UNIVERSIDADE FEDERAL DE SANTA CATARINA
CURSO DE BACHARELADO EM CIÊNCIAS DA COMPUTAÇÃO

Julio Cezar Rodrigues Sincero

Pilha TCP/IP Adaptável à Aplicação

Trabalho de Conclusão de Curso submetido à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Bacharel em Ciências da Computação.

Prof. Dr. Antônio Augusto Medeiros Fröhlich

Florianópolis, fevereiro de 2004

Pilha TCP/IP Adaptável à Aplicação

Julio Cezar Rodrigues Sincero

Este Trabalho de Conclusão de Curso foi julgado adequado para a obtenção do título de Bacharel em Ciências da Computação, e aprovado em sua forma final pela Coordenadoria do Curso de Bacharelado em Ciências da Computação.

_____

Prof. Dr. José Mazzucco Júnior

Banca Examinadora

_____

Prof. Dr. Antônio Augusto Medeiros Fröhlich

_____

Prof. Dr. José Mazzucco Júnior

_____

André Luís Gobbi Sanches

A Jesus Cristo nosso Deus e Salvador.

Resumo

Este trabalho descreve a aplicação das técnicas descritas em AOSD (Application-Oriented System Desing) a área de protocolos de comunicação. Mais especificamente aos protocolos da pilha TCP/IP.

O uso de tais técnicas tem por objetivo a construção de um conjunto de componentes de *software* que sejam altamente configuráveis e adaptáveis de tal forma que possam ser usados nos mais diversos projetos de sistemas embutidos e atendam a aplicação de forma ótima, entregando a esta somente a funcionalidade requerida sem o acrécimo de *overhead*.

Foi também avaliado o tamanho do código objeto gerado em comparação as diferentes funcionalidades (features) exigidas pelas aplicações. Para isto foram realizados testes com aplicações que exigem diferentes funcionalidades do sistema de comunicação.

Abstract

This work describes the use of AOSD (Application Oriented System Design) techniques in the area of communication protocols. Specifically the TCP/IP stack protocols.

The use of these techniques aims at the construction of a set of componentes highly configurable and adaptable in order to be used in embedded systems projects. These components are supposed to deliver the requested functionality by the application, without any overhead of unnecessary functionality.

The code size evaluation was performed in comparison with different application requirements.

Sumário

Lista de Figuras

Capítulo 1

Introdução

Nos útimos anos a eletrônica teve um avanço surpreendente. Nos dias atuais nos encontramos rodeados de dispositivos eletrônicos que são usados para as mais diversas tarefas, desde dispositivos de pequeno porte, como tocadores de Mp3 e telefones celulares até dispositivos de grande porte como aviões e carros de luxo. É indiscutível que os microprocessadores são encontrados em quase todas as atividades do nosso dia a dia, esses sistemas que em geral têm um único propósito são conhecidos como sistemas embutidos.

Os sistemas embutidos possuem algumas particularidades quando comparados aos sistemas de propósito geral (por exemplo computadores padrão PC). Geralmente tais sistemas realizam somente um ou um reduzido número de funções, características como consumo de energia, custo de *hardware* e confiabilidade, são fatores essenciais durante a realização de seu projeto, além de poderem ser concebidos de uma enorme gama de processadores e soluções de *hardware*.

Entretando, sistemas embutidos não são apenas aglomerados de *hardware*. São compostos também de aplicações (*software*) que fazem uso dos recursos de *hardware* para que as operações desejadas possam ser executadas. Devido às características dos sistemas embutidos como baixo custo e hardware simplificado, as aplicações precisam fazer um uso ótimo do hardware, ou seja, evitar qualquer tipo de uso desnecessário dos recursos do sistema. Entre os requisitos da aplicação e de *hardware* que esses sitemas são projetados, e toda forma de otimização que visa diminuir o custo destes sistemas são comumente empregadas. Redução do código da aplicação, otimização do uso do processador são alguns exemplos de otimizações que automaticamente reduziriam os custos de *hardware* e conseqüentemente custos do projeto.

A partir do momento que se percebe que os requisitos de *software* influenciam diretamente os requisitos de *hardware*, se nota a importância do *software* responsável por mediar

o acesso ao hardware à aplicação, pois esse controle precisa ser feito da forma mais otimizada possível de modo a não desperdiçar recursos do sistema. Essa é a responsabilidade do sistema operacional.

O desenvolvimento de sistemas operacionais é uma tarefa trabalhosa e consequentemente cara. Se a cada projeto de sistema embutido um novo sistema operacional for desenvolvido, certamente o custo de tais projetos será consideravelmente incrementado. A solução para tal problema é o desenvolvimento de sistemas operacionais que sejam configuráveis e adaptáveis, de modo que possam ser usados em diferentes projetos de sistemas embutidos. Recentemente diversas técnicas de engenharia de software têm sido aplicadas à área de sistemas operacionas com o intuito de que tais sistemas atinjam um alto grau de configurabilidade e adaptabilidade, características estas que reduziriam o gasto com software no desenvolvimento de sistemas embutidos.

Nesse contexto se encontra o EPOS [dMF 02] (*Embedded Parallel Operating System*). Esse é um sistema operacional que faz uso de diversas das mais avançadas técnicas de engenharia de software para obter um alto grau de configurabilidade e adaptabilidade visando entregar a cada aplicação (paralela ou embutida), a partir de um conjunto de componentes de software, um sistema operacional personalizado.

O uso cada vez maior de sistemas embutidos gera a necessidade de interconexão entre os mesmos. E não só entre os mesmos, mas também com sistemas de grande porte ou de propósito geral. Assim, a possibilidade de se fazer uso destes dispositivos na *Internet* (rede mundial de computadores) se mostra bastante atraente. Pois assim, poderíamos ter dispositvos eletrônicos captando, processando e disponibilizando o mais variado tipo de informações na *Internet* para o nosso uso.

Deste modo, visando um sistema de comunicação apropriado para sistemas embutidos, o desenvolvimento de uma pilha de protocolos para o sistema operacional EPOS, se mostra uma alternativa extremamente interessante.

1.1 Objetivos

O objetivo deste trabalho é fazer uso das técnicas descritas em AOSD[dMF 01] (*Application-Oriented System Design*), que foram usadas no desenvolvimento do EPOS[dMF 02], para a construção de uma pilha de protocolos de comunicação.

O uso de tais técnicas tem por objetivo a construção de um conjunto de componentes de *software* que sejam altamente configuráveis e adaptáveis de tal forma que possam ser

usados nos mais diversos projetos de sistemas embutidos e atendam a aplicação de forma ótima, entregando a esta somente a funcionalidade requerida sem o acrécimo de *overhead*.

Por conseguinte, uma análise da aplicação de AOSD à area de protocolos de comunicação e uma análise da configurabilidade alcançada (comparativa a outras implementações que também visam sistemas embutidos), serão um resultado direto deste trabalho.

Avaliar o tamanho do código objeto gerado em comparação as diferentes funcionalidades (*features*) exigidas pelas aplicacações. Para isto serão realizado testes com aplicações que exigem diferentes funcionalidades do sistema de comunicação.

1.2 Justificativa

Atualmente existe um grande número de implementações da pilha de protocolos TCP/IP para sistemas embutidos, este é sem dúvida o conjunto de protocolos mais utilizado no mundo. A Internet (a rede mundial de computadores) é o exemplo mais claro disto, pois esta faz uso dos protocolos pertencentes ao TCP/IP.

A maioria dessas implementações suprime as funcionalidades dos protocolos e usam *smart programming* (programação inteligente) para reduzir o tamanho do código objeto gerado, afim de serem apropriadas para sistemas embutidos (recursos limitados). Em geral, possuem um nível de configurabilidade muito baixo, assim, as funcionalidades da pilha foram escolhidas durante a implementação e a aplicação que fará uso é forçada a ter alguma funcionalidade não requerida (*overhead*), ou impedida de exigir alguma funcionalidade extra.

Entretanto, uma pilha de protocolos que implemente completamente os protocolos de comunicação, e permita que a aplicação que fará seu uso escolha a funcionalidade desejada, parece ser uma opção muito mais atraente. Pois assim o *overhead* imposto pelas funcionalidades não desejadas da pilha seria evitado. De maneira simples, isto pode ser visto como redução do uso de memória e do processador.

As técnicas descritas em AOSD visam permitir a criação de um conjunto de componentes de software que atinjam exatamente este grau de configurabilidade.

1.3 Visão Geral do Documento

No capítulo 2 é feita a introdução teórica sobre os conceitos da pilha TCP/IP. No capítulo 3 é feita a descrição da técnica AOSD (Application-Oriented System Design). O

capítulo 4 descreve algumas implementações da pilha TCP/IP para sistemas embutidos, e uma análise sobre as mesmas é feita. No capítulo 5 é apresentada toda a modelagem da pilha TCP/IP adaptável à aplicação proposta por este trabalho. O capítulo 6 é feita uma discussão sobre aspectos da implementação e sobre os componentes gerados. O capítulo 7 traz as conclusões e os resultados obtidos.

Capítulo 2

A Pilha TCP/IP

Este capítulo apresenta os conceitos referentes à pilha de protocolos TCP/IP. Serão apresentados os protocolos de comunicação pertecentes a esta assim como sua estrutura e funcionamento.

2.1 Introdução

O protocolo TCP nascido na década de 70, patrocinado pela DARPA (Defense Advanced Research Projects Agency) tinha o objetivo de possuir flexibilidade suficiente para lidar com computadores e redes das mais variadas tecnologias.

Após a especificação dos protocolos TCP e IP várias redes de computadores do departamento de defesa americano foram conectadas por meio desta tecnologia. Essa rede era conhecida como ARPANet. Mas com o passar dos anos, essa rede deixou de ter um uso exclusivamente militar. Intituições governamentais, educacionais assim como comerciais, também foram sendo incluídas . No início da década de 80 a combinação entre os protocolos de transporte TCP e de rede IP foram adotados como os protocolos oficiais da Arpanet e todas as redes conectadas a esta foram requisitadas a suportar o TCP/IP. A partir deste momente essa rede ficou conhecida como Internet e possuia um crescente número de usuários espalhados por todo o mundo.

Uma das razões deste sucesso foi a especificação dos protocolos. Essa especificação é obtida por meio de documentos chamados RFCs (Request for Comments). Também existem RFCs que propõem os mais variados tipos de sugestões, como melhorias e otimizações aos protocolos. Para que um protocolo se torne um Internet Standard, é necessário que seja publicado seu RFC, outros RFCs podem ser produzidos com sugestões a este. Caso se torne estável,

um membro do IAB (Internet Activity Board) pode sugerir ao comitê que o protocolo seja um padrão. Então, um RFC é publicado dando este anúncio, caso não haja objeção em um período determinado pela entidade, o IAP declara o protocolo como um Internet Standard.

Atualmente a pilha TCP/IP é composta por uma grande número de protocolos, e não somente pelos dois que dão nome a esta. Dentre eles podemos citar: ARP (Address Resolution Protocol/Reverse Address), DHCP (Dynamic Host Configuration Protocol), ICMP (Internet Control Message Protocol), UDP (User Datagram Protocol), entre muitos outros.

2.2 Estrutura

O termo *pilha de protocolos* define um conjunto de protocolos que são comumente usados em conjunto. Protocolos de comunicação são um conjunto de regras que visam permitir a comunicação íntegra entre máquinas. Como o conjunto de regras para garantir esse comunicação é extremamente grande, essas regras (os protocolos de comunicação) são divididos em níveis hierárquicos, um protocolo sobre o outro.

Como pode ser visto na figura 2.1 [1], a pilha TCP/IP é dividida em cinco camadas. Cada camada possui uma responsabilidade bem específica, visando a redução da comlexidade de implementação de cada protocolo. Cada camada executa um conjunto de funções afim de prover serviços para a camada imediatamente acima. Deste modo as camadas podem ser vistas como prestadoras de serviço para as camadas acima e usuárias dos serviços prestados pelas camadas imediatamente abaixo.

O fluxo de dados se dá da seguinte maneira: a camada de transporte (camada 4) recebe os dados da aplicação (camada 7), adiciona a este o cabeçalho do protocolo de transmissão sendo usado e passa os dados para a camada de rede (camada 3), então é adicionado o cabeçalho do protocolo de rede em uso e os dados são passados para a camada de enlace de dados(camada 2) que também incluirá seu cabeçalho e finalmente repassará os dados para a camada física (camada 1), esta é responsável pela transmissão dos bits através do canal de comunicação.

Nas próximas seções serão apresentados mais detalhadamante os protocolos pertecentes à pilha TCP/IP que serão mencionados no decorrer deste trabalho.

[1] Fonte: http://www.govtech.net/magazine/govinternetguide/gig98/tcp.phtml

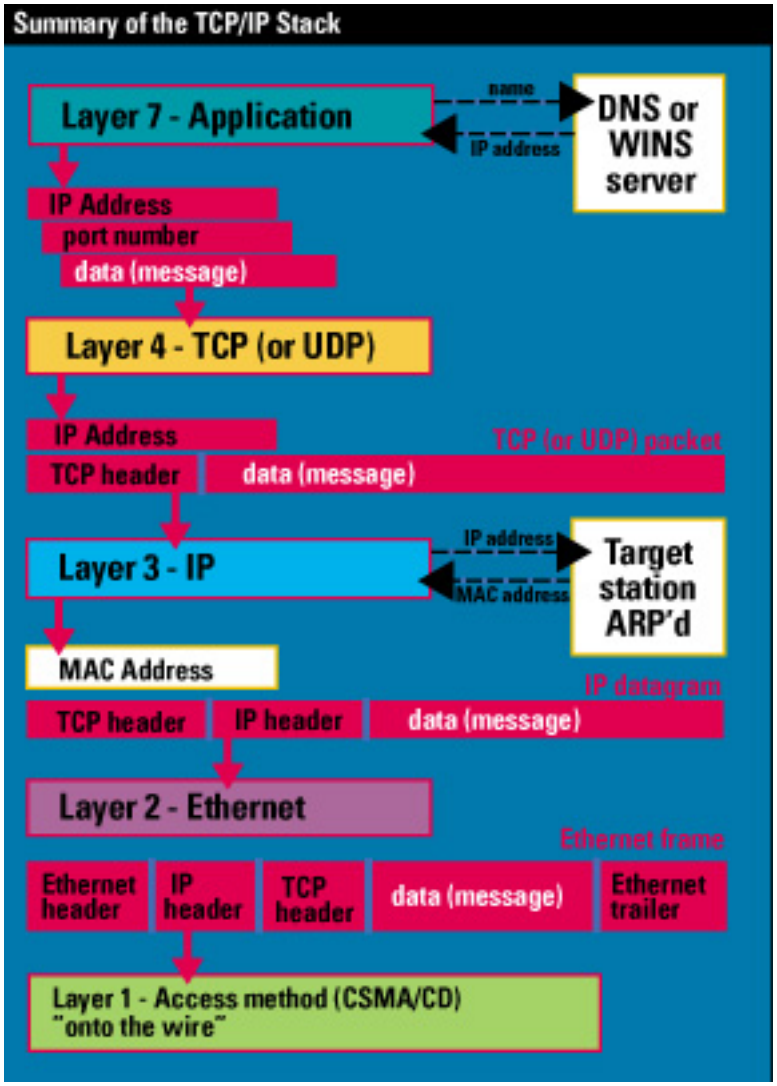

Figura 2.1: Pilha de protocolos TCP/IP

2.3 Ethernet

O termo *Ethernet* se refere ao padrão publicado no inicio da década de 80 pela Digital Equipment Corp., Intel Corp., e Xerox Corp. Atualmente esta é a tecnologia de rede local (Local Area Network - LAN) mais utilizada no mundo em conjunto com a pilha TCP/IP.

Utiliza um método de acesso ao meio chamado CSMA/CD (Carrier Sense, Multiple Access with Collision Detection) e utiliza endereçamento de 48 bits.

O nome *Ethernet* é usado como referência ao padrão IEEE 802.3. Esse padrão, preocupa-se com elementos da física e da camada de enlace de dados. Ao nível de enlace de dados, existem subcamadas de controle lógico de link (Logical Link Control - LLC) e MAC, que são as mesmas para as diversas variedades e velocidades do padrão *ethernet*. A camada LLC é

responsável por identificar os dados transportados em um quadro ethernet. A camada MAC é responsável pelo protocolo usado para arbitrar o acesso ao sistema ethernet [STE 00b].

O núcleo do sistema ethernet é o quadro. O hardware da rede (interfaces Ethernet, cabos, etc) existe simplesmente para mover quadros ethernet entre as estações. Os bits do quadro ethernet são distribuidos em campos especificados. A figura 2.2 [2] mostra a distribuição dos campos no quadro.

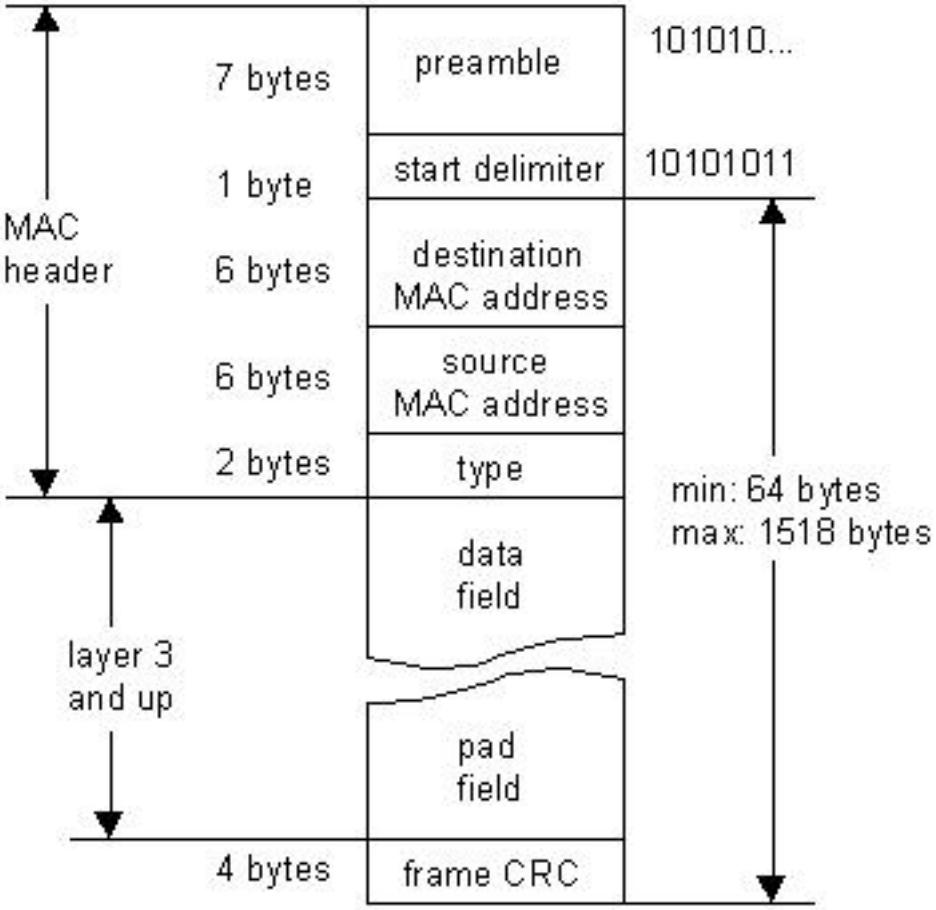

Figura 2.2: Quadro Ethernet

O primeiro campo é o preâmbulo, que é responsável pela sincronização do receptor com o sinal. Pode ser visto como um aviso de que um quadro ethernet está sendo transmitido. É composto por uma sequencia de 64 bits de uns e zeros alternados.

Na sequência vem os endereços de físicos do destino e da fonte respectivamente. Cada um ocupando 48 bits. É de responsabiliadde do procolo ARP a tradução destes endereços

[2]Fonte: http://www.mattjustice.com/ethernet/layer2.html

físicos em endereços IP.

Os próximos dois bytes identificam o tipo do dado que está encapsulado pelo quadro. O RCF 1340 possui os valores que podem ser usados neste campo com relação ao tipo de dado sendo transmitido.

Em seguida vem os dados que estão sendo tranmitidos, pode ser um pacote IP, ARP, etc. Em conformidade com o valor especificado no campo tipo.

Ao fim vem o campo CRC (cyclic redundancy check) que tem por finalidade a detecção de erros no quadro transmitido.

Exite um tamanho mínimo para o quado. São necessários pelo menos 46 bytes no campo de dados. Caso necessário, *padding bytes* devem ser adicionados no campo de dados afim de atingir o tamanho mínimo requerido.

2.4 ARP

O procolo ARP (Address Resolution Protocol) tem por finalidade resolver o problema de mapeamente entre endereços da camada de enlace de dados e da camada de rede. Sua epecificação é dada pelo RFC 826.

Quando um quadro ethernet é enviado de uma estação para outra, é o endereço ethernet (48 bits) que identifica a interface que deve receber o quadro. E não o endereço destino que está no cabeçalho do protocolo da camada superior. Portanto, a finalidade do protocolo é coverter endereços de protocolos (endereços IP) para endereços de redes locais (endereços ethernet).

Os campos do cabeçalho são mostrados na figura 2.3 [3]. Os dois primeiros campos de 16 bits fazem a identificação entre quais tipos de endereço a conversão será realizada. Os dois campos seguintes de 8 bits determinam o tamanho em bytes dos campos de endereço que devem ser convertidos. Esses quatro primeiros campos garantem uma ótima flexibilidade ao protocolo. Pois permite que quaisquer endereços de protocolos sejam convedrtidos em qualquer tipo endereço de hardware. O campo seguinte, especifica o tipo de operação, poder ser uma requisição ou uma resposta.

Os dois campos seguintes devem ser completos com os endereços de hardware e protocolo do emissor e em seguida vem os enderecos de hardware e software do destinatário da mensagem.

[3]Fonte: http://www.pop-rs.rnp.br/ovni/tcpip/t_resol.htm

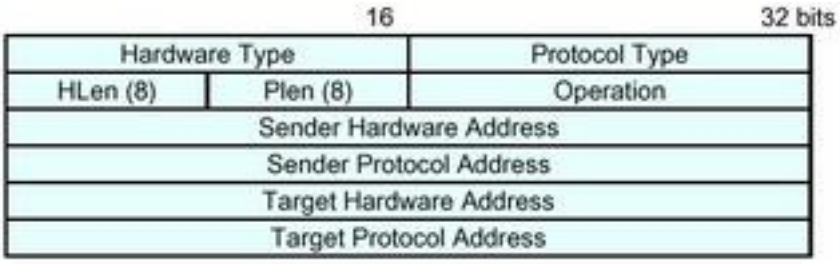

Figura 2.3: Cabeçalho ARP

2.5 IP

O protocolo IP pode ser visto como o coração da pilha TCP/IP. Os protocolos TCP, UDP, ICMP, e IGMP são transmitidos como datagramas IP. O IP fornece um serviço de entrega de datagrama sem conexão pre-estabelecida e não confiável. Sendo assim um serviço que não garante a entrega do pacote, ou seja, provê um serviço de melhor esforço possível. Quando alguma falha acontece, é usado seu simples mecanismo de manipulação de erros, que joga o datagrama fora e tenta enviar uma mensagem ICMP de volta para o emissor do pacote. Pelo fato de não manter uma conexão, cada datagrama é manipulado independentemente dos demais. Isto significa que os pacotes podem chegar fora de ordem no destinatário [STE 00b]. Caso a confiabilidade da entrega seja necessária, esta deve ser implementada por algum protocolo das camadas superiores.

Na figura 2.4 [4] pode ser obervado o cabeçalho IP. O primeiro campo, que possui 4 bits, faz a identificação da versão do protocolo. O campo seguinte, identifica o tamanho do cabeçalho em unidades de 4 bytes. Na sequência vem o campo tipo de serviço, os três primeiros bits são ignorados, os quatro na sequência sinalizam atraso mínimo, máxima vazão (throughput), máxima confiabilidade e mínimo custo monetário, respectivamente. O ultimo bit deste campo não usado e deve ser zero [STE 00b].

O campo tamanho total, informa o tamanho completo do datagrama IP em bytes. Com o uso deste campo e do campo tamanho do cabeçalho, pode-se saber exatamente onde se inicia o campo de dados dentro do datagrama IP.

O campo *identification*, identifica unicamente os datagramas enviados por determinada estação. Normalmente a cada envio este número é incrementado em um, assim como os campos a seguir *flags* e *fragmentation offset* é usado durante a fragmentação e remontagem de datagramas. O campo TTL (time-to-live) especifica o número máximo de roteadores pelos quais

[4] Fonte: http://www.wickiup.com/wickiup/net/

o datagrama pode passar. Ou seja, é limitado o tempo de vida do pacote. A cada roteador esse número é decrementado. Quando este valor atinge zero, o pacote deve ser descartado e uma mensagem ICMP enviado ao emissor do datagrama. Isso previne que o pacote fique preso em um *looping* entre roteadores para sempre.

O campo protocolo identifica o tipo de dado que está encapsulado no datagrama IP. É usado para demultiplexar os pacotes recebidos e identificar para qual protocolo o IP deve repassar o pacote. O campo a seguir armazena o valor do *checksum* que é realizado somente no cabeçalho do datagrama.

A seguir vem os campos com os endereços da fonte e destino da mensagem respectivamete. São estes endereços de 32 bits. Então, podem aparecer os campos com as opções do protocolo IP. Estas podem ser referentes a segurança, diversas opções de roteamento e controle de tempo. Mas já estão em desuso e a minoria das implementações provêem suporte.

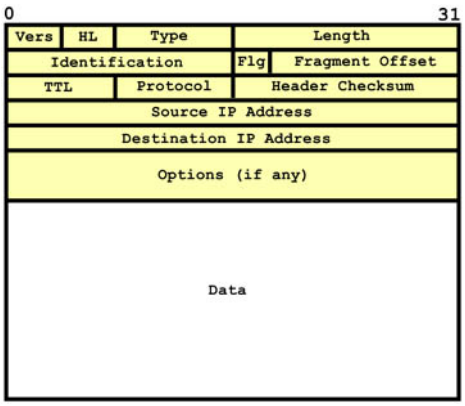

Figura 2.4: Cabeçalho IP

2.5.1 Roteamento IP

O roteamento de pacotes IP é teoricamente simples. Caso o destino esteja conctado diretamente ao emissor ou em uma rede local compartilhada (por exemplo ethernet ou token ring), o datagrama IP é enviado diretamente para o destino. Caso contrário, o emissor envia o pacote para um roteador padrão, e então é de reponsabilidade do roteador entregar a mensagem ao destinatário [STE 00b].

De modo geral, o serviço de IP pode receber datagramas do TCP, UDP, ICMP ou IGMP (ou seja, pacotes gerados localmente) para serem enviados, ou ainda datagramas que foram recebidos de uma interface de rede. A camada de IP deve possuir uma tabela de roteamento

Classe	Variação
A	0.0.0.0 até 127.255.255.255
B	128.0.0.0 to 191.255.255.255
C	192.0.0.0 to 223.255.255.255
D	224.0.0.0 to 239.255.255.255
E	240.0.0.0 to 247.255.255.255

Tabela 2.1: Classes dos endereços IP

que é checada cada vez que existe a necessidade de um datagrama ser enviado. Quando recebido pela interface de rede, a camada IP confere se o endereço de destino do pacote é um de seus próprios endereços de IP ou um endereço de *broadcast*. Caso seja, o datagrama é entregue à camada responsável por este, caso contrário, se a camada IP estiver configurada para atuar como roteador, o pacote é passado adiante (*forwarded*), senão, o datagrama é descatado [STE 00b].

O roteamento IP não sabe a rota completa para nehum destino (com excessão para as estações diretamente conectadas). O que o IP faz é fornecer o endereço para o próximo roteador que o datagrama deve ser enviado. Assume-se que o próximo roteador esteja "mais perto"do destino do que o emissor.

2.5.2 Endereçamento IP

Toda estação configurada para receber datagramas IP deve possuir um enderço IP. Os endereços IP são de 32 bits e possuem uma estrutura particular. A tabela 2.1 faz a classificação das classes de endereços IP.

Exitem três tipos de endereços IP: *unicast* (destinado para uma única estação), *broadcast* (detinada para todas as estações de uma rede), e *multicast* (detinada para um conjunto de estações que pertencem a um grupo multicast).

2.6 ICMP

O protocolo ICMP é considerado parte da camada IP. Comunica mensagens de erro e condições que requerem atenção. Mensagens ICMP são produzidas pela camada IP ou por um protocolo de nível mais alto. Algumas mensagens ICMP causam erro que devem ser

informados aos processos de usuário [STE 00b].

As mensagens ICMP são definidas por um tipo e por um código. Existem um grande número de tipos e código, sendo assim a solução para a detecção de erros no envio de pacotes IP.

2.7 UDP

O protocolo UDP é um simple protocolo de transporte orientado a datagramas. Cada operação de envio por um processo, produz exatamente um datagrama UDP, o causa a geração de também um datagrama IP a ser enviado. Assim, é diferente dos protocolos de transporte orientados a *stream* [STE 00b].

O UDP não é confiável. Não existe garatia que os datagramas alcançarão o destino final. Caso a quantidade de informação sendo enviada seja maior que o MTU (maximum transport unit) o datagrama IP será fragmentado.

O cabeçalho UDP pode ser visto na figura 2.5 [5]. O campo *source port* identifica o processo que enviou a mensagem, e o campo *destination port* identifica o processo que deve receber a mensagem. O campo *length* informa o tamanho do datagrama UDP (cabeçalho e dados) em bytes. O valor mínimo deste campo é oito, ou seja, é possível enviar um datagrama UDP com zero bytes de dados. O próximo campo mantém o valor do *checksum* que é calculado nos campos do cabeçalho e de dados.

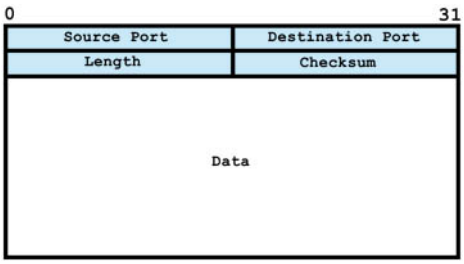

Figura 2.5: Cabeçalho UDP

[5]Fonte: http://www.wickiup.com/wickiup/net/

Capítulo 3

Application-Oriented System Design

Este capítulo tem o objetivo de apresentar a técnica Application-Oriented System Design [dMF 01]. Após uma visão geral da técnica serão apresentadas suas particularidades.

3.1 Introdução

No título deste trabalho, o termo *adaptável à aplicação*, caracteriza um sistema fortemente comprometido com a aplicação que fará seu uso. Um sistema customizável é dito *adaptável à aplicação* se este puder ser customizado de modo a atender os requisitos de uma aplicação em particular, e não somente atender a padrões ou aspectos de hardware.

Um sistema modelado como um arranjo de componentes de software pode alcançar um alto grau de customização. A configuração é então realizada através da seleção, adaptação e combinação dos componentes de software. Metodologias como: *family-based design*, *collaboration-based design*, *subject-oriented programming*, *aspect-oriented programming* e uma combinação multi-paradigma destas, tem o potencial de guiar o projeto de sistemas customizáveis [dMF 01].

Baseado nas técnicas de engenharia de software citadas acima, nasceu o *Application-Oriented System Design*. Esta técnica multi-paradigma guia a decomposição de domínio em direção a *famílias de abstrações independentes de cenário* que podem ser re-usadas para a construção de diversos sistemas. Dependências do ambiente de execução identificadas durante a decomposição do domínio são modeladas separadamente como *aspectos de cenário*. Estes podem ser aplicados transparentemente as abstrações do sistema com a ajuda de *adaptadores de cenário*. Funcionalidades opcionais são agrupadas em estrutudas chamadas *configurable features* [dMF 01].

3.2 Decomposição de domínio

De um modo geral, a decomposição de domínio descrita em AOSD[dMF 01] é similiar a decomposição da metodologia de construção de sistemas orientados a objetos. Entretanto, existem algumas diferenças de grande importância que precisam ser consideradas. Ao invés de agrupar objetos com funcionalidades similares em uma hierarquia de classes aplicando análise de variabilidade e identificar como uma entidade especializa a outra, é aplicada uma análise de variabilidade seguindo a técnica *family-based design*. Assim, são geradas abstrações idependentes umas das outras, eliminado uma das restrições da decomposição orientada a objetos onde especializações (subclasse) de uma abstração só podem ser usadas na presença da versão mais genérica (superclasse). Deste modo, melhora-se a noção de *adaptável à aplicação*. Isto não significa que possa existir uma hieraquia de especializações e sim que esse tipo de hierarquia não é uma regra obrigatória [dMF 01].

Outra diferença que precisa ser considerada se trata das dependências do *ambiente de execução*. A análise de variabilidade executada na decomposição de domínio orientada a objetos não enfatiza a diferença das variacões que pertencem a abstração, das variações que pertencem ao *ambiente de execução* sendo considerado. Abstrações que imcorporam dependências do *ambiente de execução* geralmente possuem uma chance mais reduzida de serem utilizadas em novos cenários (ambientes de execução). Essa dependência pode ser reduzida fazendo-se uso das técnicas de *aspect-oriented programming*, assim, durante a decomposição do domínio, é realizada a separação dos aspectos. Deste modo, variações no domínio podem resultar em novos membros de famílias ou ainda *aspectos de cenário* [dMF 01].

A construção de *famílias de abstrações independentes de cenário* e a identificação de *aspectos de cenário* são as atividades principais da decomposição do domínio. A estratégia inicial para adicionar funcionalidade a uma família, é a adição de novos membros. Mas, também pode ser desejada a extensão de uma funcionalidade de todos os membros da família, estes casos são tratados modelando-se essa extensão como uma *configurable feature*.

3.3 Famílias de abstrações independentes de cenário

Durante a decomposição de domínio, *abstrações idependentes de cenário* são identificadas e agrupadas em famílias. O proximo passo é o refinamento das abstrações e a definição dos componentes que irão implementá-las.

O primeiro refinamento é o ajuste de magnitude entre componentes e abstrações. A idéia básica é manter 1 para 1 e preservar conformidade com o domínio. Entretando algumas entidades do domínio podem ser muito complexas para serem modeladas em um único componente, ou muito primitivas para serem expostas as aplicações. Componentes muito complexos não são apropriados à noção de *adaptável a aplicação* enquanto componentes muito primitivos aumentam a complexidade da configuração do sistema. A idéia proposta por AOSD é que os componentes sejam: "O mais simples possível, mas ainda prontos para a aplicação"[dMF 01]. Assim evitando o aparecimento de abstrações intermediárias, as quais precisam ser compostas umas com as outras para estarem prontas a serem utilizadas pela aplicação. Facilitando assim a configuração do sistema e a identificação das funcionalidadaes exigidas pela aplicação.

Depois do balanço entre componentes e abstrações estar realizado, cada família de abstrações pode ser refinada considerando detalhes quanto a estrutura e o comportamento de seus membros visando ratificar sua interface. Idealmente, as abstrações devem ser entregues como *tipos abstratos de dados*. Assim, cada abstração seria exportada através de uma interface que claramente identifica suas responsabilidades. Estas interfaces são de fundamental importância para o projeto, pois estas constituem o ponto principal de interação entre o sistema e os programadores das aplicações. [dMF 01].

Durante este refinamento das famílias, também é importante a indentificação do relacionamento entre as famílias. Famílias normalmente confiam em abstrações de outras famílias afim de entregar seus serviços corretamente [dMF 01].

3.4 Configurable features

Uma *configurable feature* encapsula estruturas de dados e algoritmos que são usados para implementar uma funcionalidade opcional de uma família. Mas cada membro da família é responsável por sua implementação. As abstrações podem re-usar, extender ou sobrepor as funcionalidades de uma *configurable feature*, mas é necessário que as abstrações se comportem de acordo com a *configurable feature* quando a mesma está habilitada [dMF 01]. E caso não esteja, esta não deve impor *overhead* algum ao sistema. O sistema gerado com a *configurable feature* desabilitada deve operar como se esta funcionalidade não existisse.

3.5 Aspectos

Durante a decomposição de domínio de AOSD [dMF 01], abstrações são especificadas evitando dependências de possívies *ambientes de execução*. Aspectos de *ambientes de execução*, são capturados em construções separadas. Deste modo, abstrações dependentes do ambiente de execução são evitadas, enquanto o grau de re-usabilidade das abstrações é aumentado. Entretanto, as construções usadas para capturar os aspectos dependentes de cenário (ambiente de execução) devem ser modeladas de modo a tornar possível forçar as abstrações as condições ditadas por um possível cenário de execução, sem a necessidade de modificá-las explicitamente [dMF 01].

Em AOSD as abstrações são forçadas as condições de determinado cenário, fazendo-se uso de *scenario adapters*. Estes, podem ser vistos como um tipo de agente que engloba uma *abstração indenpendente de cenário* e faz a mediação de sua interação com um cliente dependente de cenário. Assim, abstrações podem adquirir as propriedades necessárias para executar em um dado cenário, sem a necessidade de ser modificado [dMF 01].

Capítulo 4

Implementações para Sistemas Embutidos

Este capítulo apresentará algumas implementações da pilha TCP/IP para sistemas embutidos. Também serão apresentados os pontos fortes e fracos destas implementações.

4.1 Lightweight TCP/IP Stack

Atualmente mantida por um grupo de desenvolvedores na internet, esta implementação de código fonte aberto visa a redução do consumo de memória e do código objeto. Existem portes desta pilha para diversos sistemas operacionais e um grande número de usuários e projetos que fazem uso desta implementação, dentre os quais podemos citar os projetos PURE [BEU 99] e ARENA [HTT 04].

Nesta pilha existe suporte aos seguintes protocolos: ARP, IP, ICMP, UDP, e TCP. Também inclui suporte a múltiplas interfaces de rede. Possui um sistema comlexo de gerenciamento de buffers de memória, possuindo três tipos distintos de buffers.

Implementa uma camada de emulação de sistema operacional, com o intuito de facilitar o porte desta implementação para diferentes sistemas operacionais e também cria a possibilidade do uso desta mesmo sem o suporte de um sistema operacional. Com relação a sua modularização, é considerada modular o suficiente para ser facilmente extendida com a adição de novos protocolos.

O sucesso desta implementação se dá principalmente pois esta se compromete a ser uma pilha TCP/IP completa e para seu uso é necessário aproximadamente 15 kilobytes de memória disponível. Além de possuir uma boa documentação, um grupo ativo de desenvolvedores e usuários, e mecanismos que visam facilitar o porte para diferentes arquiteturas e sistemas

operacionais.

4.2 uIP TCP/IP Stack

Mantida pelo mesmo grupo que desenvolve a Lightweight TCP/IP Stack, esta pilha visa ser a menor pilha TCP/IP completa do mundo, é destinada principalmente a sistemas microcontrolados. Atualmente existem portes para diferentes arquiteturas, dentre as quais incluem-se: AVR, H8S/300, 8051, Z80, ARM e Lego Mindstorms.

É projetada para oferecer somente o conjunto mínimo de funcionalidades requeridas por uma pilha TCP/IP completa. Não possui suporte a múltiplas interfaces de rede e implementa os seguintes protocolos: ARP, IP, ICMP e TCP.

O sistema de bufferização é simples, não faz uso de alocação dinâmica. Um buffer único e global é usado(tanto para pacotes de entrada quanto saída), e possui tamanho suficiente para manter um pacote do tamanho máximo suportado.

Assim como a lwIP, possui mecanismos que visam facilitar o porte para outras arquiteturas, mas possui um nível muito baixo de modularização. Existem rotinas que agrupam funcianalidades de diferentes protocolos e fazem uso extensivo da estrutura de controle *goto*. O que impede que esta possa ser extendida com a adição de novos protocolos.

Entretanto, essa pilha exige pouco uso de memória. Este uso é definido por alguns parametros de configuração como número de conexões TCP alocadas, número de entradas na tabela ARP, e qual tamaho máximo dos pacotes.

Uma uIP configurada para uma porta de escuta TCP, dez slots de conexão, dez entradas na tabela ARP e com um tamanho máximo de pacote de 400 bytes, fará uso de aproximadamente 980 bytes de memória RAM e o tamanho do código objeto será em torno de 6 kilobytes. Assim, tornando-a apropriada para sistemas com recursos bastante limitados.

4.3 Ethernut TCP/IP Stack

Esta pilha pertence a um projeto de hardware e software de código aberto para a construção de dispositivos ethernet embutidos. A implementação da pilha é usada pelo Nut/OS , que é o sistema operacional para a solução de hardware deste mesmo projeto.

Esta também pode ser considarada uma pilha TCP/IP completa pois implementa os protocolos ARP, DHCP, IP, ICMP, UDP e TCP. Possui um sistema de bufferização característico,

e pode fazer uso de alocação estática e dinâmica.

Na configuração da Nut/Net para operar como um servidor HTTP simples, serão necessário aproximadamente 15 kilobytes para o código objeto e 200 bytes para a alocação dinâmica.

Dentre as implementações avaliadas, essa é a mais dependente do sistema operacional, mas possui um nível de modularização suficiente para a adição de novos protocolos.

4.4 Discussão

Todas as implementações analisadas são mantidas por um grupo organizado e ativo de desenvolvedores. Observando-se o histórico destas implementações, nota-se o crescimento do número de usuários e os constantes esforços para a correcão de *bugs* e adição de novas funcionalidades.

Como todas essas implementações têm propósitos similares, é óbvio que, quando comparados os códigos fonte destas, são encontradas várias rotinas muito similares. Além de todas estarem modularizadas com base no paradigma estruturado e escritas em linguagem C.

Desta maneira, nota-se um deperdício de esforço nestas implementações, pois todas compartilham um grande número de rotinas exatamente iguais ou muito similares. Em todos estes projetos, não notam-se decisões que visam o reuso da implementação. Tal fato dificulta um possível aproveitamento das funcionalidades providas por cada uma destas pilhas em outras implementações.

Em uma situação hipotética, se no projeto de alguma destas implementações fosse usada alguma técnica de engenharia de software visando aumentar a reusabilidade da implementação em questão, os outros projetos poderiam fazer um reuso desta em seus projetos específicos, fato que muito provavelmente influenciaria os custos do projeto.

Outro fator bastante similar nestas implementações é a configurabilidade. Em todas, isto se dá majoritariamente por meio de compilação condicional. Este recurso quando amplamente utilizado permite atingir um alto grau de configurabilidade, mas inevitavelmente eleva complexidade do código fonte em níveis muito maiores. Por isso, é usado apenas para abilitar (ou desabilitar) funcionalidades cosideravelmente grandes. Não sendo recomendável como única fonte para garantir um bom nível de configurabilidade.

Sem dúvida, a solução para se atingir bons níveis reusuabilidade e configurabilidade se dá por meio da engenharia de software. As técnicas de AOSD[dMF 01], visam a construção de componentes de software que agrupem bons níveis de configurabilidade, reusabilidade,

manutenabilidade e consequentemente adaptabilidade.

Capítulo 5

Pilha TCP/IP Adaptável à Aplicação

Este capítulo tem por objetivo apresentar os resultados da aplicação das técnicas de AOSD[dMF 01] no domínio protocolos de comunicação da pilha TCP/IP. Após a apresentação da metodologia utilizada, serão apresentadas as *famílias de abstrações independentes de cenário* que resultaram deste trabalho.

5.1 Metodologia

Conforme descrito no capítulo 3, o estudo do domínio que será modelado pelas técnicas de AOSD[dMF 01] é de extrema importância. É necessario um estudo com um alto nível de detalhamento. Como a técnica é muito expressiva, visando proporcionar a geração de componentes de software de qualidade (reusáveis, configuráveis, adaptáveis, etc), toda e qualquer característica do domínio pode causar um impacto considerável nos componentes de software gerados. Assim, pode se dizer que quanto maior conhecimento do domínio for obtido, com certeza maior será a qualidade da modelagem obtida e consequentemente do software resultante.

Deste modo, a primeira tarefa para a aplicação das técnicas de AOSD[dMF 01] é a definição do domínio a ser estudado. Isto implica na identificação do que pertence a este domínio e o que não faz parte. Todas as decisões tomadas durente o decorrer deste trabalho levaram em consideração que o mesmo visa contruir um conjunto de componentes de software que forneçam as funcionalidades da pilha TCP/IP para *sistemas embutidos*.

O domínio dos protocolos de comunicação da pilha TCP/IP é extremamente extenso. O número de protocolos é muito grande e a complexidade dos mesmos é bastante considerável. O sistema que é usado para a definição, manutenção e evolução dos protocolos é bastante

eficiente. É feita por meio da publicação de RFCs (Request for Comments), e estas publicações não são feitas por entidades regulamentadoras, e sim é aberta a todos que fazem uso dos protocolos. Portanto, uma pesquisa na base de RFCs publicados, retorna não somente a definição dos protocolos mas também melhorias, otimizações e diversos tipos de recomendações para seu uso. Tal fato, mantém a evolução dos protocolos e também propicia o aumento de sua qualidade. Mas quando uma implementação é proposta, decisões sobre qual das otimizações ou melhorias existentes serão usadas, precisam ser tomadas.

Como este trabalho propõe a contrução de uma pilha TCP/IP com um alto grau de flexibilidade, todas as decisões foram tomadas de modo a não restringir de maneira alguma a evolução da mesma. Pois com a constante evolução dos protocolos, extensões aos componentes de software gerados serão necessárias.

Assim, durante a modelagem das famílias optou-se por uma postura conservadora. Foram modeladas as funcionalidades base dos protocolos e que fossem pertinentes aos *sistemas embutidos*. De modo que qualquer otimização ou extensão que possa ser feita futuramente ocorra de modo fácil e simples. Isto é possível devido às características das técnicas de AOSD, podendo ocorrer pela adição de um novo membro em alguma família, a adição de uma nova *configurable feature* ou a introdução de um novo cenário com a adição de um aspecto.

Afim de obter um conhecimento pleno do domíno, renomados livros sobre a pilha TCP/IP foram consultados [STE 00b, STE 00a, DEC 98, BEN 02]. Também foram analisadas diversas implementações, desde implementações para sistemas de propósito geral como o Linux até implementações específicas para os sistemas embutidos. Um grande número de RFCs também fizeram parte do estudo.

Após o levantamente de toda informação referente aos protocolos de interesse, as técnicas de AOSD[dMF 01] puderam ser aplicadas. Primeiramente foi realizada a decomposição do domínio de cada protocolo. Em seguida a modelagem das famílias foi realizada, neste ponto são definidos dentre as funcionalidades dos protocolos, quais partes serão *membros* da família, *configurable features* ou *aspectos de cenário*. Após todas essas decisões tomadas, diagramas de classe de cada família foram modelados. Então, a implementação pôde transcorrer diretamente baseda nos diagramas de classe elaborados.

As próximas seções deste capítulo apresentarão as famílias desenvolvidas no decorrer deste trabalho, serão apresentadas as funcionalidades de cada membro da família, assim como suas *configurable features* e aspectos de cenário.

5.2 Família Ethernet

Esta família tem por finalidade implementar as funcionalidades do padrão *ethernet*. Ou seja, encapsular os dados a serem enviados em um quadro, fazer uso da resolução de endereços e repassar o quadro para que este seja enviado pela interface de rede. Ou ainda, receber o quadro que foi recebido pela interface de rede e caso seja necessário, entregá-lo ao protocolo pertinente.

Como a maioria dos protocolos, suas funcionalidades podem ser divididas em funcionalidades de envio e de recepção. Por isso, na família que é descrita na figura 5.1, precebe-se uma divisão, existem membros responsáveis por transmissão e pela recepção.

As responsabilidades mais importantes de cada membro podem ser descritas da seguinte maneira:

EthOut: É responsavel pelo encapsulamento dos dados que serão enviados e a comunicação com o *driver* que enviará o quadro para a placa de rede.

EthRes: Além das fucionalidades do membro EthOut, faz a tradução do endereço de protocolo em endereço físico antes de entregar o quadro ao *driver* da interface de rede.

EthIn: Responsável por retirar os dados recebidos do quadro, e entregar ao único nível superior possível.

EthDemux: Além de retirar os dados recebidos do quadro, analisa o tipo do dado que está encapsulado e faz a entrega ao protocolo apropriado.

As *configurable features*, conforme descritas pelo AOSD[dMF 01], englobam funcionalidades que são opcionais e que afetam caso estejam ativadas afetam todos os membros da família.

As principais funcionalidades das *configurable features* desta família são:

Broadcast: Esta opção, habilita ou desabilita a identificação de mensagens do tipo *broadcast*.

Multicast: Caso esta funcionalidade esteja habilidade a identificação de endereços de grupo (*multicast*) estará ativa.

Reliability: Esta *configurable feature* define que o pacote não é confiável, podendo estar corrompido, então diversos tipos de checagem são feitos para a sua validação. Caso o quadro não seja validado, o mesmo é descartado.

Os aspectos até então identificados, tem a seguinte funcionalidade:

Statistics: Com a aplicação deste aspecto, todo e qualquer tipo de estatística pode ser levantado. Como por exemplo: número de bytes enviados/recebidos, quantidades de quadros descartados, etc...

Buffering: Neste cenário, antes dos membros da família receberem o quadro pra sua manipulação, o mesmo é adicionado a um buffer, que tem por objetivo armazenar os quadros até que os membros estejam prontos para a manipulação de um novo quadro.

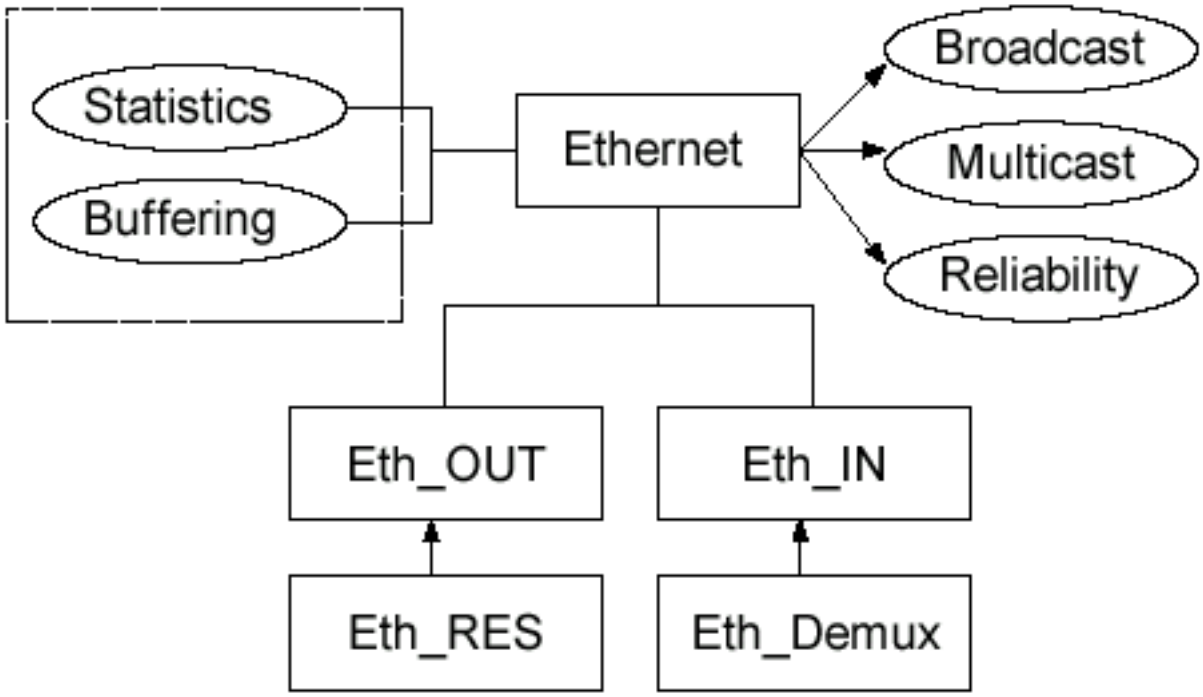

Figura 5.1: Família Ethernet

5.3 Família ARP

Esta família implementa a funcionalidade da tradução de endereços de protocolo em endereços de hardware. Durante sua modelagemm três membros foram identificados, sua funcionalidades são descritar a seguir:

Static: Este membro possui uma tabela, que é preenchida em tempo de compilação e mantém todo o mapeamente entre os endereços. Caso uma tradução de endereço seja requisitada, a tabela é consultada e o endereço traduzido é retornado.

Dynamic: Implementa as funcionalidades do protocolo ARP. Possui a capacidade de preparar requisições e repostas de traduções de endereços. Mas não possui uma estrutura para armazenar os endereços já resolvidos.

Cached: Possui as funcionalidades do membro Dynamic, e possui uma estrutura para que os endereços já resolvidos possam ser mantidos. Assim, requisições já feitas serão consultadas na tabela local e uma nova requisição ARP não será necessária.

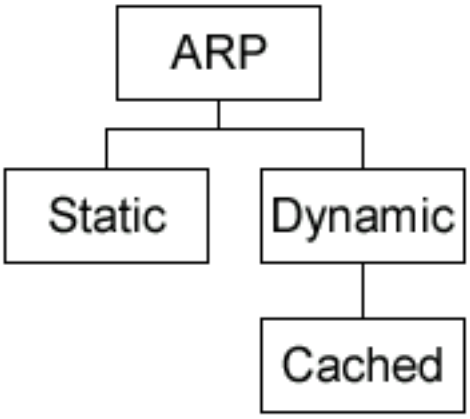

Figura 5.2: Família Address Resolution

5.4 Família IP

Esta família é responsável por implementar as funcionalidade do protocolo IP. Pelos mesmos motivos apresentados nas outras famílias, pode se notar que existe uma hierarquia para os membros de envio e uma para os de recepção.

Deviso sua complexidade foi necessária a definição de várias *configurable features* e membros para esta família, como pode ser visto na figura 5.3.

As funcionalidades dos membros são:

IpOut: Este membro é responsável pelas funcionalidades básicas de envio de datagramas IP. Entre elas: construção do cabeçalho, definição do endereço de destino, etc.

IpFragmented: Além das funcionalidades do membro IpOut, este membro possui a funcionalidade necessária para a fragmentação de datagramas. Este membro se faz necessário quando a aplicação pode gerar o envio de dados maior que o MTU (maximum transport unit).

IpRouted: Este membro implementa todas as funcionalidades do membro IPOut, uma tabela de roteamento e mecanismos para a decisão de escolha de rotas.

IpIn: Implementa as funções básicas de recebimento de datagramas IP, como por exemplo: análise se o datagrama chegou na estação certa, validação do pacote e entrega ao protocolo superior apropriado.

IpReassemble: Além das funcionalidades do membro IPIn, possui mecanismos para a remontagem de pacotes que sofreram fragmentação.

A expressividade dessa família aumenta bastante com a habilitação das *configurable features*, são elas:

Broadcast: Habilita a manipulação de endereços broadcast. Isto se reflete tanto no envio quanto no recebimento.

Multicast: Habilita a manipulação de endereços multicast. Esta funcionalidade interfere nas rotinas de definição do endereço de destino e validação do datagrama recebido.

Forwarding: Com a habilitação desta funcionalidade, caso a estação receba um pacote que não pertence a esta, o pacote não é simplesmente descartado, e sim enviado ao próximo roteador.

CtrlMsg: Caso esta funcionalidade esteja habilitada, em qualquer caso de erro uma mensagem ICMP será gerado. E caso uma mensagem ICMP seja recebida, será corretamente analisado e caso necessário uma resposta será gerada.

Checking: Esta funcionalidade deve ser habilitada caso os datagramas recebidos não sejam confiáveis. Assim, diversos tipos de verificações serão feitas afim de validar o datagrama antes de prosseguir o processo de rebimento do pacote.

Os apectos identificados para esta família foram os mesmos da família Ethernet: um para controle de *buffer* e outro para a captura de estatísticas.

5.5 Família ICMP

Esta família implementa funções para a manipulação de mensagens ICMP. Cada membro da família é responsável pela geração de novas mensagens ICMP a serem enviadas e a análise de mensagens ICMP recebidas.

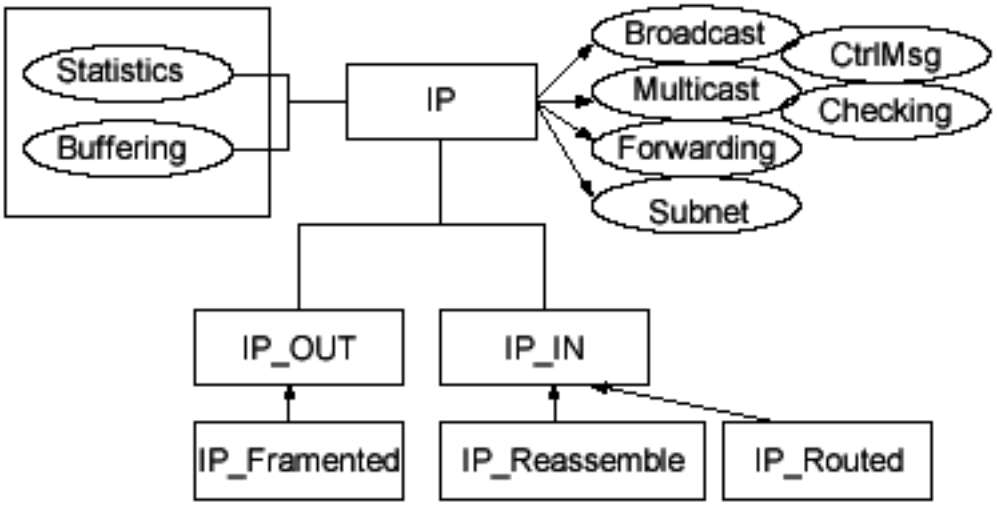

Figura 5.3: Família Internet Protocol

Devido ao fato de existir um grande número de tipos de mensagens, o diagrama desta família apresentado na figura 5.4 apresenta somente dois membros, afim de simplificar a visualização.

Estes membros têm a seguinte funcionalidade:

Echo: Implementa a funcionalidade para o recebimento e envio de mensagens do tipo *Echo*. Este tipo de mensagem que permite a implementação do utilitário ping.

DstUn: É responsável por lidar com mensagens ICMP de *destination unreachable*.

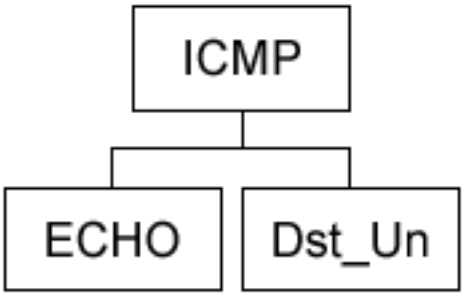

Figura 5.4: Família ICMP

5.6 Família UDP

O protocolo UDP é realmente simples. Isto se reflete na família UDP que implementa as funcionalidades deste protocolo, como pode ser visto na figura 5.5.

Assim como as famílias já apresentados, exitem membros para a manipulação de envio e da recepção de datagramas, são estes:

UDPOut: Possui as funcionalidades para a criação e o preenchimento do cabeçalho UDP.

UDPIn: A funcionalidade deste membro é a remoção do cabeçalho e a entrega dos dados encapsulados para a aplicação em espera.

UDPDemux: Possui toda a funcionalidade do membro UDPIn e a capacidade de demultiplexar o pacote recebido, fazendo a escolha dentre as aplicações esperando pacotes, qual deve receber.

Os aspectos *buffering* e *statistics* também podem ser aplicados a esta família, com o mesmo objetivo da aplicação nas demais famílias apresentadas.

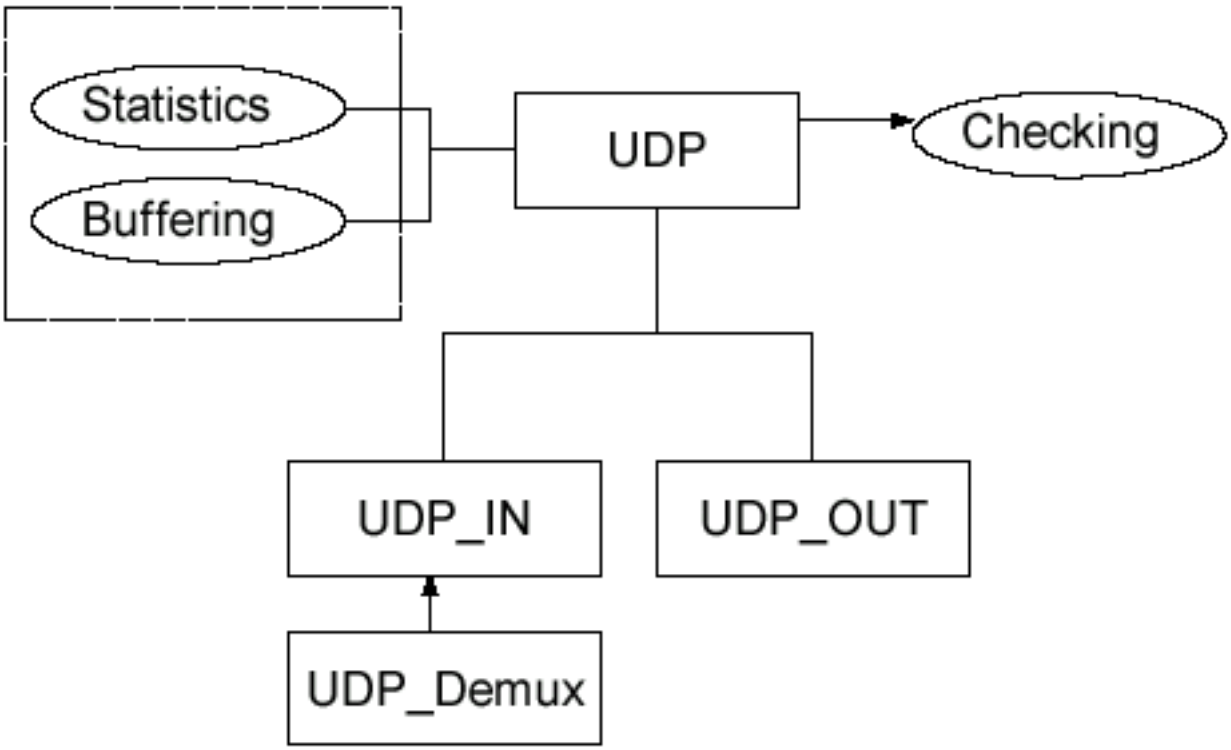

Figura 5.5: Família UDP

5.7 Família BufferFactory

O controle dos *buffers* que armazenam os pacotes que estão sendo preparados para o envio, assim como os pacotes que estão sendo processados pelas rotinas de recebimento, é uma das tarefas mais complexas de uma pilha de protocolos.

Para que esta manipulação seja eficiente, é necessário se fazer uso de recursos de sistema operacional em uso. Mas como as abstrações desenvolvidas no curso deste trabalho visam ser *independentes de cenário*, não poderiam aparecer dependências de possíveis cenários de execução.

Portanto essa família foi modelada com o intuito de ser capaz de criar *buffers* para o armazenamento de pacotes sendo manipulados, da forma mais otimizado possível, e sem implicar em depedência alguma.

Para isso, são implementados membros para cada arquitetura específica, durante o decorrer deste trabalho, foram utilizadas implementações para os sistemas operacionais Linux e Epos, como pode ser visto na figura 5.6.

Mas esta família não define somente arquiteturas, também podem ser criados membros para a mesma arquitetura, e com políticas diferentes para a criação de *buffers*.

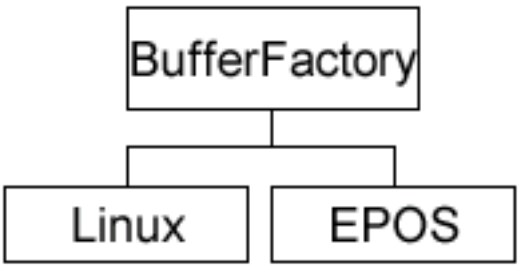

Figura 5.6: Família Memory Buffer

5.8 Família Device

Apesar de não pertencer à pilha TCP/IP, os membros dessa família são de extrema importância. Pois são estes que recebem os dados da família Ethernet e comunicam-se com a interface de rede, passado os pacotes para que possam ser transmitidos no meio de comunicação.

Os membros desta família podem ser vistos como *mediadores de hardware* ou também device drivers. Pelo fato fazerem acesso ao hardware, também são dependentes do sistema

operacional em uso. Assim, como no caso da família *BufferFactory*, são necessários membros para cada arquitetura em questão.

Foram desenvolvidos membros para o Linux e para o Epos, para a manipulação da interface de rede 3Com 905C-TX [3Co 98]. Esta é uma interface de rede bastante moderna, e que possui bastante aceitação no mercado atual.

Os testes das outra famílias foram realizados fazendo-se uso dos membros desta família para entrega dos quadros no meio de tranmissão. Assim, permitiu-se a interação da pilha de TCP/IP proposta neste trabalho, com pilha TCP/IP de sistemas de propósito geral como o Linux.

Capítulo 6

Aspectos de Implementação

6.1 Implementação

Após a modelagem, foram implementados membros de todas as famílias apresentadas no capítulo anterior. Como a técnica usada se compromete a entregar componentes de software que sejam configurados, e tenham suas funcionalidades selecionadas de modo a não gerar overhead algum caso alguma funcionalidade não seja requerida, durante a implementação é necessário o uso de técnicas que permitam tal fim.

De um modo geral a implementação ocorreu usando-se a linguagem de programação C++. Mas alguns pontos deve ser levantados. Quando uma configurable feature não foi requisitada, a mesma não pode gerar um único byte que seja no código objeto. É como se esta não existisse no componente em questão. Para conseguir isto, foi necessária a utilização dos recursos de *static metaprogramming* da linguagem de programação C++.

O trecho de código abaixo apresenta um exemplo de como são implementadas as configurable features. Com essa implementação caso esssa funcionalidade não esteja habilitada no arquivo de configuração, o compilador não gerará código para a mesma.

```
template< int configs, typename UIP = UpperIP >
class IPInCF {
  public:
  static inline bool checks(IPHeader& iph, EthernetNic* n)
  {
    if (!IPInCF<configs & Traits<IP>::VERSION>::checks(iph, n)){
```

```
      return false;
    }

    if (!IPInCF<configs & Traits<IP>::HSIZE>::checks(iph, n)){
      return false;
    }

    if (!IPInCF<configs & Traits<IP>::CHKSUM>::checks(iph, n)){
      return false;
    }
  }

...
```

Um outro ponto interessante da implementação, foi a aplicação dos *aspectos*. Os aspectos foram desenvolvidos idependentes de como seriam aplicados aos componentes de software. A aplicação ocorreu de duas maneiras. Utilizando-se a técnica de *scenario adapters* [dMF 01]. E utilizando recursos da linguagem AspectC++ [AG 00].

Como os componentes de software em questão são configuráveis, foi necessária a definição de um mecanismo para essa configuração. Neste trabalho isto se deu com a criação de um aquivo *header* que informa quais *configurable features* e aspectos devem estar habilitados. Abaixo segue um trecho deste arquivo:

```
class IP;
template <> struct Traits<IP>: public Traits<void>
{
        static const int VERSION = 1;
        static const int HSIZE = 2;
        static const int CHECKSUM = 4;
        static const int NOADDRESS = 8;
        static const int IP_BROADCAST = 16;
```

```
        static const int ICMP = 32;
        static const int UDP = 64;

        static const int CF =  VERSION + HSIZE + CHECKSUM
         + NOADDRESS + IP_BROADCAST;
};
```

6.2 Avaliação

Como o principal intuito deste trabalho é a criação de componentes de softare que possam ser costurados a aplicação, e forneçam somente a funcionalidade requisita, foram preparadas aplicações com diferentes requisitos e então foi feita uma análise do tamanho do código objeto gerado. Abaixo segue a descrição das aplicações implementados com intuito de testar os componentes desenvolvidos e sua configurabilidade:

Aplicação 1: Esta aplicação visava somente o recebimento de dados. Sendo conectada diretamente ao emissor, e com um hardware de rede de alta qualidade, não sendo necessário então a aplicação de nenhuma configurable feature ou aspecto. Sendo então utilizados somente os membros: *EthIn, IPIn e UDPIn.* Com essa configuração o código objeto gerado foi de **3420 bytes**.

Aplicação 2: Esta segunda aplicação, tinha por objetivo fazer uso das funcionalidades básicas de envio e recepção. Assim como na anterior, as estações estarão conectadas diretamente uma a outra. Para se obter essas funcionalidades, não foi necessária a adiçao de nenhuma configurable feature ou aspecto. Os seguintes membros foram utilizados: *EthOut, IPOut, UDPOut, EthIn, IPIn, UDPIn, ArpS e ICMP*. Com essa configuração o código objeto gerado foi de **5632 bytes**.

Aplicação 3: Esta aplicação tinha o objetivo de implementar a funcionalidade de um roteador comum, receber os pacotes, selecionar uma rote e re-enviá-los. Para isto, as seguintes configurable features foram necessárias: *Forwarding, Multicast, Broadcast e Cheking*. Os membros usados foram os seguintes: *EthRes, EthDemux, IPOut, IPIn, ArpD e ICMP*. Com essa configuração o código objeto gerado foi de **8856 bytes**.

Aplicação 4: O objetivo desta aplicação era funcionar como uma implementação de propósito geral. Onde as funcionalidades requeridas não são conhecidas em tempo de compilação. Assim, todas as funcionalidades foram selecionadas. Com essa configuração o código objeto gerado foi de **9376 bytes**.

Todas as aplicações foram compiladas usando-se o compilador C++ da GNU versão 3.2.2. Com geração de código para a arquitetura IA32 e usando o *flag* de otimização -O2.

Capítulo 7

Conclusão

Todos os objetivos propostos por este trabalho foram satisfatoriamente alcançados. As técnicas de AOSD[dMF 01] foram aplicadas ao domíno TCP/IP, os componentes de software resultantes da implementação obtiveram um alto grau de configurabilidade. Tal fato pode ser notado pela quantidade de funcionalidades que podem ser escolhidas durante a compilação dos mesmos. Além dessa configurabilidade ter sido alcançada de uma forma otimizada, pois as funcionalidades não requeridas não causam *overhead* algum ao sistema gerado.

Mas o sucesso da implementação se deu na avaliação do código gerado por aplicações com diferentes requisitos. Ficou claro que o tamanho do código objeto gerado é diretamente proporcional ao número de funcionalidades selecionadas. Pode se dizer que o o código gerado reflete exatamente as funcionalidades que são exigidas pela aplicação, nem menos nem mais.

A área de protocolos de comunicação se mostrou uma área extremamente atraente para a aplicação das técnicas de AOSD. Pois os protocolos de comunicação são normalmente muito complexos e a maioria das aplicações não faz uso de todas as funcionalidades impostas por estes.

Portanto, o conjunto de componentes de software desenvolvidos neste trabalho podem ser considerados altamente recomendáveis e apropriados para o uso em projetos de *sistemas embutidos* .

7.1 Trabalhos Futuros

Como um trabalho de desenvolvimento de software, este não encontra-se complemtamente finalizado.

A adição de novas famílias (novos protocolo), configurable features e aspectos

seria muito interessante. Também a extensão dos já existentes, poderia resultar na avaliação da manutenibilidade dos componentes.

Por se tratar de componentes de software que visam implementar funções de comunicação, a avaliação da *performance* destes componentes deve ser a continuação natural deste trabalho.

Referências Bibliográficas

[3Co 98] 3Com. **3c90x and 3c90xB NICs Technical Reference**, Agosto, 1998.

[AG 00] ANDREAS GAL, WOLFGAN SCHRöDER-PREIKSCHAT, O. S. Aspectc++ :language proposal and prototype implementation. **OOPSLA 2001 Workshop on Advanced Separation of Concerns in Object-Oriented Systems**, [S.l.], v., 2000.

[AND 92] ANDERSON, T. The Case for Application-Specific Operating Systems. In: PROCEEDINGS OF THE THIRD WORKSHOP ON WORKSTATION OPERATING SYSTEMS, 1992. **Proceedings...** Key Biscayne, U.S.A.: [s.n.], 1992. p.92–94.

[BAR 99] BARR, M. **Programming Embedded Systems in C and C++**. O'Reilly, Janeiro, 1999.

[BAR 00] BAR, M. **Linux Internals**. Osborne McGraw-Hill, 2000.

[BEN 02] BENTHAM, J. **TCP/IP Lean: Web Servers for Embedded Systems**. CMP Books, 2002.

[BEU 99] BEUCHE, D. et al. The PURE Family of Object-Oriented Operating Systems for Deeply Embedded Systems. In: PROCEEDINGS OF THE 2ND IEEE INTERNATIONAL SYMPOSIUM ON OBJECT-ORIENTED REAL-TIME DISTRIBUTED COMPUTING, 1999. **Proceedings...** St Malo, France: [s.n.], 1999.

[BOL 97] BOLOSKY, W. J. et al. Operating System Directions for the Next Millennium. In: PROCEEDINGS OF THE SIXTH WORKSHOP ON HOT TOPICS IN OPERATING SYSTEMS, 1997. **Proceedings...** Cape Cod, U.S.A.: [s.n.], 1997. p.106–110.

[DAL 68] DALEY, R. C.; DENNIS, J. B. Virtual memory, processes, and sharing in multics. **Communications of the ACM**, [S.l.], v.11, n.5, p.306–313, 1968.

[DEC 98] DOUGLAS E. COMER, D. L. S. **Internetworking with TCP/IP Vol. II: ANSI C Version: Design, Implementation, and Internals**. Prentice Hall, 1998.

[dMF 01] DE MEDEIROS FRöHLICH, A. A. **Application-Oriented Operating Systems**. Number17 in GMD Research Series. GMD - Forschungszentrum Informationstechnik GmbH, 2001.

[dMF 02] DE MEDEIROS FRöHLICH, A. A. Epos - the reference manual. UFSC, 2002. Relatório técnico.

[HTT 04] HTTP://ARENA.INTERNET2.EDU/. **Arena Project**. Site na Internet.

[IBM 01] IBM Corporation. **PowerPC 405GP Embedded Processor User's Manual**, 2001.

[Int 98] Intel Co. **Intel 440BX AGPset: 82443BX Host Bridge/Controller**, 1998.

[JN 02] JUAN NAVARRO, SITARAM IYER, P. D.; COX, A. Practical, transparent operating system support for superpages. In: PROCEEDINGS OF THE 5TH SYMPOSIUM ON OPERATING SYSTEMS DESIGN AND IMPLEMENTATION, 2002. **Proceedings...** Boston, U.S.A: ACM Press, 2002. Operating Systems Review, p.89–104.

[KOL 92] KOLDINGER, E. J.; CHASE, J. S.; EGGERS, S. J. Architectural Support for Single Address Space Operating Systems. **Operating Systems Review**, [S.l.], v.26, p.175–186, Outubro, 1992.

[LAR 01] LARMAN, C. **Applying UML and Patterns**. second. ed. Unknown, 2001.

[RUB 97] RUBINI, A. **Linux Device Drivers**. Sebastopol, U.K.: O'Reilly, 1997.

[SMA 98] SMARAGDAKIS, Y.; BATORY, D. Implementing Reusable Object-Oriented Components. In: PROCEEDINGS OF THE FIFTH INTERNATIONAL CONFERENCE ON SOFTWARE REUSE, 1998. **Proceedings...** Victoria, Canada: [s.n.], 1998.

[SPU 00] SPURGEON, C. E. **Ethernet: the definitive guide**. O'Reilly & Associates, Inc., 2000.

[STE 00a] STEVENS, W. R. **TCP / IP Ilustrated: The Implementation**. Addison Wesley, 2000.

[STE 00b] STEVENS, W. R. **TCP / IP Ilustrated: The Protocols**. Addison Wesley, 2000.

[THO 01] THOMSEN, C. et al. Chaos — an rcx os where chaos is only in the name. Aalborg University, Maio, 2001. Relatório TécnicoE2-113-f1a.

Apêndice A

Código Fonte

Apêndice B

Artigo