2º SIMPÓSIO BRASILEIRO SOBRE REDES DE COMPUTADORES (2º SBRC)

COMPOSIÇÃO DE CAMADAS DE PROTOCOLOS

Ruy José Guerra Barretto de Queiroz (*)
Divisão de Suporte
Núcleo de Processamento de Dados da UFPE
Av. dos Reitores, s/n
Cidade Universitária 50.000 Recife - PE

Paulo Roberto Freire Cunha (**)
Departamento de Informática da UFPE
Centro de Ciências Exatas e da Natureza
Av. Prof. Luiz Freire s/n
Cidade Universitária 50.000 Recife - PE

RESUMO

Considerando que implementação de protocolos é essencialmente programação, buscamos construir uma metodologia baseada nas idéias de programação estruturada e nas técnicas formais de especificação de protocolos, que facilite a tarefa de implementação de arquiteturas hierárquicas de protocolos. Esperamos aproximar tanto quanto possível as técnicas formais de projeto, especificação e validação, e as metodologias de programação empregadas na implementação de protocolos em arquiteturas hierárquicas. Procuramos vivenciar o problema através de um exercício prático de implementação e composição de dois níveis de protocolos, utilizando uma linguagem de programação do tipo PASCAL.

(*) Aluno do Mestrado em Informática do Departamento de Informática da UFPE.
(**) Professor Adjunto do Departamento de Informática da UFPE.

1. INTRODUÇÃO

A comunicação entre computadores é um problema de domínio tão amplo, e envolve um número tão grande de mecanismos de transporte, integridade e segurança de informações, que a comunidade científica da computação decidiu dividi-lo em subproblemas de domínios menores, e estabelecer faixas de responsabilidade, seguindo um critério hierárquico-construtivo. Desta maneira, alguém se serve dos serviços oferecidos por aquele que se encontra hierarquicamente inferior, para realizar tarefas solicitadas por outrem numa posição superior, segundo a hierarquia ([16]). A cada nível ou camada resultante dessa estratificação cabe um problema menos complexo de comunicação entre computadores, e está associado um conjunto regras disciplinando a troca de mensagens. A esse conjunto de regras atribui-se um termo já utilizado, no mesmo sentido, inclusive, pelas Ciências Sociais: protocolo de comunicação.

Os computadores contemporâneos são máquinas programáveis, e portanto, para implementar um certo protocolo de comunicação é preciso construir um programa que execute o protocolo nesse computador, realizando as tarefas destinadas a cada camada da arquitetura. É de se esperar que a implementação ou programação seja, de fato, independente entre os níveis ou camadas adjacentes, e que a composição da arquitetura hierárquica não apresente dificuldades. Na prática, a programação tem sido feita de forma independente, até mesmo porque em geral trata-se de implementação de um certo conjunto de regras de procedimento à ocorrência de eventos conhecidos. A composição dos diversos níveis para formar uma arquitetura, entretanto, revela problemas que, na maioria das vezes, é tratado de uma forma "ad hoc". A escolha de uma disciplina que evite conflitos no acesso a dados compartilhados por níveis adjacentes, e a própria passagem de dados e comandos entre as camadas, constituem-se em principais exemplos desses problemas.

O trabalho aqui descrito procura, através de uma experiência prática de implementação, visualizar caminhos para o estabelecimento de uma metodologia que não só facilite e permita a sistematização do problema da implementação de protocolos em arquiteturas hierárquicas, como também proporcione um maior entendimento do assunto. Com ela esperamos tornar possível e natural o uso de técnicas formais no projeto, especificação, implementação e validação de protocolos, programas e arquiteturas desse tipo. Procuramos nos servir das idéias e conceitos firmados a respeito de Programação Estrutura

da ([2], [9], [10] e [15]) e Tipos Abstratos de Dados ([6], [7], e [12]), Especificação e Verificação de Protocolos usando Técnicas Formais ([13]), Programação para Sistemas Distribuídos orientada por Mensagens ([3] e [4]), Modelagem de Sistemas Distribuídos ([11]), e, finalmente, Especificação de Protocolos usando uma linguagem de Programação do tipo PASCAL ([5] e [14]).

2. MODELO DE PROGRAMAÇÃO POR EVENTOS

A exemplo da linguagem "SPEX" de especificação de protocolos ([13]), adotamos o modelo de programação em que tudo acontece por demanda, ou seja, à medida que existe informação disponível para processamento, um determinado conjunto de ações pode ser acionado para dar o devido tratamento a essa informação. As condições de estabelecimento da disponibilidade de dados para processamento é representada por sentenças de "pré-condição". Cada "pré-condição" é uma sentença "booleana", predicando sobre valores de variáveis do programa, que, resultando verdadeira, "habilita" a execução do conjunto de ações associado. O estabelecimento de uma "pré-condição" representa para nosso modelo, a ocorrência de habilitação de um evento.

Para garantir a integridade dos dados manipulados pelo respectivo conjunto de ações, e assegurar que as condições do ambiente de execução são as mesmas estabelecidas no instante do disparo do evento, acrescenta-se o caráter de atomicidade aos eventos. O que significa dizer que as ações associadas a um certo evento são indivisíveis, atômicas. A fidelidade às características de não-determinismo de um sistema distribuído é representada pela avaliação e disparo de "pré-condições" e eventos, respectivamente, de uma forma não-determinística.

Ainda com relação ao caráter de atomicidade dos eventos, é importante observar o seguinte: uma vez disparado um evento, e estando em execução suas respectivas ações, todas as pré-condições passam a ser indefinidas ou irrelevantes, inibindo a habilitação de outro evento, e, ao término da execução, todas as sentenças de pré-condição são reavaliadas. Por outro lado, a certeza de que o mesmo evento não estará indefinidamente habilitado reside na obrigatoriedade de existência, dentro do conjunto de ações associadas, de uma operação com mudança de valor sobre alguma variável envolvida na sentença de pré-condição, de forma a torná-la falsa ou indefinida.

O paralelismo do sistema fica modelado pela seleção e disparo não determinísticos de eventos, e, como é de se esperar, o grau de paralelismo está intimamente relacionado com o grau de independência das operações sobre as variáveis mencionadas nas sentenças de pré-condição.

3. ESTRUTURAÇÃO DE DADOS

Reconhecendo que a estruturação de um programa pede a estruturação dos dados por ele manipulados, e que esta, por sua vez, ajuda a modularização do programa, permite a construção de programas mais claros e mais elegantes, e facilita a própria construção de programas mais confiáveis ao "esconder" do programador os detalhes irrelevantes, procuramos usar a idéia de abstração de dados, definindo tipos estruturados não necessariamente disponíveis na linguagem de programação. A idéia é permitir que a estrutura e os tipos de dados do programa se aproximem tanto quanto possível do problema que o programa é suposto resolver. O tipo "fila de itens (mensagens, pacotes, quadros, etc.)", por exemplo, bastante utilizado em ambientes de processamento distribuído, foi definido segundo o modelo de classes de linguagem SIMULA: as operações sobre o tipo são os seus atributos. Assim, o acesso e a manipulação das variáveis do tipo "fila" são feitos unicamente através das operações definidas sobre o tipo.

As operações sobre o tipo fila, do ponto de vista funcional, são divididas em três grupos bem distintos, de forma a facilitar a implementação e o tratamento sistemático da especificação do tipo: a) operações de construção, tais como, no caso do tipo "fila", a.1) criar uma fila, a.2) inserir um item em uma fila; b) operações de seleção, tais como b.1) ler o conteúdo da cabeça de uma fila, b.2) testar se uma certa fila vazia; e, finalmente, c) operações de extensão, como por exemplo, c.1) concatenar duas filas, c.2) limpar uma fila tornando-a vazia.

4. GERENCIADOR DE INTERFACE

A experiência que descrevemos neste trabalho tem como um dos principais objetivos, dentro da idéia de estabelecer uma metodologia para implementação de protocolos em arquiteturas hierárquicas, a criação de um mecanismo que faça a associação de mensagens e comandos entre camadas de protocolo adjacentes, e resolva os casos de conflito de acesso a dados compartilhados entre essas camadas. De-

sejamos que o referido mecanismo realize estas tarefas de uma forma tão transparente quanto possível para os programas que implementam os protocolos, de forma a facilitar o trabalho do programador. Esse mecanismo, que decidimos chamar de "gerenciador de interface", deve ser idêntico, do ponto de vista funcional, qualquer que sejam os níveis sobre cuja interligação ele seja o responsável.

Uma representação conveniente para o canal de chegada e saída de mensagens em um determinado ponto de um programa para sistemas distribuídos, é aquela feita através de filas, estruturas de dados conhecidas também como "FIFO" (o primeiro item que entra é o primeiro que sai). Essa representação se presta particularmente bem para o caso de um canal que liga entidades que se comunicam assincronamente, com velocodades de transmissão e recepção não necessariamente iguais. Não é certamente diferente do tão conhecido "buffer", que tem o pepel de compensar variações da velocidade de produção e consumo de mensagens entre entidades comunicantes.

Numa arquitetura hierárquica de protocolos, espera-se que cada camada tenha autonomia para enviar e receber mensagens. Por estar lidando justamente com esse tipo de arquitetura é que decidimos que a comunicação entre camadas seria feita através de filas e variáveis compartilhadas e sob responsabilidade do "gerenciador da interface". Dessa forma, um certo nível "N" desejando enviar mensagem ao nível inferior, simplesmente solicitaria inserção de mensagem na fila de saída do nível "N", com destino ao nível "N-1". O nível "N", então, "vê" 4(quatro) filas distintas: um par compartilhado com cada nível adjacente, sendo uma de entrada e uma de saída em cada interface. (Fig. 1)

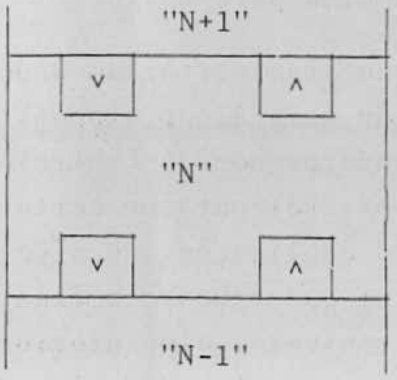

Fig. 1 Nível "N" "vê" 4 filas, um par em cada interface: uma saindo e outra entrando.

Assim, cada nível preocupa-se em manter apenas variáveis de sua privativa propriedade. As variáveis compartilhadas entre os níveis adjacentes são de inteira responsabilidade do gerenciador de interface: ele é quem deve tratar da disponibilidade de mensagens nas suas filas, além de resolver possíveis conflitos de acesso a variáveis compartilhadas. O programa que implementa o nível "N", por exemplo, conhece as filas compartilhadas com os níveis "N + 1" e "N - 1" através do produto cartesiano (interface, sentido), onde a primeira coordenada pode assumir os valores "inferior" e "superior", e a segunda pode assumir os valores "entra" e "sai".

O gerenciador de interface é o responsável pela associação da fila de saída do nível superior com a fila de entrada do nível inferior, e vice-versa. Para ele, então, existem apenas 2(duas) filas na interface sob sua responsabilidade: uma com fluxo de dados de baixo para cima, e outra com fluxo de dados de cima para baixo. (Fig. 2)

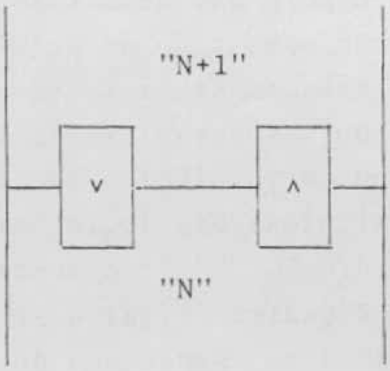

Fig. 2 O gerenciador da interface "vê" 2 filas: uma de baixo para cima e outra de cima para baixo.

Além das filas que transportariam dados entre as camadas, sentimos a necessidade de mais algum dispositivo de comunicação que permitisse a troca de informações de controle entre as camadas. Seria necessário tornar possível para um certo nível, por exemplo, a abertura ou o fechamento de ligações do nível inferior a ele. Imaginamos então, uma variável que tivesse a finalidade de passar um comando de um nível para o nível a ele inferior. Essa variável deveria ser, evidentemente, compartilhada entre os níveis adjacentes, segundo a idéia do nosso modelo de programação por eventos delineado na seção 2 deste trabalho: o nível superior ao emitir um comando através dessa variável compartilhada, estaria habilitando um certo

evento na camada inferior, como por exemplo, abrir ou fechar uma ligação. O conjunto de ações associadas a este evento deveria, ao ser acionado, modificar o valor de tal variável, indicando que já recebeu e processou o comando. Essa modificação, por sua vez, habilitaria um evento na camada superior, indicando que a ligação estaria disponível ou que não estaria sendo possível completá-la, dependendo do resultado da tentativa de abertura dessa ligação ou que os recursos previamente alocados para a conexão estariam disponíveis no caso de fechamento dessa ligação.

A troca de mensagens entre os níveis adjacentes é então feita sob responsabilidade do gerenciador, que deve resolver as questões de conflito de acesso aos "recursos" (variáveis) compartilhados entre as camadas. Considerando o gerenciador como o único caminho de acesso aos recursos compartilhados, alcançamos um estágio de modularização e estruturação de dados tal que a implementação dos protocolos e do próprio gerenciador pode adotar qualquer umas das técnicas de programação concorrente, seja através de variáveis compartilhadas (semáforos, região crítica, monitor), seja através de passagem de mensagens. Neste último caso, se poderia viabilizar a implementação de uma arquitetura hierárquica de protocolos em processadores distintos, com espaços de endereçamento de memória não necessariamente compartilhados.

5. CONSTRUÇÃO E COMPOSIÇÃO DAS CAMADAS DA ARQUITETURA

Estabelecido o modelo de programação e o mecanismo de comunicação entre os programas que realizam os protocolos das camadas de uma certa arquitetura hierárquica, propomos uma metodologia onde identificamos e executamos os passos fundamentais a serem dados para a construção e composição dos níveis da arquitetura. A identificação desses passos precisa ser bem sucedida, pois, dela dependem a eficiência e a viabilidade da tarefa de implementação da arquitetura hierárquica de protocolos. Procuramos então percorrer todo esse processo, abordando sucintamente desde os aspectos de especificação até considerações de eficiência e viabilização do paralelismo da implementação.

5.1. Especificação

Seguindo o caminho aparentemente natural para resolução de problemas, começamos a analisar os aspectos relativos à especificação do ambiente. Diversas técnicas de especificação de sistemas con

correntes, aplicadas a projeto e validação de protocolos de comunicação, têm sido usadas para descrição dos serviços oferecidos e dos protocolos propriamente ditos, de cada camada em separado. A grande maioria dessas técnicas tem procurado aproveitar o formalismo e os resultados analíticos já obtidos através dos estudos da teoria de autômatos.

Muitas dessas técnicas têm sido bem sucedidas quando aplicadas a protocolos de complexidade relativamente baixa, segundo o critério que considera o estreito relacionamento entre o grau de complexidade e a diversidade de tipos e valores de dados com os quais o protocolo está envolvido. Essa restrição se deve principalmente à relativa pobreza de expressão da linguagem dos autômatos para o ambiente em questão, pois ela provê a possibilidade de representação do comportamento de uma máquina apenas através de estados e entradas (ou eventos). Quando um certo protocolo leva em consideração os valores de determinadas variáveis (número de sequência das mensagens, por exemplo), observa-se que sua descrição através de um autômato é bastante prejudicada na sua calreza e na sua complexidade: a explosão de estados de forma a representar os diversos valores que podem assumir tais variáveis, torna-se inevitável. Algumas adaptações e até extensões à linguagem pura têm sido feitas na tentativa de contornar o problema. Porém, como na maioria dos casos de extensão, o modelo que surge a partir da linguagem original estendida perde o caráter de generalidade, e muitas vezes torna-se mais difícil de ser analisado. O que acontece normalmente é que as extensões são fortemente influenciadas pelo contexto em que estão inseridas, resultando daí particularidades não raro indesejáveis.

Em uma outra abordagem ao problema da especificação de protocolos, procura-se tratar protocolos como programas não necessariamente sequenciais. Dada a ainda pouca maturidade das ferramentas de tratamento formal de programas, e, sobretudo de programas concorentes ou paralelos, essa abordagem não tem sido escolhida para as tarefas de validação de protocolos. O fato dela ser preterida, em confronto com as abordagens baseadas em autômatos, está diretamente relacionado com a dificuldade ainda encontrada para se verificar propriedades fundamentais de protocolos usando os mecanismos formais confiáveis que ela oferece, que são poucos ou ainda não significativos do ponto de vista da sua utilização. As técnicas que fazem uso da matemática das máquinas de estado, entretanto, têm propiciado a verificação de propriedades inportantes tais como ausência de impasse, progresso, ausência de estados inoperantes ("liveness"), etc.,

embora que restrita a protocolos de complexidade relativamente baixa.

Apesar da temporária desvantagem no tratamento dos problemas de verificação, as técnicas de especificação de protocolos através de linguagens de programação parecem ganhar terreno em relação às demais, devido sobretudo aos enormes esforços que têm sido realizados em prol do tratamento sistemático de programas. Todo desenvolvimento nas áreas de Metodologia de Programação, Síntese de Programas, e Prova Automática de Teoremas, redundará certamente no enriquecimento dessa abordagem. Além do mais, a representação de um protocolo através de um programa é aceita bem mais naturalmente que através de um autômato, visto que o executor das tarefas do protocolo nada mais é do que um programa.

Evidentemente que a qualidade da especificação dependerá da clareza, da estruturação e dos critérios de corretude da linguagem de programação. Exemplos dignos de serem mencionados são o Relatório da Especificação dos Protocolos da Rde Local Ethernet ([5]), e os protocolos apresentados com fins didáticos no livro de Tanenbaum ([14]). Ambos utilizam trechos de programas escritos em uma linguagem do tipo PASCAL para representar protocolos preenchendo os requisitos acima.

No nosso trabalho, que tem preocupação não só com a especificação e a verificação propriamente dita, mas também com os aspectos relativos a implementação de protocolos, procuramos seguir a abordagem que utiliza linguagens de programação, aproveitando, entretanto, algumas características dos modelos baseados na máquina de estados finita. Na tentativa de modularização por eventos segundo um critério de coesão funcional ([15]), procuramos tornar fácil a composição dos programas e a viabilização do paralelismo do sistema, mesmo em ambientes sequenciais, adotando para isso um modelo de programação que chamamos de programação por eventos. Esse modelo, que está descrito na seção 2, é fortemente influenciado pela idéia de programação por fluxo de dados ("data flow"), ou programação centrada em transformação ([15]), representada formalmente pela máquina de estados finita. Segundo ele, um programa é contituído basicamente de um conjunto de eventos, um conjunto de pré-condições cada uma associada a um evento, e um conjunto de funções de transformação, que no ambiente de protocolos, costuma-se chamar de ações.

No que diz respeito à especificação propriamente dita dos protocolos da arquitetura a ser implementada, nossa metodologia recomenda procurar construí-la (ou transformá-la, caso já exista al-

guma) através de uma linguagem do tipo PASCAL, seguindo as idéias do modelo adotado, e procurando sempre se manter num nível de abstração tal que não seja influenciado por detalhes da implementação.

5.2. Implementação

A partir da especificação dos protocolos que comporão a arquitetura hierárquica, precisamos estebelecer um plano de transformação do que está especificado, em programas que executem as tarefas desejadas. Enumeramos abaixo, os diversos aspectos a serem levados em consideração:

5.2.1. Em cada nível:

a) Inicialização

No que concerne à inicialização do protocolo de uma certa camada, procuramos dar especial atenção aos aspectos referentes à propriedade de auto-sincronização do protocolo e aos valores iniciais que devem assumir não só as variáveis locais mas também as compartilhadas. A estas últimas atribuímos uma precedência de operação: ou ambos os níveis que eles têm acesso as inicializam; ou, apenas um dos níveis o faz, seja o superior ou o inferios. No primeiro caso, o maior benefício é a independência de cada nível quanto à inicialização de todas as variáveis de seu escopo. No outro caso, renuncia-se a uma total independência para, em compensação, evitar duplicação de procedimentos: um certo nível só inicializa as variáveis compartilhadas com o nível e ele inferior, e espera que as compartilhadas com o nível superior sejam inicializadas por este. O importante é que essa decisão seja tomada com a devida antecedência. Quanto aos valores propriamente ditos que devem ser atribuídos inicialmente às variáveis, devem ser observadas as condições para que o protocolo seja auto-sincronizável, ou seja, mesmo na ocorrência de erros durante a inicialização, o sincronismo da comunicação entre as entidades possa ser estabelecido ([8]).

b) Mensagens

Visando a estruturação dos dados manipulados pelo programa que implementa o protocolo no nível, definimos o tipo estruturado de dados "fila de mensagens" levando em conta os diversos tipos, formatos e tamanhos das possíveis mensagens entre as entidades do nível. Quanto às mensagens trocadas com os níveis adjacentes, a estruturação deve ser feita em consonância com gerenciador da interfa

ce e com o programa que implementa o nível adjacente.

c) Estado

A identificação dos estados possíveis de uma entidade da camada, se não está completamente explícita na especificação, deve ser feita de forma "top-down" (de cima para baixo), seguindo um processo de refinamento passo-a-passo. Tratamos os estados inicialmente segundo uma visão macroscópica, e em seguida fazemos o detalhamento necessário, onde alguns estados podem, inclusive, ser "explodidos" (subdivididos).

d) Eventos

Há necessidade de identificação dos possíveis eventos em cada estado, associando a cada um deles: uma sentença "booleana" que determine suas condições de habilitação; e uma função de transformação (conjunto de ações) a ser aplicada. As sentenças "booleanas", chamadas de "pré-condições" no modelo, devem ser conhecidas por um certo mecanismo de baixo nível que faça a avaliação e o disparo não determinístico de eventos. Uma maneira de fazer isso é definir duas variáveis do tipo "conjunto de eventos" e passá-las a esse mecanismo, uma contendo os eventos possíveis naquele estado, e outra conteria, após a avaliação, os eventos habilitados. O mecanismo de baixo nível então avaliaria não deterministicamente as sentenças de cada evento do primeiro conjunto, inserindo aqueles cuja sentença resultasse verdadeira, no segundo conjunto. O "disparador" de eventos teria então que escolher aleatoriamente um elemento do conjunto de eventos habilitados.

5.2.2. Nas interfaces:

a) Comunicação com camadas adjacentes

Como cada interface está sob a responsabilidade de um certo "gerenciador", a disciplina de acesso aos dados compartilhados entre níveis adjacentes deve ser da forma mais abstrata e simples possível para os programas que implementam os níveis, deixando as complicações e os detalhes de implementação para o gerenciador. As questões das chamadas ao gerenciador serem blocadas ou não, e da representação das diversas situações através de uma codificação dos valores resultantes das chamadas, devem ser tratadas antes de implementação deste. As chamadas blocadas podem ser feitas simplesmente através das chamadas do tipo "função" (function"), ou através da

inibição dos eventos possíveis em função do resultado.

5.2.3. Na arquitetura:

a) Chaveamento de contexto e coexecução

Para garantir o não-determinismo e prover paralelismo à implementação, precisamos verificar as condições de chaveamento de contexto existentes no ambiente hospedeiro. Se implementamos a arquitetura em um único processador que disponha de sistemas operacional multi-usuário por partilha de tempo, temos unicamente de verificar se é possível implementar o programa de cada camada executando em uma unidade de processamento ("job", processo, usuário, etc.). Em caso positivo, não há a necessidade de preocupação com o chaveamento de contexto e a coexecução de fato, pois o próprio sistema operacional com essas características, ao chavear a execução das unidades de processamento, já o estará fazendo. Se, por outro lado, mais de uma camada está executando em uma única unidade de processamento, aí então surge a necessidade de criação de um mecanismo local de baixo nível que faça a troca de contexto.

b) Eficiência

Em se tratando de problemas relacionados com implementação de programas, não há como evitar a preocupação com a eficiência do produto final. No modelo que propomos, dois aspectos têm importância fundamental no que concerne ao grau de eficiência do sistema: a complexidade de cada sentença de "pré-condição", e as condições de chaveamento de contexto.

Cada sentença de "pré-condição" deve envolver operações "booleanas" sobre parcelas funcionais que resultem em valor "booleano". Como exemplo, imaginemos o caso de um evento do tipo "recepção de uma mensagem do tipo X" em um certo nível da arquitetura. A "pré-condição" associada a esse evento seria, por exemplo, "(fila de recepção não vazia) e (tipo da mensagem que está na frente da fila é igual a X)". Segundo nosso modelo, ambos os operandos da operação "e" da sentença são invocações do tipo "function" ao gerenciador da interface inferior. A preocupação com a eficiência nesse caso deve resultar na otimização do cálculo da sentença "booleana", tal como fazem os compiladores otimizantes: se a operação é o "e" lógico, e o primeiro operando resulta falso, então o resultado será falso, sem a necessidade de avaliação do valor do segundo operando.

De uma maneira geral, a eficiência do cálculo das sentenças de "pré-condição" dos eventos pode ser incrementada se, dentro de cada uma, a avaliação do seu valor é feita de forma hierárquica.

No que diz respeito ao chaveamento de contexto, a eficiência é algo intimamente relacionado com o ambiente hospedeiro, ficando, portanto, dependente da arquitetura deste.

6. O AMBIENTE E AS PECULIARIDADES DA EXPERIÊNCIA

Procurando vivenciar o mais realisticamente possível o processo de implementação de protocolos em arquiteturas hierárquicas, embora em caráter experimental, construímos a experiência sob forma de simulação, atentos e com um apurado espírito de investigação das dificuldades que poderiam surgir durante o processo.

O ambiente escolhido deveria ter simplicidade suficiente, de forma a não obscurecer e não dificultar as intenções e idéias básicas do projeto, permitindo a vivência com os diversos aspectos do problema de implementação de protocolos sem interferência de qualquer facilidade específica contextual, dado o caráter de generalidade pretendido. Além do mais, a linguagem de programação, os protocolos e o ambiente deveriam nos ser suficientemente familiares a ponto de não se constituírem em problemas adicionais.

6.1. A Linguagem de Programação

Embora não sendo um exemplo ideal de linguagem para programação estruturada utilizando Tipos Abstratos de Dados, nem para programação concorrente, escolhemos a linguagem PASCAL Sequencial. Diversas razões nos levaram a tal escolha, entre as quais nossa atual disponibilidade de linguagens estruturadas, sua relativa simplicidade, sua larga popularidade e aceitação, e, finalmente, a possibilidade de estruturação dos dados a serem manipulados por um programa escrito na linguagem. Ainda mais, tínhamos o propósito de construir uma metodologia o mais geral possível, e que pudesse ser aplicada em uma grande variedade de ambientes.

6.2. O Hospedeiro e seu Sistema Operacional

Utilizamos o sistema computacional instalado no Núcleo de Processamento de Dados da Universidade Federal de Pernambuco. Trata-se de um "DECsystem 10" da "DIGITAL Equipment Corporation", com sistema operacional por partilha de tempo ("TOPS-10"). Esse Sistema

Operacional faz o chaveamento de contexto considerando um processo ("job") como unidade de processamento, e provê um mecanismo de comunicação entre processos chamado "Inter Process Communication Facility (IPCF)". O IPCF estebelece uma disciplina de comunicação direta, através de um canal com memória. Ele conhece o endereço e mantém as filas de recepção e de transmissão de cada processo. Para faser uso dos recursos do IPCF, cada processo se identifica ao sistema operacional, podendo, a partir de então, enviar mensagens para um outro processo já conhecido pelo IPCF, um de cada vez, e receber mensagens de um determinado processo ou de qualquer processo, Usando termos conhecidos nos ambiente de programação distribuída, diríamos que um certo processo P poderia se comunicar através das primitivas "send(Q)", "receive(Q)" e "receive-any", ou seja, envie para Q, receba de Q, e receba de qualquer processo.

6.3. A Arquitetura de Protocolos

Devido sobretudo aos aspectos de simplicidade e familiaridade relacionados com a linguagem, os protocolos e o ambiente, dos quais já falamos, escolhemos uma arquitetura de dois níveis: um nível de controle de enlace, e um nível físico, correspondendo aproximadamente aos níveis 1 e 2 da arquitetura dos sistemas abertos, definida pela ISO ([16]). Para o nível 2 escolhemos um subconjunto do protocolo definido pela pela Recomendação X.25 da CCITT ([1]) para controle de enlace (*): o LAPB ("Link Access Procedures Dalanced"). O nosso nível 1 é uma simulação por "software" de uma linha física, onde as extremidades da ligação se comunicam através do "IPCF", trocando sempre alguma mensagem ao estouro de um temporizador.

6.4. A comunicação entre camadas com o gerenciador de interface

Quanto à implementação do gerenciador da interface, decidimos pela utilização de duas filas transportando dados, que chamamos de "InteriorParaSuperior" e "SuperiorParaInferior", e uma variável do tipo "comando", que poderia assumir os valores "AbraLigação", "FecheLigação" e "ComandoNulo", Essas variáveis, tanto as filas de

---

(*) Nossa familiaridade advém do fato de estarmos no Grupo de Redes de Computadores e Sistemas Distribuídos da UFPE atualmente envolvidos com um projeto de desenvolvimento de uma comporta X.25 para o DEC-10 da UFPE.

mensagens como o "comando", são compartilhadas entre as camadas adjacentes, e o acesso a elas só pode ser feito através de chamadas ao gerenciador. A disciplina utilizada para resolver os conflitos de acesso é do tipo "semáforo", onde a cada variável está associada uma "Atividade Corrente", que pode assumir os valores "Leitura", "Escrita" ou "Nenhuma". A única atividade com exclusivo acesso é a de "Escrita".

As consultas que o nível superior faz para avaliar a disponibilidade das ligações no nível inferior são feitas através de chamadas "blocadas" ao gerenciador, do tipo "function". Por exemplo, uma função denominada "DisponibilidadeDeLigação" retorna um dos seguintes valores: "Disponível", "Ocupada" ou Desfeita".

6.5. Chaveamento de contexto e habilitação/disparo de eventos

Implementamos cada arquitetura completa, ou cada nó, em um único processo ("job") do sistema operacional, portanto, para permitir a coexecução e o não-determinismo de cada sistema multi-nível, foi necessário criar um mecanismo local de chaveamento de contexto. Esse mecanismo, implementado em baixo nível, foi aproveitado também para realizar as tarefas de avaliação e disparo não determinístico das "pré-condições" e dos eventos, com a obrigação adicional de selecionar a priori e aleatoriamente qual a camada que deveria ganhar controle a seguir.

7. CONSIDERAÇÕES FINAIS

Com o presente trabalho, tencionamos demonstrar a generalidade de aplicação de uma metodologia para implementação de protocolos em arquitetura hierárquica: a) independência do(s) nível(is) de protocolo; b) independência do ambiente em que será implementado, b.1) se em Rede Local ou em Rede a Longa Distância, b.2) se em ambiente distribuído ou sequencial, b.3) se em um ou diversos processadores, b.4) utilizando ou não mecanismos de comunicação entre processos e de chaveamento de contexto em nível adequado; c) e, finalmente, independência quanto à linguagem da programação a ser utilizada.

REFERÊNCIAS:

[1] CCITT: Interface Between Data Terminal Eqipment (DTE) and Data Circuit Equipment (DCE) for Terminals Operating in the Packet Mode on Public Data Networks; Yellow Book, Volume VIII, Fascículo VIII.2, Recomendação X.25, Genève, Suíça, Novembro 1980, (pp. 100-190).

[2] Cowan, D.D., Graham, J.W., Welch, J.W., Lucena, C.J.P.: A Data-directed Approach to Program Construction; Software-Practice and Experience, Vol. 10, 1980, (pp. 355-372).

[3] Cunha, P.R.F.: Design and Analysis of Message Oriented Programs; Tese de Doutoramento, Universidade de Waterloo, Waterloo, Ontário, Canadá, 1981, 199 pp.

[4] Cunha, P.R.F., Lucena, C.J., Maibaum, T.S.E.: On the Design and Specification of Message Oriented Programs; International Journal of Computer and Information Sciences, vol. 9, nº 3. Junho 1980, (pp. 161-191).

[5] DIGITAL Equipment Corporation, INTEL Corporation, XEROX Corporation: The Ethernet - A Local Area Network - Data Link Layer and Physical Layer Specifications; Versão 1.0,Setembro 1980, 82 pp.

[6] Goguen, J.A., Thatcher, J.W., Wagner, E.G.: An Initial Algebra Approach to the Specification, Correctness, and Implementation of Abstract Data Types; Current Trends in Programming Methodology, Ed. R.T. Yeh, Prentice-Hall, 1978, (pp. 80-149).

[7] Guttag, J.V., Horowitz, E., Musser, D.R.: Abstract Data Types and Software Validation; Communications of the ACM, volume 21, número 12, Dezembro 1978, (pp. 1048-1064).

[8] Hajek, J.: Automatically Verified Data Transfer Protocols; Proceedings of the International Computer Communication Conference, Kyoto, Japão, Setembro 1978, (pp. 749-756).

[9] Hughes, J.W.: A Formalization and Explication of the Michael Jackson Method of Program Design; Software-Practice and Experience, vol. 9, 1979, (pp. 191-202).

[10] Jackson, M.A.: Principles of Program Design; A.P.I.C. Studies in Data Processing; nº 12, Academic Press Inc. Corp., Londres, Inglaterra, 1975, 299 pp.

[11] MacQueen, D.B.: Models for Distributed Computing; Proceedings of EEC/IRIA Course on the Design of Distributed Processing Systems, Nice, França, Julho 1978, 35 pp.

[12] Queiroz, R.J.G.B. de, Lopes, M.A.: Especificação e Confiabilidade de Software; Relatório Técnico RT-DI/UFPE-005/83, Departamento de Informática, UFPE, Recife, Brasil, 1983, 33 pp.

[13] Schwabe, D.: Formal Techniques for the Specification and Verification of Protocols; Tese de Doutorado, Report nº CSD-810401, Computer Science Department, UCLA, Los Angeles, Estados Unidos, Abril 1981, 240 pp.

[14] Tanenbaum, A.S.: Computer Networks; Prentice-Hall, Inc., Englewood Cliffs, New Jersey, Estados Unidos, 1981, 515 pp.

[15] Yourdon, E., Constantine, L.L.: STRUCTURED DESIGN - Fundamentals of a Discipline of Computer Program and Systems Design; Prentice-Hall, Inc., Englewood Cliffs, New Jersey, Estados Unidos, 1979, 474 pp.

[16] Zimmermann, H.: OSI Reference Model - The ISO Model of Architecture for Open Systems Interconnection; IEEE Transactions on Communications, vol. COM 28, nº 4, Abril 1980, (pp. 425-432).