9,7 (nove e sete)

ESCOLA POLITECNICA DA UNIVESRIDADE DE SÃO PAULO
PMR - DEPARTAMENTO DA ENGENHARIA MECATRÔNICA

Sistema de CAD para a Web

Projeto de Formatura

Autor: Nelson Vogel

Orientador: Prof. Dr. Marcos de Sales Guerra Tsuzuki

Coordenador da disciplina: Prof.Dr. Lucas Moscato

São Paulo - 2001

RESUMO

Os sistemas de computação com a arquitetura de clientes finos ("thin client"), também conhecidos como arquitetura em três camadas, estão se propagando cada vez mais. Isso pelas suas características de facilitar a manutenção dos sistemas, tanto em hardware quanto em software. Se houver a necessidade de aumentarmos a capacidade de processamento do sistema, será suficiente (em primeira instância) apenas aumentar a capacidade apenas da máquina servidora. Cada vez que surgir uma nova versão melhorada, ela pode estar disponível apenas na máquina servidora. Estas características também são desejáveis para os sistemas CAD.

Neste trabalho alteramos a arquitetura de um Modelador de Sólidos B-Rep para permitir sua execução em um ambiente três camadas. Para isto foi utilizado o modelo ASP ("Application Service Provider") que está sendo altamente difundido. Este modelo permite que um software seja alugado ao invés de ser vendido. O resultado é um programa CAD que roda em um servidor, podendo ser acessado (utilizado) a partir de um browser via uma rede local (LAN) ou a Internet.

A aplicação servidora inclui um Modelador de Sólidos B-Rep desenvolvido em linguagem C++. A aplicação cliente possui uma interface com o usuário e está implementada em Java, sendo que para exibir os objetos tridimensionalmente utiliza o OpenGL. O Cliente possui um mapeamento da estrutura de dados suficiente para realizar seleções e exibir o sólido em wireframe (e em shading, no futuro). Além da linguagem Java, outras tecnologias abertas tais como o XML e o socket (TCP/IP) estão sendo utilizadas, garantindo a escalabilidade e portabilidade para diversas plataformas.

O uso de uma arquitetura aberta é uma tendência mundial no mundo da Tecnologia da Informação, o que nos leva a crer que será também uma tendência para os programas CAD no futuro próximo.

ABSTRACT

The thin client architecture computer systems, also known as three-layer architecture, are being used more and more over the years. This is happening because of its features that facilitate the systems maintenance, in both hardware and software. If there is a need to increase the system processment capacity, it is enough (in first instance) to only increase the capacity of the server machine. Each new improved version should be available only in the server machine. These features are also desirable in the CAD systems.

In this work we've modified a B-Rep Solid Modeller architecture to allow its execution in a three-layer environment. The ASP (Application Service Provider) model, which is being highly spread out, was used to achieve that. This model allows a software to be rented instead of being sold. The result is a CAD program that wheel in a server, and which can be accessed (used) from browser in a local area network (LAN) or the Internet.

The server application includes a B-Rep Solid Modeller, developed in C++ programming language. The client side application will have a user interface and is implemented in Java. To show objects three-dimensionally it uses the OpenGL. The client will have a data structure mapping, enough to carry through selections and to show the solid in wireframe (and with shading, in the future). Beyond the open Java language, other technologies such as the XML and socket (TCP/IP) are being used, guaranteeing the scalability and portability to many platforms.

The use of an open architecture is a world-wide trend in the Information Technology business, which leads us to believe that this will be also a trend for CAD applications in the near future.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTAGENS

1 INTRODUÇÃO

O trabalho aqui apresentado é um software CAD, utilizando a modelagem de sólidos B-Rep, que utiliza uma arquitetura distribuída. Com isso podemos dividir o programa em duas partes: o servidor e o cliente.

O servidor que está sendo desenvolvido pelo mestrando Marcelo Shimada contém a maior parte da inteligência do sistema, desde a estrutura de dados necessária para representar os sólidos até os operadores de Euler e booleanos.

O cliente, que é o foco desse trabalho, contém apenas as informações e inteligência necessárias para a visualização e manipulação dos sólidos na tela no micro do usuário, utilizando para tanto as bibliotecas do OpenGL. Também é foco desse trabalho definir a forma com que o cliente e o servidor trocam informações entre si, através de uma arquitetura simples, aberta e que está preparada para aceitar novas features que venham a surgir no programa.

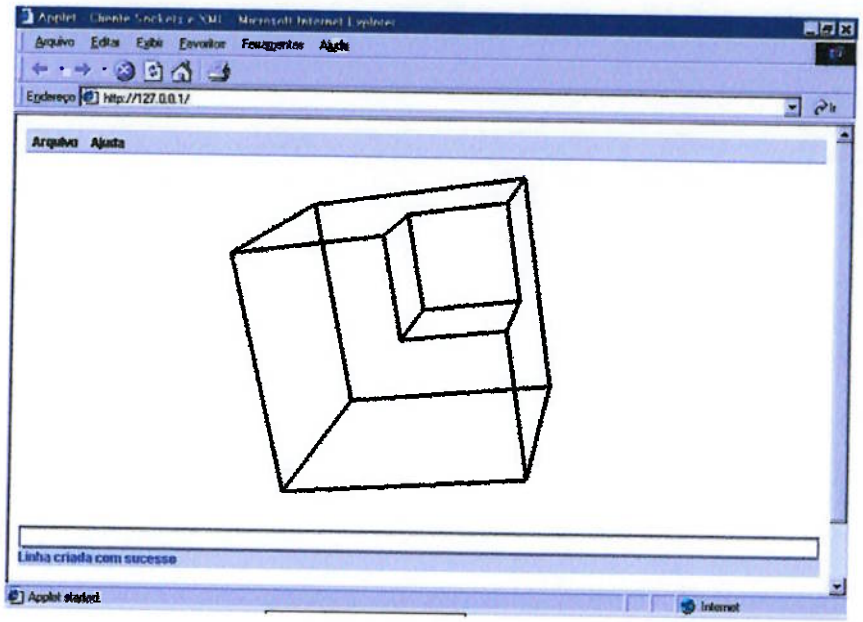

Figura 1.1 – Interface do programa cliente.

2 ARQUITETURA DA SOLUÇÃO

Foram estudadas algumas arquiteturas para a solução, todas utilizando o sistema CAD via internet, através do modelo cliente-servidor.

Para o lado cliente da aplicação havia duas possibilidades: ser instalado na máquina de cada usuário, ou ser executado através de um browser. A segunda opção foi escolhida, evitando a instalação do software e facilitando a manutenção do sistema.

O papel que cada uma das partes envolvidas (cliente e servidor) deveria desempenhar gerou uma grande discussão. Podíamos optar pelo cliente ter um mapeamento de toda a estrutura dos sólidos, ou então apenas uma interface suficiente para exibir os sólidos na tela do usuário. Na primeira opção, temos uma maior flexibilidade de fazer alterações no sólido sem a necessidade de utilizar a inteligência do servidor, otimizando o fluxo de comunicação. Por outro lado, é preciso se preocupar com a escalabilidade do sistema, onde novas funções que forem desenvolvidas no servidor devem estar disponíveis na Web, sem a necessidade de alterar o programa cliente. A divisão ideal é detalhada mais adiante.

Como forma de comunicação pensou-se inicialmente em utilizar um sistema baseado no CORBA (*Common Object Request Broker Architecture*). O CORBA define as interfaces que cada programa oferece, ou seja, funções que podem ser executadas remotamente a partir de outros aplicativos. Com esse protocolo teríamos dificuldade para manter a escalabilidade do sistema, uma vez que para novas funções criadas no USPDesigner teríamos que atualizar tanto o programa servidor como o cliente, adaptando-os à nova interface. Assim, optou-se por uma forma mais simples – o socket – para enviar os comandos necessários para o servidor. No retorno, as informações são enviadas para os clientes através de um arquivo texto, que está em conformidade com o padrão XML. Com isso simplifica-se tanto a comunicação com vários usuários simultâneos como também a manipulação dos dados, e ainda permite a interoperabilidade com outros sistemas que trabalham com o XML.

3 AMBIENTE

3.1 Applet

Os applets, também conhecidos como miniaplicativos, são programas em Java que são executados a partir de um browser. Ao invés de chamar o programa a partir de uma linha de comando, é possível colocar no browser um endereço de uma página Web, que por sua vez chama um ou mais applets, executando-os como se fossem programas normais. Esse processo elimina a necessidade de instalação do programa nos micros dos usuários, deixando apenas os arquivos na máquina servidora. Para esclarecer isso, o seguinte código html chama o applet "cliente.class" (arquivos com extensão ".class" são programas em Java):

```
<HTML>
<HEAD>
<META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=windows-1252">
<TITLE>
Arquivo HTML chamando o Applet - Cliente.class
</TITLE>
</HEAD>
<BODY>
O Applet aparecera logo abaixo:<BR>
<APPLET
  CODEBASE = "."
  CODE     = "Cliente.class"
  NAME     = "Cliente"
  WIDTH    = 100%
  HEIGHT   = 100%
>
</APPLET>
</BODY>
</HTML>
```

Listagem 3.1 – Arquivo html chamando o applet "Cliente.class".

Para utilizar bibliotecas de extensão ao Java, como é o caso do J3D, é preciso converter esse arquivo html para utilizar o Java Plug-in no lugar do JVM (Java Virtual Machine) padrão do browser. O Java Plug-in permite o aproveitamento de funcionalidades que vão além da linguagem Java pura, como é o caso do OpenGL. O arquivo html resultante pode ser utilizado em diferentes browsers e plataformas:

```
<HTML>
 <HEAD>
  <META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=windows-1252">
  <TITLE>
   Applet - Cliente Sockets e XML
  </TITLE>
 </HEAD>

 <BODY>

<!--"CONVERTED_APPLET"-->
<!-- CONVERTER VERSION 1.3 -->
<SCRIPT LANGUAGE="JavaScript"><!--
    var _info = navigator.userAgent; var _ns = false;
    var _ie = (_info.indexOf("MSIE") > 0 && _info.indexOf("Win") > 0 &&
_info.indexOf("Windows 3.1") < 0);
//--></SCRIPT>
<COMMENT><SCRIPT LANGUAGE="JavaScript1.1"><!--
    var _ns = (navigator.appName.indexOf("Netscape") >= 0 && ((_info.indexOf("Win") > 0
&& _info.indexOf("Win16") < 0 &&
java.lang.System.getProperty("os.version").indexOf("3.5") < 0) || (_info.indexOf("Sun") >
0) || (_info.indexOf("Linux") > 0)));
//--></SCRIPT></COMMENT>

<SCRIPT LANGUAGE="JavaScript"><!--
    if (_ie == true) document.writeln('<OBJECT classid="clsid:8AD9C840-044E-11D1-B3E9-
00805F499D93" WIDTH = 100% HEIGHT = 100%
codebase="http://java.sun.com/products/plugin/1.3/jinstall-13-
win32.cab#Version=1,3,0,0"><NOEMBED><XMP>');
    else if (_ns == true) document.writeln('<EMBED type="application/x-java-
applet;version=1.3" CODE = "Cliente.class" WIDTH = 100% HEIGHT = 100% scriptable=false
pluginspage="http://java.sun.com/products/plugin/1.3/plugin-
install.html"><NOEMBED><XMP>');
//--></SCRIPT>
<APPLET CODE = "Cliente.class" WIDTH = 100% HEIGHT = 100%></XMP>
<PARAM NAME = CODE VALUE = "Cliente.class" >

<PARAM NAME="type" VALUE="application/x-java-applet;version=1.3">
<PARAM NAME="scriptable" VALUE="false">

</APPLET>

</NOEMBED></EMBED></OBJECT>

<!--
<APPLET CODE = "Cliente.class" WIDTH = 100% HEIGHT = 100%>

</APPLET>
-->
<!--"END_CONVERTED_APPLET"-->

</BODY>
</HTML>
```

Listagem 3.2 – Arquivo html convertido para suportar o Java Plug-in.

Dessa forma é possível criar páginas na internet capazes de fazer praticamente qualquer coisa que um programa instalado no micro faz. Isso ainda com a vantagem de que o aplicativo pode ser executado de diversas máquinas simultaneamente, em diversas plataformas, bastando para isso apenas um browser com suporte ao Java Plug-in.

Algumas restrições ou desvantagens dos applets que podemos citar:

- ✓ Programas complexos/grandes demoram a carregar, dependendo da conexão do usuário com a Internet.
- ✓ Existe uma limitação de segurança que restringe o acesso ao disco local. Essa limitação é imposta pelo browser utilizado (Netscape, IExplorer, Opera, etc.).
- ✓ O applet pode se comunicar apenas com o host de onde está sendo executado, por motivos de segurança. Com isso o servidor de aplicativo (USPDesigner em C++) deve estar sendo executado na mesma máquina do servidor Web.

3.2 OpenGL (J3D)

O OpenGL é uma biblioteca de funções para manipulação gráfica. Essa biblioteca oferece uma facilidade para a programação além de um alto desempenho, que pode ainda ser aumentado quando utilizando hardware compatível com o OpenGL.

Apesar de ser multi-plataforma, o OpenGL é uma linguagem compilada e, portanto, depende do sistema operacional. Isso significa que cada sistema operacional tem sua implementação do OpenGL.

Em contrapartida, a linguagem Java se caracteriza por ser uma linguagem interpretada e multi-plataforma.

Para resolver esse paradoxo existem algumas implementações de APIs para o Java que aproveitam as bibliotecas de funções do OpenGL nativo do sistema operacional, seja ele qual for (por exemplo, o opengl32.dll do Windows). Assim, utilizando essas APIs, é possível criar um aplicativo em Java que é independente de plataforma e, ao mesmo tempo, utiliza o OpenGL que está instalado no sistema operacional (de forma transparente para o programador Java). Obviamente, é necessário que o OpenGL esteja instalado no computador. Assim, o aplicativo estará

aproveitando inclusive recursos de aceleração por hardware, quando um dispositivo desse estiver presente no computador.

No início do trabalho estava sendo utilizada a biblioteca GL4Java, da Jausoft (http://www.jausoft.com/), que se mostrou bastante interessante por ser fácil de usar e oferecer um bom desempenho. No decorrer do trabalho decidiu-se por usar o J3D (http://java.sun.com/products/java-media/3D), que é uma implementação da mesma natureza do GL4Java, porém feita pela Sun. Com isso supõem-se que a implementação terá maior escalabilidade, uma vez que o produto está sendo feito por uma empresa de grande porte, criadora do Java. Além disso, o J3D oferece uma maior facilidade para programação e uma documentação mais completa.

3.3 *Aspectos de Segurança*

A linguagem Java oferece ao programador e ao usuário ferramentas para permitir uma alta segurança nos aplicativos evitando, por exemplo, a proliferação de vírus.

A primeira ferramenta disponível é o *Security Manager*. É nele onde se configura, dentro do código fonte de um programa, quais são as permissões desse software. Quando utilizamos um applet a partir de um browser não é possível redefinir as regras de segurança do *Security Manager* uma vez que o applet utiliza a regra do browser que o está executando.

Além disso, cada usuário pode, em seu micro, definir as regras do JVM (*Java Virtual Machine*). Com isso, o JVM não executa nada que não seja permitido nas definições que são configuradas através do utilitário *policytool* ou diretamente no arquivo `java.policy`. Por exemplo, o JVM não permite, em sua configuração padrão, que sejam carregadas bibliotecas nativas do sistema operacional durante a execução de um programa. Portanto, para viabilizar o uso do OpenGL, cada usuário deve alterar a permissão de seu micro para permitir o uso das bibliotecas opengl32.dll, J3D.dll, j3daudio.dll ou equivalentes.

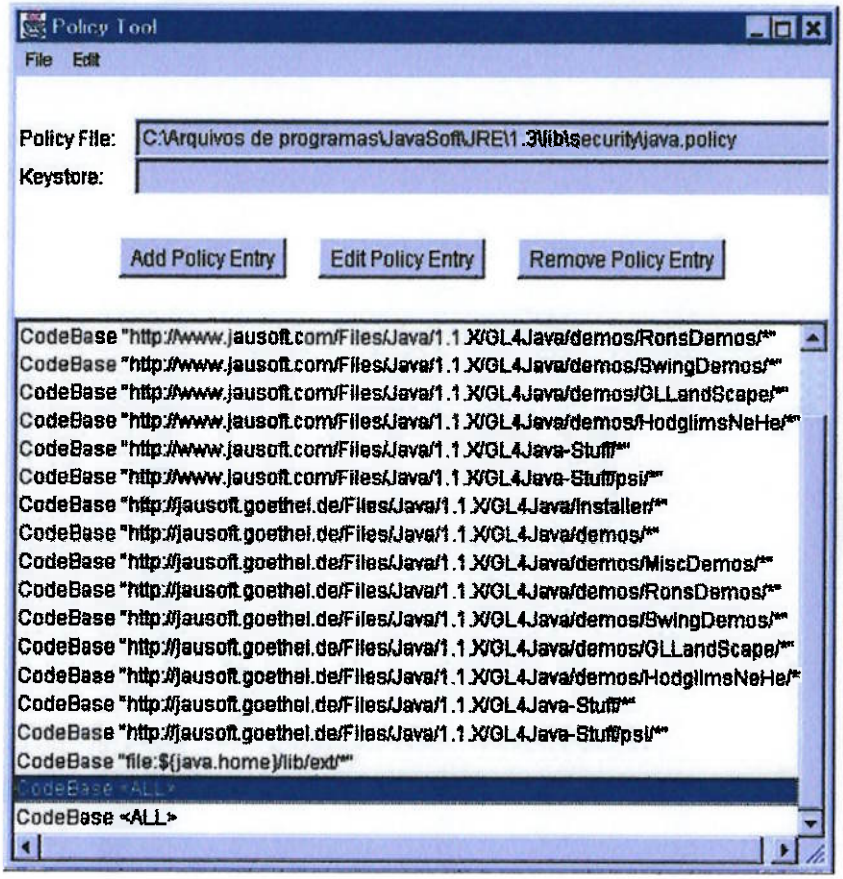

Figura 3.1 – Utilitário policytool.

3.4 *Cliente-Servidor*

Nos programas cliente-servidor é preciso definir qual é o papel de cada uma das partes e, em função disso, definir a forma de comunicação.

Antigamente sistemas multi-usuários eram baseados em *main-frames*, máquinas UNIX ou semelhantes, onde diversos terminais emulavam as telas do servidor, no qual ocorria todo o processamento. Isso era (e ainda é) muito utilizado em sistemas baseados em caracteres, nos quais não há o aproveitamento dos recursos gráficos hoje disponíveis em qualquer PC.

Existe uma tecnologia recente que permite o uso de emulação de programas baseados em janelas (windows). Essa tecnologia, o MetaFrame, da empresa Citrix, (ainda) não se tornou tão popular, devido ao seu custo e o uso de tecnologia proprietária.

Por isso os sistemas cliente-servidor correspondem hoje a maior parte dos aplicativos sendo criados. Neles, parte do processamento passa a ocorrer na máquina cliente, aproveitando dessa maneira o poder de processamento da mesma. Isso viabilizou a proliferação de uso dos sistemas gráficos distribuídos.

O browser, que surgiu com a expansão da Internet, é um sistema que tem ao mesmo tempo a flexibilidade de um emulador e a escalabilidade de um sistema cliente-sevidor. A flexibilidade do browser existe uma vez que ele surgiu para acessar diferentes sistemas remotos no mundo cibernético. Sua escalabilidade existe porque a Internet cresceu junto com o surgimento e aumento de uso dos sistemas operacionais gráficos, dentre os quais se destaca o Windows da Microsoft. Assim, os browsers tiveram que se adaptar aos recursos gráficos e figuras, que passaram a ser uma exigência dos usuários dos microcomputadores.

No início, a função do browser era apenas de exibir páginas html, uma linguagem que se tornou padrão na Internet. Com o tempo percebeu-se o poder da rede quando criadas páginas dinâmicas, e então surgiram tecnologias linguagens de programação para gerar páginas dinâmicas (ASP, Java Servlet, Cold Fusion, etc.) e recursos que foram agregando funcionalidades ao browser, dentre as podemos citar o Flash e o Java. Esses *Plug-ins* tiveram um importante papel na sobrevivência e consolidação do browser como a ferramenta padrão para a Internet.

Nesse trabalho estamos utilizando o Java Plug-in para transformar o browser em uma ferramenta CAD.

3.5 *Fluxograma da Comunicação*

Um dos grandes desafios nesse projeto é manter o fluxo de informações eficiente e coerente, e ainda permitir que os vários usuários trabalhem simultaneamente.

Existem duas possibilidades para o sistema CAD multi-usuário. Na primeira opção, cada usuário trabalha de forma independente dos outros usuários. Ou seja, para *n* usuários teríamos *n* estruturas de sólidos independentes. No segundo modo, todos trabalham em uma única estrutura de dados, portanto para *n* usuários existe apenas uma estrutura de dados. Optamos nesse projeto por usar a segunda forma, incentivando o trabalho em grupo.

Quando o usuário envia um comando, o servidor precisa processar o comando e avisar a todos os usuários de que houve uma mudança na estrutura de dados. Para facilitar a transferência da nova estrutura de dados, a mesma é colocada em um arquivo XML, e esse arquivo será lido por cada um dos usuários. Cabe ao servidor notificar todos os clientes (usuários) de que o arquivo com a nova estrutura está disponível. Enquanto processa um determinado comando, o servidor precisa também "bloquear" as linhas de comando para que um segundo usuário não envie um novo comando antes do primeiro ser processado.

Analisando a situação do ponto de vista do programa cliente é possível notar que existem duas situações onde haverá comunicação. A primeira é quando o próprio usuário envia um comando, ou seja, a iniciativa de comunicação parte do cliente. A segunda situação surge quando um outro usuário envia um comando, e como conseqüência o servidor precisa notificar todos os clientes. Assim, a iniciativa de comunicação partiu do servidor, se analisarmos olhando o ponto de vista de um dos clientes (usuários) que não enviou o comando.

Utilizamos as conexões socket (TPC/IP) com um protocolo proprietário e bem simples para ambas as situações.

O servidor tem uma porta socket (servidora) aguardando as conexões dos clientes, por onde serão enviados os comandos. Cada cliente cria uma conexão

(cliente) para enviar comandos, conexão através da qual também recebe uma resposta sobre o sucesso no processamento do comando enviado.

Além disso, uma segunda porta (servidora) é criada em cada cliente, porta através da qual o servidor se comunica com os clientes bloqueando a linha de comando, notificando alterações na estrutura de dados, e qualquer outra notificação que seja necessária e que se enquadre na segunda situação discutida acima, onde a iniciativa de comunicação parta do servidor.

Após receberem os sinais, cada um dos clientes busca o arquivo XML que foi gerado pelo servidor e que contém todos os dados necessários para a visualização dos objetos na tela.

O fluxograma a seguir ilustra essa comunicação:

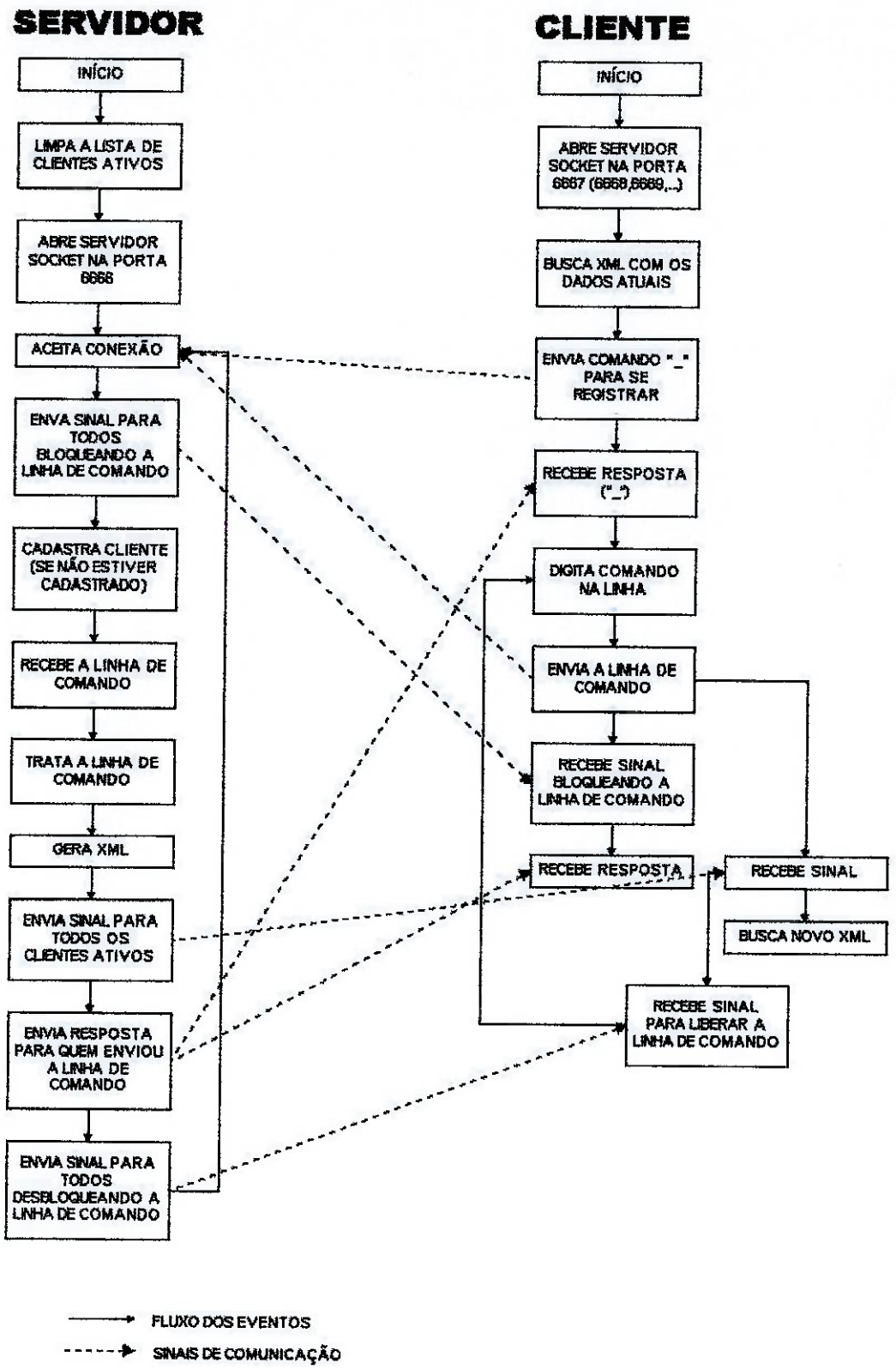

Figura 3.2 - Fluxo de comunicação entre o cliente e o servidor.

Para que o servidor saiba quais são os clientes que ele deve notificar, é feito um controle de todos os clientes que estão ativos, ou seja, de todos os usuários que estão usando o sistema a cada momento. A cada comunicação realizada, o usuário (cliente) envia para o servidor a informação do número da porta socket em que está aguardando as devidas sinalizações.

3.6 *Threads*

Enquanto que um usuário está visualizando um sólido, outro usuário pode enviar um comando que deverá alterar a estrutura de sólidos que o primeiro usuário está vendo. Assim, o programa deve estar preparado para realizar tarefas de naturezas totalmente diferentes de forma independente. Em outras palavras, o fato do usuário estar rotacionando o sólido na tela não deve ser um impedimento para que a estrutura de dados seja modificada. Com isso, temos a necessidade de realizar diferentes tarefas simultâneas independentemente. Essas filas de execução de tarefas chamam-se *threads*.

Em geral, em uma linguagem de programação, o processador executa linha a linha os comandos do programa. Para fazer duas tarefas distintas simultaneamente, é preciso utilizar o recurso de *threads*. As *threads* são filas de instrução que o processador executa em paralelo, às vezes independentemente uma da outra, em um sistema operacional multi-tarefa (também chamado de *multi-thread*), tal como o Windows, UNIX, Linux, etc.

O melhor exemplo para essa necessidade é quando um programa está aguardando uma entrada de dados por parte do usuário. No código, isso se traduz como um dos comandos a seguir (na linguagem C++):

```
cin >> s;
```

ou

```
scanf("%s",&s);
```

Sabemos que quando o processador encontra um desses comandos, ele aguarda uma entrada do usuário e só após isso prossegue nas linhas de código seguintes.

Muitas vezes queremos que o programa continue sua execução normal (cálculos, exibições na tela, etc.) e ao mesmo tempo fique aguardando a entrada de algum comando por parte do usuário. Quando isso acontecer, pode haver alguma mudança na execução do programa.

Em nosso caso, o uso de múltiplas *threads* nos permite resolver as seguintes situações:

- ✓ No programa cliente, é preciso exibir o sólido, enviar comandos e receber sinais de alteração na estrutura de dados, cada tarefa independente das outras.

- ✓ No servidor, é preciso que o programa continue executando normalmente, podendo inclusive utilizar o servidor como um programa *stand-alone*, da mesma forma que o USPDesigner era utilizado até agora, independente da nova interface via Web, e ainda aceitar os comandos dos usuários via Web.

Para explicar melhor cada uma dessas tarefas, estão detalhados a seguir os programas do lado do servidor e do cliente.

3.7 Servidor

Na solução aqui apresentada o servidor é o USPDesigner, que já estava (e ainda está) em fase de implementação. Esse software é um modelador de sólidos didático B-Rep, que está sendo traduzido para a linguagem C++ (o programa original foi escrito em C). Essa "tradução" compreende na remodelagem do sistema para permitir um melhor desempenho do sistema utilizando o OpenGL e facilidade de implementação de novas funções, dentre as quais podemos citar o suporte a Web e o uso de álgebra intervalar. Esse trabalho está sendo feito pelo mestrando Eng. Marcelo Shimada.

Apesar do modelador em si não fazer parte desse trabalho, é importante entender seu funcionamento para viabilizar o uso da nova interface, uma vez que os

são sistemas que se complementam. Ainda, é preciso fazer algumas modificações para viabilizar a implementação da interface Web.

As adaptações que estão sendo feitas no USPDesigner são basicamente a geração de um arquivo XML a cada alteração na estrutura de dados, o envio de sinais de aviso para todos os clientes (usuários remotos) ativos sempre que houver alguma mudança (ou seja, a cada arquivo gerado), e ainda a aceitação de comandos via conexões socket. Esses comandos devem ser tratados de maneira igual aos comandos que o usuário deveria digitar na linha de comando do USPDesigner original. Toda a lógica de criação de sólidos através de operadores de Euler será mantida.

Como já foi dito anteriormente, a nova interface não deve interferir na execução do programa, devendo o mesmo continuar sendo usado também como um programa *stand-alone*. Para isso cria-se uma fila de execução (*thread*) separada da fila principal, especialmente para aceitar a nova interface Web.

4 PROGRAMA CLIENTE

Esse tópico deveria ser um subtópico do item anterior, mas como o foco do trabalho está no programa cliente, ele merece um tópico à parte e será detalhado aqui.

O programa cliente é um programa que deve oferecer uma interface adequada para que o usuário possa utilizar o sistema com certa facilidade. Basicamente essa interface consiste em um mostrar o sólido na tela, além de ter uma linha de comando onde o usuário deverá digitar os comandos que serão processados pelo servidor.

Para o desenvolvimento do programa cliente está sendo utilizado o JBuilder4 Foundation, que pode ser obtido gratuitamente na Internet (http://community.borland.com/). Ainda, como principal fonte de informações sobre a linguagem Java, está sendo utilizado o tutorial fornecido pela Sun, que explica em detalhes e até exemplifica os principais componentes da linguagem (http://java.sun.com/).

4.1 Classes

Uma vez que o Java é uma linguagem orientada a objetos, foram criadas algumas classes representando as entidades necessárias para a execução do programa. Algumas classes não representam exatamente um objeto e foram criadas com a finalidade de organizar as funções que o programa deve exercer. A seguir está um breve descritivo de cada classe:

Cliente: Essa é a classe que cria a janela principal do programa, além de instanciar e chamar funções das outras classes.

ServidorSocket: Essa classe cria o servidor socket no programa cliente. Esse servidor já foi explicado nos tópicos anteriores e é utilizado para que o cliente receba notificações sobre a mudança da estrutura de sólidos (para que busque o arquivo XML com a nova estrutura de dados) e outros comandos tais como bloqueio e desbloqueio da linha de comando.

ClienteSocket: Essa classe é instanciada cada vez que o usuário digita um comando. A classe cria uma conexão com o servidor e envia a linha de comando, recebendo um retorno sobre o sucesso/insucesso da operação.

ClienteXML: Responsável por buscar o arquivo XML e gerar, a partir desse, a nova estrutura que será mostrada e manipulada na tela.

Propriedades: Abre uma caixa de diálogo com as opções de visualização do sólido. Por enquanto existe apenas a opção de Antialiasing, mas novas funções deverão ser adicionadas em breve, como a visualização em Wireframe ou Shading.

Ajuda: Abre uma caixa de diálogo com textos dando um auxilio ao usuário como, por exemplo, o uso do mouse para mover e rotacionar objetos.

Com essas classes foi criado o programa de forma que fosse bastante simples sua manutenção e a implementação de futuras melhorias.

4.2 Técnicas de Programação

Esse tópico tem como objetivo explicar as técnicas de programação empregadas nesse projeto, facilitando a compreensão do funcionamento do sistema e da leitura do código fonte do programa (ANEXO I).

Na classe principal foram utilizadas variáveis estáticas, que são alteradas a partir de outras classes. Isso foi feito para permitir a alteração da estrutura de sólidos (que por enquanto nada mais é do que uma lista de arestas), limpeza da linha de comando e outras alterações necessárias. Com a utilização de vários threads seria necessário configurar aqueles que alteram a estrutura de dados (ClienteXML, que é chamado a partir do ServidorSocket, Propriedades, etc.) para notificar aquele thread que mostra os objetos na tela de que houve tal mudança. O uso de variáveis estáticas evita essa técnica de programação que é um pouco complexa.

Tentou-se na medida do possível aproveitar os recursos facilitadores da biblioteca J3D, que por estar voltada para aplicações gráficas já tem diversas

funcionalidades implementadas. Exemplos disso são a lista de arestas (foi utilizada a lista que vem com o J3D) e as funções de mouse.

4.3 Uso do Mouse

A biblioteca do J3D oferece objetos prontos para o uso de mouse como ferramenta de manipulação dos sólidos na tela. Esses objetos são muito simples de serem implementados e seguem um padrão que é bastante fácil de usar.

Até o momento foram implementadas três funcionalidades para o mouse:

- ✓ Clicando com o botão direito e arrastando o mouse na tela o usuário pode mover os objetos, colocando-os em uma posição ideal para visualização.
- ✓ Com o botão esquerdo do mouse, novamente arrastando o mouse na tela, é possível rotacionar o objeto, podendo assim enxergar o objeto de pontos de vista diferentes.
- ✓ Segurando a tecla ALT, e com o botão esquerdo do mouse pode ser mudado o zoom, aproximando-se ou distanciando-se dos objetos que estão sendo exibidos.

Quando o programa for alterado para poder exibir, além das arestas, as faces dos objetos (vide itens posteriores), será implementada a função de seleção de faces, arestas, pontos ou outro elemento relevante. Assim como as funcionalidades descritas acima, essa feature é considerada padrão no mundo da modelagem de sólidos e por isso já está pronta na biblioteca J3D. Sendo assim, não se espera encontrar nenhuma dificuldade com relação a sua implementação.

4.4 *Arquivo XML*

O arquivo XML foi citado até agora diversas vezes, porém em nenhum momento foi explicada sua natureza.

O XML (eXtensible Markup Language) é uma meta-linguagem que já se tornou padrão e seu uso está sendo difundido no mundo cibernético a uma velocidade incrível. Isso se deve graças à simplicidade e escalabilidade dessa meta-linguagem.

Esse texto não tem como objetivo ensinar sobre XML, mas sim como ele foi utilizado nesse projeto. Para isso está colocado a seguir o cabeçalho do arquivo (DTD – Data Type Definition), que define o tipo de dados que o arquivo deve conter, com uma explicação:

Tabela 4.1 – DTD utilizado no projeto.

`<?xml version="1.0"?>`	Define a versão da meta-linguagem (atualmente 1.0)
`<!DOCTYPE TUDO[`	Inicia definindo que o conteúdo do documento é um objeto do tipo "TUDO"
`<!ELEMENT TUDO (ARESTA)*>`	Define o objeto do tipo "TUDO" é formado por várias "ARESTA"s (o asterisco significa vários)
`<!ELEMENT ARESTA (PONTO,PONTO)>`	Cada aresta é formada por dois "PONTO"s
`<!ELEMENT PONTO (X,Y,Z)>`	Cada ponto é formado por três elementos: "X", "Y" e "Z"
`<!ELEMENT X (#PCDATA)>`	Elemento "X" contém dados
`<!ELEMENT Y (#PCDATA)>`	Elemento "Y" contém dados
`<!ELEMENT Z (#PCDATA)>`	Elemento "Z" contém dados
`]>`	Fim do DTD (Data Type Definition)

Após esse cabeçalho são colocados os dados, ou seja, as informações sobre as coordenadas das arestas. O exemplo seguinte mostra um arquivo contendo uma aresta entre os pontos (0,0,0) e (1,1,1) e outra aresta entre os pontos (0,-2,3) e (2,4,2). Comentários não foram colocados por ser de fácil entendimento:

```
<?xml version="1.0"?>
<!DOCTYPE TUDO[
<!ELEMENT TUDO (ARESTA)*>
<!ELEMENT ARESTA (PONTO,PONTO)>
<!ELEMENT PONTO (X,Y,Z)>
<!ELEMENT X (#PCDATA)>
<!ELEMENT Y (#PCDATA)>
<!ELEMENT Z (#PCDATA)>
]>

<TUDO>
 <ARESTA>
  <PONTO>
   <X>0</X>
   <Y>0</Y>
   <Z>0</Z>
  </PONTO>
  <PONTO>
   <X>1</X>
   <Y>1</Y>
   <Z>1</Z>
  </PONTO>
 </ARESTA>
 <ARESTA>
  <PONTO>
   <Z>3</Z>
   <X>0</X>
   <Y>-2</Y>
  </PONTO>
  <PONTO>
   <Z>2</Z>
   <X>2</X>
   <Y>4</Y>
  </PONTO>
 </ARESTA>
</TUDO>
```

Listagem 4.1 – Exemplo de arquivo XML.

Esse arquivo é o que está sendo utilizado até o momento.

É fácil notar que a única entidade definida é a aresta, não estando presentes no arquivo as definições dos sólidos, faces, e outros objetos que fazem parte da estrutura B-Rep que foi sugerida por Mantyla e que está sendo utilizada no USPDesigner.

Para o desenvolvimento de novas funcionalidades, poderá surgir a necessidade de se fazer alterações de maneira que o programa cliente entenda o que é uma face, por exemplo. Para isso deverá ser criada uma nova classe para cada nível de entidade a ser definido (sólido, face, etc.) que terá as propriedades da mesma.

Uma das funcionalidades que podem exigir tal alteração é a possibilidade de exibir os sólidos em shading, missão que só pode ser feita se forem definidas as faces no arquivo XML.

4.5 Antialiasing

Por ser de implementação bastante simples quando utilizado o OpenGL, foi colocada a opção de antialiasing para a exibição dos objetos na tela.

Com essa opção obtém-se figuras mais nítidas, sem as discretizações causadas pela resolução limitada da tela. Em contrapartida, o uso desse recurso acarreta em uma queda de desempenho do sistema. Nos testes realizados até o momento essa variação de desempenho foi insignificante.

A figura seguinte mostra a melhoria atingida quando utilizado o antialiasing.

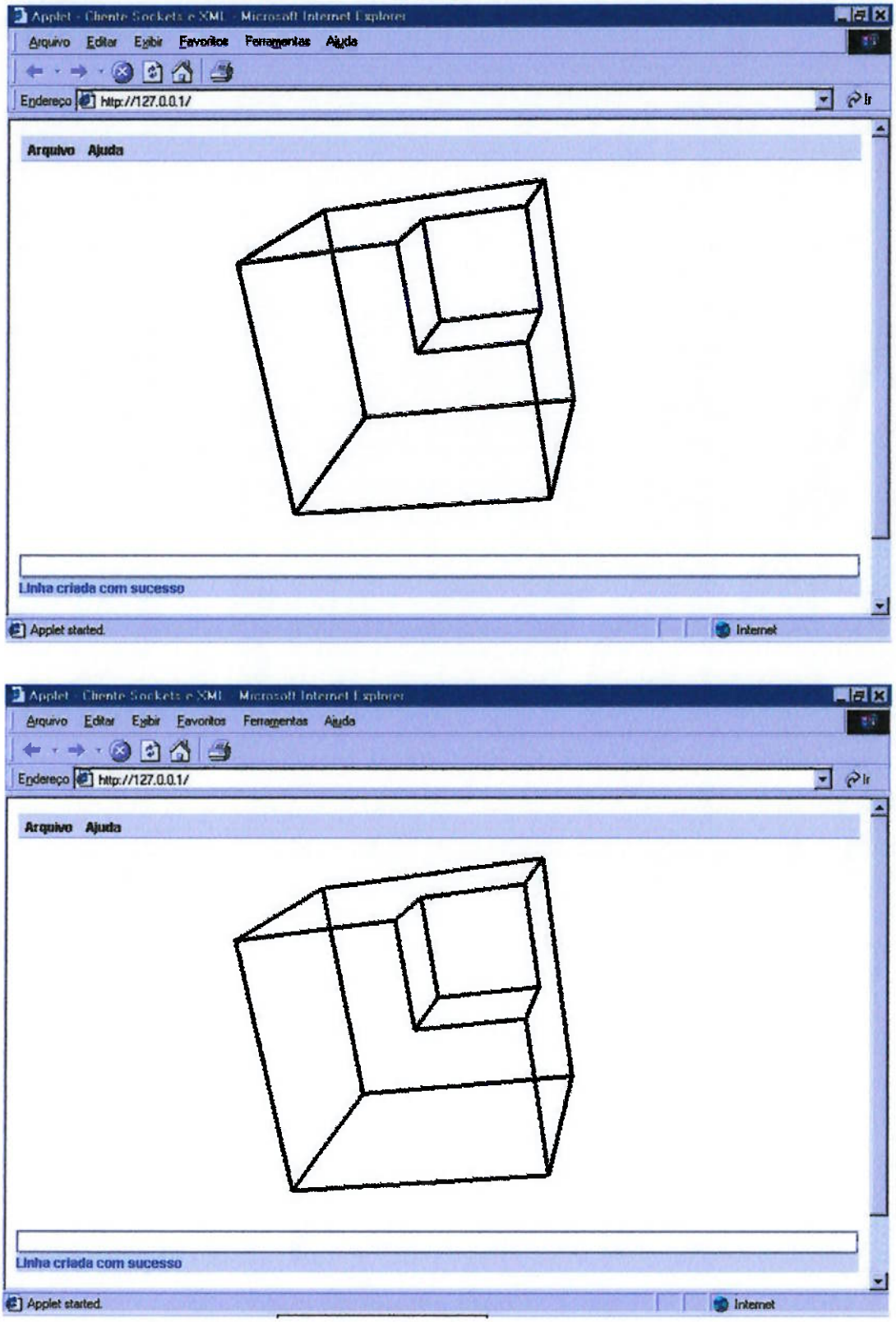

Figura 4.1 – Sólido exibido com antialiasing (superior) e sem antialiasing (inferior).

4.6 Melhorias Futuras

O tempo dedicado até agora nesse trabalho foi limitado. Por isso não foi possível criar todas as funcionalidades que gostaríamos, optando-se ao invés disse por deixar o sistema na forma atual bastante estável.

Os principais itens da lista do "gostaria", ou seja, daquelas funções que gostaríamos de ter implementado e que não o foram por falta de tempo estão explicadas aqui.

Visualização em Shading

Para que possamos visualizar os objetos em shading, é preciso definir uma estrutura de dados compatível. A estrutura de arestas serve para mostrarmos a estrutura wireframe. Para que possamos optar pelo shading é necessário que tenhamos, no mínimo, uma estrutura de faces definida no lado do cliente.

Para implementar essa função é preciso alterar o programa cliente para que passe a conter uma estrutura de dados mais completa. Essa estrutura pode variar desde uma lista de faces até uma lista dos sólidos, shells, faces e half-edges, da mesma forma que a estrutura está definida no USPDesigner (servidor). É preciso fazer uma análise dos impactos no desempenho e limitações de cada uma das alternativas, para podermos escolher a opção mais adequada às nossas necessidades.

Em função dessa definição é preciso alterar a estrutura do arquivo XML, e por último configurar o servidor para gerar o arquivo no novo formato e o cliente para ler a estrutura completa do mesmo.

Um exemplo de arquivo XML que poderia ser usado, contendo as faces, seus loops e os pontos correspondentes, é o seguinte (contém duas faces triangulares):

```
<?xml version="1.0"?>
<!DOCTYPE TUDO[
<!ELEMENT TUDO (FACE)*>
<!ELEMENT FACE (ID,LOOP*)>
<!ELEMENT LOOP (PONTO)*>
<!ELEMENT PONTO (ID,X,Y,Z)>
<!ELEMENT ID (#PCDATA)>
<!ELEMENT X (#PCDATA)>
<!ELEMENT Y (#PCDATA)>
<!ELEMENT Z (#PCDATA)>
]>

<TUDO>

 <FACE>
  <ID>F1</ID>
  <LOOP>
   <PONTO> <ID>P1</ID> <X>0</X> <Y>0</Y> <Z>0</Z> </PONTO>
   <PONTO> <ID>P2</ID> <X>1</X> <Y>1</Y> <Z>0</Z> </PONTO>
   <PONTO> <ID>P3</ID> <X>1</X> <Y>0</Y> <Z>0</Z> </PONTO>
  </LOOP>
 </FACE>

 <FACE>
  <ID>F2</ID>
  <LOOP>
   <PONTO> <ID>P4</ID> <X>3</X> <Y>4</Y> <Z>1</Z> </PONTO>
   <PONTO> <ID>P5</ID> <X>1</X> <Y>1</Y> <Z>2</Z> </PONTO>
   <PONTO> <ID>P6</ID> <X>1</X> <Y>3</Y> <Z>0</Z> </PONTO>
  </LOOP>
 </FACE>
</TUDO>
```

Listagem 4.2 – Possível arquivo XML (contendo faces e arestas).

Esse arquivo também considera que cada entidade (ponto, face) têm um identificado (ID), conceito que já é utilizado no USPDesigner e é muito importante para a implementação da seleção de sólidos.

Seleções de arestas ou sólidos

A possibilidade de selecionar arestas, sólidos, pontos ou qualquer elemento é bastante útil nos programas CAD. Por isso existem no J3D classes prontas que tratam dessa manipulação. As principais classes para essa finalidade são: PickShape, PickBounds, PickRay, PickSegment, e PickPoint.

Essa funcionalidade só não foi implementada por falta de tempo para realizar as pesquisas, desenvolvimento e testes necessários.

Lista dos comandos válidos (servidor)

Imaginemos a seguinte situação: sempre que for criada uma nova função no servidor, ela aparecerá no "help" – ajuda – do programa cliente, ou eventualmente pode ser criada uma entrada no menu para esse comando.

Para tanto é preciso que haja um cadastro de todas os comandos no servidor e esses comandos podem ser enviados para os "clientes". Assim, o menu no programa cliente será criado a partir de dados recebidos do servidor (por exemplo, em um segundo arquivo XML).

A implementação disso exige uma alteração significativa no programa servidor, além de alteração no programa cliente e na comunicação.

Comunicação: broadcast, chat

A arquitetura desse programa CAD via a Web foi feita para facilitar o trabalho de diversos usuários em um único projeto simultaneamente. Portanto cabe ao programa ter uma forma de comunicação para que os usuários (engenheiros, projetistas, etc.) possam discutir alterações que estão sendo feitas, podendo essa discussão ser pública (entre várias pessoas) ou particular (entre duas pessoas), facilitando o trabalho em grupo.

Últimos comandos, menus com comandos, múltiplas visões do sólido

Algumas alterações na interface do cliente podem ser feitas para simplificar o uso do programa.

Uma delas é a inclusão de comandos nos menus. Essa inclusão pode ser feita da maneira tradicional ou então de uma forma mais dinâmica e flexível, utilizando uma lista de possíveis comandos que seria gerada pelo servidor (vide melhoria sobre lista de comandos válidos).

Outra melhoria pode ser uma memória dos últimos comandos digitados e enviados. Assim evita-se digitar o mesmo comando várias vezes. Isso poderia ser feito, por exemplo, na forma de um "combo box".

Ainda, poder-se-ia colocar na interface do programa mais de uma tela de visualização (objeto Canvas3D). Assim, para ver o objeto de vários pontos de vista simultaneamente o usuário precisaria abrir apenas uma única janela do browser.

Múltiplas estruturas para múltiplos usuários (instâncias independentes no servidor)

Conforme foi mencionado no início do trabalho, optamos até agora por utilizar uma única estrutura de dados para todos os usuários conectados ao sistema.

É interessante, porém, avaliar também a opção de se fazer com que cada usuário tenha uma estrutura de dados independente. Para isso, haverá de ser feita uma grande alteração no servidor, tendo que instanciar diversas classes conforme o número de usuários conectados.

5 CONCLUSÕES

Nesse trabalho foi desenvolvida uma interface para o USPDesigner, que é um modelador de sólidos B-Rep, programa CAD que pode ser utilizado via a Web, no modelo ASP.

Foram criadas as funcionalidades básicas para uso do modelador, sendo que a arquitetura da solução é bastante flexível, permitindo a ampliação de tais funcionalidades no futuro.

6 BIBLIOGRAFIA

MÄNTYLÄ, M. – An Introduction to Solid Modeling – Computer Science Press

WOO, NEIDER, DAVIS & SHREINER - OpenGL Programming Guide - Addison-Wesley Publishing Company, Massachusetts, 1999.

TSUZUKI, MARCOS DE SALES GUERRA - Notas de aula (Transformações geométricas e projeções, Curvas e superfícies) - São Paulo: USP, 1996.

AUTODESK – White Paper: The XML Revolution – Autocad 2002

HALL, BRIAN "BEEJ" - Beej's Guide to Network Programming (Using Internet Sockets), Revision Version 2.3.0 July 25, 2001

BOUVIER, DENNIS J. (K Computing) - Getting Started with the Java 3D™ API, © 1999 Sun Microsystems, Inc.

VOGEL, ANDREAS; DUDDY, KEITH. - Java programming with CORBA, 2nd ed. New York : Wiley computer publishing, c1998.

RADEMACHER, PAUL – Glui Manual; http://www.cs.unc.edu/~rademach/glui – 1998.

http://java.sun.com

http://www.dcc.unicamp.br/~aacesta

http://www.pajato.com/jogl/

http://www.palevich.com/3dfx/JavaOpenGL.html

http://www.jausoft.com/gl4java/

http://romka.demonews.com/opengl/doc/opengl_java_eng.htm

http://www.wwa.com/~razeh/YAJOGLB/doc/YAJOGLB.html

http://arcana.symbolstone.org/products/magician/

http://community.borland.com/

http://adams.patriot.net/~tvalesky/freecorba.html

http://www.omg.org/

http://www.iona.com/

http://www.yahoo.com/, http://www.altavista.com/, http://infoseek.go.com/

http://tucows.uol.com/

http://xml.locaweb.com.br

http://www.viewpoint.com

http://www.yahoogroups.com

http://www.analogx.com/

http://www.rational.com

http://www.vrml.org

http://www.x3d.org

http://www.research.att.com/~bs

7 ANEXO I – CÓDIGOS FONTE DOS PROGRAMAS

7.1 Servidor (main)

```
main (void)
{
  _beginthread(cria_thread,0,NULL);
  ...
  ...
}

void cria_thread(void *arg)
{
  char *st = (char *)malloc(sizeof(char)*MAX_LEN);
  char txt[MAX_LEN];
  float x1, y1, z1, x2, y2, z2;

  while(1) // entra em loop infinito
  {
    if (!sockXML.espera_conexao()) // espera um cliente se conectar
    if (!sockXML.recebe_comando(&st)) // recebe o comando e coloca-o na variável st
    {
      sockXML.avisa_todos("AGUARDE\n"); // aviso para que ninguém envie mais um
comando, antes do primeiro ser tratado
      trata_comando(st); // trata o comando, gerando a nova estrutura, o novo arquivo
XML, e notificando os usuários

// a resposta vai apenas para o cliente que enviou o comando
      if (sockXML.envia_resposta("%s\n",resposta))
        else if (strcmp(txt,"_"))
// esse é o texto que o cliente envia ao iniciar, avisando que ele esta ativo
          sockXML.envia_resposta("_\n");
   sockXML.avisa_todos("LIVRE\n"); // libera as linhas de comando
}
```

7.2 Servidor (Classe sock_xml)

```
socket_xml::socket_xml()
{
// limpa a lista de clientes ativos
  int i;
  for (i=0;i<NUM_CONEXOES;i++)
    clientes_ativos[i].sin_addr.s_addr = htonl(0);

// cria um servidor socket e associa à variável sock
  if (WSAStartup(MAKEWORD(1, 1), &wsaData) != 0)
  {
    cout << "WSAStartup falhou." << endl;
    exit(1);
  }
  serv.sin_addr.s_addr = htonl(INADDR_ANY);
  serv.sin_family = AF_INET;
  serv.sin_port = htons(PORTA);
  i = sizeof(serv);
  sock = socket(AF_INET,SOCK_STREAM,0);
  if (i = bind(sock,(sockaddr*)&serv,i))
  {
    cout << "Erro na criacao do servidor socket: " << i << endl;
    exit(1);
  }

//  Servidor socket criado...

  listen(sock,NUM_CONEXOES);
}
```

```
socket_xml::~socket_xml()
{
  WSACleanup();
}

int socket_xml::espera_conexao()
{
  int i, j;

  i = sizeof(serv);
  clisock = accept(sock,(sockaddr*)&(serv),&(i));
  if (clisock==INVALID_SOCKET)
  {
    cout << "invalid socket: " << clisock << endl;
    return(1);
  }

  i = sizeof(struct sockaddr_in);
  j = getpeername(clisock, (sockaddr*)&(client), &i);
  i = ntohs(client.sin_port);

//  Conexao criada na porta remota i

  return(0);
}

int socket_xml::recebe_comando(char **st)
{
  char *texto = (char *)malloc(sizeof(char)*MAX_LEN);
  int i, porta;

// recebe a porta remota onde o cliente vai ser sinalizado
  i = recv(clisock,texto,MAX_LEN,0);
  if (i==SOCKET_ERROR)
  {
    cout << "erro ao receber dados" << endl;
    delete texto;
    return(1);
  }
  texto[i] = '\0';
  sscanf(texto,"%d",&porta);

  cadastra(porta, client);
  texto = strcpy(texto,"");
// recebe a linha de comando
  i = recv(clisock,texto,MAX_LEN,0);
  if (i==SOCKET_ERROR)
  {
    cout << "erro ao receber dados" << endl;
    delete texto;
    return(1);
  }
  texto[i] = '\0';

  *st = texto;
  return(0);
}

int socket_xml::envia_resposta(char *retorno)
{
  unsigned int i;

  i = send(clisock, retorno, strlen(retorno), 0);
  if (i!=strlen(retorno))
    return(1);
  return(0);
}

int socket_xml::avisa_todos(char *aviso)
{
  int i, j;

  for (i=0;i<NUM_CONEXOES;i++)
    if (clientes_ativos[i].sin_addr.s_addr!=htonl(0))
```

```
    {
// se ainda estiver ativo, envia sinal para ler o XML
      j = send(clientes[i],aviso,strlen(aviso),0);
      if (j==-1)
      {
// se nao estiver ativos, tira da lista
        clientes_ativos[i].sin_addr.s_addr = htonl(0);
        closesocket(clientes[i]);
      }
      else
      {
// Enviou sinal para IP: ntohl(clientes_ativos[i].sin_addr.s_addr)
      }
    }
  return(0);
}

int socket_xml::cadastra(int porta, sockaddr_in cli)
{
  int i, j, cadastrado = 0;
  for (i=0;i<NUM_CONEXOES;i++)
    if ((clientes_ativos[i].sin_addr.s_addr==cli.sin_addr.s_addr)
      && (clientes_ativos[i].sin_port==htons(porta)))
    {
      sockaddr_in temp;
      int temp2 = sizeof(struct sockaddr);
      j = getpeername(clientes[i],(struct sockaddr*)&temp,&temp2);
      if (j!=-1)
      {
        cout << "j!=-1" << endl;
        cadastrado = 1;
      }
      else
      {
        cout << "j==-1" << endl;
        clientes_ativos[i].sin_addr.s_addr = htonl(0);
      }
    }
    if (cadastrado)
    {
//  Cliente ja estava cadastrado
      return(1);
    }
  for (i=0;i<NUM_CONEXOES;i++)
    if (clientes_ativos[i].sin_addr.s_addr==htonl(0))
    {
      clientes_ativos[i].sin_addr.s_addr = cli.sin_addr.s_addr;
      clientes_ativos[i].sin_port = htons(porta);
      clientes_ativos[i].sin_family = AF_INET;
// cria uma conexao socket para avisos
      clientes[i] = socket(AF_INET,SOCK_STREAM,0);
      if (clientes[i]==-1)
      {
        clientes_ativos[i].sin_addr.s_addr = htonl(0);
        cout << "ERRO NA CRIACAO DO SOCKET" << endl;
        return(1);
      }
// se conectar no IP + porta registrados
      j  =  connect(clientes[i],  (struct  sockaddr*)&clientes_ativos[i],sizeof(struct
sockaddr_in));
      if (j==-1)
      {
        clientes_ativos[i].sin_addr.s_addr = htonl(0);
        cout << "ERRO NA CONEXAO" << endl;
        return(1);
      }
    return(0);
  }
  return(1); // nao consegui cadastrar (limite de clientes atingido)
}
```

7.3 Cliente.java

```
package tf2;

import java.awt.*;
import java.awt.event.*;
import java.applet.*;
import javax.swing.*;
import java.net.*;
import java.io.*;
import com.sun.j3d.utils.applet.MainFrame;
import com.sun.j3d.*;
import javax.media.j3d.*;

import javax.vecmath.*;
import com.sun.j3d.utils.universe.SimpleUniverse;
import java.awt.event.*;
import com.sun.j3d.utils.behaviors.mouse.*;

/**
 * Title:
 * Description:
 * Copyright:    Copyright (c) 2001
 * Company:
 * @author
 * @version 1.0
 */

public class Cliente extends JApplet
{
  static ServidorSocket ss;
  static Canvas3D canvas3D;
  static BranchGroup scene;
  static TransformGroup objTransform;
  static Shape3D s3d;
  static SimpleUniverse simpleU;

  boolean isStandalone = false;
  static boolean antialiasing = false;
  static String host;
  static String port;

  JPanel principal;
  JMenuBar menu;
    JMenu jMenuFile;
      JMenuItem jMenuFileExit;
    JMenu jMenuHelp;
      JMenuItem jMenuHelpAbout;
      JMenuItem jMenuHelpAbout2;

  JPanel grafico;
  JPanel panelsouth;
    JPanel panelcom;
      static JTextField linha;
      JButton botao;
    static JLabel retorno;

  /**Get a parameter value*/
  public String getParameter(String key, String def)
  {
    return isStandalone ? System.getProperty(key, def) :
      (getParameter(key) != null ? getParameter(key) : def);
  }

  /**Construct the applet*/
  public Cliente()
  {
    principal = new JPanel();
    menu = new JMenuBar();
    jMenuFile = new JMenu();
      jMenuFileExit = new JMenuItem();
    jMenuHelp = new JMenu();
      jMenuHelpAbout = new JMenuItem();
```

```
        jMenuHelpAbout2 = new JMenuItem();

    grafico = new JPanel();
    panelsouth = new JPanel();
      panelcom = new JPanel();
        linha = new JTextField();
        botao = new JButton();
      retorno = new JLabel();

      scene = new BranchGroup();
      objTransform = new TransformGroup();
      s3d = new Shape3D();

      JPopupMenu.setDefaultLightWeightPopupEnabled(false);
      GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
      canvas3D = new Canvas3D(config);
      simpleU = new SimpleUniverse(canvas3D);

      scene.setCapability(BranchGroup.ALLOW_CHILDREN_WRITE);
      scene.setCapability(BranchGroup.ALLOW_CHILDREN_READ);
      scene.setCapability(BranchGroup.ALLOW_DETACH);
      objTransform.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
      objTransform.setCapability(TransformGroup.ALLOW_TRANSFORM_READ);
      objTransform.setCapability(TransformGroup.ALLOW_CHILDREN_READ);
      objTransform.setCapability(TransformGroup.ALLOW_CHILDREN_WRITE);
      s3d.setCapability(Shape3D.ALLOW_GEOMETRY_WRITE);
      s3d.setCapability(Shape3D.ALLOW_GEOMETRY_READ);
      s3d.setCapability(Shape3D.ALLOW_APPEARANCE_WRITE);
      s3d.setCapability(Shape3D.ALLOW_APPEARANCE_READ);
    }

    /**Initialize the applet*/
    public void init()
    {
      try {host = this.getParameter("host", "127.0.0.1");}
      catch(Exception e){e.printStackTrace();}
      try{port = this.getParameter("port", "6666");}
      catch(Exception e){e.printStackTrace();}
      try
      {
        jbInit();
      }
      catch(Exception e)
      {
        e.printStackTrace();
      }
    }
    /**Component initialization*/
    private void jbInit() throws Exception
    {
      this.setJMenuBar(menu);
      menu.setToolTipText("");

      jMenuFile.setText("Arquivo");
      jMenuFileExit.setText("Propriedades");
      jMenuFileExit.addActionListener(new ActionListener()
      {public void actionPerformed(ActionEvent e)
        {jMenuFileExit_actionPerformed(e);} });

      jMenuHelp.setText("Ajuda");
      jMenuHelpAbout.setText("Sobre");
      jMenuHelpAbout.addActionListener(new ActionListener()
      {public void actionPerformed(ActionEvent e)
        {jMenuHelpAbout_actionPerformed(e,"USPDesigner para a web\nNelson Vogel -
2001");} });

      jMenuHelpAbout2.setText("Uso do Mouse");
      jMenuHelpAbout2.addActionListener(new ActionListener()
      {public void actionPerformed(ActionEvent e)
        {jMenuHelpAbout_actionPerformed(e,"BOTAO ESQUERDO - ROTACIONA\nBOTAO DIREITO -
MOVE\nBOTAO DO MEIO (ALT+ESQ) - ZOOM");} });

      jMenuFile.add(jMenuFileExit);
      jMenuHelp.add(jMenuHelpAbout);
```

```
    jMenuHelp.add(jMenuHelpAbout2);
    menu.add(jMenuFile);
    menu.add(jMenuHelp);

    this.setContentPane(principal);
    principal.setLayout(new BorderLayout());

    principal.add(canvas3D, BorderLayout.CENTER);

    principal.add(panelsouth, BorderLayout.SOUTH);
    panelsouth.setLayout(new BorderLayout());
    panelsouth.add(panelcom, BorderLayout.NORTH);
      panelcom.setLayout(new BorderLayout());
      panelcom.add(linha, BorderLayout.NORTH);
        linha.setText("");
        linha.addActionListener(new ActionListener()
          {public void actionPerformed(ActionEvent e)
          {linha_enterClicked(e);} });

    panelsouth.add(retorno, BorderLayout.SOUTH);
    retorno.setText("AQUI VOCE IRA VER O TEXTO DE RETORNO");
  }
  /**Get Applet information*/
  public String getAppletInfo()
  {
    return "Applet Information";
  }
  /**Get parameter info*/
  public String[][] getParameterInfo()
  {
    String[][] pinfo =
      {
      {"host", "String", "IP do host"},
      {"port", "String", "Porta para conexao"},
      };
    return pinfo;
  }
  public void start()
  {
    ss = new ServidorSocket(6667);
    ss.start();
// espera o servidor socket ser criado
    while (ServidorSocket.pronto==0);
// tenta se conectar no servidor assim que liga
    ClienteSocket cliente = new ClienteSocket(ServidorSocket.porta, "_", host,
Integer.valueOf(port).intValue());
    createSceneGraph();
  }

// This function is called, when applet is stopped
  public void stop()
  {
    scene = null;
    simpleU = null;
    objTransform = null;
    s3d = null;
    ss = null;
  }

// This function is called, when applet is destroyed
  public void destroy()
  {
  }

  //static initializer for setting look & feel
  static
  {
    try
    {
      //UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
      //UIManager.setLookAndFeel(UIManager.getCrossPlatformLookAndFeelClassName());
    }
    catch(Exception e)
    {
```

```
    }
  }

  /**File | Exit action performed*/
  public void jMenuFileExit_actionPerformed(ActionEvent e)
  {
    Propriedades dlg = new Propriedades();
    Dimension dlgSize = dlg.getPreferredSize();
    Dimension frmSize = getSize();
    Point loc = getLocation();
    dlg.setLocation((frmSize.width - dlgSize.width) / 2 + loc.x, (frmSize.height -
dlgSize.height) / 2 + loc.y);
    dlg.setModal(true);
    dlg.show();
  }
  /**Help | About action performed*/
  public void jMenuHelpAbout_actionPerformed(ActionEvent e, String st)
  {
    Ajuda dlg = new Ajuda(st);
    Dimension dlgSize = dlg.getPreferredSize();
    Dimension frmSize = getSize();
    Point loc = getLocation();
    dlg.setLocation((frmSize.width - dlgSize.width) / 2 + loc.x, (frmSize.height -
dlgSize.height) / 2 + loc.y);
    dlg.setModal(true);
    dlg.show();
  }

  void linha_enterClicked(ActionEvent e)
  {
    ClienteSocket cliente = new ClienteSocket(ServidorSocket.porta, linha.getText(),
host, Integer.valueOf(port).intValue());
    linha.setText("");
  }

  public static void createSceneGraph()
  {
    objTransform.addChild(s3d);
    scene.addChild(objTransform);
    Background background = new Background(new Color3f(Color.white));
    background.setApplicationBounds(new BoundingSphere());
    scene.addChild(background);

    MouseRotate myMouseRotate = new MouseRotate();
    myMouseRotate.setTransformGroup(objTransform);
    myMouseRotate.setSchedulingBounds(new BoundingSphere());
    scene.addChild(myMouseRotate);

    MouseTranslate myMouseTranslate = new MouseTranslate();
    myMouseTranslate.setTransformGroup(objTransform);
    myMouseTranslate.setSchedulingBounds(new BoundingSphere());
    scene.addChild(myMouseTranslate);

    MouseZoom myMouseZoom = new MouseZoom();
    myMouseZoom.setTransformGroup(objTransform);
    myMouseZoom.setSchedulingBounds(new BoundingSphere());
    scene.addChild(myMouseZoom);

    ClienteXML.cria();
    scene.compile();

    simpleU.getViewingPlatform().setNominalViewingTransform();
    simpleU.addBranchGraph(scene);
  }

}
```

7.4 ServidorSocket.java

```
package tf2;

import java.awt.*;
import java.net.*;
import java.io.*;

/**
 * Title:
 * Description:
 * Copyright:    Copyright (c) 2001
 * Company:
 * @author
 * @version 1.0
 */

public class ServidorSocket extends Thread
{
  private Socket socket = null;
  static int porta;
  static int pronto;
  public ServidorSocket(int port)
  {
    porta = port;
    pronto = 0;
  }
  public void run()
  {
    ServerSocket serverSocket = null;
    try
    {
      while (pronto==0)
      {
        try
        {
          serverSocket = new ServerSocket(porta);
          pronto = 1;
        }
        catch (IOException e)
        {
          porta++; // tenta outra porta
          pronto = 0; // redundante
        }
      }
      Socket socket = serverSocket.accept();
      BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
      String inputLine;
      boolean exit = false;
      while(!exit)
      {
        inputLine = in.readLine();
        if (inputLine.compareTo("PERPETUO")==0)
        {
            socket.close();
            exit = true;
        }
        else if (inputLine.compareTo("AGUARDE")==0)
        {
          Cliente.linha.setEditable(false);
          Cliente.retorno.setBackground(Color.red);
        }
        else if (inputLine.compareTo("LIVRE")==0)
        {
          Cliente.linha.setEditable(true);
          Cliente.retorno.setBackground(Color.lightGray);
        }
        else if (inputLine.compareTo("XML")==0)
        {
          ClienteXML.cria();
          Cliente.linha.setEditable(true);
          Cliente.retorno.setBackground(Color.lightGray);
```

```
        }
        else
        {
          Cliente.retorno.setText(":>" + inputLine);
        }
      }
      in.close();
      socket.close();
      System.exit(1);
    }
    catch (IOException e)
    {
      e.printStackTrace();
      System.out.println("MORRA");
      System.exit(1);
    }
  }
}
```

7.5 ClienteSocket.java

```
package tf2;
import java.io.*;
import java.net.*;

/**
 * Title:
 * Description:
 * Copyright:    Copyright (c) 2001
 * Company:
 * @author
 * @version 1.0
 */

public class ClienteSocket
{
  public ClienteSocket(int portserv, String st, String host, int port)
  {
    Socket sk = null;
    PrintWriter out = null;
    BufferedReader in = null;
    try
    {
      sk = new Socket(host, port);
      out = new PrintWriter(sk.getOutputStream(), true);
      in = new BufferedReader(new InputStreamReader(sk.getInputStream()));
    }
    catch (UnknownHostException e)
    {
      System.err.println("host " + host + " nao encontrado.");
      System.exit(1);
    }
    catch (IOException e)
    {
      System.err.println("nao foi possivel se comunicar com o host " + host);
      System.exit(1);
    }
    out.println(portserv); // envia a porta onde está o servidor
    out.println(st); // envia a linha de comando
    String linha;
    try
    {
      linha = in.readLine();
      Cliente.linha.setEditable(true);
      if (linha.compareTo("_")!=0)
        Cliente.retorno.setText(linha);

      out.close();
      in.close();
      sk.close();
    }
```

```
      catch (Exception e)
      {
        System.err.println("Erro na leitura / fechamento dos arquivos");
        System.exit(1);
      }
    }
  }
```

7.6 ClienteXML.java

```
package tf2;
import com.sun.xml.tree.*;
import org.w3c.dom.*;
import java.io.*;
import com.sun.j3d.*;
import javax.media.j3d.*;
import javax.vecmath.*;
import java.awt.*;

/**
 * Title:
 * Description:
 * Copyright:    Copyright (c) 2001
 * Company:
 * @author
 * @version 1.0
 */

public class ClienteXML
{
  static XmlDocument doc = null;

  public ClienteXML()
  {
  }
  public static void cria()
  {
    try
    {
      String url = "http://" + Cliente.host + "/arestas.xml";
      doc = XmlDocument.createXmlDocument(url);
    }
    catch (Exception e)
    {
      System.err.println("Erro na criacao do DOM: \n" + e.toString());
    }
    NodeList listaArestas = doc.getElementsByTagName("ARESTA");
    int j = 2*listaArestas.getLength();
    if (j!=0)
    {
      LineArray minhasLinhas = new LineArray(j,LineArray.COORDINATES |
LineArray.COLOR_3);
      LineAttributes la = new LineAttributes();
      la.setLineWidth(3.0f);
      la.setLineAntialiasingEnable(Cliente.antialiasing);
      Appearance appla = new Appearance();
      appla.setLineAttributes(la);
      Cliente.s3d.setAppearance(appla);
      minhasLinhas.setCapability(LineArray.ALLOW_COORDINATE_WRITE);
      minhasLinhas.setCapability(LineArray.ALLOW_COORDINATE_READ);
      minhasLinhas.setCapability(LineArray.ALLOW_COLOR_WRITE);
      Cliente.s3d.setGeometry(minhasLinhas);
      for (int i=0;i<listaArestas.getLength();i++)
      {
        Element aresta = (Element)listaArestas.item(i);
        NodeList listaPontos = aresta.getElementsByTagName("PONTO");
        Element ponto = (Element)listaPontos.item(0);
        NodeList listaX = ponto.getElementsByTagName("X");
        NodeList listaY = ponto.getElementsByTagName("Y");
        NodeList listaZ = ponto.getElementsByTagName("Z");
        String x1 = listaX.item(0).getFirstChild().getNodeValue().trim();
```

```
            String y1 = listaY.item(0).getFirstChild().getNodeValue().trim();
            String z1 = listaZ.item(0).getFirstChild().getNodeValue().trim();
            ponto = (Element)listaPontos.item(1);
            listaX = ponto.getElementsByTagName("X");
            listaY = ponto.getElementsByTagName("Y");
            listaZ = ponto.getElementsByTagName("Z");
            String x2 = listaX.item(0).getFirstChild().getNodeValue().trim();
            String y2 = listaY.item(0).getFirstChild().getNodeValue().trim();
            String z2 = listaZ.item(0).getFirstChild().getNodeValue().trim();
            minhasLinhas.setCoordinate(2*i,new
Point3f(Float.valueOf(x1).floatValue(),Float.valueOf(y1).floatValue(),Float.valueOf(z
1).floatValue()));
            minhasLinhas.setCoordinate(2*i+1,new
Point3f(Float.valueOf(x2).floatValue(),Float.valueOf(y2).floatValue(),Float.valueOf(z
2).floatValue()));
           minhasLinhas.setColor(2*i,new Color3f(Color.black));
           minhasLinhas.setColor(2*i+1,new Color3f(Color.black));
         }  // for (i
      } // if (j!=0)
    } // cria()
}
```

7.7 *Propriedades.java*

```
package tf2;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

/**
 * Title:
 * Description:
 * Copyright:    Copyright (c) 2001
 * Company:
 * @author
 * @version 1.0
 */

public class Propriedades extends JDialog implements ActionListener
{
  JPanel contentPane;
  BorderLayout borderLayout1 = new BorderLayout();
  JCheckBox jCheckBox1 = new JCheckBox();
  JToggleButton jToggleButton1 = new JToggleButton();

  /**Construct the frame*/
  public Propriedades()
  {
//    super(parent);
    enableEvents(AWTEvent.WINDOW_EVENT_MASK);
    try
    {
      jbInit();
    }
    catch(Exception e)
    {
      e.printStackTrace();
    }
    pack();
  }

  /**Component initialization*/
  private void jbInit() throws Exception
  {
    jCheckBox1.setText("Antialiasing");
    jCheckBox1.setSelected(Cliente.antialiasing);
    contentPane = (JPanel) this.getContentPane();
    contentPane.setLayout(borderLayout1);
    this.setSize(new Dimension(400, 300));
    this.setTitle("Propriedades");
    jToggleButton1.setText("OK");
```

```
    jToggleButton1.addActionListener(this);
    contentPane.add(jCheckBox1, BorderLayout.NORTH);
    contentPane.add(jToggleButton1, BorderLayout.SOUTH);
  }
  /**Overridden so we can exit when window is closed*/
  protected void processWindowEvent(WindowEvent e)
  {
    if (e.getID() == WindowEvent.WINDOW_CLOSING)
    {
      cancel();
    }
    super.processWindowEvent(e);
  }
  /**Close the dialog*/
  void cancel()
  {
    if (jCheckBox1.isSelected())
      Cliente.antialiasing = true;
    else
      Cliente.antialiasing = false;
    ClienteXML.cria();
    dispose();
  }
  /**Close the dialog on a button event*/
  public void actionPerformed(ActionEvent e)
  {
    if (e.getSource() == jToggleButton1)
    {
      cancel();
    }
  }
}
```

7.8 *Ajuda.java*

```
package tf2;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

/**
 * Title:
 * Description:
 * Copyright:    Copyright (c) 2001
 * Company:
 * @author
 * @version 1.0
 */

public class Ajuda extends JDialog implements ActionListener
{
  JPanel contentPane;
  JLabel jLabel1 = new JLabel();
  BorderLayout borderLayout1 = new BorderLayout();

  JToggleButton jToggleButton1 = new JToggleButton();

  /**Construct the frame*/
  public Ajuda(String st)
  {
//    super(parent);
    jLabel1.setText(st);
    enableEvents(AWTEvent.WINDOW_EVENT_MASK);
    try
    {
      jbInit();
    }
    catch(Exception e)
    {
      e.printStackTrace();
    }
```

```
    pack();
  }

  /**Component initialization*/
  private void jbInit() throws Exception
  {
    contentPane = (JPanel) this.getContentPane();
    contentPane.setLayout(borderLayout1);
    this.setSize(new Dimension(400, 300));
    this.setTitle("Ajuda");
    jToggleButton1.setText("OK");
    jToggleButton1.addActionListener(this);
    contentPane.add(jLabel1, BorderLayout.CENTER);
    contentPane.add(jToggleButton1, BorderLayout.SOUTH);
  }
  /**Overridden so we can exit when window is closed*/
  protected void processWindowEvent(WindowEvent e)
  {
    if (e.getID() == WindowEvent.WINDOW_CLOSING)
    {
      cancel();
    }
    super.processWindowEvent(e);
  }
  /**Close the dialog*/
  void cancel()
  {
    dispose();
  }
  /**Close the dialog on a button event*/
  public void actionPerformed(ActionEvent e)
  {
    if (e.getSource() == jToggleButton1)
    {
      cancel();
    }
  }
}
```