

ecai

Universidade Federal de Santa
Catarina
Centro Tecnológico
Curso de Engenharia de Controle e
Automação Industrial

ufsc

Um Sistema para Desenvolvimento Cooperativo de Software

*Monografia submetida à Universidade Federal de Santa Catarina
como requisito para a aprovação na disciplina:
EEL 5901: Projeto de Fim de Curso*

Eduardo José Pereira

Florianópolis, Julho de 1996

Um Sistema para Desenvolvimento Cooperativo de Software

Eduardo José Pereira

Esta monografia foi julgada no contexto da disciplina
EEL 5901: Projeto de Fim de Curso
e aprovada na sua forma final pelo
Curso de Engenharia de Controle e Automação Industrial

Banca Examinadora:

Luiz Fernando Bier Melgarejo
Orientador do Curso

Prof. Augusto Humberto Bruciapaglia
Responsável pela disciplina e Coordenador do Curso

Prof. Jean Marie Farines, Avaliador

Vladimir H. Gaidzinski, Debatedor

Robson Júnior Biff, Debatedor

AGRADECIMENTOS

Este projeto não deve ser analisado como um trabalho desenvolvido em seis meses, mas como fruto de uma pesquisa desenvolvida ao longo de um curso de graduação.

Por me ensinar os valores que uma pessoa deve cultivar em si mesma e por me dar liberdade para cultivá-los por mim mesmo, eu gostaria de agradecer os meus pais: Olímpia Gonçalves e José Nélio Pereira.

Por me dar a oportunidade de participar de sua equipe de pesquisa e me convidar para trabalhar no EDUGRAF, eu gostaria de agradecer ao Luiz Fernando Bier Melgarejo, que, mais do que um orientador, vem sendo um amigo.

Por ajudar, dia a dia, a construir e manter um ambiente divertido de pesquisa séria, eu gostaria de agradecer a todos os amigos do EDUGRAF.

Resumo

Este projeto foi implementado tendo em vista dois objetivos: o estudo de técnicas para desenvolvimento cooperativo de software e o uso dessas técnicas no desenvolvimento de protótipos para redes.

Dois protótipos foram desenvolvidos. O primeiro deles, o EduFórum, é um software cliente que roda sobre uma rede TCP/IP e implementa o acesso a uma base de dados Gopher [Anklesaria 93]. A novidade apresenta é a possibilidade que o usuário tem de poder interagir com a base de dados. O EduFórum está sendo desenvolvido e vem sendo utilizado dentro do projeto hiperNet [Melgarejo 91]. Esta sendo analisado ergonomicamente, nos proporcionando um grande aprendizado, principalmente no desenvolvimento de interfaces homem/máquina. O Outro protótipo desenvolvido, o WinSdca (Cliente e Servidor), é um software para controle e automação industrial que foi distribuído sobre uma rede TCP/IP. Foi um desenvolvimento multi-linguagem: um módulo tem a função de achar um modelo para o processo industrial e depois controlá-lo através de um PID; e outro implementa um servidor provendo serviços sobre os quais pode-se executar um cliente que poderá comanda o processo à distância.

Sumário

1. INTRODUÇÃO	7
2. APRESENTAÇÃO DO AMBIENTE DE DESENVOLVIMENTO DO PROJETO (O EDUGRAF)	8
3. PROGRAMAÇÃO ORIENTADA A OBJETOS	10
3.1. Vantagens da Orientação a Objetos	10
4. MODELAGEM DE SISTEMAS	12
4.1. OMT	12
4.2. Patterns	13
5. PROGRAMAÇÃO PARALELA	14
6. PROGRAMAÇÃO DISTRIBUÍDA	15
7. REDES	16
7.1. O modelo Cliente/Servidor	16
7.2. Protocolo	16
7.3. TCP/IP	16
7.4. Endereços e Portas	17
7.5. Vantagens do TCP/IP	17
7.6. Um Exemplo	17
8. SMALLTALK	19
9. EHS (EXCEPTION HANDLING SYSTEM)	20
10. NLS (NACIONAL LANGUAGE SUPPORT)	23
11. DLL - DYNAMIC LINK LIBRARY	24
11.1. O que é uma DLL?	24
11.2. Como funciona uma DLL?	24
11.3. Vantagens do uso de DLLs.	24

11.4. Exemplos de uso de DLLs.	25
11.4.1. As DLLs do Windows	25
11.4.2. Controle de periféricos via DLL	25
11.4.3. Bancos de dados, editores gráficos e muito mais.	26
11.5. Como chamar rotinas de DLLs.	26
12. SMALL-WINSOCK	27
12.1. Windows Sockets	27
12.2. Interfaceando Smalltalk com Windows Sockets	27
13. DESENVOLVIMENTO DE SOFTWARE COOPERATIVO	29
13.1. ENVY/MANAGER	29
13.1.1. Arquitetura do ENVY	29
13.1.2. Componentes de administração	30
13.1.3. Administração de alterações	31
13.1.4. Administração do Histórico de Desenvolvimento	33
13.2. ENVY/Swaper	33
13.2.1. Utilizando o ENVY/Swaper	34
13.3. O ENVY/Stats	34
13.4. O ENVY/Packager (Como fazer um Runtime ?).	36
13.4.1. O que é o ENVY/Packager ?	37
13.4.2. Passos para geração de um runtime.	38
13.4.3. Limitações do ENVY/Packager	40
13.4.4. Debugando erros em runtimes	40
14. UM MODELO PADRÃO PARA DESENVOLVIMENTO DE SOFTWARE	41
15. EDUFÓRUM	44
15.1. O projeto hiperNet	44
15.2. O EduFórum	44
15.3. Implementação do cliente EduFórum	44
16. WINDCA (CLIENTE E SERVIDOR)	47
16.1. O WinSdca	47
16.2. Um novo protótipo	47
16.3. Os tipos de conexão	47
16.4. A divisão em camadas	48
16.5. O protocolo de comunicação entre o Cliente e o Servidor	49

16.6. A conexão de visualização	50
17. CONCLUSÕES E PERSPECTIVAS	51
18. BIBLIOGRAFIA	52

1. Introdução

Este projeto foi dividido em três partes. A primeira trata-se de uma consolidação teórica, onde foram revisados conceitos de programação paralela, programação distribuída, programação orientada a objetos, tratamento de exceções, NLS (National Language Support), modelagem orientada a objetos etc. Na segunda estudou-se um ambiente para desenvolvimento cooperativo de software, o ENVY. A terceira parte é caracterizada pelo desenvolvimento de dois protótipos o EduFórum e o WinSdca (Cliente e Servidor).

As técnicas estudadas na primeira parte são a base fundamental para o desenvolvimento dos protótipos. Eles foram desenvolvidos segundo uma filosofia de programação orientada a objetos seguindo uma metodologia de prototipação. Por serem softwares de rede e por terem uma interface orientada a eventos eles exigem a utilização de conceitos de programação paralela. Técnicas como, tratamento de exceções e NLS também foram estudadas e utilizadas no desenvolvimento dos protótipos.

Dada a complexidade dos sistemas que o mercado vem exigindo, a cooperação entre especialistas das mais diversas áreas é indispensável. Um dos objetivos principais deste projeto foi o estudo e utilização de um ambiente de desenvolvimento cooperativo de software. O ENVY é um ambiente que vem se destacando nos últimos anos e será o foco da segunda parte desta monografia.

O EduFórum tornou-se fonte de um estudo ergonômico e o *feedback* deste estudo nos proporcionou um aprendizado muito grande principalmente no desenvolvimento de interfaces homem/máquina. O WinSdca, por ser um projeto multi-linguagem, propiciou-nos a experiência de um desenvolvimento cooperativo com restrição de tempo de desenvolvimento e restrições de comunicação entre as duas linguagens com necessidade de análise e alteração do projeto durante o desenvolvimento.

Esta monografia, também dividida em três partes. Apresentará, na primeira, uma análise de cada uma das técnicas utilizadas no desenvolvimento do projeto, bem como o ambiente Smalltalk, onde ele foi desenvolvido. Ainda nesta parte inicial apresentaremos resumidamente alguns conceitos básicos de rede e o conceito de DLLs (Dynamic Link Library), recurso disponível no Windows que nos exploramos ao longo do projeto. A segunda parte é dedicada ao ENVY, apresentando algumas de suas ferramentas e funcionalidades, e também o modelo que estamos utilizando como padrão no desenvolvimento de nossas aplicações. Por último vamos descrever os protótipos desenvolvidos, enfocando suas modelagens.

2. Apresentação do Ambiente de Desenvolvimento do Projeto (O EDUGRAF)

O Laboratório de Software Educacional foi criado em novembro de 1985. Está vinculado ao Departamento de Informática e Estatística do Centro Tecnológico da Universidade Federal de Santa Catarina.

O objetivo principal deste Laboratório é promover a investigação e a pesquisa na área de desenvolvimento de ambientes computacionais de boa qualidade técnica que auxiliem na implantação de um novo paradigma psico-pedagógico.

Percebendo as atividades básicas da Universidade (ensino, pesquisa e extensão) como intrinsecamente associadas, são também objetivos do EDUGRAF:

- interagir com a comunidade de forma a aplicar o conhecimento que vem sendo sistematizado e produzido desde a sua criação, promovendo, assim, a reconstrução e a reorganização desse mesmo conhecimento;
- estimular nos estudantes a criação de atitudes próprias e adequadas a um bom pesquisador, quais sejam: autonomia, criatividade, gosto e prazer pelo aprender.

Atualmente, o EDUGRAF concentra sua atenção, principalmente, nas seguintes áreas de pesquisa, as quais serão brevemente descritas abaixo:

- A área de Engenharia de Conhecimento procura investigar os sistemas baseados em conhecimento e suas aplicações. A área engloba atividades como: investigação teórica de modelos de representação de conhecimento, estabelecimento de métodos de comparação, tanto do ponto de vista formal como experimental entre os diferentes modelos, desenvolvimento de sistemas baseados em conhecimento e estudo das relações entre sistemas e o processo ensino/aprendizagem.
- Dados os objetivos descritos acima, a abordagem clássica de Engenharia de Software não é apropriada para a construção de um ambiente de desenvolvimento para programas de experimentação. As técnicas clássicas solicitam a definição das funções a que os softwares se destinam já nas fases iniciais do projeto. Esta definição preliminar influencia todo o desenvolvimento, de forma a gerar impacto em caso de redefinição do comportamento esperado do programa. A investigação de novas abordagens tem contribuído para o chamado Modelo Orientado a Objetos. Essa abordagem não está completamente consolidada. No entanto, têm surgido continuamente novas ferramentas seguindo este paradigma, o que demonstra que, ao menos em algumas áreas, já se alcançou um bom nível de maturidade.

PARTE I:
ETAPA DE CAPACITAÇÃO.
CONCEITOS BÁSICOS.

3. Programação Orientada a Objetos

O modelo de programação orientada a objetos (POO) é baseado na idéia de trazer os objetos do domínio de mundo real para o domínio da computação.

Enquanto na programação tradicional, o programa era escrito visando a funcionalidade do sistema, agora, na programação orientada a objetos, nós descrevemos os objetos que compõem este sistema.

Um sistema orientado a objetos pode ser descrito como um conjunto de objetos que se comunicam entre si através da troca de mensagens. Quando um objeto recebe uma mensagem ele executa um método, responde a mensagem retornando um outro objeto, resultado da execução do método. O conjunto de mensagens com o qual os objetos se comunicam determina um protocolo de comunicação entre eles.

Assim como no mundo real, os objetos do mundo computacional são classificados conforme sua estrutura. Então, “uma classe é uma abstração que captura os atributos e operações comuns a um conjunto de objetos” [LaLonde 90] e uma instância é um objeto individual que descreve um membro desta classe.

Como muitos destes objetos são especializações de outros, a POO traz o conceito de hierarquia do mundo real para o mundo computacional.

Desta forma, os objetos são classificados em classes que, por sua vez, são organizadas em uma árvore de superclasses e subclasses. Baseado neste idéia, pode-se dizer que um objeto pertence a uma classe e herda todas as características da superclasse e também passa a responder as mensagens definidas pelo protocolo da superclasse.

Palavras como objetos, mensagens, métodos, hierarquia, podem inicialmente assustar qualquer programador que pense em utilizar esta filosofia. Compreender tais definições talvez não seja tão simples. A simplicidade desta filosofia está no fato de que seus próprios conceitos podem ser firmados com sua utilização. Quando auxiliado por um bom ambiente de programação isto pode se tornar ainda mais fácil.

3.1. Vantagens da Orientação a Objetos

Clareza e Simplicidade - Esta filosofia, por modelar os sistemas de uma maneira bem mais próxima à realidade, tem a vantagem de tornar mais fácil a leitura do programa escrito.

Reusabilidade - Esta é uma das maiores vantagens da programação orientada a objetos, ela será ainda mais evidente se o sistema for bem modelado.

Decomposição de Problemas - Como os sistemas passam a ser descritos pelos objetos que o compõe e não mais por sua funcionalidade, a decomposição dos problemas em problemas menores passa a ser mais natural.

Adaptação - Através do mecanismo de herança a alteração do comportamento de um objeto torna-se simples.

Modularidade - A própria divisão do sistema em classes facilita a modularização deste sistema.

4. Modelagem de Sistemas

“Um modelo é uma abstração de alguma coisa, cujo propósito é permitir que se conheça essa coisa antes de se construí-la” [Rumbaugh 91].

Entre os pesquisadores que estudam as técnicas de modelagem de softwares, uns defendem e tentam impor sua utilização e outros as criticam por sua dificuldade de uso. De qualquer forma somos obrigados a concordar quanto a necessidade de um certo formalismo na modelagem de um software.

Especificar um software em todos os seus detalhes é algo impossível. Particularmente, acredito que a especificação deve ser feita em etapas, paralela ao desenvolvimento, ou melhor, um passo a frente do desenvolvimento, seguindo uma filosofia de prototipação.

O advento da programação orientada a objetos como solução para muitos problemas computacionais trouxe consigo uma série de outras necessidades. A principal delas era a necessidade de uma metodologia de modelagem orientada a objetos.

Embora esta nova filosofia de programação simplifique a etapa de modelagem, esta não pode ser dispensada. Podemos ter certeza que: “antigamente toda parte de análise tinha que ser completada para que o projeto pudesse entrar em desenvolvimento” [Calvet 96]. Hoje não. Por outro lado, outros autores dizem que esta ênfase muito grande na implementação pode ser um passo atrás na engenharia de software [Rumbaugh 91].

Duas foram as técnicas estudadas durante a etapa de capacitação deste projeto. A primeira, OMT (Object Modeling Technique), caracteriza-se por uma modelagem orientada a objetos tentando ser independente do sistema no qual ele será desenvolvido [Rumbaugh 91]. A segunda, chamada de Patterns [Gamma 95], é uma técnica para especificação de problemas, também sob uma filosofia orientada a objetos. Esta segunda é basicamente uma tentativa de comparar os problemas a serem solucionados com outros padronizados anteriormente, baseado no fato de que a solução de problemas similares também deve ser similar.

4.1. OMT

A especificação de sistemas usando OMT consiste na definição de três modelos numa tentativa de tornar mais simples e abrangente a modelagem de sistemas complexos.

O *modelo de objetos* descreve a estrutura hierárquica, o relacionamento entre as classes de objetos encontrados no sistema e os atributos destes objetos. O autor propõe um diagrama onde uma classe é representada por um bloco contendo o seu

nome, os atributos e as principais mensagens entendidas pelas instâncias da classe. Um conjunto de símbolos incorporados a este diagrama representarão o relacionamento entre as classes.

O *modelo dinâmico* é representado por um ou mais diagramas de estados. Este modelo consegue descrever o comportamento dinâmico dos objetos através dos seus estados e dos eventos que levam os objetos de um estado para outro.

O *modelo funcional* é representado por DFDs (Diagramas de Fluxo de Dados).

4.2. Patterns

Repasar experiências ou lembrá-las geralmente não é uma tarefa fácil, principalmente quando temos necessidade delas.

A idéia é catalogar todas as experiências anteriores segundo um conjunto de padrões. Estes padrões são baseados na descrição do problema, da solução e das conseqüências desta solução.

[Gamma 95] propõe um conjunto de padrões para ajudar na modelagem de softwares orientados a objetos.

5. Programação Paralela

Embora a filosofia de programação orientada a objetos tente fazer uma simulação dos objetos do mundo real no mundo computacional [LaLonde 90], ela não traz consigo a propriedade de paralelismo, uma propriedade inerente dos objetos reais.

As mensagens enviadas aos objetos são respondidas com o resultado da execução de um método. Estes métodos são executados de forma seqüencial, ou seja, o envio de uma mensagem a um objeto não passa de uma chamada a um procedimento.

Os objetos reais não agem de forma seqüencial, portanto, seqüencialização de tarefas é uma restrição no desenvolvimento de um software. Dependerá da linguagem de desenvolvimento a possibilidade de utilização de técnicas de programação paralela.

Nesta modelagem, os processos são pedaços de programas seqüenciais que vão compartilhar o mesmo processador. Embora possa parecer uma mistura de modelagens, a modelagem de sistemas segundo os processos que ele apresenta pode “conviver” com uma modelagem orientada a objetos.

Utilizando unicamente uma modelagem por processos, as necessidades de modularização e reusabilidade ainda aparecem e estas são devidamente solucionadas quando mesclamos estas duas modelagens.

6. Programação Distribuída

Quando falamos em programação distribuída, estamos falando em um sistema implementado em várias partes e estas partes sendo executadas em máquinas diferentes.

Um sistema distribuído muitas vezes é a solução para restrições de tempo, periculosidade, etc. Por exemplo, em um sistema de automação industrial pode-se utilizar um CLP para controlar o processo, responsabilizando-se pela aquisição de sinais, matemática de controle e atuação; um PC pode ficar a cargo do armazenamento das informações do processo ao longo do tempo; e outro PC pode servir de interface homem/máquina. É claro que esta distribuição de tarefas em vários equipamentos acarreta em um custo elevado, mas o aumento na utilização de tais sistemas vem provar seu benefício e lucratividade.

Ainda não existe nenhuma padronização na comunicação destes sistemas computacionais de arquiteturas tão diferentes. Acredito que o mercado venha a definir tais padrões ao longo do tempo.

A grande vantagem da utilização de sistemas distribuídos é a especialização dos hardwares utilizados: no chão de fábrica, ao lado do processo podem estar sistemas com controladores embutidos, sistemas pequenos e facilmente substituíveis com interfaces baratas (led's, serial para comunicação com PC). Em uma sala confortável estariam máquinas com interfaces gráficas. As máquinas responsáveis em armazenar os dados teriam HD's de grande capacidade de armazenamento e alta velocidade de acesso.

7. Redes

7.1. O modelo Cliente/Servidor

Cliente/Servidor [Comer 91] é o principal modelo de comunicação entre softwares de rede. Por ser um modelo simples ele tornou-se um padrão para maioria dos softwares conhecidos.

O **servidor** é um programa, sem interface, que oferece um serviço. Ele geralmente fica esperando um pedido que chega pela rede, analisa este pedido e então envia uma resposta.

O programa que envia pela rede os pedidos para o servidor é chamado de **cliente**. Ele é o “negativo” do servidor, ou seja, geralmente ele envia um pedido e fica esperando uma resposta.

Netscape, Eudora, FtpTool, entre outros, são softwares clientes que se comunicam com servidores disponibilizando, numa interface amigável, o serviço oferecido pelo respectivo servidor.

7.2. Protocolo

Estes clientes e servidores se comunicam entre si seguindo um conjunto rígido de normas, estas normas são chamadas de **protocolo**.

Fazendo uma analogia com um fato real, podemos nos colocar como um cliente que deseja ir a uma loja comprar uma calça. Você chega em uma loja, avista um balconista, ele diz: “bom dia”; você repete: “bom dia”; ele pergunta: “o que o senhor deseja?”; você responde: “uma calça”; ele pergunta: “qual o seu numero?”; você responde: “40”; e continua até você pagar e ir embora ou dar uma outra finalização prevista.

Geralmente, os protocolos também definem algumas mensagens de erro. Por exemplo, se você ao invés de responder “40”; você respondesse “15 e 1/4”; o balconista iria dizer “como senhor? Não entendi”.

No mundo da computação os programas são implementados da mesma forma. Um protocolo de comunicação é definido rigorosamente e o programa cliente se comunica com o servidor segundo este protocolo.

7.3. TCP/IP

TCP/IP é um conjunto de protocolos, devidamente organizados e estruturados, que permite a comunicação (interligação) entre quaisquer redes de computadores. O

termo TCP/IP deriva das iniciais dos dois protocolos principais que o compõem: IP (Internet Protocol) e TCP (Transmission Control Protocol).

O protocolo IP é o responsável pelo roteamento das mensagens que transitam na rede, escolhendo os caminhos pelos quais os pacotes devem "viajar". Ou seja, até que uma mensagem chegue ao seu destino pode ser necessário percorrer várias máquinas intermediárias.

A responsabilidade do protocolo TCP é implementar um serviço de entrega de pacotes seguro, com conexão.

7.4. Endereços e Portas

O cliente geralmente é o software que está sendo executado na sua máquina, o servidor pode estar rodando na mesma máquina ou do outro lado do mundo, por exemplo. Então é necessário informar ao cliente com qual servidor ele deverá conversar.

Todas as máquinas em uma rede TCP/IP tem um endereço (ex.: 150.162.61.70) e um nome associado (ex.: desterro.edugraf.ufsc.br). A partir do nome o cliente pode descobrir facilmente qual é o endereço. Como em uma mesma máquina podem rodar vários programas servidores é necessário configurar qual a porta onde servidor esta rodando.

Vamos fazer uma analogia entre máquinas e shoppings. O nome do shopping poderia ser "shopping.beira-mar" ou "shopping.itaguaçu". Olhando um guia turístico de Florianópolis descobriríamos que o primeiro fica no "centro" e o segundo no "continente". Dentro destes shoppings temos várias lojas, que, se fossem identificadas por um número, poderíamos chamá-lo de porta.

7.5. Vantagens do TCP/IP

- Independência do tipo de tecnologia de rede: ao mesmo tempo em que o TCP/IP está baseado na tecnologia convencional de comutação por pacotes, é independente de qualquer interface de hardware particular utilizada (TNC-rádio, placa de rede, linha serial, ...). Além disso, está apto a trabalhar com redes locais (LAN's), metropolitanas (MAN's) e de longa distância (WAN's).
- Interconexão universal: o protocolo TCP/IP permite a qualquer par de computadores estabelecerem uma ou mais conexões.

7.6. Um Exemplo

Coloquemo-nos como usuários de uma rede TCP/IP que querem enviar e receber correspondências (e-mail) usando um software como o Eudora, Netscape ou qualquer outro "mailer".

Os mail são recebidos e ficam armazenados em uma máquina cujo nome você deverá conhecer. Nesta máquina está rodando um programa servidor chamado POP que geralmente roda na porta 110. O seu mailer implementa um cliente POP que vai se conectar ao servidor, os dois conversarão segundo o protocolo POP e todos os seus *mails* serão transferidos para sua máquina onde você poderá lê-los.

Para enviar um mail, o seu mailer implementa um cliente SMTP que vai conversar com um servidor SMTP que deverá estar rodando em uma máquina configurada por você (geralmente na porta 25). Eles conversarão segundo o protocolo SMTP. O *mail* que você escreveu será transmitido para o servidor que vai transmitir para outro (e para outro) até que o mail chegue ao destinatário.

8. Smalltalk

Muito mais do que uma linguagem de programação, Smalltalk é um ambiente de desenvolvimento. Este ambiente é caracterizado por uma total integração de ferramentas disponíveis ao programador. Browsers, editores, debuggers tornam Smalltalk um sofisticado sistema de desenvolvimento suportado por uma linguagem simples, de alto nível, dotada de um imenso conjunto de classes.

Além do ambiente, Smalltalk pode ser diferenciado de linguagens como C, Pascal, Fortran, Modula 2, por ser uma linguagem puramente orientada a objetos, ou seja, ela não traz consigo a necessidade de compatibilização com outras filosofias fazendo uma mistura complicada. Esta diferença se caracteriza com a frase: Tudo em Smalltalk é objeto, inclusive o próprio Smalltalk.

A filosofia de programação orientada a objetos associada a este poderoso ambiente de programação dão ao programador facilidades que aceleram o desenvolvimento de suas aplicações.

A simplicidade da linguagem unida a simplicidade do ambiente reduz o tempo de formação de um programador. É claro que modelagem de sistemas não é algo tão simples, mas sob a orientação de um programador mais experiente torna-se fácil e rápido montar-se uma equipe "Smalltalk" de desenvolvimento.

Uma experiência recente feita no EDUGRAF, provou que em dois meses de trabalho, dez horas por semana, é possível transformar um estudante com mínima formação de computação em um programador Smalltalk. É claro que ainda nesta fase a necessidade de comunicação é muito forte, exigindo reuniões e análise de código, mas nota-se uma tendência descendente desta necessidade.

Talvez o grande segredo esteja no fato de que em Smalltalk aprende-se olhando a implementação do próprio Smalltalk à medida em que você vai desenvolvendo alguma uma aplicação.

9. EHS (Exception Handling System)

Dos conceitos básicos para desenvolvimento de software de qualidade, este não é o mais novo, mas talvez seja o menos difundido. Embora muitos pesquisadores afirmem que esta técnica entra em conflito com a idéia de programação estruturada, ela vem sendo incorporada à maioria dos sistemas de desenvolvimento.

A utilização de um mecanismo central de tratamento de exceções em qualquer metodologia de programação tende a simplificar a maneira como o programa é escrito. Em Smalltalk, linguagem e ambiente de programação puramente orientado à objetos, implementações de mecanismos de tratamento de exceções são rotineiras. Em aplicações como monitoramento e aquisição de dados, o tratamento de exceções é um fator importante para o bom funcionamento do programa.

Um software bem estruturado, geralmente, além de modularizado, também é dividido em camadas. Esta divisão em camadas apresenta uma dificuldade no momento de reportar os erros para as camadas superiores. Poucas são as linguagens de programação que apresentam suporte para solucionar tal problema.

Geralmente o que os programadores fazem são uma série de estruturas de seleção (IF's) e variáveis a serem testadas. Não seria difícil mostrar que um código simples transforma-se em algo monstruoso e de difícil leitura. Isto torna a manutenção do software muito mais complexa.

Ao longo do projeto estudamos dois tipos de tratamento de exceção: um baseado no artigo do Christophe Dony [Dony 90] e o outro baseado no Smalltalk 80 [ParkPlace 92].

A grande diferença entre os dois sistemas é que o primeiro trata os tipos de exceções como sendo classes com suas hierarquias; o segundo, embora não tão elegante quanto o primeiro, é mais utilizado e trata as exceções como variáveis.

Outra diferença entre os dois modelos é que o primeiro é um sistema desenvolvido para Smalltalk e o segundo é utilizado no desenvolvimento de um Smalltalk. Quando uma linguagem não utiliza EHS em sua própria implementação, teremos um sistema híbrido, ou seja, as exceções ocorridas em métodos da própria linguagem são difíceis de serem tratadas. Por exemplo, se ao tentarmos abrir um arquivo para escrita não obtivermos permissão do sistema operacional para isto, o Smalltalk V/Win da Digitalk gera um erro enquanto o VisualWorks da ParcPlace gera uma exceção.

Analisemos o exemplo simples abaixo:

Exemplo.

```
Object errorSignal
  handle: [: theException | Transcript show: 'During!!!'.
          theException returnWith: 'NOT everything executed'
] do: [ Transcript show: 'Before...'.
      Object errorSignal raise.
      Transcript show: 'After'.
    ].
```

Resultado 1.

```
Transcript shows: Before...During!!!
Result: 'NOT everything executed'
```

Este exemplo está escrito segundo o modelo descrito em [ParkPlace 92]. A classe *Objetc* recebe a mensagem *#errorSignal* que retorna um objeto da classe *Exception*. Este, por sua vez, recebe a mensagem *#handle:do:* com dois blocos como parametro. O primeiro especifica o tratador que, neste caso, mostra o string *'During!!!'* na janela *Transcript* e depois finaliza o tratamento retornando o string *'NOT everything executed'*. O segundo bloco (bloco principal) especifica o código original do programa, ele mostra o string *'Before ...'*, dispara a exceção e mostra o string *'After'*.

Avaliando o exemplo, obtemos como resultado o string *'NOT everything executed'* e na janela *Transcript* aparece o string *'Before...During!!!'*. Isto demonstra que o programa iniciou no segundo bloco, ocorreu uma exceção, o tratamento foi executado e o segundo bloco foi abortado.

Percebamos que, a título de exemplo, a exceção está sendo sinalizada (*Object errorSignal raise.*) na mesma camada que o tratamento mas geralmente isto ocorre em camadas inferiores.

Este código é um exemplo onde deseja-se abortar o segundo bloco independentemente da exceção que venha a acontecer. Pode-se:

- alterar o tipo de exceção a ser tratada mudando-se a primeira linha onde ela foi especificada.
- tratar mais de um tipo de exceção alterando a mensagem *#handle:do:* por uma mais genérica:

```
HandleCollection new
  on: Signal genericSignal
  handle: [ "Bloco de tratamento" ];
  on: File errorSignal
  handle: [ "Bloco de tratamento" ];
  "Pode-se repetir estas duas linhas indefinidamente"
```

handleDo: [Bloco principal].

- corrigir o possível erro e prosseguir o bloco principal trocando a mensagem *#returnWith:* enviada para o objeto *theException*, dentro do bloco de tratamento, pela mensagem *#proceedWith:*.
- fazer um tratamento parcial e deixar que a camada acima termine o tratamento trocando *#returnWith:* por *#reject:*

Talves este exemplo não venha a mostrar a simplificação final conseguida no código escrito mas ela torna-se evidente quando imaginamos um sistema maior e subdividido em camadas.

10. NLS (Nacional Language Support)

Infelizmente a língua que domina o mundo não é o Português, portanto, outra preocupação que temos que ter na hora de desenvolver um software, é a língua na qual ele será desenvolvido.

NLS é um sistema (uma metodologia) que vem sendo disponibilizado pelas melhores linguagens de programação conhecidas. Este sistema, além de facilitar a troca da língua utilizada na interface, também auxilia na forma de mostrar os números, valores monetários, data, hora, etc.

As várias implementações de Smalltalk oferecidas no mercado (Digitalk, ParcPlace, OTI/IBM) trazem consigo um conjunto de classes para prover este suporte.

No caso do Smalltalk da IBM, que vem se destacando como a linguagem de programação multi-usuário e multi-plataforma, há suporte para que o software pergunte ao sistema operacional sobre o qual ele está rodando, qual a língua em que ele deve se apresentar.

Estas classes também devem dar suporte para o armazenamento destas mensagens em arquivos que deverão ser lidos em tempo de execução.

Podemos imaginar que se o programador não vai utilizar os *strings* diretamente no seu código ele deverá utilizar 'chaves' para substituí-los. A utilização destas chaves não torna mais fácil a leitura do código escrito, de forma que o programador deverá seguir alguns padrões [IBM-ST 94] para que isto não torne-se um problema.

11. DLL - Dynamic Link Library

11.1. O que é uma DLL?

Uma DLL (Dynamic Link Library) é um módulo (extensão .DLL e, por vezes, .EXE) que contém código e/ou '*resources*' (dados como ícones, caixas de diálogo, etc) que serão posteriormente usados por outras aplicações. No ambiente Windows, DLLs permitem que várias aplicações diferentes possam compartilhar os mesmos códigos (programas), variáveis e '*resources*'.

Um conceito próximo ao de uma DLL é o de uma UNIT do Pascal (TPU). A diferença de uma DLL para uma TPU está no processo de "linkagem". Enquanto as TPUs são "linkadas" ao programa na hora da compilação (o que resulta na incorporação do código da TPU no código do programa), as DLL são chamadas apenas quando se roda o programa.

11.2. Como funciona uma DLL?

Uma DLL funciona da seguinte forma: a aplicação que vai usar a DLL de nome X pede para o sistema carregar essa DLL. O sistema verifica se X existe e, em caso afirmativo, aloca a memória requerida por X para rodar e incrementa um contador que marca quantas aplicações estão usando X naquele momento. Assim que todas as aplicações deixam de usar a DLL X, o sistema encerra a execução de X e libera a memória usada por X para outras aplicações.

11.3. Vantagens do uso de DLLs.

O uso de DLLs propicia certas vantagens ao programador, como modularidade, racionalização do uso de memória e processamento, ou mesmo a garantia de compatibilidade de código entre diferentes linguagens.

A primeira vantagem das DLLs é exatamente a modularidade. Através do uso de DLLs é possível a atualização dos módulos DLL de uma aplicação sem alteração do código da aplicação em si. Isso possibilita ao programador uma maior flexibilidade tanto no desenvolvimento como na manutenção de aplicações, já que o código da DLL fica independente do código do programa. Além disso, o programador pode, ainda, adicionar DLLs "comerciais" às suas aplicações, bastando, para isso, apenas a documentação da DLL que irá utilizar.

Outro benefício conseguido é a otimização do uso de memória do equipamento, pois mesmo que se tenha mais de uma aplicação chamando uma mesma DLL, como exemplificado na figura abaixo, esta DLL só será carregada uma vez. Isso evita a replicação de código e dados no uso de aplicações multi-tarefa, o que, de outra forma, dificilmente seria conseguido.

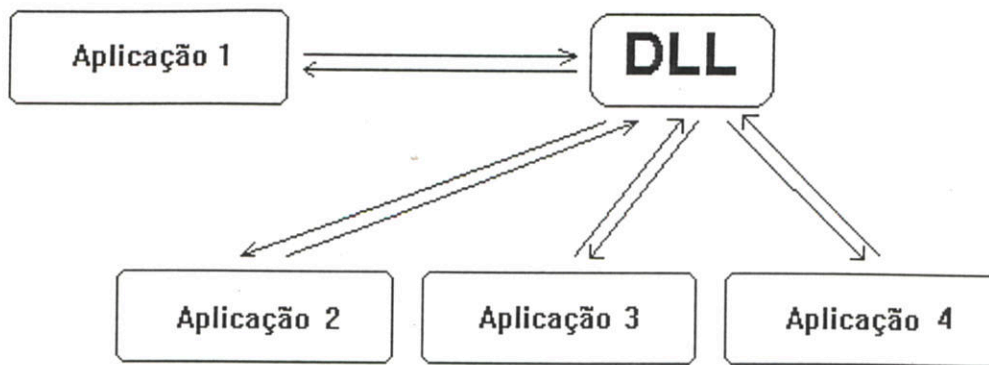


Figura 11.1

Ainda tem-se a vantagem de poder usar uma DLL específica sobre qualquer linguagem que suporte DLLs, não importando em que linguagem foi a DLL escrita originalmente. Isto equivale a dizer que se pode ter uma aplicação escrita em 'C' chamando uma DLL desenvolvida em Pascal ou em qualquer outra linguagem.

11.4.Exemplos de uso de DLLs.

Mas para que servem todos os benefícios trazidos pelo uso de DLLs, se o programador ignora onde possa ser utilizada uma DLL no desenvolvimento de sua aplicação? Veremos a seguir alguns exemplos do uso de DLLs no desenvolvimento de aplicações e, através deles, podemos pensar onde uma DLL se encaixaria no desenvolvimento de determinada aplicação.

11.4.1.As DLLs do Windows

O primeiro e mais importante exemplo de uso de DLLs é o próprio Windows, que é composto de várias DLLs. Entre as DLLs usadas pelo Windows estão KERNEL, GDI e USER. A DLL GDI por exemplo, é a que cuida da parte gráfica do sistema. O Windows se vale desse recurso para evitar a duplicação de código, o que tornaria o sistema inviável. Já imaginou se cada programa tivesse de incorporar uma biblioteca de tratamento gráfico ?

11.4.2.Controle de periféricos via DLL

Outro exemplo do uso de DLL é o controle de hardware e outros dispositivos, como no caso do acesso de aplicações a uma placa. Se cada aplicação tiver seu próprio código de controle dessa placa (figura 11.2a), haverá a possibilidade de se terem colisões no acesso ao periférico (coisa que o programador terá de resolver de algum jeito). Através do uso de uma DLL específica para o controle dessa placa, não existirá o problema de colisões no acesso, nem será replicado o código de controle (figura 11.2b).

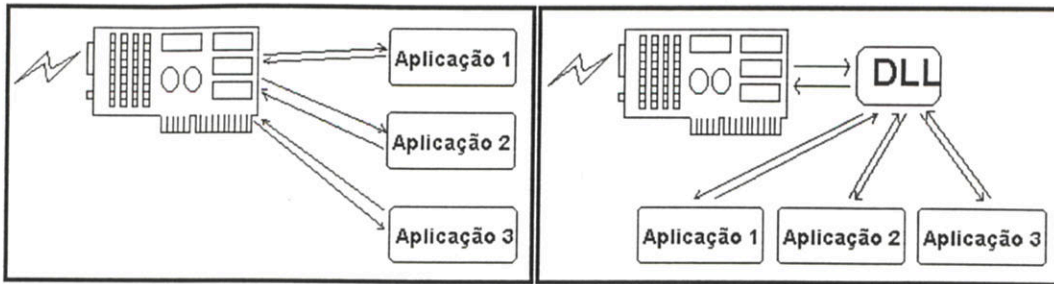


Figura 11.2a

Figura 11.2b

11.4.3. Bancos de dados, editores gráficos e muito mais.

Várias outras aplicações podem ser desenvolvidas através do uso de DLLs. Aplicações como banco de dados podem usar DLLs para fazerem a conversão de formatos de banco de dados diferentes. Muitas DLLs com essa finalidade já existem no mercado, o que libera o programador de construir um mini-sistema para essa tarefa. Aplicações gráficas podem compartilhar funções de uma mesma DLL, o que possibilita a criação de sistemas gráficos completos sem desperdício de código e/ou memória. Isso sem falar nas aplicações de DLLs na área da multimídia, em que uma aplicação pode ter acesso a informações criadas a partir de outros programas pelo uso de DLLs que fazem a interconexão entre as duas aplicações. Enfim, pode-se dizer que onde houver programação voltada à multi-tarefa, se tem a necessidade de um mecanismo como DLL.

11.5. Como chamar rotinas de DLLs.

As DLLs, como vem sendo dito até agora, podem ser chamadas de qualquer linguagem que as suporte. Embora sejam de uso genérico e qualquer linguagem que as suporte possa evocá-las, o procedimento de chamada a uma DLL varia de linguagem para linguagem.

12. Small-WinSock

12.1. Windows Sockets

Maior utilização do UNIX e domínio do mercado pelo Windows ou Novel são motivos de intensas discussões entre os vários especialistas. De uma maneira ou outra, todos concordamos que a Internet vem tomando conta de uma grande fatia no que diz respeito a intercomunicação de computadores no mundo. Com a Internet o TCP/IP passa a ser o protocolo de comunicação dominante. Também devemos acreditar que o Windows está dominando a maior fatia do mercado no que diz respeito a sistema para computador pessoal. Baseados nestes fatos podemos concluir que o desenvolvimento de software sobre Windows para TCP/IP vem se tornando uma área de pesquisa acentuada.

Windows Sockets [Hall 92] é um padrão de comunicação TCP/IP para Windows e sobre eles vem sendo desenvolvida uma gama de aplicativos Windows. Aplicativos como Gopher, FTP, TELNET, POP, SMTP e WWW estão disponíveis para o usuário INTERNET.

12.2. Interfaceando Smalltalk com Windows Sockets

No projeto **Small-WinSock** [Pereira 95] implementamos um conjunto de classes e métodos para fazer o interfaceamento do Smalltalk com a DLL WinSock, tanto para Smalltalk V/Win 2.0 da Digitalk quanto para o VisualWorks 1.0 da ParcPlace.

O Small-WinSock é composto por três subconjuntos de classes:

- as classes que representam as estruturas "C";
- a classe que faz o interfaceamento propriamente dito;
- as classes que elevam o nível do Windows Sockets.

Esta aplicação tem como pré-requisito um sistema de tratamento de exceções utilizado para reportar os erros às camadas superiores. Desta forma os erros não são mais reportados como números que estariam sob uma estrutura *case*.

PARTE II:
ESTUDO DE UM AMBIENTE DE
DESENVOLVIMENTO COOPERATIVO DE
SOFTWARE.

O ENVY.

13. Desenvolvimento de software cooperativo

prol
A complexidade dos sistemas computacionais demandados pelo mercado, vem denotando uma necessidade de um desenvolvimento cooperativo destes sistemas, de forma que uma equipe com especialistas nas mais diversas áreas possam cooperar em pró deste desenvolvimento.

Problemas como comunicação, distribuição e sincronização de tarefas tornam esta cooperação algo muito complexo. Um dos principais objetivos deste projeto foi o estudo e utilização de um ambiente para desenvolvimento cooperativo de software.

O ENVY [OTI-ED 96] é um sistema desenvolvido pela OTI (Object Technology International Inc.), que roda sobre Smalltalk em várias plataformas e traz consigo várias ferramentas com o objetivo de auxiliar os programadores no desenvolvimento de suas aplicações.

13.1. ENVY/MANAGER

Nos últimos anos verificamos um grande avanço na área de informática, tanto a nível de hardware quanto a nível dos softwares disponíveis. Desta forma, podemos afirmar que tornou-se mais complexo o desenvolvimento de tais softwares. Esta complexidade vem exigir uma série de requisitos para desenvolvimento e estes nos mostram que tornou-se impossível o desenvolvimento de software que não seja em equipe, onde cada membro da equipe se especializa em uma determinada área.

Sabemos que agora um desenvolvimento em equipe vem exigir um ambiente que o suporte. Este ambiente deve satisfazer uma série de exigências tais como: facilitar a comunicação entre os membros da equipe, fazer o controle de versões, facilitar a modularidade e ter suporte de trabalho em rede. Podemos perceber que não estamos falando de uma linguagem para desenvolvimento, mas sim de um ambiente que tenha tais facilidades. A linguagem de programação seria somente uma parte de tal ambiente.

O ENVY, muito mais que uma ferramenta para auxílio no desenvolvimento de software, se propõe a ser este ambiente. Hoje já sabemos que ele preenche muito dos requisitos de um ambiente de desenvolvimento, mas peca em outros.

13.1.1. Arquitetura do ENVY

A arquitetura do sistema ENVY/MANAGER é baseada na idéia de múltiplos *images* Smalltalk acessando concorrentemente um repositório de código sobre uma LAN (figura 13.1) Este repositório é conhecido como LIBRARY, onde ficam armazenados todos os fontes e métodos compilados ficam guardados nela.

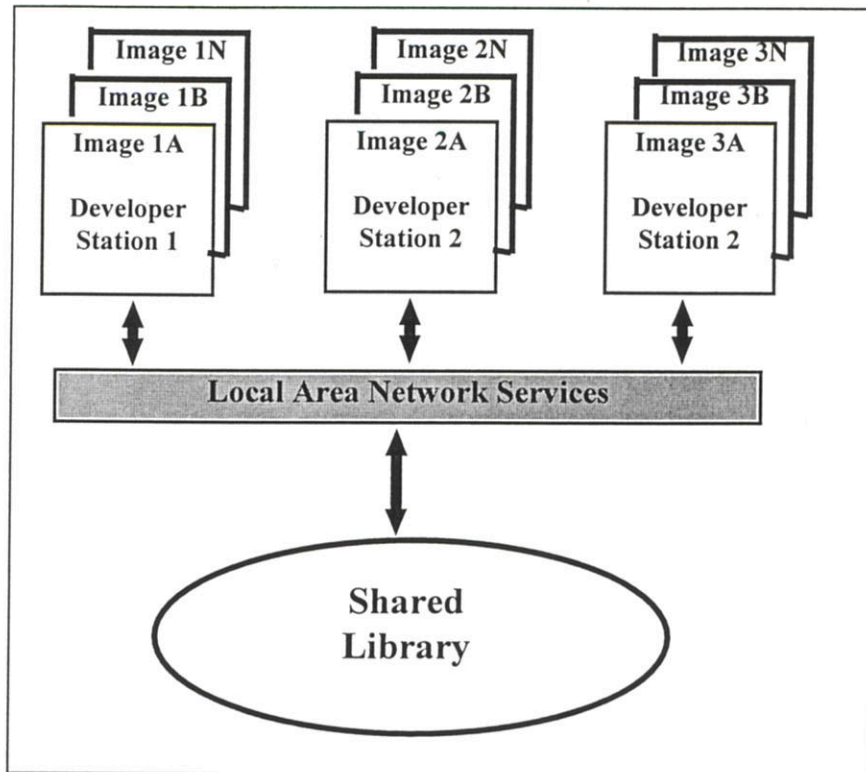


Figura 13.1

O usuário pode carregar os componentes de um software da biblioteca para seu image. Observemos aqui que o método não é recompilado ao ser recarregado em seu image, pois este já estava guardado compilado na LIBRARY. As classes e os métodos são apenas linkados ao image, resultando em uma alta performance.

Cada alteração feita será salva na biblioteca de forma que o desenvolvedor terá acesso a cada versão de seu código, podendo retornar a qualquer versão em qualquer momento. Cada uma destas versões também estará disponíveis aos outros usuários para que estes também possam utilizá-las.

É importante lembrar que o carregamento (e descarregamento) de uma aplicação da LIBRARY para o image (do image) é uma operação atômica. Esta operação poderá resultar em um erro, mas não deverá deixar o image inconsistente.

13.1.2. Componentes de administração

13.1.2.1. Componentes do sistema

Library- A library é um repositório multi-usuário que administra os vários componentes desenvolvidos pelos vários usuários. Cada usuário terá seu image Smalltalk conectado via rede local a esta biblioteca.

Esta biblioteca é composta de um único arquivo e um software administrador.

Users- Cada biblioteca mantém uma lista de usuários representando cada desenvolvedor. Um destes usuários será o Library Supervisor. Somente ele poderá criar ou modificar usuários.

13.1.2.2. Componentes de Software

Conceitos de classes, métodos e herança já são próprios do Smalltalk, mas conceitos de *aplicações*, *configurations maps*, *pré-requisitos*, *definição* e *extensão* são introduzidos pelo ENVY.

Aplicações- A grande promessa da programação orientada a objetos é baseada no alto grau de reusabilidade. Esta idéia só é possível graças a estruturação dos objetos em classes reusáveis. O ENVY introduz um novo componente para aumentar este grau de modularização e reusabilidade.

Aplicações representam um conjunto de classes que servem a um mesmo propósito. Estas classes podem ser definidas ou estendidas na aplicação.

As aplicações podem manter dois tipos de relações entre si: pré-requisitos e sub-aplicação.

Configuration Maps- Um "Configuration Map" tem dois propósitos:

- durante o desenvolvimento ele provê aos usuários a facilidade de carregar novas alterações em seus sistemas;
- depois do projeto finalizado, ele é usado para construir um conjunto final de aplicações. De forma que o desenvolvedor, usuário deste conjunto, possa carregá-lo com facilidade.

13.1.3. Administração de alterações

Um ambiente como o ENVY provê facilidades para o desenvolvimento de software e também para a manutenção deste. Além desta ferramenta para o auxílio ao desenvolvimento, o ENVY também provê mecanismos para desenvolvimento em equipe e coordenação desta equipe. Como subconjunto destas facilidades podemos citar: controle de versões, edições, acesso e alteração.

O **change management** é responsável pelos seguintes aspectos:

- habilitar ou restringir a possibilidade de alguma alteração conforme as permissões do usuário;
- permitir a concorrência no desenvolvimento das diversas partes do software tratando as políticas de permissões;
- habilitar ou restringir o simples acesso conforme as permissões do desenvolvedor, o usuário ENVY.

13.1.3.1. Controle de alterações

O controle de alterações no ENVY é baseado em regras distintas para os desenvolvedores envolvidos na aplicação.

O **class owner** é o responsável pela integridade do código da classe. Ele vai definir a política de utilização para cada desenvolvedor, tais como: criar novas versões, apagar a classe da aplicação, etc.

O **class developer** é o usuário que está desenvolvendo a classe no momento. Ele pode ser o próprio owner ou não. Somente o desenvolvedor da edição da classe pode:

- carregar a edição em seu image;
- fazer uma nova versão da edição;
- alterar a definição da classe;
- alterar, carregar, apagar ou salvar um método;
- tornar um método público ou privado;
- alterar os comentários ou notas dos métodos e da classe.

X O **application manager** é o responsável pela aplicação como um todo, é ele quem deve coordenar as atividades entre os vários desenvolvedores. Somente ele poderá:

- X • adicionar um novo usuário ^{ou} no grupo de desenvolvedores da aplicação;
- apagar um usuário;
- determinar o owner de cada classe;
- repassar a responsabilidade de manager da aplicação;
- criar uma nova edição da aplicação;
- alterar os pré-requisitos da aplicação;
- alterar a lista de compatibilidade entre versões;
- adicionar novas subaplicações;
- criar uma versão da edição.

* O **configuration map manager** é o responsável por manter a integridade do "configuration map" que ele administra. Somente ele poderá:

- criar versões do "configuration map"
- criar novas edições;
- adicionar novas aplicações;
- apagar aplicações.

13.1.3.2. Controle de acesso

X O controle de acesso foi concebido para assegurar uma política de controle nas permissões de leitura ou modificações. As principais categorias de controle de acesso são em relação ao controle de acesso do image, da library e da aplicação. são a

O **controle de acesso ao image e à library** são feitos através do network name do usuário e de sua senha.

O **controle de acesso à aplicação** é baseado nos diferentes privilégios que podem ser dados a todos os usuários da library, ao grupo, ou somente ao owner. Privilégios tipo:

- de execução, ou seja, a aplicação pode ser carregada;
- leitura do fonte dos métodos públicos;
- leitura do fonte dos métodos privados;
- criação de novas edições.

13.1.4. Administração do Histórico de Desenvolvimento

A administração do histórico de desenvolvimento também é baseada em uma série de regras distintas:

- cada edição de um componente de software terá além do seu nome, uma data hora associados a ela;
- cada versão terá um número associado;
- toda e qualquer edição e versão de um componente de software será guardada na library;
- novas edições deverão ser geradas a partir de versões;
- múltiplas edições poderão ser criadas a partir de uma única versão;
- uma versão de um componente de software é imutável;
- só poderão ser criadas novas edições de um sub-componente a partir de uma edição do seu componente superior;
- para criar-se uma nova versão de um componente todos os sub-componentes deverão ser versionados.

13.2. ENVY/Swaper

X Uma grande faixa de aplicações desenvolvidas necessitam de arquivamento de dados. Até o momento este problema parecia estar resolvido com a utilização de bancos de dados relacionais. Problemas como segurança, integridade, extensibilidade, recuperação frente a problemas de hardware já eram assuntos bem estudados. Outros

problemas como compartilhamento de usuários e aplicações e distribuição de dados compõem um vasto campo de pesquisa.

Com a ascensão da utilização de programação orientada a objetos tais assuntos voltaram a ser discutidos. Esta nova filosofia de programação gerou consigo alguns problemas que ainda estão sendo pesquisados. A bibliografia se divide entre autores mais conservadores que sugerem o mapeamento de objetos em entidades e relacionamentos e outros autores que mostram sua pesquisa em bancos de dados orientados a objetos.

Ainda fica a cargo da equipe de desenvolvimento a escolha entre um mapeamento para um banco de dados relacional ou um banco de dados orientado a objetos.

Ainda mais indefinidas são as relações entre a orientação a objetos e a distribuição de tais dados. Talvez possamos afirmar que os grandes problemas são: achar a melhor solução nesta grande árvore de soluções e achar a forma com que cada uma se identifica com uma aplicação diferente.

O ENVY/Swaper é uma ferramenta *high speed* para arquivamento de objetos.

13.2.1. Utilizando o ENVY/Swaper

ENVY/Swaper é usado para baixar e carregar objetos em arquivos. Veja o exemplo:

```
ObjectDumper new  
unload: objetoASerSalvo intoFile: 'fileName'.
```

ou

```
objetoRecuperado := ObjectLoader new loadFromFile: 'fileName'.
```

Uma grande vantagem da utilização do ENVY/Swaper é a possibilidade de desligar (unlink) objetos durante o arquivamento.

13.3. O ENVY/Stats

Uma necessidade no desenvolvimento de um Software profissional é a análise dos gargalos deste software.

Muitas vezes uma simples análise do código pode facilitar sua otimização. Mas sem uma ferramenta de auxílio esta análise se torna muito difícil. O ENVY/Stats se propõe a ser esta ferramenta.

Vamos ver o seguinte exemplo:

```
| dic |
```



```

(MessageProfiler spyOn: [
  dic := Dictionary new: 8 * 1024.
  1 to: 20 * 1024
    do: [:i |
      dic
        at: (String new: (i \\ 100 + 1))
        put: 0.
    ].
]) browse.

```

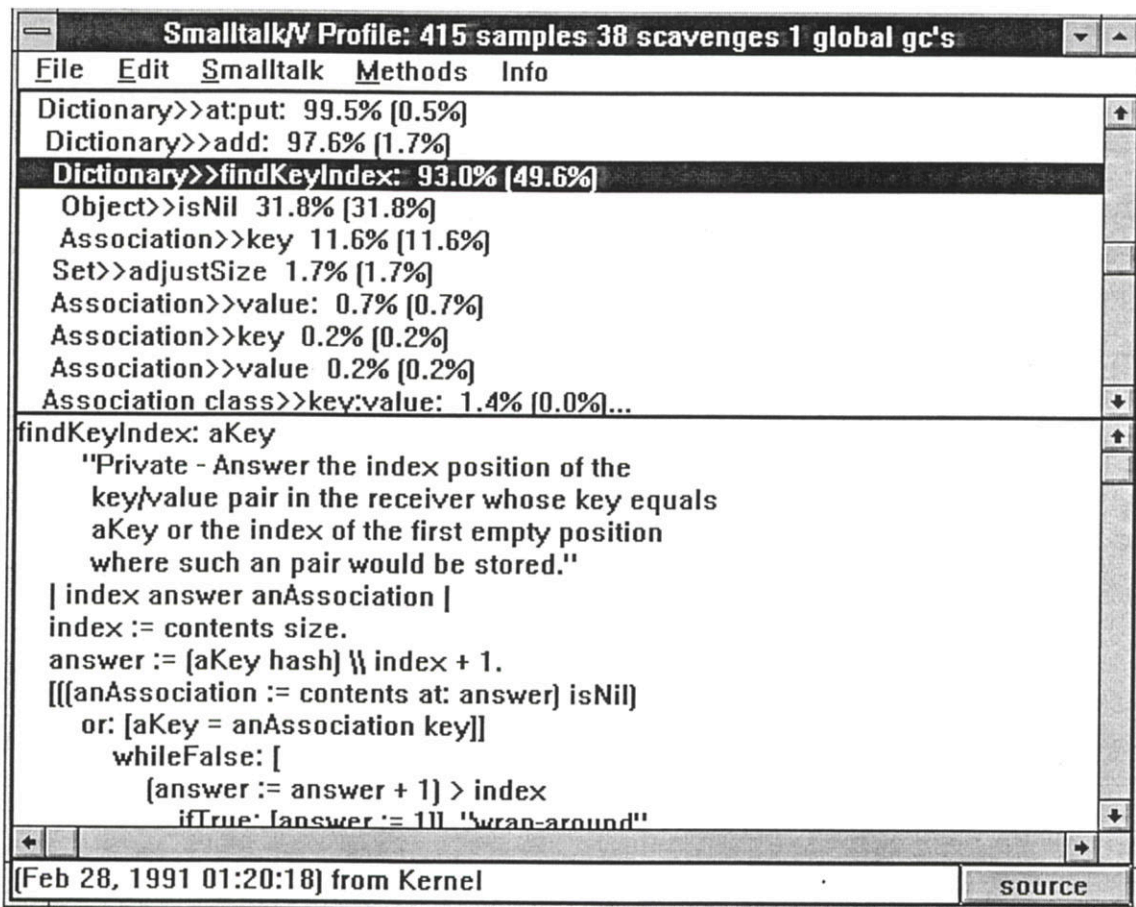


Figura 13.2

Este código cria um dicionário de tamanho igual a 8K e insere 20K strings nele. Como o dicionário é pequeno ele terá que crescer. No final do processamento um browser é aberto (figura acima) nos mostrando a distribuição do tempo ao longo dos vários métodos que foram chamados

Este browser é composto por duas "panes" (partes de uma janela). A "pane" superior mostra uma hierarquia de métodos com a porcentagem de tempo que seus sub-métodos gastaram e a porcentagem que ele gastou em si mesmo. A pane inferior mostra o código do método selecionado na superior.

Uma análise das relações de tempo levarão o desenvolvedor aos pontos que ele deverá otimizar.

13.4. O ENVY/Packager (Como fazer um Runtime ?).

Embora os programadores Smalltalk não gostem da palavra "interpretada", podemos ver, a priori, Smalltalk como sendo uma linguagem interpretada.

X Um dos grandes problemas das linguagens^{ns} interpretadas é o fato delas não gerarem programas executáveis. Alguns interpretadores exigem que o desenvolvedor entregue seu código fonte ao seu cliente juntamente com o interpretador para que este possa usar a aplicação desenvolvida. Smalltalk tem o poder de incorporar o código desenvolvido fornecendo ao desenvolvedor o poder de fazer runtimes de suas aplicações sem mostrar o código fonte.

Bom ambiente de desenvolvimento, Smalltalk traz consigo um grande conjunto de classes reusáveis para o desenvolvedor. Analisando Smalltalk como sendo uma linguagem interpretada, podemos mostrar a dificuldade de selecionar o conjunto mínimo e completo destas classes e métodos que a aplicação desenvolvida irá utilizar.

Dado o fato que Smalltalk não é uma linguagem tipada, as classes dos objetos só serão definidas em tempo de execução, então torna-se ainda mais difícil selecionar este conjunto de classes.

Como o caminho: desenvolve, salva, compila, linka e executa já não existe em Smalltalk, qualquer conjunto de classes mal selecionado só vai ser detectado em tempo de execução.

Desta forma, um conjunto de regras se faz necessário para que seja possível gerar-se um runtime, além de uma boa ferramenta de auxílio.

Grande parte das implementações de Smalltalk não fornecem esta ferramenta, de forma que toda a escolha deste conjunto deve ser feita pelo desenvolvedor através da procura dos "senders" dos métodos, das referências das classes e pela inicialização das variáveis globais.

* Além de toda uma melhora no ambiente de desenvolvimento Smalltalk, o ENVY traz consigo algumas ferramentas e entre elas o ENVY/Packager.

senders ?

13.4.1.O que é o ENVY/Packager ?

Para podermos seleccionar este conjunto de classes, métodos e outros objetos, necessitamos de uma ferramenta de auxílio. O ENVY/Packager se propõe a ser esta ferramenta.

O Smalltalk IV Windows pode ser dividido em três grandes arquivos: uma DLL onde ficam as classes e objetos que serão utilizados no desenvolvimento de aplicações, outra DLL onde está o compilador e um arquivo executável onde ficam as classes e objetos desenvolvidos pelo programador. Este último arquivo também guardará todas as alterações feitas pelo programador nas classes do sistema.

De uma forma não ideal, estas classes alteradas são inteiramente replicadas neste último arquivo (chamado de image). O ENVY/Packager, além de auxiliar na escolha do conjunto mínimo necessário de classes, se propõe a consertar este erro, ou seja, as classes continuam replicadas em tempo de desenvolvimento, mas esta replicação é anulada no momento da confecção do runtime.

13.4.1.1.Regras para desenvolvimento de uma aplicação.

Se por um lado Smalltalk tem uma série de benefícios que outras linguagens não tem, por outro lado ele também dá ao desenvolvedor uma série de liberdades com a qual o desenvolvedor deverá conviver. Algumas destas liberdades vem a facilitar o desenvolvimento, outras, aparentemente vantajosas, trarão incômodos futuros.

Vamos descrever abaixo algumas regras práticas que devem ser seguidas no momento do desenvolvimento para facilitar a futura geração de um runtime enxuto. Algumas delas vem frisar aquilo que não se deve fazer em momento algum. De qualquer forma estas regras não liberam o programador de todos os seus compromissos de engenharia.

1. O uso de variáveis globais: Todas as variáveis globais devem ser inicializadas antes da geração do runtime, de forma que o objeto referenciado pela variável não venha a fazer volume desnecessário no runtime. Desta forma, cabe ao desenvolvedor "anotar" suas variáveis globais para não esquecê-las no momento de configuração do packager. Um programador mais experiente sabe que o uso de variáveis globais é condenado por uma série de motivos. Embora muitas vezes o seu uso torne o desenvolvimento mais simples.

2. A menção indireta à classes: Sabe-se que tudo em Smalltalk é um objeto pertencente a uma classe e que as classes também são instâncias de uma classe. Esta é uma liberdade que deverá ser usada em raros casos. Por exemplo: podemos criar um dicionário onde as chaves tenham valores quaisquer e os valores sejam classes. Se agora fizermos menção a estas classes através deste dicionário, o ENVY/Packager não vai detectar esta menção não incluindo estas classes ao runtime.

| var1 var2 |


```
var1 := Dictionary new.  
var1 at: 'c1' put: OrderedCollection.  
var2 := (var1 at: 'c1') new.
```

O ENVY/Packager não vai detectar que (var1 at: 'c1') é uma menção a OrderedCollection não incluindo esta classe no runtime. Torna-se grande a probabilidade de esquecimento da inclusão de tais classes pelo desenvolvedor na configuração do Packager.

* 3. Menção indireta a métodos: Os métodos são objetos da classe Message e portando também poderiam ser mencionados do forma indireta, mas isto não deve ser feito porque o Packager não conseguiria detectar os "senders" destes métodos.

4. Variáveis de classe: O cuidado dispensado às variáveis de classe deve ser igual ao cuidado dispensado às variáveis globais. As classes referenciadas na aplicação a ser "empacotada" farão parte do runtime gerado e junto com estas classes irão as variáveis de classe. Estas deverão ser inicializadas antes da geração do runtime, de forma a não fazer volume desnecessário.

13.4.2. Passos para geração de um runtime.

O ENVY/Packager geralmente é a última preocupação no ciclo de desenvolvimento de um protótipo, mas não deveria ser assim. Esta estrutura de pensamento vai fazer com que sejam esquecidos pré-requisitos importantes. Durante todo o desenvolvimento, o programador deve se preocupar com alguns detalhes de forma a não encontrar barreiras na hora de gerar o runtime, para que ele não tenha que voltar, corrigir estes problemas e depois novamente se deparar com outros problemas na confecção do runtime. Tendo que repetir este ciclo para cada pré-requisito esquecido.

Passos para geração do runtime:

1. **Carregar o ENVY/Swaper:** O ENVY/Packager tem como pré-requisito o ENVY/Swaper, pois este último tem a função de guardar os objetos em arquivo. Desta forma, podemos definir o Packager como uma aplicação que utiliza o Swaper para auxiliar o desenvolvedor a gerar um runtime.

Obs.: O ENVY/Swaper é carregado através de um "Configuration Map".

2. **Carregar o ENVY/Packager:** Assim como o Swaper, o ENVY/Packager também é carregado através de um "Configuration Map".

3. **Configurar o Packager:** Uma vez carregado o ENVY/Packager, devemos configurá-lo para que ele possa empacotar um "runtime". Vejamos os itens mais importantes nesta configuração:

a: Escolha das aplicações que vão formar o corpo do "runtime".

b: Verificação das classes que compõem as aplicações selecionadas em "a" que não precisam estar no "runtime", ou seja, classes não referenciadas. Neste momento podemos dizer que seria difícil construir uma ferramenta inteligente o suficiente para ser automática. De fato o ENVY/Packager não exclui as classes não referenciadas, ele só fornece uma lista delas. Depois disso o desenvolvedor deverá analisar cada uma destas classes listadas e excluí-las uma a uma através de um browser. Dado o fato de que este browser não é muito cômodo para a exclusão de várias classes, segue abaixo um código para facilitar o trabalho.

*(PackagerLauncher allInstances at: 1)
selectedImage excludeClassesNamed:
#("conjunto de classes apresentado pelo packager).*

Obs. 1. Este passo deverá ser repetido várias vezes até que não hajam mais classes a serem excluídas.

Obs. 2. Se o desenvolvedor não seguir algumas das regras descritas na sessão "Regras para desenvolvimento de uma aplicação", ele deverá se preocupar em não excluir nenhuma classe referenciada indiretamente.

c: Procura de métodos sem "senders": Depois de excluídas algumas das classes não utilizadas pela aplicação, pode-se procurar os métodos não chamados.

Depois disto podemos voltar ao passo "b", pois, tais métodos poderiam fazer referências a algumas classes que agora tornam-se desnecessárias.

Neste momento tem-se a impressão que vamos ficar em loop nos passos "b" e "c" por algum tempo, mas isto não é verdade. Bastam umas três iterações para conseguirmos chegar a um estado final de exclusões.

Em algumas aplicações, o desenvolvedor poderá ter a intenção de transformar palavras (por exemplo, lidas de um arquivo) em mensagens enviadas a algum objeto. Neste caso ele deverá incluir estes métodos manualmente, dado o fato de que a exclusão de métodos é automática. O macete, neste caso, é incluir o nome destes métodos como símbolos em algum método que necessariamente não será excluído.

d: Procurar globais não referenciadas.

e: Inicializar globais e variáveis de classe.

f: Fazer um método similar ao **#startUpApplication:** da classe **NotificationManager** e fazer uma troca destes dois (Menu: ReplaceWith no Classes Mapping Browser).

evitar! substituir por Sugere-se

Obs. 1. Encontramos alguns problemas neste momento, por isso, implementamos este método da seguinte forma:

newStartupApplicationMethodo: oldWindow

InputEvent addEvent:

*(Message new selector: #openApplicationX
receiver: ClasseX).*

13.4.3.Limitações do ENVY/Packager

O ENVY/Packager para Smalltalk /V Windows limita os objetos a serem salvos em 64k. Qualquer objeto de tamanho maior que este deverá ser inicializado depois do startup da aplicação.

Muitas vezes a grande necessidade de uma ferramenta para auxílio na geração de runtime nos faz acreditar que o Packager seria uma verdadeira automação deste processo. No entanto, tomando ciência da dificuldade deste processo podemos ver que sua completa automação não seria algo muito simples.

13.4.4.Debugando erros em runtimes

Quando um erro ocorre em um teste durante o desenvolvimento, uma janela (walkback) é aberta. Através dela o desenvolvedor poderá identificar que erro ocorreu ou executar o "programa" passo a passo.

No runtime torna-se difícil visualizar o erro ocorrido. Idéia simples e muito útil o ENVY trouxe consigo uma ferramenta que escreve em arquivo a "stack" do programa, podendo depois ser analisada pelo desenvolvedor. Podemos esclarecer aqui que quando mencionamos "stack" não estamos falando de números hexadecimais de difícil análise, estamos falando de nomes de métodos e mensagens de erros.

será que já não foi criado um termo substituto em português.

a pilha ('stack')

14. Um Modelo padrão para desenvolvimento de Software

Um software desenvolvido sob uma filosofia de programação orientada a objetos é todo modelado em **classes** de objetos que por sua vez são dispostos em uma árvore hierárquica. Estas classes são agrupadas em **aplicações** dentro do ENVY (ambiente para desenvolvimento cooperativo). Uma análise de um mesmo sistema implementado por vários programadores diferentes vai nos apresentar várias modelagens para um mesmo problema, caracterizando o **estilo** de cada desenvolvedor.

Quando falamos em desenvolvimento cooperativo temos que padronizar ao máximo o modelo de cada uma das partes a serem desenvolvidas pelos vários programadores, para que elas sejam facilmente entendidas e facilite a cooperação entre os componentes da equipe.

O padrão básico utilizado no desenvolvimento dos nossos protótipos (apresentado na figura abaixo) é o seguinte:

- uma aplicação reúne as classes de interface;
- um outra, geralmente chamada de "manager", agrupa as classes responsáveis pelo Kernel do software.
- uma terceira aplicação reúne as classes que implementam os protocolos dos clientes e servidores desenvolvidos.

Estes clientes e servidores são implementados sobre uma quarta aplicação chamada Small-WinSock que, por sua vez, implementa um conjunto de classes responsáveis pela conversação com a DLL WinSock.

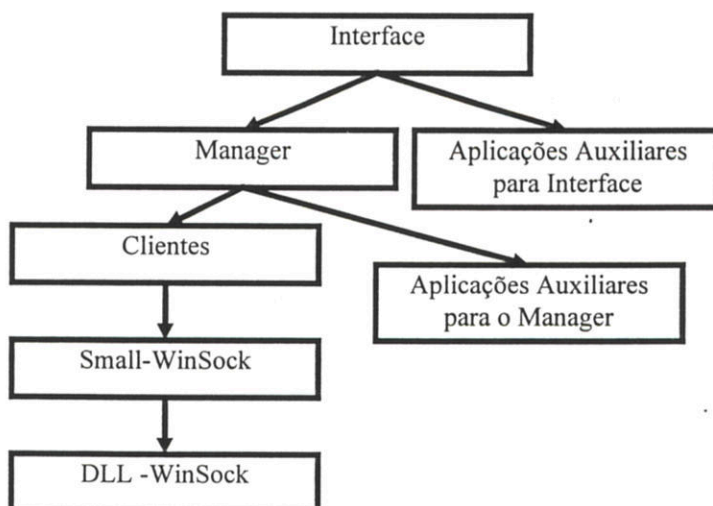


Figura 14.1

Além deste conjunto de aplicações, nós separamos em outras aplicações os elementos auxiliares de forma a aumentar a reusabilidade. Por exemplo: no decorrer do desenvolvimento de um cliente Gopher [Anklesaria 93] surgiu a necessidade de

X podemos disparar outras aplicações Windows (notpade.exe, write.exe, visualizadosⁿ de image, etc) a partir da interface do nosso cliente, a classe que implementava esta possibilidade foi separada em outra aplicação, e agora outro programador (da equipe), que venha a ter a mesma necessidade, precisa somente carregar esta aplicação em seu ambiente de trabalho e utilizá-la.

Assim,

PARTE III:
DESENVOLVIMENTO DE PROTÓTIPOS DE
REDE.

O EDUFÓRUM E O WINSOCA.

15. EduFórum

15.1. O projeto hiperNet

“Rede para nós outros”, é como o projeto hiperNet é definido pelo idealizador [Melgarejo 91]. Desenvolvido no Laboratório de Software Educacional - EDUGRAF, este projeto disponibilisa um sistema computacional a vários NETs (Núcleo de Experimentação/ Aprendizagem da Telemática) e demonstra na prática o uso de computadores na educação. A comunicação entre os usuários dos NETs se dá totalmente através de uma rede de computadores, a Rede hiperNet, que provê a infraestrutura básica através da qual irão acontecer os Fóruns.

Os Fóruns hiperNet são os debates que acontecem na Rede hiperNet, em torno de um núcleo temático. Para isto, os usuários podem utilizar o cliente EduFórum.

15.2. O EduFórum

A maioria dos aplicativos como Gopher [Anklesaria 93] e WWW se apresentam como uma grande fonte para pesquisa de dados espalhados pelos vários servidores na Internet. Existe uma necessidade de uma base de dados mais interativa, onde várias pessoas possam ler e alterar os dados com políticas estabelecidas. Podemos ver o EduFórum como um protótipo para uma base de dados deste tipo.

Baseia-se em uma base de dados Gopher onde um servidor EduFórum implementa a idéia de vários murais onde os usuários podem, através de um cliente, colocar novas informações e descrevê-las.

O servidor foi desenvolvido para UNIX e pode esperar a conexão de vários clientes. Implementa um protocolo que possui comandos para criar diretório, mudar de diretório, identificar usuário, apagar arquivo, colocar arquivo, alterar a descrição de diretório ou arquivo etc. As partes de pesquisa à base de dados são repassadas a um servidor Gopher.

Através do cliente EduFórum pode-se: enviar mensagens para os usuários hiperNet; navegar na base de dados e editá-la, explorar um ambiente de realidade virtual modo texto [Hoepers 96] onde os usuários podem, inclusive, conversar com os outros usuários conectados ao EduFórum. Tudo integrado em um ambiente de janelas.

15.3. Implementação do cliente EduFórum

Além da utilização das classes do Smalltalk e EHS, o EduFórum pode ser subdividido em três camadas (figura 15.1):

- a camada de **rede** (Small-WinSock) já apresentada;

- a camada **principal** onde é implementada toda a parte dos clientes Gopher, EduFórum e hiperMOO;

- a camada de **interface**, o mais separada possível da camada abaixo dela.

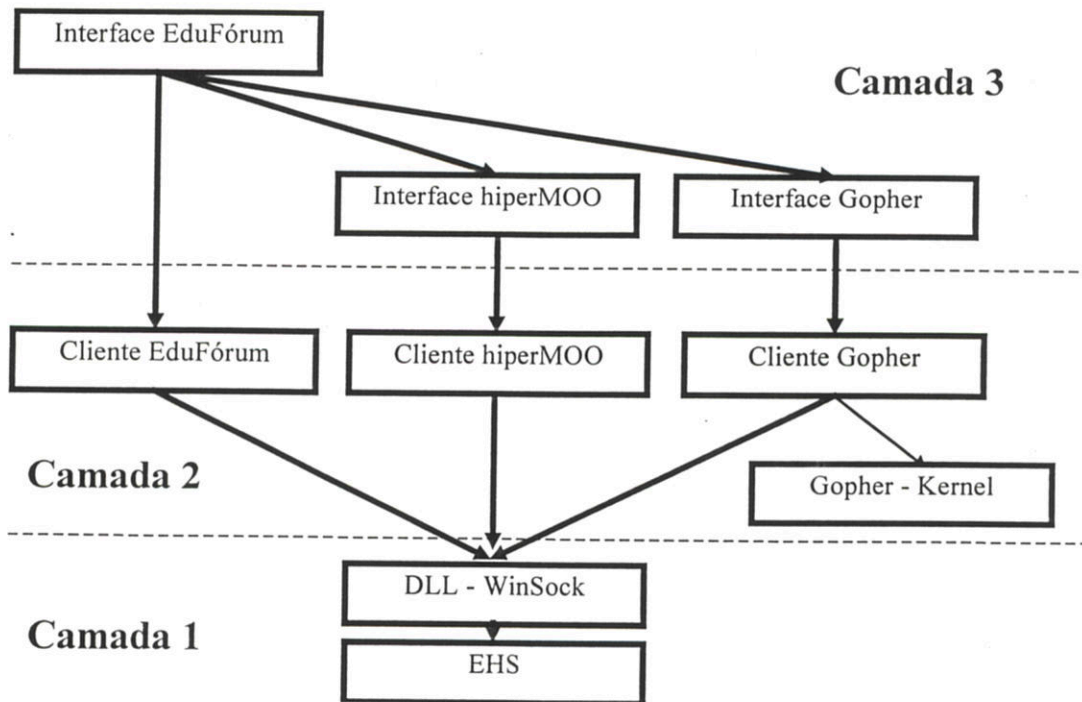


Figura 15.2

A segunda camada é composta de quatro aplicações: Gopher-Kernel, Cliente Gopher, Cliente hiperMOO e Cliente EduFórum.

O Gopher-Kernel implementa um conjunto de classes que representam os tipos Gopher (GopherDirectory, GopherBinaryFile, GopherImage, GopherCSO, GopherTelnet, GopherSearch etc.) e a classe GopherMenu responsável em converter os "strings" recebidos pela rede em objetos Gopher.

O Cliente Gopher implementa o protocolo Gopher utilizando o Small-WinSock, EHS e o Gopher-Kernel. Podemos chamá-lo de "manager" do cliente Gopher.

O Cliente hiperMOO implementa o acesso a um Ambiente de Aprendizagem Baseado em Realidade Virtual Modo Texto, o servidor hiperMOO.

O Cliente EduFórum implementa a parte de edição da base de dados.

A terceira camada compoe-se das aplicações: Interface EduFórum, Interface hiperMOO, Interface Gopher.

A Interface Gopher é responsável pela interface com o usuário do software. Ele espera as interações do usuário, repassa para o Cliente Gopher e mostra as mensagens de feedback para o usuário.

O Cliente EduForum e a Interface EduForum são extensões do Cliente Gopher e da Interface GopherClientInterface. Estas implementam a comunicação com o servidor EduFórum. Aqueles comandos do servidor que poderiam ser acessados via TELNET na porta do servidor EduFórum, agora são transformados em menus e janelas.

16. WinSdca (Cliente e Servidor)

16.1. O WinSdca

O WinSdca é um software desenvolvido no Laboratório de Controle e Micro Informática - LCMI da Universidade Federal de Santa Catarina.

Trata-se de uma aplicação para controle e automação que poderá ser conectada a um processo industrial através de uma placa A/D, D/A. Esta aplicação tem uma interface desenvolvida para rodar sobre o Windows e através dela pode-se, entre outras coisas:

- alterar o tempo de amostragem do sistema;
- obter automaticamente um modelo em malha aberta do processo (ganho estático, atraso e constante de tempo);
- obter automaticamente os parâmetros de um PID, que ^{alternativas de ajuste do} pode ser utilizado como controlador do processo. O WinSdca apresenta quatro PIDs ao usuário: referência rápida, perturbação rápida, referência lenta, perturbação lenta;
- visualizar o processo através de um ploter que apresenta três curvas: ^{comportamento do} referência, saída e atuação de controle;
- alterar a referência do processo;
- alterar ^{os parâmetros} K_c , T_i , T_d de acordo com a vontade do usuário.

16.2. Um novo protótipo

O objetivo principal deste projeto era o desenvolvimento de um protótipo onde fosse possível supervisionar o processo a distância utilizando uma rede TCP/IP. Para isso, foram desenvolvidos um cliente e um servidor e foi feita uma remodelagem do software existente, desenvolvendo-o sobre uma DLL.

A DLL mantém todas as funcionalidades do WinSdca, desligando-as da interface. O servidor roda sobre a DLL, disponibilizando as funcionalidades desta a um cliente a ele conectado através da rede. Outros clientes poderão se conectar ao servidor, podendo somente visualizar o processo. ^{simultaneamente?}

Foram desenvolvidos, então, um cliente, um servidor e um protocolo de comunicação entre eles.

16.3. Os tipos de conexão

Conforme mencionado acima, o servidor pode aceitar dois tipos de conexão:

- no primeiro tipo o cliente poderá atuar sobre o servidor, ou seja, poderá alterar os parâmetros do controlador, o valor da referência, etc;
- no segundo tipo o cliente será apenas um espectador, podendo apenas visualizar o processo.

16.4. A divisão em camadas

Com o propósito de facilitar a etapa de testes, bem como a ^{de}manutenção, este protótipo foi desenvolvido em três camadas independentes conforme a figura abaixo:

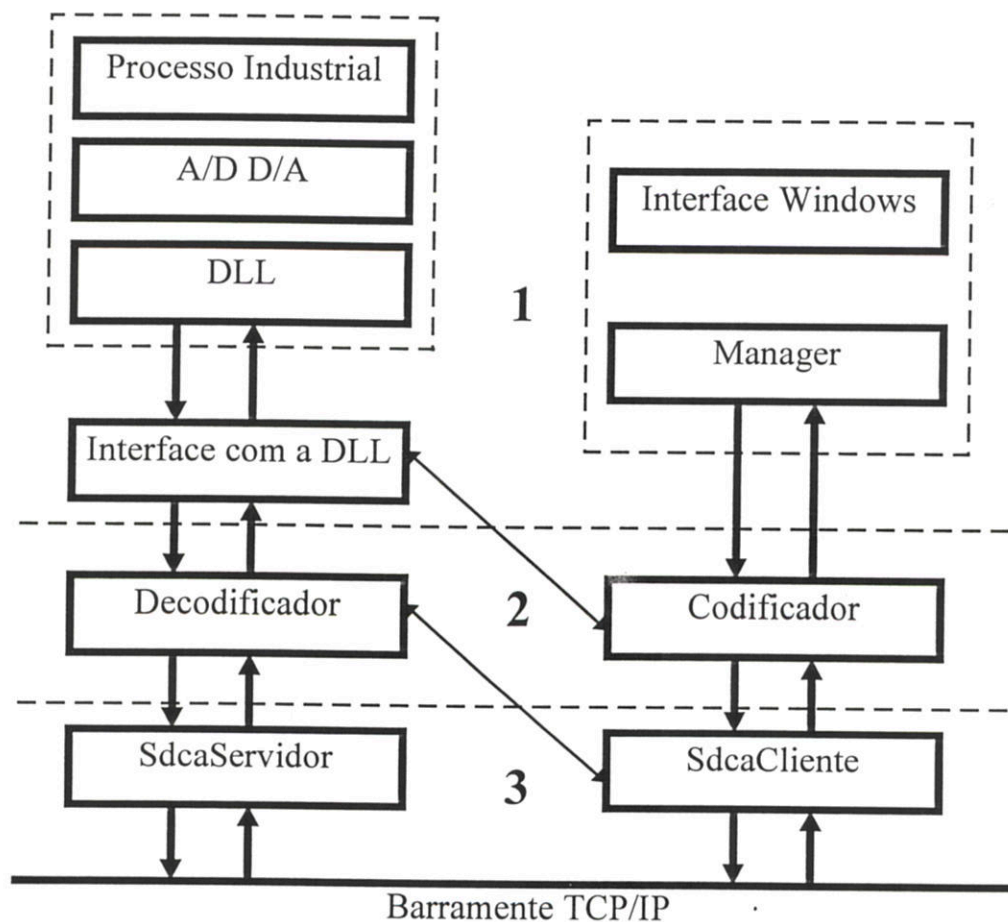


Figura 16.1

A "Interface com a DLL" é implementada por uma classe chamada "SdcaDLL" que tem a responsabilidade de converter os objetos em tipos aceitos pela DLL. Esta classe implementa um conjunto de métodos praticamente igual ao conjunto de funções oferecidos pela DLL [Cavalcante 96]. Estes métodos recebem objetos como parâmetros, transformam-os em tipos (Integer, Double, Pointer, Struct, etc), repassam estes tipos para uma função da DLL, recebem sua resposta e fazem o caminho inverso.

O "SdcaServidor" recebe um conjunto de bytes pela rede, repassa-los para o Decodificador e envia os bytes que o Decodificador lhe devolve como resposta.

O "Decodificador" decodifica um conjunto de bytes em objetos (mensagens e parâmetros) e envia estas mensagens à instancia única da classe SdcaDLL (Interface com a DLL). O protocolo de comunicação entre o SdcaServidor e o Decodificador se dá por uma única mensagem, a mensagem #launch: que recebe uma cadeia de bytes como parâmetro.

O "Codificador" implementa o mesmo conjunto de métodos implementado pela classe SdcaDLL. Este conjunto de mensagens transformam os objetos em bytes e os repassa para o "SdcaCliente".

O "SdcaCliente", por sua vez tem a função de enviar os bytes para o SdcaServidor. Assim como o decodificador, o "SdcaCliente" implementa a mensagem #launch:.

Generalizando, as manipulações feitas pelo usuário na "Interface Windows" do protótipo passam de uma camada para outra até chegarem à DLL e as respostas da DLL fazem o caminho contrario.

Como a interface entre o SdcaServidor e o Decodificador é igual a interface entre o Codificador e SdcaCliente a terceira camada poderia ser suprimida fazendo o Codificador referenciar diretamente o Decodificador.

Similarmente, a segunda camada também poderia ser suprimida fazendo o Manager referenciar a "Interface com a DLL" diretamente.

Suprimindo estas camadas tornasse possível testar somente a primeira e depois acrescentá-las uma a uma.

16.5. O protocolo de comunicação entre o Cliente e o Servidor

O quadro de bytes transmitido pela rede é definido pelo Codificador ou Decodificado e estendido pelo Cliente ou Servidor. Para manter a independência entre as camadas, todo byte acrescentado pelo Cliente deve ser suprimido pelo Servidor e vice-versa. Neste caso, para qualquer função definida pelo protocolo, a terceira camada acrescenta dois bytes no início do pacote informando o tamanho deste pacote.

A parte definida pela segunda camada especifica o nome da função que será repassada para DLL e os seus parâmetros, veja a figura abaixo.

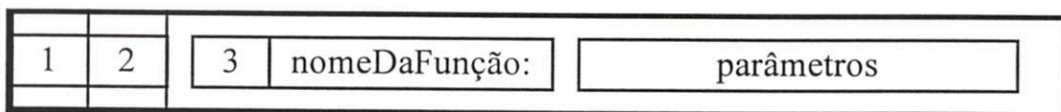



Figura 16.2

- o primeiro byte representa o byte mais significativo do número que representa o tamanho do pacote;
- o segundo byte representa o byte menos significativo;
- o terceiro representa o tamanho do nome da função.

Obs.: Todo string dentro deste pacote vai ter um byte representando o seu tamanho antecedendo-o.

Para exemplificar, vamos analisar a função 'setYr'. Esta é a função implementadas pela DLL que altera o valor da referência.

Quando o usuário altera o valor da referência na sua interface o manager envia uma mensagem para o objeto codificador que codifica esta mensagem no seguinte quadro:

- 
- o primeiro byte igual a 5;
 - um conjunto de bytes igual a: 115, 101, 116, 89, 114, representando o código ASCII de cada caracter do string "setYr";
 - um outro conjunto de dois bytes representando o novo valor de Yr (byte mais significativo e menos significativo).

O SdcaCliente acrescenta o número "8" na forma de dois bytes no início do quadro e o envia pela rede para o SdcaServer.

O SdcaServer lê dois bytes da rede, monta o número "8", (retirando a primeira parte do quadro), lê mais oito bytes e os repassa para o decodificador.

Este último transforma estes oito bytes em uma mensagem #setYr: com seu parâmetro e envia para o objeto da classe SdcaDLL.

16.6. A conexão de visualização

A cada período de tempo, independentemente do período de amostragem do processo, o servidor envia para todos os seus cliente os novos pontos amostrados pela DLL. O cliente receberá estas informações e atualizará sua interface.

Um dos clientes conectados ao servidor vai possuir duas conexões TCP/IP, os outros possuirão somente uma.

17. Conclusões e Perspectivas

O auge deste projeto é alcançado no momento da implementação dos protótipos onde se pode adquirir alguma experiência na utilização das técnicas estudadas no seu início:

- o primeiro protótipo foi desenvolvido paralelamente ao estudo de tais técnicas, portanto sua principal contribuição foi principalmente de ordem formativa. Ele vem sendo fonte de um estudo ergonômico, como parte de uma tese de doutorado e o feedback deste estudo tem nos proporcionado um grande aprendizado principalmente do desenvolvimento de sua interface (interface homem/máquina).
- o segundo protótipo, parte principal deste projeto, foi desenvolvido quando as técnicas já eram conhecidas e experimentadas, portanto, ele pode ser encarado como um teste prático. O trabalho incluiu: duas plataformas diferentes de trabalho, "C++" e "Smalltalk"; duas áreas diferentes, controle de processos e rede de computadores; e por último, dois laboratórios diferentes, EDUGRAF e LCMI.

Em ambos os protótipos tentou-se vivenciar a idéia de um desenvolvimento cooperativo:

- o EduFórum oferece ao usuário, entre outras coisas, a possibilidade de ler e enviar mails. Esta parte foi implementada por outro integrante do grupo e depois acrescentada ao EduFórum [Quarti 96], esta junção foi possível depois de um reestudo do modelo de ambas as partes. Esta cooperação foi facilitada pela utilização do ENVY (Ambiente para desenvolvimento cooperativo de Software).
- o WinSdca (Cliente/Servidor) foi desenvolvido em duas partes, uma DLL em C++ e a parte de rede em Smalltalk. Embora o WinSdca já vinha sendo desenvolvido há algum tempo no LCMI ele não era implementado sobre uma DLL. Esta re-estruturação do programa aconteceu paralelamente ao desenvolvimento do Cliente e do Servidor, as restrições para término do projeto tornaram possível confirmar que a velocidade de desenvolvimento segundo uma filosofia orientada a objetos auxiliada de bom ambiente de programação pode ser maior do que no caso contrário.

Eles ainda podem ser explorados como base de uma pesquisa:

- o EduFórum é um aplicativo para qualquer tipo de usuário sendo uma ótima ferramenta educacional que pode ser estendido com novas idéias.
- o WinSdca (Cliente e Servidor) explora uma pequena fatia dentre todas as técnicas estudadas ao longo do curso de controle e automação. Novos algoritmos de controle, de reconhecimento, controle multi-variável, entre outros, poderiam ser acrescentados a ele.

18. Bibliografia

[Anklesaria 93]

F. Anklesaria, M. McCahill, P. Lindner, D. Johnson, D. Torrey, B. Alberti, "The Internet Gopher Protocol", Request for Comments: 1436, University of Minnesota, March 1993, <http://info.internet.isi.edu:80/in-notes/rfc/files/rfc1436.txt>.

[Calvet 96]

Roberto Calvet, "Um pouco mais de OOP", Micro Sistemas, pag. 18, Ano XIX, Enter Press Editora LTDA.

[Cavalcante 96]

Carla Cavalcante,

[Comer 91]

Douglas E. Comer, "Internetworking With TCP/IP Vol I: Principles, Protocols and Architecture", Prentice-Hall, 1991.

[Dony 90]

Christophe Dony, "Exception Handling and Object-Oriented Programming: towards a synthesis.", ECOOP/OOPSLA, October 1990.

[ParkPlace 92]

"ObjectWorks/Smalltalk User's Guide", ParcPlace Systems, 1992.

[Gamma 95]

Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, "Design Patterns - Elements of Reusable Object-Oriented Software", Addison-Wesley Publishing Company, Setember 1995.

[Hall 92]

Martin Hall, MarkTowfiq, Geoff Arnold, David Treadwell, Henry Sanders, "Windows Sockets - An Open Interface for Network Programming under Microsoft Windows", November 1992, <ftp://microdyne.com/pub/winsock>.

[Hoepers 96]

Cristine Hoepers, "Ambientes de Aprendizagem Baseados em Realidade Virtual Modo Texto", Trabalho de Conclusão de Curso, Bacharelado em Ciências da Computação, Julho de 1996.

[IBM-ST 94]

International Business Machines Corporation, "IBM Smalltalk - Programmer's Reference", 1994

[LaLonde 90]

Wilf R. LaLonde, John R. Pugh, "Inside Smalltalk" Prentice Hall, 1990.

[Melgarejo 91]

L. F. B. Melgarejo, "hiperNet: um ambiente de colaboração em rede local", IX Simpósio Brasileiro de Redes de Computadores, p. 478 a 492, <http://www.hipernet.ufsc.br/hnsbrc.htm>.

[OTI-ED 96]

Objetc Technology Internacional, "ENVY/Developer: The Proven Choice for Smalltalk Application Development", <http://www.oti.com/product/ed.htm>, 1996.

[Pereira 95]

Eduardo José Pereira, et al, "Small-WinSock: Uma interface TCP/IP implementada em VisualWorks Smalltalk", V Seminário de Iniciação Científica de Santa Catarina, Agosto 1995.

[Quarti 96]

Silenio Sullivan Quarti, "EduCorreio - Um aplicativo para gerenciamento de mensagens", EDUGRAF-INE-UFSC, Julho 1996.

[Rumbaugh 91]

James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, William Lorenzen, "Modelagem e Projetos Baseados em Objetos", Editora Campus, 1994.