\epsilon_1), \ldots, \rho_\sigma(\epsilon_i)]\};$$

$$\mathcal{S}(a = \textbf{new\_actor}[\text{método}, \epsilon_1, \ldots, \epsilon_i])(\sigma[\![rec(self)/ev]\!]) = \{\emptyset, \lfloor[\rho_{\sigma'}(\text{método})\rho_{\sigma'}(\epsilon_1), \ldots, \rho_{\sigma'}(\epsilon_i)], ev\rfloor, \emptyset\}.$$

onde $\sigma' = \sigma[\![a/\text{newid}()]\!]$ e newid() é uma função que retorna um identificador único para o novo ator criado.

4.3.6 Pontos Fixos

O comportamento de um ator, de acordo com a semântica associada às expressões sintáticas descritas em 4.3.5, determina o que acontece em cada passo da evolução do sistema. Quando este número de passos é finito, como é o caso do cálculo de fatorial recursivo, apenas a evolução do diagrama até a completação da tarefa é suficiente para especificar completamente a função realizada pelo conjunto de atores.

Contudo, quando existem laços que podem perpetuar a execução durante um tempo indeterminado é necessário que se estude o comportamento do sistema na fronteira do domínio. Este estudo é realizado com a utilização de domínios de potência, que permitem expressar o não-determinismo existente no sistema, e posterior busca de um ponto de estabilidade conhecido como *ponto fixo* [80].

A base teórica necessária para a construção de um domínio de potência a partir de um domínio base incompleta é descrita em [19].

Resumidamente, a construção do domínio de potência e a determinação dos pontos fixos é realizada de acordo com as seguintes etapas:

1. Para um dado programa $\mathcal{P}$ toma-se como base a história X em um dado momento e constroi-se um conjunto de diagramas aumentados de eventos através da expansão de um evento pendente. Esta expansão ocorre de acordo com o comportamento semântico do programa $\mathcal{P}$;

2. Cada elemento resultante da expansão do evento pendente ou é um elemento isolado, ou é maximal em $\mathcal{D}$;

3. Explorando-se a continuidade em ω da função $f : \mathbf{D} \rightarrow P[\mathbf{D}]$ definida por:

$$f(x) = \bigsqcup Z \leq X g(z)$$

determina-se o menor ponto fixo como sendo

$$\bigsqcup_{i \in \omega} f_*^i(\perp) = (\bigcup_{i \in \omega} f_*^i(\perp))^c$$

onde os estágios f_*^i contêm somente histórias finitas de uma execução que inicou no elemento mínimo $\perp$, ou seja, desde o evento inicial;

4. É a operação *menor limitante superior* $\bigsqcup$ que coloca os elementos representando seqüencias de execução não concluídas no menor ponto fixo.

A construção do domónio de potência e a determinação do menor ponto fixo completam a semântica denotacional para as linguagens baseadas em atores. O exemplo a seguir mostra que esta semântica, apesar de possuir um embasamento teórico que envolve conceitos pouco convencionais, é operacionalmente bastante simples.

Exemplo 1 (Semântica para Fatorial Recursivo) *O cálculo recursivo do fatorial de um número modelado com atores foi definido na Seção 3.2.5 e retomado na Seção 4.3.3. neste exemplo, a sua semântica será precisamente definida em termos da função de comportamento* $\beta(m) = (P, G, \delta)$ *para a função que realiza o fatorial e* γ *para o multiplicador.*

$$\beta([n,u]) > \begin{cases} \{(-\langle self, [n,u], \mathcal{A}(ev)\rangle, \langle u, [1], ev\rangle), \emptyset, \beta\} & \text{se } n = 0; \\ \{(-\langle self, [n,u], \mathcal{A}(ev)\rangle, \langle self, [n-1, c], ev\rangle), \lfloor [\gamma, n, u], w \rfloor, \beta\} & \text{caso constrário.} \end{cases}$$

O comportamento dos multiplicadores criados é descrito por:

$$\gamma(k) = \{(-\langle self, [k]\rangle \mathcal{A}(ev)\rangle, \langle u, [n * k], ev\rangle), \emptyset, \emptyset\}.$$

Capítulo 5

Implementação

A Plataforma Distribuída de Agregados de Atores (PDA^2) foi desenvolvida em parceria entre o Laboratório PRiSM da Universidade de Versailles Saint-Quentin en Yvelines, na França, e o Laboratório de Telemática da Unicamp.

A PDA^2 foi implementada em C++, sob sistema operacional SunOS-4.1, e em sua infraestrutura de comunicações usou-se o ambiente ISODE. A utilização deste ambiente possui a vantagem de permitir que a plataforma opere tanto com a família de procolos OSI, quanto Internet, satisfazendo um dos requisitos apresentados no Capítulo 2. A biblioteca de processos leves[1], que é parte integrante do SunOS, foi usada para o provimento de "threads" de controle.

Como interface para esta biblioteca, construiu-se um conjunto de classes C++. O uso destas classes, ao invés da chamada direta das funções que compõem a biblioteca, permite a portabilidade mais rápida para outras implementações de "threads", como o padrão POSIX [63, 30] e Solaris [75, 27]. Além disto, ao serem transformados em objetos, os "threads" se beneficiam do encapsulamento, protegendo a sua estrutura interna contra modificações indevidas.

Para um melhor entendimento do funcionamento da plataforma, algumas definições são necessárias (ver Figura 5.1)[24]:

Processo: é uma unidade básica de alocação de recursos que inclui páginas de espaço de endereçamento e acesso protegido a recursos do sistema, tais como: descritores de arquivos, processadores e "ports".

Agregado: é um conjunto de atores prestando um serviço.

Agregados Co-localizados: são agregados dentro de um mesmo processo. Assim sendo, estes compartilham o *núcleo* da plataforma e *objetos de protocolo.*

Núcleo: é o nome dado à composição de um escalonador, um gerenciador de "locks" e um gerenciador de "threads". A sua definição serve apenas para separar a parte que interage com a biblioteca de processos leves da que interage com o ambiente ISODE.

Escalonador: é o componente da plataforma responsável pelo revezamento nas filas correspondentes aos vários níveis de prioridade atribuídos aos "threads" em atividade. O próprio

[1] "Lightweight Processes Library"

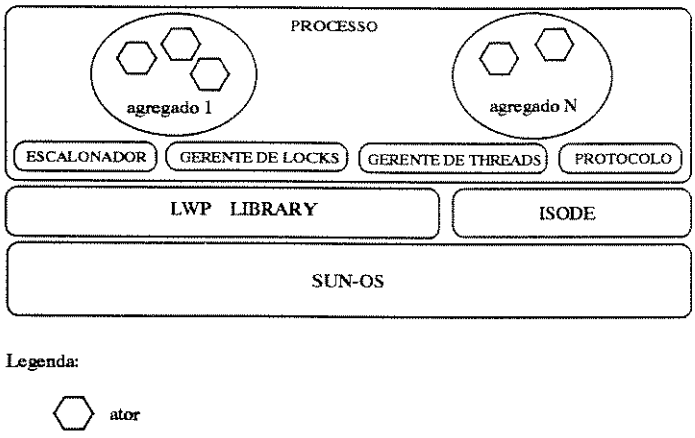


Figura 5.1 Arquitetura da plataforma.

escalonador é um ator, cujo "thread" associado sempre possui a mais alta prioridade no sistema. Além disto, quando advertido pelo gerente de "locks" de uma inversão de prioridades, o escalonador trata-a mudando temporariamente a prioridade do "thread" que detém o recurso para o mesmo valor do que está à espera deste.

Gerente de "Locks": é o componente, cuja função é fornecer, ao projetista da aplicação, um meio de sincronizar o acesso à região de dados compartilhada pelos atores. Cada processo tem seu próprio gerenciador de "locks". Se inversão de prioridades é detectada, o gerenciador envia uma mensagem ao escalonador, que tomará as medidas necessárias para reverter a situação. Não é necessário ter um gerenciador global de "locks" porque os modelos de informação OSI, Internet, TMN e IN estabelecem que a informação está dispersa em todos os seus componentes. Por exemplo, cada parte da MIB é associada a apenas um bloco funcional, como OSF, WSF, NEF, NF ou QAF.

Gerente de "Threads": é a interface entre a plataforma e a biblioteca de processos leves. Ele funciona em cooperação com o escalonador, mantendo-o atualizado com respeito à criação e destruição de "threads".

Objetos de Protocolo: são os componentes que controlam as associações ativas e possuem uma tabela relacionando agregados a associações correspondentes.

A plataforma exibe ainda as seguintes características:

- Múltiplos "threads" de controle podem ser criados para atender às necessidades dos atores, transformando o agregado em um *servidor inteligente*, porque este pode criar e destruir novos *substitutos* ("threads") dependendo do número de requisições simultâneas;
- Níveis de prioridades podem ser atribuídos aos "threads" em sua criação e modificados posteriormente, caso necessário;
- Como pode-se construir agregados co-localizados, é possível implementar várias entidades (OSF, FE, etc) compartilhando o mesmo elemento de protocolo[2]. Isto reduz significativa-

[2]Evitando a necessidade de implementar a pilha de protocolos várias vezes.

mente o tamanho do código da aplicação resultante, pois normalmente se deveria inserir os objetos de protocolo em todas as entidades;

- A comunicação entre agregados co-localizados é bastante rápida, porque envolve somente troca de mensagens entre "threads". Isto evita a sobrecarga desnecessária da pilha da protocolos de comunicação, que aconteceria se os agregados estivessem em processos diferentes;

- *Transparência de acesso* é obtida através do uso de um *compilador de interfaces*[3]. Este compilador é um intermediário[4] entre a estruturação em classes de C++ e os "stubs" em linguagem C gerados pelos compiladores *pepy* e *posy* do ambiente ISODE, ao receberem especificaçoes em ASN.1. Ele gera classes de objetos sobre as estruturas de dados em *C*, geradas por *posy*, e aproveita os conceitos de *classes abstratas* e *sobrecarga de operadores* existentes em C++ para chamar automaticamente as funções de codificação e decodificação geradas por *pepy*. O *compilador de interfaces* é parte do ambiente da plataforma, e foi implementado no desenvolvimento deste trabalho;

- *Transparência de Localização* dos agregados é realizada através do uso do serviço de diretório *quipu*, o qual é parte integrante do ambiente ISODE. Para registrar o seu serviço no diretório, a aplicação precisa antes haver definido a sintaxe abstrata em ASN.1 das mensagens a serem trocadas com o exterior do agregado. O nome associado a esta sintaxe deve ser registrado no serviço de diretório juntamente com o nome do serviço. Estes dois parâmetros são enviados ao objeto de protocolo que adiciona informações como o endereço do processo na rede (**host, port**) e as classes às quais o registro pode pertencer na hierarquia do diretório. A estrutura resultante enviada ao diretório assume a forma.

```
objectClass = top&applicationEntity
cn= NEFService
presentationAddress= #1972/Internet=richelieu.prism.uvsq.fr
supportedApplicationContext= isode NEFService ctx
acl=
execvector= NEFReceptionist
```

5.1 "Threads" de Controle

A idéia que delineia o conceito de "threads" de controle é fornecer uma abstração ao conceito de *fluxo de controle*, transformando-o em um tipo de dado. Cada "thread" possui o seu próprio contador de programa, registradores de uso geral e pilha associados.

Dentro de todo código executável tem-se pelo menos um "thread", que é o que o percorre desde a função principal até o final da execução. Quando se tem mais de uma função dentro do mesmo código, como acontece com as funções implementando um objeto, tem-se tantos "threads" em potencial quantas são estas funções.

A identificação de "threads" que podem executar em paralelo em uma aplicação apenas determina o seu grau de concorrência. Para que um "thread" execute, é necessário que os recursos do sistema operacional lhe sejam alocados.

[3] "Stub compiler"
[4] "Front end"

Se existe apenas um processador na arquitetura de "hardware", a parcela de tempo deste processador alocada pelo sistema operacional a um dado processo tem de ser dividida por todos os "threads" existentes dentro deste processo.

Em máquinas cuja arquitetura é baseada em multiprocessadores, o sistema operacional pode fornecer suporte para "threads" no nível do núcleo, reconhecer que um processo é composto de vários "threads", e realizar a alocaçação de processadores a "threads" automaticamente. Isto ocorre no sistema Solaris da SUN [75, 27]. Neste sistema, os "threads" são divididos em dois níveis, e no nível mais baixo, cada um deles pode ser visto como uma CPU virtual, escalonados independentemente pelo núcleo e possuíndo proteção para chamadas de sistema[5]

5.1.1 A Biblioteca de Processos Leves do SunOS

Na implementação da plataforma proposta, utilizou-se a biblioteca LWP[6] do sistema SunOS-4.1 [78]. Esta biblioteca foi projetada para fornecer "threads" apenas no nível do usuário, ou seja, não existe suporte para "threads" no nível do sistema operacional. O programa principal (`main`) torna-se um "thread" logo que a primeira primitiva da biblioteda é encontrada durante a execução.

Primitivas Básicas

As primitivas básicas para a gerência de "threads" fornecem suporte para:

Inicialização: `lwp_setstkcache` inicializa um "cache" de pilhas protegidas[7] que são utilizadas em chamadas subseqüentes a `lwp_newstack`, a qual é chamada a cada novo "thread" criado. As pilhas alocadas são automaticamente liberadas quando os "threads" que as utilizam deixam de existir. A primitiva `pod_setmaxpri` especifica o máximo nível de prioridade aceito na aplicação;

Criação: `lwp_create` cria um novo "thread", cujo ponto de partida para a execução é a função determinada em seus parâmetros;

Destruição: `lwp_destroy` destrói um "thread" especificado;

Término: embora a destruição de um "thread" seja automática quanto este deixa a função na qual ele começou a execução, existe o término através da primitiva `pod_exit`, ou `exit`. Para estabelecer um *status de término*, a primitiva `pod_setexit` é utilizada;

Primitivas de Escalonamento

O escalonamento básico fornecido pela biblioteca LWP é preemptivo, de acordo com uma escala de prioridades. O "thread" não bloqueado e com a maior prioridade está sempre em execução. Dentro de um determinado nível de prioridade, não existe preempção, sendo os "threads" executados seguindo a disciplina FIFO. Quando não especificado diferentemente, o programa principal (`main`) inicia executando sempre no mais alto nível de prioridade.

[5] "System calls."
[6] "Lightweight Processes Library."
[7] "Red-zone protected stacks".

Existem situações nas quais este escalonamento básico não atende às necessidades de projeto das aplicações concorrentes e, conseqüentemente, um outro tem de ser construído. Para auxiliar nesta constução, a biblioteca fornece as seguintes primitivas:

Rodízio na fila: [8] como os "threads" executando em um mesmo nível de prioridade são escalonados com a disciplina FIFO, se um deles permanece executando durante um longo intervalo, os outros não podem começar a sua execução durante este intervalo. Nestas situações, para assegurar que outros "threads" de mesma prioridade possam iniciar a sua execução, a primitiva `lwp_resched` tem de ser chamada periodicamente para garantir que haja um rodízio na fila correspondente àquele nível de prioridade;

Mudança de prioridade: a prioridade de um dado "thread" pode ser modificada através de `lwp_setpri`. Não existe o conceito de ancestral de um "thread", como no caso dos processos em UNIX nos quais cada novo processo criado é um filho do que o criou. Um "thread" recém criado pode até mesmo mudar a sua própria prioridade para um valor superior à do "thread" que o criou. Como o "thread" com maior prioridade está sempre executanto, se um "thread" altera a prioridade de um outro para um valor superior à sua própria, ele é suspenso, pois sofre preempção;

Entrega do controle: [9] o "thread" corrente entrega o recurso de processamento que lhe fora alocado a um outro pertencente ao seu mesmo nível de prioridade, e este começa a execução imediatamente após a primitiva `lwp_yield`. A escolha do próximo a executar pode ser feita explicitamente pelo "thread" que está executanto, ou implicitamente, caso em que o primeiro da fila é escolhido. No último caso, o "thread" que entrega o controle é posicionado em segundo lugar na fila, enquanto que quando a escolha é explícita, ele é posicionado em último lugar;

Suspensão e retomada da execução: através da primitiva `lwp_suspend`, um "thread" pode tornar um outro inelegível para executar nos próximos rodízios da fila em seu nível de prioridade, ou suspender a si próprio, caso o parâmetro da primitiva seja uma auto referência. Para tornar um "thread" suspenso elegível para execução, a primitiva `lwp_resume` deve ser invocada;

Temporização: a primitiva `lwp_sleep` bloqueia o "thread" durante pelo menos o período especificado como parâmetro. Não se pode assegurar uma quantidade exata de tempo se o sistema não fornece suporte para execução em *tempo real*;

Avaliação do estado: as primitivas `lwp_ping`, `lwp_getstate` e `lwp_enumerate` permitem a investigação do estado de um dado "thread", e enumerar todos os "threads" existentes;

Primitivas de Sincronização

Para a sincronização entre "threads" em execução, são oferecidos dois paradigmas: *monitores* e *rendezvous*:

[8] "Reshuffling the queue."
[9] "Relinquishing control."

"Rendezvous" Estendido: a sincronização é executada por meio das primitivas `msg_send`, `msg_receive` e `msg_reply`. Quem quer que chame a primitiva primeiro, seja este o enviador ou o recebedor, terá de esperar até que o outro esteja pronto para completar a operação. Quando ambos atingem este ponto, o recebedor chama a primitiva `msg_reply` e ambos continuam a execução independentemente;

Monitores: a sincronização pode também ser satisfeita com a utilização de monitores e variáveis de condição. Um monitor implementa uma *região crítica*, que é uma região reentrante do código com acesso seriado. Os monitores são primitivas de alto nível e seguem o conceito de encapsulamento de dados de programação orientada a objetos, pois os dados internos ao monitor somente podem ser acessados pelas funções definidas dentro dele.

Ao se associar uma variável de condição a um monitor, as primitivas `cv_wait` e `cv_notify` reduzem a notificação de liberação da região com acesso seriado a uma operação atômica. Caso existam vários "threads" esperando o accesso a uma mesma região, a primitiva `cv_broadcast` é usada para advertir todos eles.

Interrupções

Não existem interrupções assíncronas em um "thread" implementado com a biblioteca LWP do SunOS-4.1[10]. Embora o processo no qual o "thread" está contido ainda é capaz de reagir a eventos assíncronos, a comunicação entre dois "threads" tem de ser síncrona.

O mapeamento dos evento assíncronos em síncronos é realizado com a utilização de **agentes**, que são "threads" executando na mais alta prioridade de forma a poder capturar os eventos assíncronos, e transmiti-los por meio de troca de mensagens aos "threads" que estão esperando por estes eventos.

5.2 Correspondência entre "Threads" e Atores

Atores e "threads" guardam muitas semelhanças entre si, do ponto de vista operacional, pois os threads são tipos abstratos de dados representando um fluxo de controle, enquanto os atores são estruturas de dados que supostamente estão sempre ativas, precisando para tal, de um fluxo de controle sempre que uma requisição lhes é dirigida.

Os "threads" existentes dentro de um mesmo espaço de endereçamento podem fazer acesso de forma global à área de dados deste espaço. Por um lado, isto é altamente positivo, pois permite que se implemente facilmente servidores que podem tratar muitas requisições simultaneamente, como é o caso do *servidor concorrente de MIB* e das *entidades funcionais IN*. Por outro lado, quando feito de forma desordenada, este acesso simultâneo pode gerar efeitos colaterais na execução que são muito difíceis de depurar.

Felizmente, ao serem incorporados de forma organizada em uma linguagem orientada a objetos, como é o caso de C++, estes efeitos podem ser enormemente diminuídos. A forma adotada na implementação da plataforma utilizou este recurso, ao notar que um "thread" (da mesma forma que um ator, através da especificação de seu novo comportamento) pode assumir a *identidade*

[10] O Solaris permite interrupção assíncrona dos "threads" existentes em um processo, pois possui suporte no nível do núcleo.

de um objeto, quando o endereço deste objeto é passado como um parâmetro na sua função de criação. Isto deve-se a uma particularidade da linguagem C++ que coloca o endereço de um objeto na pilha de parâmetros da função chamada, permitido-o utilizar a variável **this**.

Procedendo desta forma, o "thread" passa a executar somente no contexto daquele objeto. Isto significa que ele terá conhecimento apenas do estado associado à instância na qual está executando e às variáveis de classe, além, é claro, da variáveis globais do programa. Como um programa bem estruturado deve possuir um mínimo de variáveis globais, os efeitos colaterais são bastante reduzidos.

No exemplo a seguir, tem-se o ator *lockManager* que faz acesso a duas listas de "locks": a de pendentes e a de fornecidos. A ele, foram associados dois *comportamentos*: o comportamento sem pendências, no qual ele sabe de antemão que não precisa consultar a lista de pendência dado que ela está vazia, e o comportamento quando existem elementos na lista de pendências. Como se pode observar, o encapsulamento dentro de uma classe de todos os comportamentos que um ator pode assumir é uma atividade bastante saudável do ponto de vista de projeto, pois a depuração futura torna-se mais simples. Isto não estava previsto no paradigma original de atores, no qual qualquer função dentro do código pode ser o próximo comportamento de um dado ator.

```
class lockManager: public ator {
        static  LockList      _grantedLocks;
        static  LockList      _pendingLocks;

public:
        void    noPendingLocks();
        void    withPendingLocks();
};
```

Em resumo, a junção de "threads" e objetos para implementar atores mostrou-se bastante adequada: os primeiros por fornecerem os recursos de processamento do sistema operacional de forma abstrata, sem que se precise se preocupar com questões como o número de processadores efetivamente disponíveis; e o segundo por permitir a construção de atores de uma forma bem estruturada.

5.2.1 Comportamento Semântico

Em termos semânticos, a correspondência entre as primitivas de atores e as associadas aos "threads" de controle são as seguintes:

new_actor : cria uma estrutura de dados para representar o ator; cria um "thread" mandando para ele o endereço desta estrutura e o método da classe que corresponderá ao seu primeiro comportamento; e liga o "mailbox" do ator à fila de mensagens deste primeiro "thread" associado a este ator;

new_behaviour: cria um novo "thread" e uma nova estrutura de dados, mandando para ele o endereço desta nova estrutura e o método correspondente ao próximo comportamento. Qualquer informação relevante à história do sistema deve ser transmitida ao substituto,

seja através desta estrutura de dados, seja via passagem direta de parâmetros. Em seguida, desassocia o "mailbox" do "thread" precedente, ligando-o ao novo.[11]

send: tem uma correspondência direta com a primitiva `threadSend`, só que antes de ser entregue ao ator propriamente dito, a mensagem deve ser enviada ao árbitro, pois é este que decide a ordem de entrega da mesma;

accept: tem correspondência direta com a primitiva `threadRecv`, mas uma consulta ao árbitro é efetuada para saber qual é a mensagem com maior prioridade. Ao determinar a próxima mensagem, o árbitro solicita ao escalonador de "threads" em execução que aumente a prioridade do thread correntemente ligado ao "mailbox" para o valor correspondente ao da prioridade da mensagem;

5.3 Teste da Plataforma: Servidor de MIB Concorrente

Na Seção 2.8, foi apresentada a importância do acesso concorrente aos dados contidos na MIB, e dividiram-se estes dados em três categorias: sensoriais, estruturais e de controle, e que os dados sensoriais podiam ser periódicos ou esporádicos. As mensagens associadas à manipulação destes dados foram então divididas em prioridades de acordo com as seguintes regras [23]:

Dados Sensoriais Esporádicos: as mensagens destinadas à manipulação desta categoria receberam o nível de prioridade máximo (MAX). O recepcionista do agregado as reconhece como tal porque o ator que se registrou como seu receptor, solicitou ao objeto de protocolo a abertura de um ponto extra de comunicação para a espera de mensagens ("socket"), e pediu prioridade máxima para todas as mensagens recebidas através deste. Esta prioridade é então associada à mensagem enviada ao recepcionista.

Dados Sensoriais Periódicos: os atores responsáveis pela sondagem periódica de algum dispositivo solicitam ao *serviço de relógio* que lhes envie mensagens periódicas com prioridade (MAX-1);

Requisições de Gerência: estas requisições agem tanto sobre os dados de controle quanto sobre os dados estruturais. Como não existe diferenciação destes dados do ponto de vista de gerência OSI, isto é, um M_SET ou um M_ACTION pode ser dirigidas a ambas as categorias, o seu nível de prioridade é elevada ao nível dos dados esporádicos se a primitiva é M_ACTION, e deixada inalterada, caso contrário. Isto foi feito porque é provável que uma ação seja urgente, como ocorre em caso da parada de uma conexão quando ela está sendo invadida por um intruso;

Como a interação com a MIB foi realizada através do *Serviço de Informação de Gerência (MIS)*, usou-se a implementação do MIS chamada OSIMIS. Este pacote foi desenvolvido na University College of London como um conjunto de funcionalidades para a construção de agentes e gerentes.

[11] Isto faz parte da especificação de um substituto, conforme visto no Capítulo 4.

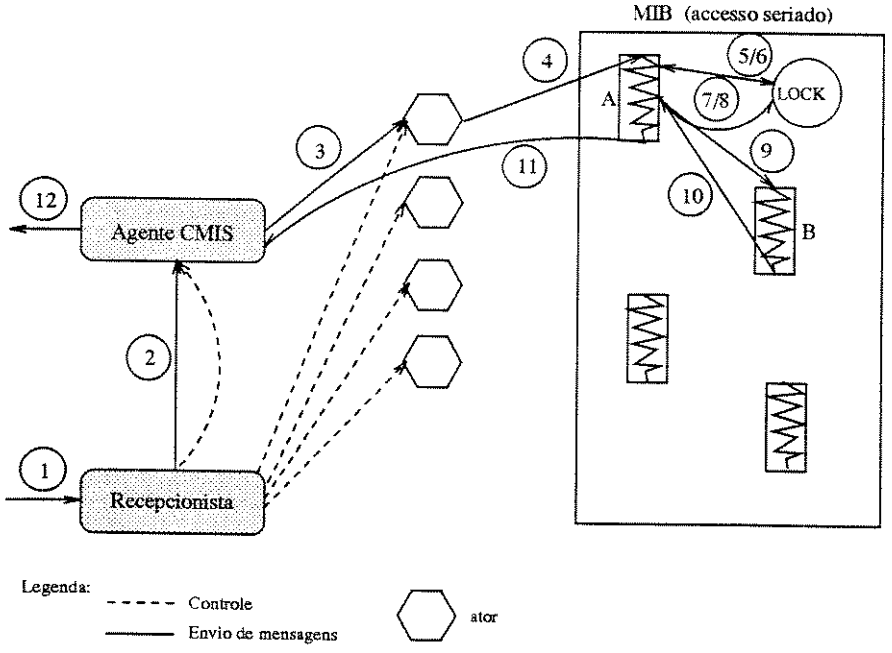


Figura 5.2 Requisição CMIS com tratamento local.

Para ilustrar como o Servidor Concorrente de MIB funciona, observe-se a Figura 5.2. A situação ilustrada corresponde a um dos muitos padrões de comportamento: o tratamento de uma requisição que não envolve qualquer iteração externa à MIB considerada.

O **Recepcionista** é o responsável pela recepção das requisições, controle de associações e interação com os objetos de protocolo.

O **Agente CMIS** é a entidade que executa o tarefas sobre a MIB. O grau de concorrência escolhido foi tratar em paralelo as requisições provenientes de associações diferentes, e seriá-las caso contrário. O raciocínio por trás desta decisão é que é bastante provável que requisições provenientes de uma mesma associação tenham de ser tratadas como um conjunto de atividades a ser executadas seqüencialmente. Portanto, o Agente CMIS cria um novo ator para cada nova associação, mas é o recepcionista que mantém o controle da ligação entre esta associação e o ator "mailbox" do ator responsável por ela, conforme ilustrado nas linhas tracejadas. Isto é necessário porque o Agente só recebe as requisições de gerência, enquanto que os atores individualmente podem receber notificações de eventos e realizar sondagens, independentemente do Agente. O tratamento de uma requisição CMIS ocorre da seguinte forma:

(1-2) A requisição chega ao recepcionista que se havia registrado no objeto de protocolo. Como ele percebe que a requisição está dirigida ao agente CMIS, ele a redireciona para este último;

(3-4) O ator responsável pela associação atende a requisição;

(5) Suponha-se, por exemplo, que a operação, um *Set*, implica na obtenção de um "lock" para realizar o acesso ao objeto alvo A. Este "lock" é, então, solicitado;

(6) O "lock" é obtido e o ator entra na região crítica de A;

	Get			Set		
	Um "thread"	Vários "Threads"		Um "thread"	Vários "Threads"	
		10 ms	50 ms		10 ms	50 ms
Get	2.09	1.95	2.04	1.78	1.95	1.76
Action	1.95	1.91	2.05	1.74	1.81	1.87

Tabela 5.1 Comparação entre a abordagem clássica e a com multiplos "Threads" de controle. (Medida em segundos)

(7-8) Durante o processamento dentro da região crítica, supondo que o objeto é composto (*Relação de Inclusão*) do objeto B, o qual deve também ser modificado. Então, outro "lock" é solicitado e obtido. Caso contrário, o objeto solicitante permaneceria bloqueado até a obtenção daquele "lock";

(9-10) O estado de B é alterado e o ator continua sua execução em A;

(11-12) O ator termina a execução da requisição e envia ao Agente CMIS a resposta, colocando-se à disposição para tratar mais uma requisição pertencente àquela associação. O Agente CMIS, empacota a resposta e a envia ao solicitante, cujo endereço lhe foi enviado pelo recepcionista como uma *continuação.*

5.3.1 Testes de Desempenho

Foram realizados testes de pior caso para avaliar a sobrecarga introduzida pela plataforma. O pior caso é representado por uma taxa de chegada de eventos tão baixa que a existência de concorrência e de vários níveis de prioridade é totalmente irrelevante, dado que um evento é tratado antes que o próximo chegue. Para a comparação, uma versão do agente com apenas um "thread" de controle, sem prioridades, mas usando C++ e ISODE foi usada.

Estes testes mostraram que o tempo médio de resposta é quase o mesmo quando comparado aos exibidos pela versão com apenas um "thread". Conseqüentemente, a sobrecarga introduzida pela estrutura de controle interna à plataforma é mínima. Os resultados são mostrados na Tabela 5.1.

As quantidades mostradas são o tempo decorrido desde o momento em que foi feita a requisição, até a recepção e exibição do resultado, medidas através do comando *time* do SunOS[12]. Elas são dependentes, naturalmente, da carga da rede local onde foi realizado o teste, mas estes foram executados durante a madrugada, quando este tráfego é mais baixo, e além disto, ambos os agentes foram testados sob as mesmas condições, quase ao mesmo tempo. Isto significa que as medidas são válidas como meio de comparação, não como valores absolutos.

Comandos típicos dirigidos à plataforma foram:

```
time maction SMA treville -c uxObj1 -i uxObj1=test -a getUserNames
```

[12]SunOS é marca registrada da SUN Microsystems.

```
time mibdump -c transportEntity -i subSystemId=transport@entityId=isode
time mset SMA treville -c uxObj1 -i uxObjId=test -w wiseSaying="..."
```

O primeiro destes comandos indica uma medida de tempo da primitiva `maction`, utilizando o contexto `SMA`, dirigida ao agente existente na máquina `treville`, realizando a ação `getUserNames`, sobre a instância `uxObj1=test` da classe `uxObj1`. No segundo comando, como o filtro é realizado sobre dois atributos, a instância é identificada por `subSystemId=transport@entityId=isode`.

Além disto, como se queria encontrar um valor ótimo para o período do escalonador, os testes foram executados para vários valores deste período, dos quais dois são exibidos: *50 ms* e *10ms*. O ambiente onde os testes foram realizados possuia apenas um processador. Os valores para o período indicam o intervalo de tempo após o qual um "thread" existente retoma a execução. Como o período de escalonamento implica na reativação do escalonador para escolher este novo "thread", espera-se uma degradação do sistema quando este período tende a zero. Para os valores examinados, no entanto, esta degradação não foi significativa.

Capítulo 6

Conclusão

O Paradigma de Atores, extendido para operar com gradação das suas atividades segundo níveis de prioridade, e para permitir agrupamentos em agregados mostrou-se vantajoso no projeto e implementação de agentes (OSI) ou (Internet), blocos funcionais TMN e entidades funcionais IN, pois:

- Cada uma destas entidades pode ser modelada como um agregado de atores. O agregado define a fronteira de um conjunto de threads, cuja finalidade é fornecer um serviço. A correspondência entre agregados e serviços é muito importante no modelamento das entidades de gerência e controle de que trata esta tese, pois o par *<serviço,protocolo>* é uma constante na definição destas entidades;

- A granularidade da concorrência interna de cada uma destas entidades pode ser determinado por seu projetista;

- Atores permitem abstração completa acerca da arquitetura do ambiente de execução. Precisa-se determinar apenas o grau máximo que se deseja explorar a concorrência, deixando todo o resto a cargo do ambiente no qual os atores foram inseridos;

- A especificação de programas que requerem eqüitatividade no tratamento de eventos que lhes são dirigidos é possível, dado que a base semântica da Teoria de Atores é fundamentada em não-determinismo ilimitado. Este não-determinismo é expresso na hipótese de tempo finito na entrega de todas as mensagens, e é representado nos diagramas aumentados de eventos, cuja seqüência determina a história de um sistema de atores. Esta é uma propriedade altamente desejável de uma plataforma de suporte de gerência e controle, pois mesmo os eventos com baixa prioridade devem ser tratados em algum momento;

- A introdução de níveis de prioridades no tratamento dos eventos pendentes, uma adição deste trabalho à teoria original, atende à necessidade de processamento de eventos que requerem uma ação urgente. A sua maior influência é nas redes de alta velocidade, como ATM, onde devem ser tomadas ações corretivas antes que a situação detectada torne-se crítica;

A plataforma PDA^2 foi implementada em C++, sob SunOS-4.1, e em sua infraestrura de comunicações usou-se o ambiente ISODE, tornando-a adequada para a implementação de aplicações de gerência tanto em redes OSI, quanto em redes Internet.

A Biblioteda de *Processos Leves (LWP)* foi usada para fazer a correspondência entre atores e "threads", os quais foram reestruturados em classes C++. Esta reestruturação permitirá a migração futura para qualquer plataforma de "threads" de controle existentes no mercado, em particular para o padrão POSIX.

Testes computacionais mostraram que a sobrecarga introduzida pela plataforma é mínima. Conseqüentemente, ela pode ser utilizada mesmo em ambientes que não exigem os seus requisitos de projeto, como concorrência e prioridades, como ocorre no caso em que a taxa de chagada de eventos seja muito baixa.

Até onde se sabe, esta é a única plataforma na qual programação concorrente com níveis de prioridades é introduzida em uma linguagem de programação como C++, e cuja infraestrutura de comunicação é realizada com o ambiente ISODE.

Como indicação de pesquisa futura sugere-se a extensão do modelo para incorporar mecanismos de transações entre agregados. Estes permitiriam a definição de serviços nos quais as entidades devem operar de forma transacional.

Glossário

ACSE: (*Application Control Service Element*) Elemento de Serviço de Controle de Associações (entre aplicações);

AD: (*Adjoint*) Adjunto. PE da IN;

ATM: (*Asynchronous Transfer Mode*) Modo de Transferência Assíncrono é um dos padrões propostos atualmente como base para B-ISDN;

ASN.1: (*Abstract Syntax Notation One*) Notação de Sintaxe Abstrata Um. Padrão usado nas camadas de apresentação e de aplicação para a representação de estruturas de dados e sua codificação;

ASE: (*Application Service Element*) Elementos de Serviço de Aplicação são uma sub-divisão dos ASOs;

ASO: (*Application Service Object*) as entidades da camada de aplicação são compostas de um ou mais Objetos de Serviço de Aplicação.

BCP: (*Basic Call Processing*) Processamento Básico de Chamadas na IN;

BER: (*Basic Encoding Rules*) Regras de Codificação Básicas são as regras utilizadas na codificação em ASN.1;

B-ISDN: (*Broadband Integrated Services Digital Network*) rede de serviços integrados de faixa larga;

CCAF: (*Call Control Agent Function*) Função Agente de Controle de Chamadas, FE da IN;

CCF: (*Call Control Function*) Função de Controle de Chamadas, FE da IN;

CCS: (*Calculus of Communicating Systems*) Cálculo de Sistemas Comunicantes. Linguagem para a descrição de sistemas concorrentes muito próxima a CSP;

CID: (*Call Instance Data*) Dados de Controle de Chamada, informação veiculada entre SIBs na IN;

CMIP: (*Common Management Information Protocol*) Protocolo de Informação Comum de Gerência;

CMIS: (*Common Management Information Service*) Serviço Comum de Informação de Gerência;

CMISE: (*Common Management Information Service Element*) Elemento de Serviço Comum de Informação de Gerência;

CSP: (*Communicating Sequential Processes*) Processos Seqüenciais Comunicantes. Linguagem para descrição de sistemas em termos de processos concorrentes;

DFP: (*Distributed Functional Plane*) Plano Funcional Distribuído na IN;

DN: (*Distinguished Name*) Nome Distinto usado para identificação de objetos gerenciados na OSI-MF;

DSL: (*Distrubuted Service Logic*) Lógica de Serviço Distribuída. comanda a troca de informações entre FEs;

FE: (*Functional Entity*) Entidade Funcional na IN, pertencente ao DFP;

FEA: (*Functional Entity Action*) Ações de Entidade Funcional na IN;

FTAM: (*File Transfer Access and Management*) Gerência, Acesso e Transferência de Arquivos;

GFP: (*Global Functional Plane*) Plano Funcional Global na IN;

GSL: (*Global Service Logic*) Lógica de Serviço Global. É a linguagem de composição de serviços no GFP da IN;

ICMP: (*Internet Control Message Protocol*) Protocolo de Mensagem de Controle Internet;

IDU: (*Interface Data Unit*) Unidade de Dados de Interface. Especifica os dados que atravessam uma interface;

IN: (*Intelligent Network*) Rede Inteligente;

INMF: (*Internet Network Management Framework*) Estrutura de Gerência Internet;

ISO: (*International Standards Organization*) Organização de Padrões Internacionais;

ISODE: (*ISO Development Environment*) Conjunto de bibliotecas e compiladores para o desenvolvimento de aplicações OSI em ambiente Internet;

ITU: (*International Telecommunications Union*) União Internacional de Telecomunicações;

ITU-T: (*Telecommunication Standardization Sector of ITU*) Setor de padronização de telecomunicações da ITU;

LLA: (*Logical Layered Architecture*) arquitetura lógica em camadas na TMN;

LM: (*Layer Manager*) Gerente de Camada na OSI-MF;

LWP: (*Lightweight Processes Library*) biblioteca de processos leves do SunOS;

MF: (*Mediation Function Block*) Bloco de Função de Mediação na TMN;

MIB: (*Management Information Base*) Base de Informações de Gerência;

MIS: (*Management Information Service*) Serviço de Informação de Gerência, padrão OSI para gerência de sistemas;

MRSW: (*Multiple Reader, Single Writer*) Múltiplos leitores, escritor único, tipo de memória distribuída com replicação;

MWMR: (*Multiple Writer, Multiple Reader*) Múltiplos escritores, múltiplos leitores, tipo de memória distribuída com replicação;

NEF: (*Network Element Function Block*) Bloco Funcional de Elemento de Rede na TMN. É o bloco que realiza a função de agente;

NSAP: (*Network Service Access Point*) SAP da camada de Rede. Fornece um endereço global para todo sistema OSI;

OID: (*Object Identifier*) Idenfiticador de Objeto. O OID é uma seqüência não ordenada de números que identifica unicamente um objeto de informação. A sintaxe para um OID é definida em ASN.1.

OSF: (*Operations Systems Function Block*) Bloco Funcional de Operações do Sistema na TMN. É o bloco que realiza a função de gerente;

OSI: (*Open Systems Interconnection*) Interconexão de Sistemas Abertos, padrão desenvolvido pela ISO;

OSI-MF: (*OSI Management Framework*) Estrutura de Gerência OSI;

PE: (*Physical Entity*) Entidade Física na IN;

PNO: (*Public Network Operator*) Operador público de rede;

POI: (*Point of Initiation*) Ponto de Início, através do qual a chamada IN deixa o SIB BCP;

POR: (*Point of Return*) Ponto de Retorno, através do qual a chamada IN retorna ao BCP;

POSIX: (*Portable Operating System Interface*) Interface Portável de Sistema Operacional, padrão da IEEE;

QAF: (*Q-Adaptor Function*) Bloco Funcional Adaptador-Q na TMN;

QoS: (*Quality of Service*) Qualidade de Serviço;

RDN: (*Relative Distinguished Name*) Nome Distinto Relativo usado na identificação de objetos gerenciados na OSI-MF;

ROSE: (*Remote Operations Service Element*) Elemento de Serviço de Operações Remotas;

RTSE: (*Reliable Transfer Service Element*) Elemento de Serviço de Transferência Confiável;

SAP: (*Service Access Point*) Ponto de Acesso ao Serviço é um ponto endereçável na fronteira entre duas camadas adjacentes;

SCEF: (*Service Creation Environment Function*) Função de Ambiente de Criação de Serviços, FE da IN;

SCEP: (*Service Creation Environment Point*) Ponto do Ambiente de Criação de Serviços, PE da IN onde os serviços são criados;

SCF: (*Service Control Function*) Função de Controle de Serviço, FE central no processamento de serviços da IN;

SCP: (*Service Control Point*) Ponto de Controle de Serviços, PE da IN;

SDF: (*Service Data Function*) Função de Serviço de Dados, FE da IN;

SF: (*Service Feature*) Característica de Serviço na IN;

SIB: (*Service Independent building Block*) Bloco Independente construtor de Serviço na IN;

SLP: (*Service Logic Processing Programs*) programas de processamento lógico de serviço na rede inteligente;

SMAF: (*Service Management Access Function*) Função de Acesso à Gerência de Serviços, FE da IN;

SMAP: (*Systems Management Application Process*) Processo de Aplicação de Gerência de Sistemas na OSI-MF;

SMAP: (*Service Management Access Point*) Ponto de Acesso de Gerência de Serviços. PE da IN;

SMF: (*Service Management Function*) Função de Gerência de Serviço, FE da IN;

SMP: (*Service Management Point*) Ponto de Gerência de Serviços. PE da IN;

SN: (*Service Node*) Nó de Serviço, PE da IN;

SNMP: (*Simple Network Management Protocol*) Protocolo Simples de Gerência de Rede usado na INMF;

SRF: (*Specialized Resource Function*) Função de Recurso Especializado, FE da IN;

SSCP: (*Service Switching Control Point*) Ponto de Controle de Comutação de Serviços, PE da IN;

SSD: (*Service Support Data*) Dados de Suporte ao Serviço, parâmetros requeridos por um SIB;

SSF: (*Service Switching Function*) Função de Chaveamento de Serviçoi, FE da IN;

SSP: (*Service Switching Point*) Ponto de Comutação de Serviços, PE da IN;

SunOS: (*Sun Operating System*) um dos sistemas operacionais utizados pela SUN Microsystems.

TMN: (*Telecommunications Management Network*) Rede de Gerência de Telecomunicações;

VASP: (*Value Added Service Provider*) Provedor de Serviço de Valor Agregado na TMN.

Apêndice

Definição 16 (Relação em um conjunto) *Uma* relação α em um conjunto D *é um subconjunto dos pares ordenados do produto cartesiano* $D \times D$.

Definição 17 (Reflexividade) *Uma relação* α *em um conjunto* D *é dita* reflexiva *se, para toda* $a \in D, a \alpha a$.

Definição 18 (Anti-simetria) *Uma relação* α *em um conjunto* D *é dita* anti-simétrica *se, para* a *e* b *em* D, $a \alpha b$ *e* $b \alpha a$ *implicam em* $a = b$.

Definição 19 (Transitividade) *Uma relação* α *em um conjunto* D *é dita* transitiva *se, para* a, b *e* c *em* D, $a \alpha b$ *e* $b \alpha c$ *implicam em* $a \alpha c$.

Definição 20 (Ordenamento Parcial) *Um ordenamento parcial, denotado por "*$\leq$*", é uma relação reflexiva, anti-simétrica e transitiva em um dado conjunto.*

Definição 21 (Conjunto Parcialmente Ordenado) *Um conjunto parcialmente ordenado, denotado por* $\langle D, \leq \rangle$ *consiste em um conjunto* D *e um ordenamento parcial* $\leq$ *em* D.

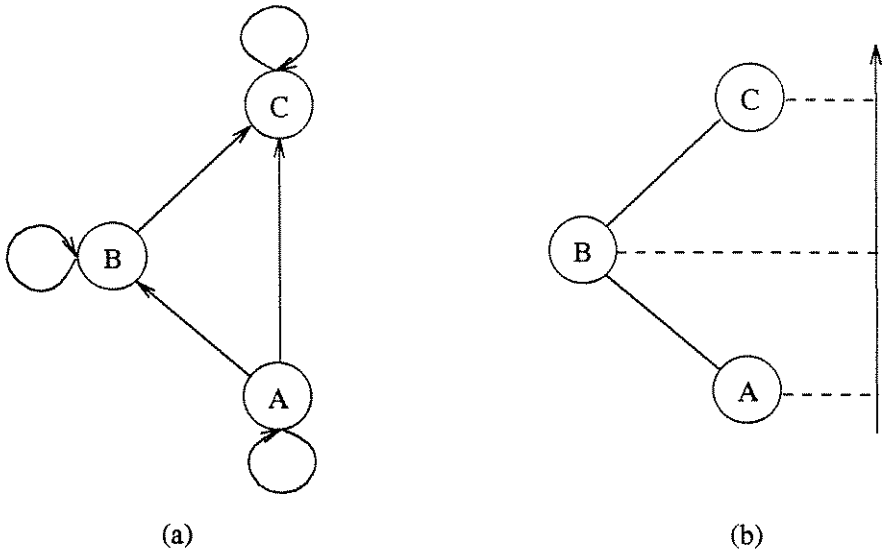
Um exemplo de conjunto parcialmente ordenado é um conjunto no qual a relação entre os elementos é dada pela *relação de inclusão*. Esta relação é um ordenamento parcial porque cada elemento contém ele mesmo (reflexividade), se $A \subseteq B$ e $B \subseteq A$, então $A = B$ (anti-simetria), e $A \subseteq B$ e $B \subseteq C$, então $A \subseteq C$ (transitividade). Uma representação possível para o conjunto $\{A, B, C\}$ onde $A \subseteq B \subseteq C$ é mostrada na Figura 6.1(a).

Geralmente esta representação é simplificada em uma forma chamada *diagrama de hasse*, mostrado na Figura 6.1(b), na qual usa-se apenas a relação de *predecessor imediato*[1] de tal forma que a projeção no eixo vertical indique uma enumeração consistente com o ordenamento entre os elementos.

Definição 22 (Menor Limitante Superior) *O menor limitante superior* $\bigvee_{x \in A} x$ *(ou* $\bigvee A$*) de um conjunto* $A \subseteq D$ *é o menor dentre todos os elementos* $u \in D$ *para os quais:*

$$\forall x \in A, x \leq u \tag{6.1}$$

[1]Se a é predecessor imediato de b, então não existe c tal que $a \leq c \leq b$.



(a) (b)

Figura 6.1 Representações para $\{A, B, C\}$, onde $A \subseteq B \subseteq C$.

É interessante observar que o menor limitante superior é comparável com todos os elementos de A e, portanto, quando este limitante existe, ele é único. Na Figura 6.2 tem-se que e é o menor limitante superior e d não o é, por não ser comparável com a e b.

A definição de **maior limitante inferior** é dual: $\bigwedge_{x \in A} \equiv l \in D \mid \forall x \in A, x \geq l$.

Definição 23 (Reticulado) *Um reticulado é um conjunto parcialmente ordenado $\langle A. \leq \rangle$, no qual cada par de elementos tem menor limitante superior e maior limitante inferior.*

Um exemplo de reticulado está ilustrado na Figura 6.3. A relação entre os elementos é a de que o elemento inferior é divisor do superior. É fácil verificar que esta relação goza das propriedades de transitividade, reflexividade e anti-simetria. Além disto, este reticulado possui uma característica especial: cada um de seus subconjuntos têm um menor limitante superior e um maior limitante inferior em A. Tome-se, por exemplo, o par $\{2, 5\}$. O maior limitante inferior é o 1 e o menor limitante superior, o 10. Quando possui esta característica, um reticulado é denominado **reticulado completo**.

Definição 24 (Conjunto Dirigido) *Um conjunto $A \subseteq D$ é dirigido se, e somente se cada par de elementos de A tem um limitante superior em A.*

Os limitantes superiores mencionados na Definição 24 não precisam necessariamente ser os *menores* limitantes superiores. Na Figura 6.4, tem-se um exemplo de conjunto dirigido que é subconjunto do reticulado da Figura 6.3.

Definição 25 (Função Monotônica) *Sejam $\langle D, \leq \rangle$ e $\langle D', \leq' \rangle$ conjuntos parcialmente ordenados. A função $f : D \rightarrow D'$ é monotônica se, e somente se preserva a ordem, tal que se $x \leq y$, então $f(x) \leq' f(y)$ para todo $x, y \in D$.*

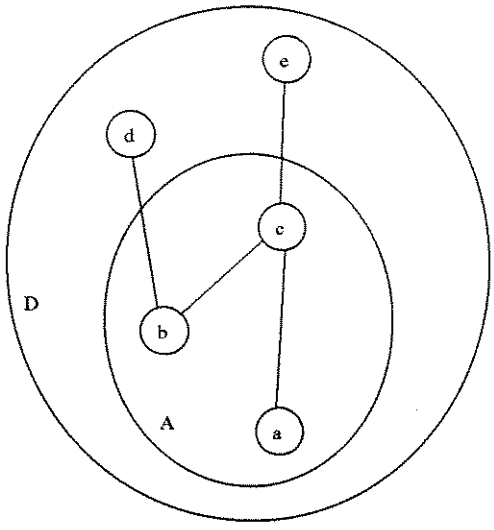


Figura 6.2 D não é o menor limitante superior do conjunto A, pois não é comparável com C e D.

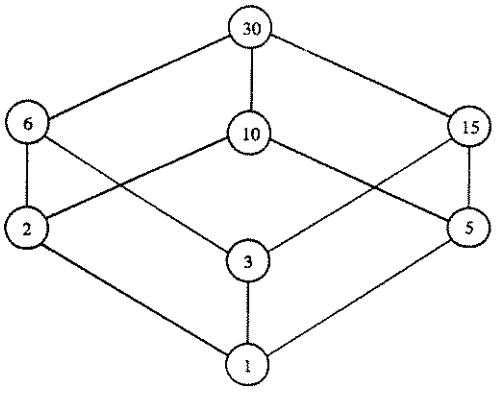


Figura 6.3 Reticulado representando a relação *é divisor de.*

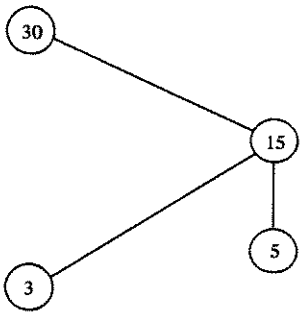


Figura 6.4 Subconjunto dirigido do reticulado da Figura 6.3.

Bibliografia

[1] W. B. Ackerman. Data flow languages. *Computer*, 15(2), 1982.

[2] T. Agerwala and Arvind. Data flow systems. *Computer*, 15(2), 1982.

[3] G. Agha. *Actors: A Model of Concurrent Computation in Distributed Systems.* MIT Press, 1986.

[4] G. Agha. An overview of actor languages. *SIGPLAN Notices*, 21(10), 1986.

[5] F. D. Anger. On lamport's interprocessor communication model. *ACM Transactions on Programming Languages and Systems*, 11(3):404–417, 1989.

[6] J. Backus. Can programming be liberated from the von neumann style? a functional style and its algebra of programs. *Communications of the ACM*, 21(8):613–641, 1978.

[7] M. Bafutto and P. C. S. Machado. Arquitetura e perspectiva de evolução da RI brasileira. In *13o. Simpósio Brasileiro de Telecomunicações.* Sociedade Brasileira de Telecomunicações, 1995.

[8] H. Baker. *Actor Systems for Real-time Computation.* PhD thesis, MIT Press, 1978.

[9] H. E. Bal, J. G. Steiner, and A. S. Tanenbaum. Programming languages for distributed computing systems. *ACM Computing Surveys*, 21(3):261–322, 1989.

[10] M. Ben-Ari. *Principles of Concurrent Programming.* Prentice-Hall Internatinal, 1982.

[11] D. Bjorner and C. B. Jones. *Formal Specification and Software Development.* Prentice-Hall International, 1982.

[12] T. Bolognesi and E. Brinksma. Introduction to the ISO specification language LOTOS. *Computer Networks and ISDN Systems*, 14:25–59, 1987.

[13] R. Braden, D. Clark, and S. Shenker. *Request for Comments 1633 - Integrated Services in the Internet Architecture: an Overview.* Network Working Group, 1994.

[14] S. D. Brookes, C. A. R. Hoare, and W. Roscoe. A theory of communicating sequential processes. *Journal of the Association of Computing Machinery*, 31(3):560–599, 1984.

[15] E. Cardozo and J. S. Sichman. DPSK+P user's manual - C++ interface, version 1.0. Technical report, FEE/Unicamp, 1992.

[16] R. Chin and S. T. Chanson. Distributed object-based programming systems. *ACM Computing Surveys*, 23(1), 1991.

[17] A. Church. *The Calculi of the Lambda-Conversion*, volume 6 of *Annals of Mathematical Studies*. Princeton University Press, 1941.

[18] J.P. Claudé. *Proposition d'une spécification d'administration de réseaux héterogènes - Thèse d'état*. PhD thesis, Université Pierre et Marie Curie, 1987.

[19] W. Clinger. *Foundations of Actor Semantics*. PhD thesis, MIT Press, 1981.

[20] R. Cmelik, N. A. Gehani, and W. D. Roome. Experience with multiple processor versions of concurrent c. *IEEE Transactions on Software Engineering*, 15(3):335–344, 1989.

[21] D. E. Comer. *Internetworking with TCP/IP: Principles, Protocols and Architecture*. Prentice Hall, 1988.

[22] A. Danthine and B. Hauzeur. Management of a backbone wideband network for LAN interconnection. In J. P. Cabanel, G. Pujolle, and A. Danthine, editors. *Local Communication Systems: LAN and PBX*. Elsevier Science Publishers, 1987.

[23] K. X. S. de Souza and N. Agoulmine. Using distributed active object model to manage broadband networks. In *Proceedings of the IEEE ATM Workshop*, 1995.

[24] K. X. S. de Souza and I. S. Bonatti. Using distributed active object model to implement TMN services. In *to appear in: Proceedings of the International Conference on Computers and Their Applications*, San Francisco, USA, 1996.

[25] G. Dickson and A. Lloyd. *Open Systems Interconnection - Computer Communications Standards and GOSIP Explained*. Prentice Hall International, 1992.

[26] A. Goldberg e D. Robson. *Smalltalk-80: The Language and its Implementation*. Addison-Wesley, 1983.

[27] J. R. Eykholt, S. R. Kleiman, S. Barton, R. Faulkner, A. Shivalingiah, M. Smith, D. Stein, J. Voll, M. Weeks, and D. Williams. Beyond multiprocessing... multithreading the SunOS kernel. In *Proceedings of Summer USENIX Conference*, 1992.

[28] J. Ferber and P. Carle. Actors and agents as reflective concurrent objects: A MERING IV perspective. *IEEE Transactions on Systems, Man and Cybernetics*, 21(6):1420–1436, 11 1991.

[29] B. Myhrhaug e K. Nygaard G. M. Birtwistle, O. J. Dahl. *Simula Begin*. van Nostrand Reinhold, 1973.

[30] B. O. Gallmeister and C. Lanier. Early experience with POSIX 1003.4 and POSIX 1003.4a. In *Proceedings of the IEEE Symposium on Real-time Systems*, pages 190–198, 1991.

[31] A. Goscinski. *Distributed Operating Systems - The Logical Design*. Addison-Wesley, 1991.

[32] I. Greif. Semantics of communicating parallel processes. Technical Report Project MAC Technical Report 154, MIT, 1975.

[33] B. Hauzeur and A. Danthine. Internet layer and management design for a backbone network. In I. N. Dallas and E. B. Spratt, editors, *Issues in LAN Management - IFIP TC6/W6.4A Workshop on LAN Management.* Elsevier Science Publishers B.V., 1988.

[34] P. Henderson. *Functional Programming Applications and Implementation.* Prentice-Hall International, 1980.

[35] C. Hewitt and H. Baker. Actors and continuous functionals. In *Proceedings of IFIP Conference on Formal Description of Programming Concepts*, 1977.

[36] C. Hewitt and H. Baker. Laws for communicating parallel processes. In *IFIP Congress Proceedings*, 1977.

[37] C. Hewitt and J. Inman. DAI betwixt and between: From "intelligent agents" to open systems science. *IEEE Transactions on Systems, Man and Cybernetics*, 21(6):1409–1419, 11 1991.

[38] C. A. R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–676, 1978.

[39] ISO. *ISO/IS 8824 - Specification of Abstract Syntax Notation One (ASN.1)*, 1987.

[40] ITU. *ITU-T Recommendation I.312/Q.1201 - Principles of Intelligent Network Architecture*, 1992.

[41] ITU. *ITU-T Recommendation I.320 - ISDN Protocol Reference Model*, 1992.

[42] ITU. *ITU-T Recommendation I.320 - ISDN Protocol Reference Model and its Application*, 1992.

[43] ITU. *ITU-T Recommendation Q.1203 - Intelligent Network Global Functional Plane Architecture*, 1992.

[44] ITU. *ITU-T Recommendation X.700 - Management Framework Definition for Open Systems Interconnection*, 1992.

[45] ITU. *ITU-T Recommendation Q.1204 - Intelligent Network Distributed Functional Plane Architecture*, 1993.

[46] ITU. *ITU-T Recommendation Q.1205 - Intelligent Network Physical Plane Architecture*, 1993.

[47] ITU. *ITU-T Recommendation Q.700 - Introduction to Signalling System No. 7*, 1993.

[48] ITU and ISO/IEC. *ITU-T Recommandation M.3010 - ODP-RM Principles pour un Réseau de Gestion de Télécommunications*, 1992.

[49] ITU,ISO/IEC. *ITU-T Draft Recommendation X.901 - ODP-RM Overview*, 1994.

[70] D. Scott. *Continuous Lattices*, volume 274 of *Lecture Notes in Mathematics*. Springer Verlag, 1971.

[71] D. Scott. Data types as lattices. *SIAM Journal of Computing*, 5(3):522–587, 1976.

[72] L. Sha, R. Rajkumar, and J. P. Lehoczky. Priority inheritance protocols. *IEEE Transactions on Computers*, 39(9), 1990.

[73] E. Shapiro. Concurrent prolog: a progress report. *Computer*, 19(8), 1986.

[74] M. Sloman. Distributed systems management. In I. N. Dallas and E. B. Spratt, editors, *Issues in LAN Management - IFIP TC6/W6.4A Workshop on LAN Management*. Elsevier Science Publishers B.V., 1988.

[75] D. Stein and D. Shah. Implementing lightweight threads. In *Proceedings of Summer USENIX Conference*, 1992.

[76] J. E. Stoy. *Denotational Semantics: The Scott-Strachey Approach to Programming Language Semantics*. MIT Press, 1977.

[77] J. E. Stoy. Mathematical foundations. In D. Bjorner and C. B. Jones, editors, *Formal Specifications and Software Development*. Prentice-Hall International, 1982.

[78] Sun Microsystems. *Lightweight Processes - Programming Utilities and Libraries Manual*, 1988.

[79] A. S. Tanenbaum. *Computer Networks*. Prentice Hall International, 1989.

[80] M. M. Tanik and E. S. Chan. *Fundamentals of Computing for Software Engineers*. Van Nostrand Reinhold, 1991.

[50] ITU,ISO/IEC. *ITU-T Draft Recommendation X.902 - ODP-RM Descriptive Model*, 1994.

[51] ITU,ISO/IEC. *ITU-T Draft Recommendation X.903 - ODP-RM Prescriptive Model*, 1994.

[52] ITU,ISO/IEC. *ITU-T Draft Recommendation X.904 - ODP-RM Architectural Semantics*, 1994.

[53] S. Khanna, M. Sebrée, and J. Zolnowsky. Realtime scheduling in SunOS 5.0. In *Proceedings of Winter USENIX Conference*, 1992.

[54] P. W. King. Formalization of protocol engineering concepts. *IEEE Transactions on Computers*, 40(4):387–403, 1991.

[55] L. Lamport. Time, clocks and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, 1978.

[56] H. Lieberman. A preview of act 1. Technical Report A.I. Memo 625, MIT, 1981.

[57] B. Liskov. Distributed programming in argus. *Communications of the ACM*, 31(3), 1988.

[58] B. H. Liskov and S. N. Zilles. An introduction to formal specifications of data abstractions. In R. T. Yeh, editor, *Current Trends in Programming Methodology*. Prentice Hall, 1977.

[59] Inmos Ltd. *Occam Programming Manual*. Prentice-Hall, 1984.

[60] R. Milne. *A Theory of Programming Language Semantics*. Chapman and Hall, 1976.

[61] R. Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer-Verlag, 1980.

[62] S. J. Mullender and A. S. Tanenbaun. The design of a capability-based distributed operating system. *Computer*, 29(4), 1986.

[63] F. Müller. A library implementation of POSIX threads under UNIX. In *Proceedings of USENIX Conference*, pages 29–41, 1993.

[64] D. A. Mundie and D. A. Fisher. Parallel processing in ADA. *Computer*, 19(8), 1986.

[65] Brisa Sociedade Brasileira para Interconexão de Sistemas Abertos. *Gerenciamento de Redes - Uma Abordagem de Sistemas Abertos*. Makron Books, 1993.

[66] A. Pnueli. The temporal semantics of concurrent programs. *Theoretical Computer Science*, 13, 1981.

[67] J. B. Postel. *Request for Comments 791 - The Internet Protocol*. DDN Network Infomation Center, 1981.

[68] R. S. Pressman. *Software Engineering - A pratictioner's approach*. McGraw-Hill Book Company, 1987.

[69] M. T. Rose. *The Simple Book - An Introduction to Management of TCP/IP-based Internets*. Prentice Hall, 1991.

[70] D. Scott. *Continuous Lattices*, volume 274 of *Lecture Notes in Mathematics*. Springer Verlag, 1971.

[71] D. Scott. Data types as lattices. *SIAM Journal of Computing*, 5(3):522–587, 1976.

[72] L. Sha, R. Rajkumar, and J. P. Lehoczky. Priority inheritance protocols. *IEEE Transactions on Computers*, 39(9), 1990.

[73] E. Shapiro. Concurrent prolog: a progress report. *Computer*, 19(8), 1986.

[74] M. Sloman. Distributed systems management. In I. N. Dallas and E. B. Spratt, editors, *Issues in LAN Management - IFIP TC6/W6.4A Workshop on LAN Management*. Elsevier Science Publishers B.V., 1988.

[75] D. Stein and D. Shah. Implementing lightweight threads. In *Proceedings of Summer USENIX Conference*, 1992.

[76] J. E. Stoy. *Denotational Semantics: The Scott-Strachey Approach to Programming Language Semantics*. MIT Press, 1977.

[77] J. E. Stoy. Mathematical foundations. In D. Bjorner and C. B. Jones, editors, *Formal Specifications and Software Development*. Prentice-Hall International, 1982.

[78] Sun Microsystems. *Lightweight Processes - Programming Utilities and Libraries Manual*, 1988.

[79] A. S. Tanenbaum. *Computer Networks*. Prentice Hall International, 1989.

[80] M. M. Tanik and E. S. Chan. *Fundamentals of Computing for Software Engineers*. Van Nostrand Reinhold, 1991.