UNIVERSIDADE ESTADUAL DE CAMPINAS

FACULDADE DE ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO

DEPARTAMENTO DE ENGENHARIA DE COMPUTAÇÃO E AUTOMAÇÃO INDUSTRIAL

Modelagem geométrica colaborativa e multiplataforma

Tese de Doutorado

Autor: **Luiz Gonzaga da Silveira Júnior**

Orientadora: **Profa. Dr-Ing. Wu, Shin-Ting**

Banca Examinadora: **Profa. Dr.-Ing. Wu, Shin-Ting (FEEC-Unicamp)**
Profa. Dra. Regina Borges de Araújo (DC-UFSCar)
Prof. Dr.-Ing. Klaus Schützer (FEAU-Unimep)
Prof. Dr. Eleri Cardozo (FEEC-Unicamp)
Prof. Dr.-Ing. Léo Pini Magalhães (FEEC-Unicamp)
Profa. Dra. Regina Coeli Ruschel (FEC-Unicamp)

Tese submetida à Faculdade de Engenharia Elétrica e de Computação da Universidade Estadual de Campinas, para preenchimento dos pré-requisitos parciais para obtenção do Título de Doutor em Engenharia Elétrica, com área de concentração em Engenharia de Computação.

2 de junho de 2005

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

Si39m

Silveira Júnior, Luiz Gonzaga da
Modelagem geométrica colaborativa e multiplataforma / Luiz Gonzaga da Silveira Júnior. --Campinas, SP: [s.n.], 2005.

Orientador: Wu, Shin-Ting
Tese (doutorado) - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Computação gráfica. 2. Imagem tridimensional. I. Wu, Shin-Ting. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

Titulo em Inglês: Multiplatform collaborative geometric modeling
Palavras-chave em Inglês: Graphics computer e Three – dimensional visualization
Área de concentração: Engenharia de Computação
Titulação: Doutor em Engenharia Elétrica
Banca examinadora: Regina Borges de Araújo, Klaus Schützer, Eleri Cardozo, Léo Pini Magalhães e Regina Coeli Ruschel
Data da defesa: 02/06/2005

COMISSÃO JULGADORA - TESE DE DOUTORADO

Candidato: Luiz Gonzaga da Silveira Júnior

Data da Defesa: 2 de junho de 2005

Título da Tese: "Modelagem Geométrica Colaborativa e Multiplataforma"

Profa. Dra. Wu Shin-Ting (Matr. 246182): _____

Profa. Dra. Regina Borges de Araújo: _____

Prof. Dr. Klaus Schützer: _____

Profa. Dra. Regina Coeli Ruschel: _____

Prof. Dr. Eleri Cardozo: _____

Prof. Dr. Léo Pini Magalhães: _____

Secretária Giane Cristina Sales Geraldo: _____

Coordenador de PG Prof. Dr. Michel Daoud Yacoub: _____

Resumo

Este trabalho aborda o problema de consistência de dados geométricos e topológicos no contexto de modelagem geométrica colaborativa, destacando o impacto do processamento distribuído sobre a robustez das operações geométricas e topológicas em cada transação em um ambiente computacional multiplataforma.

A consistência é alcançada através de uma arquitetura fracamente acoplada, caracterizada por representações independentes das informações geométricas e gráficas dos modelos, e do emprego de algoritmos geométricos já consolidados em aplicações mono-usuário. Enquanto as informações geométricas são mantidas em um servidor dedicado, as respectivas representações gráficas são replicadas, renderizadas e exibidas localmente no espaço de trabalho de cada usuário. Assim, pode-se explorar individualmente os recursos dos equipamentos utilizados, através do ajuste dos parâmetros de renderização e visualização, sem degradar a robustez do sistema. No entanto, a separação dos modelos origina dois problemas: maior tempo de resposta em cada interação e a perda de consciência coletiva.

Nós apresentamos um modelo de interação distribuída que evita a sobrecarga na comunicação, sem sacrificar a realimentação visual, através do uso de metáforas 3D. A perda de consciência coletiva é resolvida através de um conjunto de componentes de consciência coletiva e a reutilização de representações gráficas das metáforas 3D. Estes artefatos oferecem consciência de localização, perspectiva e proximidade aos participantes da colaboração.

Um protótipo de modelador geométrico colaborativo, baseado na arquitetura fracamente acoplada e nos modelos de interação distribuída e consciência coletiva, foi implementado para validar nossa proposta.

Abstract

This work tackles the consistency of geometric and topological data problems in the collaborative geometric modeling context, focusing on the robustness for each transaction that occurs in a multiplatform environment.

We propose a loosely coupled system architecture in which geometric and graphical models are represented separately. The geometric information is kept in a dedicated server and handled through consolidated geometric algorithms, while their respective graphical representations are replicated, rendered and displayed in each end user workspaces. This allows, on the one hand, better use of the computational resources, through local adjustments of rendering and visualization parameters, without loss the robustness. On the other hand, two problems may arise: longer response time in each interaction and lost of group awareness.

We present a distributed interaction model, in which the overload of the network may be avoided without sacrificing the visual feedback, through a set of 3D-metaphors. The group awareness is achieved by a set of awareness components and the reuse of graphical representations of the 3D-metaphors. They show to be very powerful in providing the awareness of the location, perspective, and proximity to the co-workers.

A collaborative geometric modeller prototype, based on loosely coupled architecture and interaction and group awareness models, has been implemented to validate our proposal.

Agradecimentos

À minha mãe, sempre presente na minha vida e ao meu pai, o coronel Luiz do Tenente, sempre preocupado com o Saber, o mais alto lugar da minha gradidão. Obrigado e muito. Dona Sônia, o caminho foi longo, mas eu cheguei lá.

Quando todos os obstáculos pareciam intransponíveis, você estava lá, vencendo-os comigo! Obrigado, minha lindinha. Sem você não teria conseguido. Obrigado, Raquel!

Ting, uma orientadora tão presente e confiante, agradeço pelo tratamento amigo durante nosso convívio. Obrigado por sempre ter acreditado que atingiríamos nossos objetivos. Agradeço ao Jaime por segurar a barra quando, por muitas vezes, precisávamos trabalhar domingos e feriados.

Daniel Tost, meu agradecimento sincero pelo seu trabalho tão dedicado. Ao amigo Marcelo Malheiros, que tanto fez para e pelo grupo. Rober, um amigo confiante. Obrigado!

Meus agradecimentos ao pessoal da FEEC, Cristina, Beti, Mazé, Gerusa, Dona Cida, Fina, Carmen, Edmundo, Cláudio, pelo tratamento amigo e pelos pepinos resolvidos. Neste campo, uma deferência toda especial a Noêmia, pela incansável presteza e atenção.

De tudo que consegui, nada se iguala ao prazer das amizades feitas e sedimentadas por todos estes anos. São tantos bons e queridos amigos que parece impossível enumerá-los...meu coração acredita que o momento pede sim para agradecer sem medo ao povo do LCA; Jussara, que sempre me apoiou muito e caminhou comigo até este momento - meu muitíssimo obrigado; os amigos do peito Rodrigo, Nicola, Mimi, Armando, Ricardo, Alexandre, Caboclo Mamador (Faina), Rex, Naur, Caçapa; aos que chegaram e ficaram Benê, Pri, Bin, Alencar, Dinho, Márcio, Ejo, Guampa, Rafael, Pedro, Wagner. Os interlocutores do saber, Chaim, Daniel e Affonso! é chegada a hora, meus amigos...

Todos têm um lugar no meu coração e não fiquem com ciúmes, mas para Rodrigo e Nicola não encontro adjetivos que sejam suficientes para descrever o tamanho da minha gratidão e admiração. Obrigado por tudo meus amigos, meus irmãos!

aos meus pais:
D. Sônia e Sr. Luiz, pela confiança
e orgulho que sempre depositaram em minha pessoa, obrigado.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

"A minha mente está cheia de idéias improváveis!"
Júlio Verne.

A consolidação de tecnologias de redes de computadores, com o aumento da largura de banda e do poder de processamento dos sistemas computacionais, aliada à necessidade de aumento de produtividade, tem contribuído de maneira decisiva para o desenvolvimento de aplicações colaborativas em arquitetura, manufatura, engenharia, simulação, educação e entretenimento. Com estas aplicações, um grupo de usuários em um mesmo ambiente ou dispersos geograficamente pode desenvolver tarefas de forma conjunta e numa rede multiplataforma. Nas últimas décadas, têm surgido alguns projetos relacionados ao desenvolvimento de plataformas computacionais neste contexto e em áreas correlatas, como visualização colaborativa, jogos e simulação [74]. Cada contexto de aplicação tem necessidades específicas, que exigem diretrizes distintas no desenvolvimento de ambientes computacionais adequados.

Consideramos como um sistema de modelagem geométrica colaborativa e multiplataforma um conjunto de aplicações capaz de prover funcionalidades de modelagem geométrica para um grupo de usuários inter-conectados numa rede multiplataforma. Esses usuários, através de sessões de trabalho colaborativo, interagem diretamente com os modelos 3D de forma concorrente, síncrona e coordenada [3, 4, 29, 76]. Van Den Berg [74] classifica diversos sistemas de modelagem geométrica colaborativa, destacando os benefícios destes sistemas comparados com aplicações mono-usuário e as dificuldades encontradas no desenvolvimento de aplicações colaborativas. Neste trabalho, estudamos alguns aspectos fundamentais relacionados ao desenvolvimento de modeladores

geométricos colaborativos e multiplataforma, identificamos problemas relacionados ao contexto de transações em um sistema distribuído e propomos um conjunto de soluções para estes problemas.

Entendemos por transação como uma sequência de requisições feitas ao sistema de forma atômica e consistente. A atomicidade exige que cada transação deve estar livre de interferências de operações concorrentes e todas as operações devem ser completadas com sucesso ou não devem ter efeito por ocasião da ocorrência de problemas no sistema ou componente do sistema. A consistência exige que uma transação deve levar um sistema de um estado consistente para outro estado também consistente. Especificamente, abordamos aspectos relacionados com a robustez de operações geométricas, adaptabilidade à heterogeneidade da plataforma computacional, usabilidade com ênfase na interatividade e à consciência coletiva[1]. Além disso, apresentamos um protótipo de um modelador colaborativo e mecanismos para validação das soluções propostas ao longo da tese.

1.1 Análise de requisitos

Um cenário típico para modelagem geométrica colaborativa é um espaço de trabalho compartilhado, no qual um grupo de usuários geograficamente dispersos (seja em uma mesma sala, empresa ou até em cidades e países distintos) trabalham em conjunto para criar, analisar, modificar e manipular diretamente um modelo 3D. Especificamente, consideramos para nosso estudo um ambiente computacional heterogêneo, no qual as aplicações utilizam diferentes equipamentos (*hardware*), sistemas operacionais, linguagens de programação e compiladores, Figura 1.1.

Proveniente do contexto de modelagem geométrica mono-usuário, a necessidade de se garantir fortemente que os objetos resultantes de operações geométricas sejam consistentes, sem ambigüidades na interpretação dos resultados e reproduzíveis para mesma seqüência de operações, são transportados para o ambiente colaborativo. O fato de que todos os participantes possam manipular os objetos direta e interativamente num ambiente colaborativo aumenta a complexidade da preservação da robustez das operações. A parceria entre participantes de um sistema colaborativo se torna ainda

[1] A consciência coletiva é também chamada de consciência de grupo.

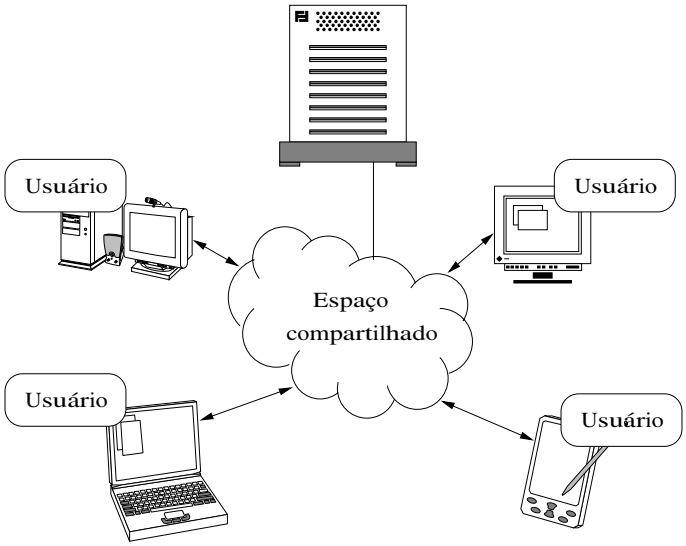


Figura 1.1: Um cenário para modelagem geométrica colaborativa.

mais crítica quando as aplicações rodam em uma plataforma computacional heterogênea. Neste tipo de plataforma, a tarefa de manter modelos consistentes se torna difícil, pois não se pode garantir que um mesmo conjunto de operações forneça os mesmos resultados, quando executadas em máquinas distintas, com diferentes sistemas operacionais e compiladores projetados com base em distintos padrões de representação de informação.

Em uma plataforma computacional heterogênea é imprescindível que uma mesma informação possa ser visualizada em distintas resoluções, de acordo com a capacidade gráfica de cada estação, e os resultados de qualquer sequência de operações sejam independentes do processo adotado para gerar as imagens, através das quais foram feitas as interações. A garantia da integridade e consistência das informações geométricas, no entanto, não são suficientes para se ter um ambiente propício à interação e/ou colaboração, quando o tempo de resposta do sistema e a percepção da presença de outros participantes são também fatores relevantes, conforme os resultados das pesquisas na área de interface humano-computador [48].

Sintetizando, enfocamos nesta tese quatro requisitos que acreditamos serem essenciais para assegurar um desempenho satisfatório de um sistema de modelagem geométrica colaborativa numa plataforma heterogênea:

1. robustez geométrica;
2. adaptabilidade;
3. interatividade;
4. consciência coletiva.

Dada a amplitude dos requisitos a serem atendidos, consideramos restrições ao cenário de colaboração no que se refere à quantidade de usuários em sessões de trabalho. Em nosso estudo, consideramos sessões de trabalhos formadas por pequenos grupos, com no máximo 5 a 10 usuários. Esta decisão é baseada em suposições de ordem prática, pois acreditamos que a maioria das atividades produtivas de modelagem são desenvolvidas por grupos reduzidos de usuários, ao contrário do que ocorre em aplicações de entretenimento, que congregam grupos formados por uma grande quantidade de usuários.

1.2 Problemas

Estabelecidos os requisitos básicos para se desenvolver uma aplicação colaborativa em modelagem geométrica, resta-nos identificar os problemas relacionados a cada um deles, para que possamos propor soluções adequadas.

1.2.1 Robustez

Inconsistências no modelo geométrico em uma plataforma heterogênea podem ocorrer decorrentes dos seguintes fatores:

① realização de um mesmo conjunto de operações em uma máquina sem um controle adequado nos arredondamentos dos valores representados em ponto flutuante;

② realização de um mesmo conjunto de operações em máquinas de arquitetura e capacidade de processamento numérica distintas, levando a decisões geométricas distintas;

③ ausência de controle de concorrência de acesso aos modelos compartilhados.

A garantia da consistência geométrica e topológica em uma máquina pode ser vista como um problema semelhante à garantia de consistência em sistemas mono-usuário, para a qual recomenda-se o uso de algoritmos robustos já existentes [1, 45]. No entanto, em um ambiente colaborativo sobre plataforma heterogênea persiste o problema da inconsistência, pois pode-se ter modelos geométricos localmente consistentes, mas diferentes entre si, mesmo que se tenha empregado algum mecanismo de controle de concorrência no acesso aos dados compartilhados e algoritmos robustos em cada cópia do modelo. Este fenômeno ocorre devido principalmente às diferentes representações numéricas e aos erros de arredondamentos nas decisões geométricas.

1.2.2 Adaptabilidade

Adaptabilidade à capacidade de processamento gráfico de cada máquina é uma propriedade essencial para interfaces com manipulação direta, uma vez que as imagens rasterizadas e exibidas intermediam as ações dos usuários (eventos) e as respostas do sistema. Além disso, deseja-se explorar maximamente os recursos das máquinas, sem sobrecarregá-las.

Em se tratando de uma plataforma heterogênea, onde as máquinas podem apresentar diferentes capacidades de processamento gráfico, o desempenho na renderização e o suporte à manipulação direta podem variar muito de uma estação para outra, embora as funcionalidades e os modelos geométricos disponíveis sejam os mesmos para todos. Neste caso, é desejável que cada participante possa personalizar a sua interface gráfica e os parâmetros de renderização de acordo com os recursos locais. Isso requer uma especificação precisa da interface interativa e mecanismos para o processamento independente em cada equipamento utilizado na colaboração.

1.2.3 Interatividade

Um aspecto relevante para qualquer usuário de aplicação interativa é o tempo de resposta da aplicação. Quando o tempo de resposta do sistema excede os valores limites aceitáveis, além dos quais o usuário chega até a se colocar em dúvida sobre o funcionamento ou não do sistema, é necessário que o sistema disponha de algum mecanismo de sinalização sobre o progresso das operações em curso.

O tempo de uma interação num sistema mono-usuário consiste essencialmente de três parcelas: tempo da captura da operação desejada pelo participante através dos eventos sobre as imagens representadas por matrizes de *pixels*, tempo de processamento da operação e o tempo de síntese da imagem (renderização e exibição) do resultado da operação numa matriz de *pixels*. Num sistema distribuído, deve-se adicionar ainda os atrasos ocorridos na comunicação entre as máquinas ao tempo total de uma interação. A redução de qualquer uma destas parcelas pode contribuir na diminuição do tempo de resposta de um sistema, e portanto, no aumento da sua interatividade.

Outro problema que surge em um espaço compartilhado 3D é a restrição quanto à visualização, forçando que todos os usuários tenham o mesmo ponto-vista da cena [34, 68]. Este problema ocorre em sistemas que compartilham a interface com o usuário, com processamento centralizado [10, 26].

1.2.4 Consciência coletiva

O trabalho colaborativo não-presencial perde muito da sua eficácia, dado que o espaço de trabalho compartilhado tridimensional é mapeado e exibido geralmente em dispositivos 2D, como monitores, enquanto a comunicação presencial se dá por meio de uma variedade de formas complementares: verbais, escritas, gestuais e tácteis. Apesar da vídeo-conferência ser uma das sugestões naturais para enriquecer a comunicação mediada por computadores, é considerada insuficiente para transmitir o foco de interesse e a tarefa de interação corrente dos participantes de uma sessão, considerados essenciais para evitar interpretações equivocadas ou ambíguas [27]. Por outro lado, não é trivial identificar quais mecanismos são suficientes para suprir as limitações do trabalho em grupo não-presencial via rede de computadores.

Um recurso muito difundido nos sistemas computacionais de apoio aos trabalhos colaborativos (CSCW) é o tele-apontamento [27]. Este recurso considera que todos os participantes compartilham a mesma vista do espaço de trabalho, o que não é o caso quando se deseja alto grau de flexibilidade no processo interativo. Necessitamos, então, descobrir informações relevantes acerca das atividades de cada membro do grupo, como capturá-las e como exibi-las num espaço "nobre" de forma concisa e clara.

1.3 Visão geral da proposta

O objetivo central deste trabalho é abordar os problemas destacados na seção 1.2, que equivale a responder as seguintes questões:

1. como garantir a robustez dos algoritmos de modelagem geométrica concebidos para sistemas mono-usuário num sistema colaborativo (multiusuário)?
2. qual deve ser a interface de um modelador geométrico de dados compartilhados para suportar técnicas de interação implementadas em cima de distintos recursos computacionais, mais especificamente, de placas gráficas de distintas tecnologias?
3. qual deve ser a granularidade de uma unidade de informação de comunicação entre as máquinas participantes de uma sessão de trabalho colaborativa?
4. como podemos melhorar a consciência coletiva de cada membro numa plataforma heterogênea onde os participantes possam ter pontos de vista distintos?

Acreditamos que uma possível solução para a questão (1) seja o projeto de uma arquitetura de *software*, que tem como característica fundamental o tratamento independente dos processos de modelagem e renderização. Isto é, as operações geométricas e os processos de renderização, manipulação e visualização são tratados separadamente. Com isso, um modelador dedicado poderia empregar algoritmos geométricos robustos utilizados em sistemas mono-usuário, sem a necessidade de adaptá-los para o sistema colaborativo, bem como mecanismos de controle de concorrência para coordenar o acesso aos dados compartilhados.

Sob o ponto de vista da interação usuário/aplicação, a arquitetura proposta permite, com o tratamento em separado da modelagem e renderização, a capacidade de exploração do poder computacional de cada equipamento individualmente. Isso facilitaria a implementação de uma interface interativa (com usuário) de fácil uso e capaz de suportar a visualização independente da cena por cada usuário e de suportar os mecanismos de interação que melhorem o tempo de resposta, solucionando as questões (2) e (3). A separação se dá pela criação de duas representações para o mesmo conjunto de objetos: o *modelo geométrico* e o *modelo gráfico*. O modelo geométrico é utilizado nas operações geométricas robustas e o modelo gráfico, uma representação alternativa do modelo geométrico, é apropriado para a renderização. Assim, os parâmetros de

renderização podem ser ajustados de acordo com a capacidade de processamento de cada equipamento. Adicionalmente, os modelos gráficos proporcionam um fraco acoplamento entre as manipulações dos usuários e o servidor de modelagem – uma característica que exploramos para minimizar o tráfego na rede a fim de manter o tempo de resposta do sistema em níveis aceitáveis de interatividade.

A solução para a flexibilização do ajuste dos parâmetros de visualização e para o aumento da granularidade da unidade de informação (soluções para as questões dois e três) impõem restrições ao processo colaborativo, uma vez que podem levar à perda de consciência coletiva e aumentar a dispersão do trabalho em grupo [33]. Para proporcionar a cada usuário conhecimento sobre localização e atividades realizadas pelos seus colegas, um conjunto de componentes de consciência pode ser valoroso. Este conjunto de componentes resolve a questão (4). Os componentes de consciência darão suporte à consciência de grupo, coletando e compartilhando informações de visualização e manipulação entre todos os participantes da sessão de trabalho.

1.3.1 Arquitetura fracamente acoplada

Partindo da hipótese de uma arquitetura de *software* baseada na separação entre modelos geométricos (representação geométrica/topológica a ser armazenada num servidor) e das imagens através das quais os participantes de uma sessão podem interagir com a cena, propomos um modelo gráfico adicional para servir como o elo de comunicação entre o servidor e as máquinas participantes.

Um único modelo geométrico colocado no servidor dedicado é compartilhado por todos os participantes através da replicação de cópias do seu modelo gráfico correspondente. A arquitetura prevê um fraco acoplamento entre os dois modelos, isto é, sobre ambos são aplicados conjuntos de operadores totalmente distintos mas consistentes entre si – um atendendo a diversidade de visualização e interação e o outro para as particularidades de modelagem geométrica, como será detalhado no Capítulo 2. Além da vantagem de reuso dos algoritmos robustos desenvolvidos para sistemas mono-usuário, destacamos ainda:

- adaptabilidade: é possível implementar para cada estação participante técnicas de interação mais apropriadas.

- modularidade: há uma interface clara entre os modelos geométricos e gráficos.
- facilidade no controle de concorrência: sendo os modelos geométricos centralizados num repositório, o gerenciamento de acessos é mais simples [28].

1.3.2 Um modelo de interação distribuída

A arquitetura fracamente acoplada, com modelador dedicado, possibilita o tratamento do problema de inconsistência geométrica numa plataforma computacional heterogênea, através de algoritmos robustos já consolidados nas aplicações mono-usuário. Em conjunto com um mecanismo de replicação passiva, ela propicia o sincronismo entre os dados gerados no modelador geométrico e as instâncias das aplicações interativas. No entanto, o acesso simultâneo aos modelos no servidor de modelagem pode aumentar o tráfego na rede, se qualquer operação sobre o modelo geométrico gera uma atualização das cópias do modelo gráfico nas aplicações interativas, e vice-versa, se a granularidade da unidade de informação entre o servidor e as máquinas participantes for muito pequena. Por outro lado, se aumentarmos em demasia a granularidade das informações, podem ocorrer discrepâncias entre as ações dos usuários nas máquinas locais e as atualizações feitas no modelo geométrico do servidor.

Propomos, como uma solução para reduzir o tráfego na rede e manter o tempo de resposta em níveis aceitáveis de interatividade, um modelo de interação distribuída baseado no uso de metáforas 3D. Esse modelo de interação introduz um controle no acoplamento entre as metáforas 3D, que são processadas localmente, e os objetos geométricos localizados remotamente no servidor de modelagem dedicado. Desta forma, manipulações realizadas somente nas metáforas não geram nenhum tráfego na rede. Partindo desta condição e da premissa de que as metáforas oferecem uma boa realimentação das ações realizadas, o modelo de interação proposto permite a realização de manipulações com realimentação visual instantânea e a efetivação controlada das operações no modelo geométrico. Por exemplo, somente após efetuado um conjunto de operações locais, a requisição da operação resultante é feita ao modelo geométrico remoto.

O compromisso entre a consistência dos dados e a interatividade no contexto de modelagem geométrica é discutido detalhadamente no Capítulo 3. Adaptamos as métricas de avaliação de tempo de respostas de sistemas interativos mono-usuário para

o contexto colaborativo e utilizamos estas métricas na análise quantitativa da interatividade [48].

1.3.3 Um modelo de consciência coletiva

Entende-se como consciência coletiva, o conhecimento sobre a interação dos outros dentro de um espaço de trabalho compartilhado. Este conhecimento inclui o entendimento de quem está no espaço de trabalho (consciência de participação), onde se está trabalhando (consciência de localização e proximidade), o que está se vendo (consciência de perspectiva) e o que está se fazendo (consciência de atividade) [34].

A consciência coletiva é essencial às interações individuais num espaço de trabalho compartilhado, dado que se deseja ter o conhecimento sobre as atividades dos outros usuários, a fim de evitar ações incoerentes ou redundantes. Ao mesmo tempo, é desejável que cada usuário possa exercer suas atividades com um grande grau de liberdade. Neste contexto, propomos um conjunto de componentes capaz de suprir cada usuário com informações específicas sobre visualização e atividades de manipulação dos demais membros do grupo, mesmo que eles não compartilhem as mesmas vistas do modelo de interesse. Detalhes dos componentes de consciência coletiva são apresentados no Capítulo 4.

O conjunto de componentes requer informações usualmente providas pelos serviços de apoio às atividades do grupo, como serviço de sessão[2], controle de concorrência no acesso às informações distribuídas e infra-estrutura de comunicação. O servidor de grupo e o aplicativo de grupo complementam, portanto, a arquitetura fracamente acoplada proposta. O primeiro é responsável pelas informações sobre a sessão e pelos mecanismos de controle de concorrência, que se utilizam da informação de sessão para promover a exclusividade de acesso aos objetos do modelo geométrico; enquanto o segundo é utilizado para a configuração da sessão e coordenação dos trabalhos.

O modelo de consciência coletiva proposto é avaliado de forma subjetiva, através dos usuários voluntários em nosso laboratório. Os critérios para esta avaliação foram adaptados de aplicações interativas mono-usuário para o contexto colaborativo.

[2]Uma sessão de trabalho caracteriza-se por um grupo de usuários trabalhando juntos através de um sistema computacional com a finalidade de resolver um problema específico.

É importante frisar que concentramos nosso trabalho de tese sobretudo nos aspectos relacionados diretamente à modelagem geométrica no contexto colaborativo, pois acreditamos que aspectos gerais de colaboração como controle de concorrência, gerência de grupo, vídeo e áudio conferências, dentre outros, têm sido explorados com sucesso em aplicações colaborativas de uso geral [19, 28, 72, 58, 59, 20].

1.4 Organização da tese

Nos Capítulos 2, 3, e 4 discutimos aspectos de implementação dos modelos propostos e as tecnologias utilizadas na nossa implementação. Para validação da arquitetura proposta, implementamos uma aplicação colaborativa simples, capaz de oferecer as funcionalidades básicas de modelagem geométrica colaborativa para demonstrar uma visão integrada dos modelos propostos. No Capítulo 5 são apresentadas as principais funcionalidades providas pelo protótipo, denominado *CoMo* (Modelador Colaborativo), construído com base nas propostas apresentadas no decorrer da tese. As conclusões e propostas para trabalhos futuros estão no Capítulo 6.

Para facilitar a leitura desta tese são incluídos ainda dois apêndices. A biblioteca de desenvolvimento de aplicações gráficas 3D *OpenGL* é descrita sucintamente no Apêndice A. No Apêndice B apresentamos um resumo do padrão CORBA para comunicação entre objetos distribuídos.

Capítulo 2

Uma arquitetura fracamente acoplada

Valeu a pena? Tudo vale a pena
Se a alma não é pequena.
Quem quere passar além do Bojador
Tem que passar além da dor.
Fernando Pessoa.

Um cenário comum para aplicações colaborativas consiste na formação de um grupo de usuários que cooperam para solução de um mesmo problema através de máquinas distintas, porém interligadas por uma rede de computadores. A diversidade dos *hardwares*, sistemas operacionais, compiladores e da infra-estrutura de comunicação torna cada vez mais difícil satisfazer o requisito da homogeneidade das máquinas participantes de uma sessão de colaboração. No contexto de modelagem geométrica colaborativa, por exemplo, alguns participantes podem utilizar estações com *hardware* gráfico dedicado de alta capacidade de processamento, enquanto outros utilizam computadores pessoais de uso geral. Apesar dos esforços para padronização [37, 38, 39], diferentes resultados podem ser gerados para uma mesma operação numérica, devido principalmente à precisão de representação e à política de arredondamento adotadas por cada arquitetura. Assim, os problemas de inconsistências geométricas e topológicas se tornam ainda mais críticos nas aplicações de modelagem geométrica colaborativa, mesmo no contexto de uma operação (atômica[1]) geométrica distribuída.

A inconsistência geométrica no contexto colaborativo pode ocorrer local ou globalmente, como será detalhado na Seção 2.1. A inconsistência local está relacionada

[1]O termo "atômica" será omitido quando não houve ambiguidade no contexto.

com as ambigüidades decorrentes das distintas seqüências de processamento sobre um mesmo objeto numa mesma máquina, levando à geração de objetos inválidos ou não-representáveis pelo esquema de representação utilizado [1, 6, 23, 35, 36, 45, 63]. Já a inconsistência global diz respeito às ambigüidades nos resultados das mesmas operações sobre as cópias de um mesmo objeto em diferentes equipamentos, mesmo que o resultado em cada máquina seja consistente. Este tipo de inconsistência pode ser ainda agravada pelo acesso concorrente a ums mesma versão de um objeto geométrico. Este é um dos problemas críticos de todas as aplicações colaborativas síncronas e tem sido tratado no contexto de aplicações colaborativas de uso geral [28, 57, 71].

A renderização e a visualização do modelo geométrico são também dependentes das características do *hardware* e do *software*. Certamente as estações de trabalho equipadas com placas aceleradoras 3D terão melhor desempenho, enquanto os usuários com equipamentos mais modestos seriam penalizados, já que teriam uma capacidade de processamento bem menor. Isso poderia comprometer a interatividade do sistema e produtividade do grupo como um todo. Uma alternativa seria nivelar a qualidade da imagem de saída pelo desempenho do *hardware* com menos recursos, penalizando os usuários que dispõem de equipamentos com melhor desempenho. Acreditamos, no entanto, que nenhuma dessas possibilidades atenderia o nosso requisito de adaptabilidade e interatividade.

Neste capítulo, propomos uma terceira alternativa que distingue duas diferentes representações fracamente acopladas[2] para objetos modelados: as informações geométricas e topológicas, que formam o *modelo geométrico* e uma representação alternativa deste modelo, usada na renderização/visualização interação, denominada *modelo gráfico*. Esta separação torna possível não só utilizar um modelador dedicado para processar o modelo geométrico adequadamente, assegurando as consistências geométricas locais e globais, como também compartilhar um modelo geométrico pelas instâncias de uma aplicação com distintas funcionalidades, viabilizando sempre a realização de uma interface gráfica interativa em cada máquina.

Na prática da atividade de modelagem é comum o envolvimento de grupos de trabalho de tamanho reduzido. Desta forma, decidimos restringir o nosso foco de estudo à atividade colaborativa envolvendo cinco a dez usuários (Seção 1.1), utilizando uma

[2]O acoplamento diz respeito às relações entre as entidades de um sistema, tais como o grau de conectividade e a dependência entre elas.

infra-estrutura de comunicação de alta velocidade.

2.1 Robustez: estudo de casos

O problema de inconsistência geométrica ocorre em aplicações de modelagem mono-usuário quando uma dada operação geométrica gera resultados ambíguos, produzindo objetos inválidos para um determinado esquema de representação. O problema torna-se mais complexo em um ambiente colaborativo sobre uma plataforma computacional heterogênea, quando uma determinada operação geométrica ou uma seqüência de operações é aplicada a diferentes cópias de um mesmo objeto. Além da possibilidade de inconsistências locais em cada instância, não se pode garantir que os resultados sejam iguais entre si, mesmo que individualmente sejam consistentes. Nesta seção apresentamos um estudo de caso para ilustrar melhor o problema.

2.1.1 Inconsistência local

Analisamos uma situação bastante comum em modelagem geométrica: o teste simples de igualdade entre três pontos A, B e C. Sob o ponto de vista teórico, este teste pode ser reduzido em dois testes de igualdade, pois pela propriedade de transitividade, se A for igual a B e B for igual a C, então A será igual a C.

Sendo as coordenadas representadas em ponto flutuante, a verificação de igualdade entre dois pontos se baseia freqüentemente na heurística ε (tolerância), pela qual dois pontos são iguais se a distância entre eles é menor do que o valor ε [9, 56]. Desta forma, dados três pontos representados por coordenadas em ponto flutuante, se $A =_\varepsilon B$ e $B =_\varepsilon C$, então seria natural inferirmos que $A =_\varepsilon C$. No entanto, a Figura 2.1 ilustra um caso em que o teste de igualdade entre A e C gera um resultado inconsistente com a nossa inferência, pois $A \neq_\varepsilon C$ mesmo que $A =_\varepsilon B$ e $B =_\varepsilon C$. Isso significa que, devido à natureza digital dos sistemas computacionais, a seqüência de operações poderia influenciar nos resultados de um processamento.

Para resolver os problemas de decisões ambíguas, vários trabalhos foram desenvolvidos com ênfase no conceito de heurística de tolerância [6, 7, 21, 44, 63]. Como conseqüência destes trabalhos, surgiram algoritmos robustos capazes de tratar de forma

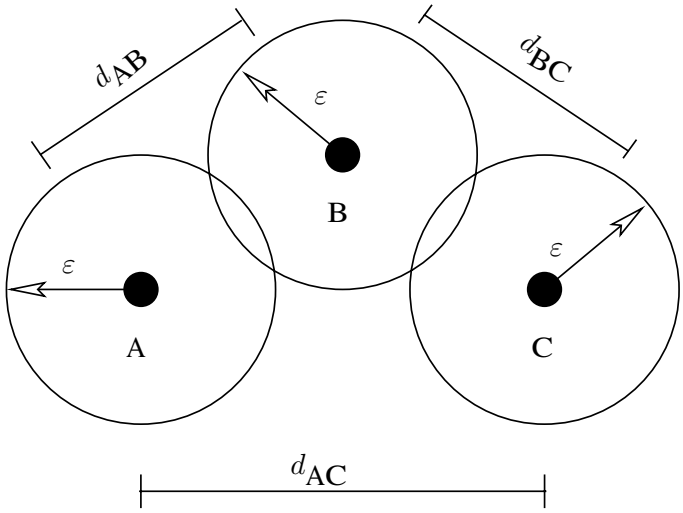


Figura 2.1: Violação da propriedade de transitividade.

consistente as imprecisões dos resultados numéricos e fornecer objetos resultantes também consistentes. Neste trabalho, propomos uma arquitetura que permite o reuso destes algoritmos no controle de consistência geométrica [21, 63] .

2.1.2 Inconsistência global

Em sistemas colaborativos para modelagem geométrica, os problemas de inconsistências podem ser mais graves, uma vez que os dados distribuídos são processados por uma plataforma computacional heterogênea e com isso, podem ocasionar o que chamamos *inconsistência global*. A inconsistência global diz respeito ao contexto distribuído, no qual um conjunto de operações aplicado às diferentes instâncias de um mesmo modelo geométrico em diferentes máquinas não garante que os objetos produzidos sejam iguais, mesmo que individualmente sejam consistentes. Para ilustrar possíveis problemas, realizamos testes de igualdade entre valores representados em ponto flutuante em diferentes máquinas, utilizando compiladores distintos e valores de tolerância ε definidos pelos respectivos compiladores. A Tabela 2.1 sintetiza os principais resultados dos nossos experimentos com diferentes precisões em representação de ponto flutuante para os ambientes *MS-Windows*, *Solaris* e *Linux*.

Para representação em precisão dupla, os valores da tolerâncias são distintos entre os compiladores, podendo acarretar também possíveis discrepâncias na realização de

S.O.	MS−Windows		Solaris		Linux
Rep. \ Comp.	DeV-C++	MSVC++	GCC-2.95	Sun-SC4.0	GCC-3.3.4
float	1,19209E-07	1,19209E-07	1,19209E-07	1,19209E-07	1,19209E-07
double	2,22045E-16	2,22045E-16	2,22045E-16	2,22045E-16	2,22045E-16
ldouble	1,08420E-19	2,22045E-16	1,92593E-34	1,92593E-34	1,08420E-19

Tabela 2.1: Valores de tolerâncias de diversos compiladores

testes de igualdade. As Tabelas 2.2 e 2.3 apresentam resultados de duas comparações. As células marcadas com (•) indicam que os valores são iguais, enquanto as células marcadas com (X) indicam que a comparação retornou um valor lógico falso. Em precisão *float* e *double*, todos os compiladores que avaliamos utilizam os mesmos valores para tolerância, garantindo a consistência dos testes de igualdade entre pares de valores.

S.O.	MS−Windows		Solaris		Linux
Rep. \ Comp.	DeV-C++	MSVC++	GCC-2.95	Sun-SC4.0	GCC-3.3.4
float	•	•	•	•	•
double	•	•	•	•	•
ldouble	X	•	X	X	X

Tabela 2.2: Teste de igualdade entre os valores 0.0 e 1.0E-17

S.O.	MS−Windows		Solaris		Linux
Rep. \ Comp.	DeV-C++	MSVC++	GCC-2.95	Sun-SC4.0	GCC-3.3.4
float	•	•	•	•	•
double	•	•	•	•	•
ldouble	•	•	X	X	•

Tabela 2.3: Teste de igualdade entre os valores 0.0 e 1.0E-20

Pelos resultados colhidos, uma solução trivial para o problema de inconsistência global seria adotar precisão simples (*float*) na aritmética em ponto flutuante, dado que os resultados dos testes mostram consistência entre os valores obtidos. No entanto, esta direção pode ser refutada quando os testes envolvem a propriedade de transitividade na comparação entre três ou mais valores, conforme ilustram as Tabelas 2.4, 2.5 e 2.6.

S.O.	MS−Windows		Solaris		Linux
Rep. \ Comp.	DeV-C++	MSVC++	GCC-2.95	Sun-SC4.0	GCC-3.3.4
float	A=B, B=C e A=C	A=B, B=C e A=C	A=B, B=C e A=C	A=B, B=C e A=C	A=B, B=C e A=C
double	A=B, B=C e A≠C	A=B, B=C e A≠C	A=B, B=C e A≠C	A=B, B=C e A≠C	A=B, B=C e A≠C
ldouble	A≠B, B≠C e A≠C	A=B, B=C e A≠C	A≠B, B≠C e A≠C	A≠B, B≠C e A≠C	A≠B, B≠C e A≠C

Tabela 2.4: Teste de igualdade entre os valores A=0.0, B=2.0E-16 e C=3.0E-16

S.O.	MS−Windows		Solaris		Linux
Rep. \ Comp.	DeV-C++	MSVC++	GCC-2.95	Sun-SC4.0	GCC-3.3.4
float	A=B, B=C e A=C	A=B, B=C e A=C	A=B, B=C e A=C	A=B, B=C e A=C	A=B, B=C e A=C
double	A≠B, B=C e A≠C	A≠B, B=C e A≠C	A≠B, B=C e A≠C	A=B, B=C e A≠C	A=B, B=C e A≠C
ldouble	A≠B, B=C e A≠C	A≠B, B=C e A≠C	A≠B, B≠C e A≠C	A≠B, B≠C e A≠C	A≠B, B=C e A≠C

Tabela 2.5: Teste de igualdade entre os valores A=-3.0E-16, B=0.0 e C=2.0E-20

Representação \ Compilador	Todos
float	A=B, B=C e A≠C
double	A≠B, B≠C e A≠C
ldouble	A≠B, B≠C e A≠C

Tabela 2.6: Teste de igualdade entre os valores A=-1.0E-7, B=0.0 e C=1.0E-7

Os resultados dos nossos experimentos reforçam a nossa hipótese de que tanto inconsistência local quanto global podem ocorrer entre os diversos compiladores e arquiteturas de computadores. Desta feita, a garantia da consistência local é condição necessária, mas não suficiente, para a robustez de uma aplicação colaborativa baseada em uma arquitetura replicada.

A comparação entre valores é uma função básica para as operações geométricas. O uso de algoritmos geométricos robustos poderia garantir apenas a consistência nos resultados em cada instância da aplicação, isto é, eles seriam independentes da seqüência de operações em cada máquina individualmente. Por exemplo, o resultado do primeiro teste na Tabela 2.6 seria que os três valores fossem iguais na precisão *float* e distintos nas precisões *double* e *ldouble*. No entanto, não se pode garantir que os resultados sejam coerentes em todas as plataformas computacionais para uma mesma precisão. Apesar do desenvolvimento de técnicas sofisticadas de robustez em operações geométricas para resolver problemas de inconsistências [45], a eficácia destas técnicas no contexto colaborativo depende geralmente da homogeneidade da plataforma computacional, o que não se pode garantir com facilidade no cenário da computação atual.

A Figura 2.2 mostra mais um exemplo de inconsistência global. É um exemplo de interseção entre um objeto formado por dois polígonos (F_1 e F_2) e um segmento de reta (E_1), realizada em máquinas diferentes e utilizando algoritmos de interseção robustos.

Realizamos estas interseções em diferentes plataformas computacionais, e obtivemos diferentes resultados. O resultado da interseção na primeira instância é um ponto sobre a aresta comum aos polígonos do primeiro objeto e o segmento de reta (Figura 2.2.(a)). Já o resultado da interseção na segunda instância da mesma aplicação são dois pontos no interior de cada polígono do primeiro objeto e a divisão do segmento de reta em três arestas (Figura 2.2.(b)). Analisando os resultados, é fácil notar que as operações fornecem resultados individualmente consistentes, mas que os objetos resultantes são topologicamente distintos entre si.

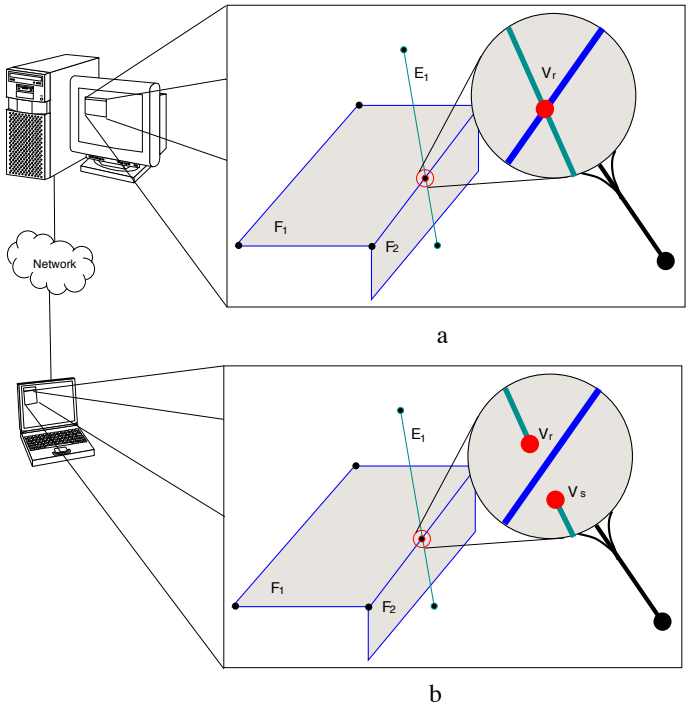


Figura 2.2: Inconsistências na interseção entre 2 objetos.

Na seção 2.3, detalhamos nossa solução de arquitetura para a garantia da consistência geométrica global do modelo geométrico. A consistência local nesta arquitetura é assegurada pelo emprego de algoritmos robustos e a consistência global, pela replicação passiva dos objetos gráficos.

2.2 Trabalhos correlatos

A classificação das diferentes arquiteturas de distribuição mostra a disposição dos diversos componentes em um sistema colaborativo, como estes componentes trocam informações entre si, bem como enumera as vantagens, desvantagens e aplicabilidade de cada uma das soluções. Uma classificação aceita e referenciada atualmente considera duas arquiteturas para aplicações colaborativas: centralizada e replicada [2, 27].

Na arquitetura centralizada, uma instância da aplicação roda em um servidor central, que controla o trabalho colaborativo, e a interface compartilhada é replicada para cada usuário do grupo. Na arquitetura replicada, em contra-partida à centra-

lizada, cada usuário dispõe de uma instância da aplicação e cópias das informações compartilhadas. As instâncias das aplicações trocam informações (dados) diretamente entre si, além do que, cada instância deve processar e exibir estas informações de forma independente das demais.

Outros trabalhos [53, 60] com ênfase na taxonomia dos sistemas colaborativos síncronos, além da preocupação com a disposição das aplicações e a troca de informações entre elas, analisam e classificam os sistemas colaborativos com base em diferentes visões da informação compartilhada. Utilizaremos a primeira classificação como base para discussão e apresentação do nosso trabalho, pois é mais aceita e referenciada pela literatura e tem servido de base para a maioria dos trabalhos na área.

A escolha de uma arquitetura adequada está diretamente relacionada aos requisitos, ao contexto da aplicação e ao compromisso entre consistência e tempo de resposta do sistema. No contexto de modelagem geométrica, a consistência da informação é um requisito prioritário, ao passo que em sistemas interativos é o tempo de resposta. Temos um impasse, pois o contexto deste trabalho é a modelagem geométrica colaborativa síncrona, ou seja, engloba aplicações interativas.

Em um sistema centralizado, como ilustra a Figura 2.3, a consistência em cada transação pode ser garantida com menos esforço, dado que se tem uma única cópia dos objetos compartilhados. No entanto, pode ter que se pagar o preço da perda de interatividade inerente ao tráfego (tempo resposta) e da restrição na visualização (tipicamente WYSIWIS[3] [27]), dado que todos os usuários teriam o mesmo ponto de vista da cena tridimensional. Além disso, pela natureza desta arquitetura, alguns equipamentos poderiam ser sobrecarregados, enquanto outros poderiam ser sub-utilizados [71]. No primeiro caso, devido à incapacidade de equipamentos de uso geral em exibir adequadamente cenas complexas, produzidas para serem visualizadas por equipamentos com alta capacidade de processamento gráfico, e no segundo, quando cenas mais pobres são geradas para atender aos requisitos de equipamentos de uso geral, provocando a sub-utilização de equipamentos com alta capacidade de processamento gráfico.

Um sistema baseado em arquitetura replicada, cujo esquema é ilustrado na Figura 2.4, tende a fornecer respostas mais rápidas às ações dos usuários, e portanto, apresenta maior interatividade do que em sistemas centralizados [8, 53]. Nesta arquite-

[3]Paradigma WYSIWIS (What You See Is What I See), no qual todos os usuários tem a mesma visão da cena.

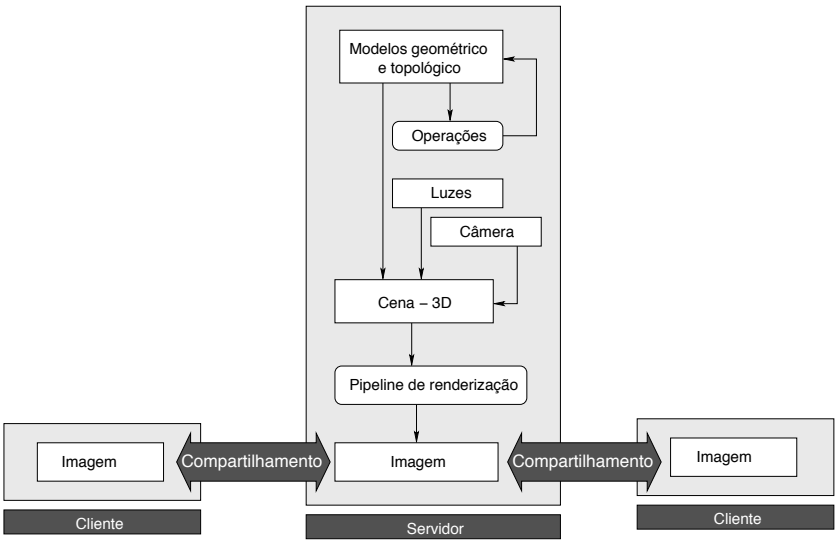


Figura 2.3: Arquitetura centralizada.

tura, cada aplicação interativa armazena e processa localmente os objetos geométricos e o tráfego fica restrito apenas às mensagens com informações sobre quais operações devem ser realizadas pelas instâncias do modelador. Além disso, a renderização e a visualização são realizadas de forma independente para cada instância da aplicação interativa, dando maior flexibilidade ao trabalho individual de cada usuário em relação à cena, pois cada um pode independentemente ajustar os seus parâmetros de visualização.

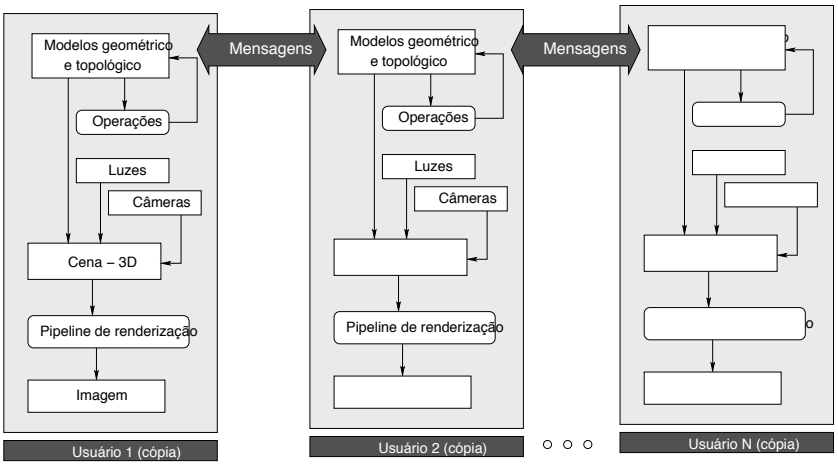


Figura 2.4: Arquitetura replicada.

No entanto, em um ambiente computacional heterogêneo a consistência dos ob-

jetos geométricos é extremamente difícil de ser alcançada em cada transação por um modelador colaborativo baseado na arquitetura replicada, mesmo que se utilize algoritmos geométricos robustos nas instâncias do modelador. Isto deve-se ao fato de que tais algoritmos geométricos garantem somente a consistência local de cada réplica e não a consistência global de todas as réplicas derivadas de um mesmo conjunto de objetos, uma vez que em uma plataforma computacional heterogênea as operações geométricas decorrentes de decisões baseadas na computação em ponto flutuante não necessariamente produzem resultados iguais em todas máquinas.

A concepção de aplicações colaborativas baseadas nestas duas arquiteturas básicas produz diferentes gargalos: de um lado, a restrição na visualização e tempo de resposta crítico na arquitetura centralizada, e do outro, a dificuldade na garantia da consistência da informação na arquitetura replicada, agravada pela heterogeneidade das plataformas computacionais muito comum atualmente [5, 27]. Isso nos motivou a adotar uma arquitetura que consiste essencialmente em separar a representação dos objetos nos modelos geométrico e gráfico, e acoplá-los fracamente.

2.3 Uma arquitetura de modelos fracamente acoplados

Motivados pelo requisito de heterogeneidade da plataforma computacional, propomos uma solução de arquitetura para suporte à resolução do problema de inconsistência geométrica global, através da criação de duas abstrações: o modelo geométrico, que retém as informações geométricas e topológicas; e o modelo gráfico, uma correspondência do modelo geométrico contendo informações necessárias para renderização e visualização (Figura 2.5).

A separação dos modelos geométrico e gráfico tem como objetivo tratar separadamente o problema da inconsistência geométrica e os processos de renderização, visualização e manipulação 3D dos objetos. Assim, pode-se ter um núcleo de modelagem dedicado, no qual as informações geométricas são processadas de forma centralizada em um servidor de modelagem. A renderização, a manipulação e a visualização do modelo gráfico passam a ser realizadas de forma independente em cada instância da aplicação interativa. Desta forma, os parâmetros de visualização e renderização podem ser ajustados de acordo com a capacidade individual de cada equipamento. Com isso,

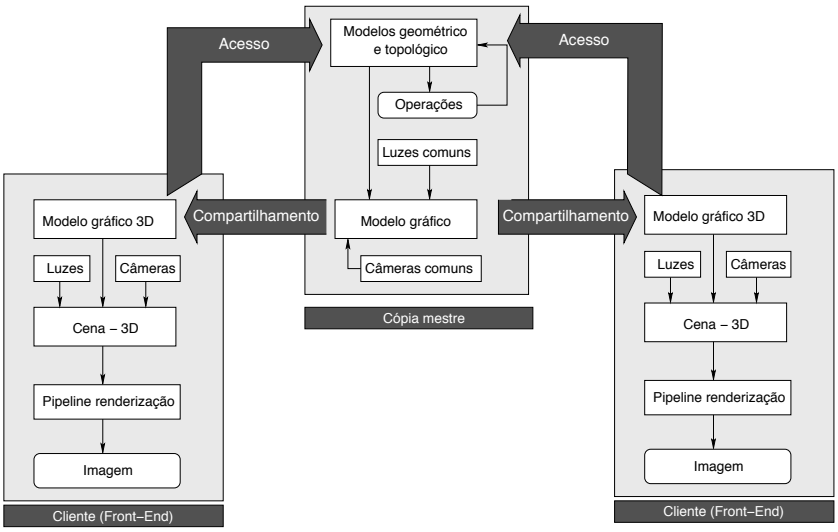


Figura 2.5: Visão da arquitetura híbrida

pretende-se mostrar que esta abordagem permite tratar o problema da inconsistência geométrica e adaptabilidade através de uma solução de arquitetura de sistema.

Para garantir a consistência geométrica local, propomos aliar à arquitetura algoritmos robustos provenientes de aplicações mono-usuário nas operações geométricas realizadas no modelador dedicado, sem qualquer necessidade de concentrar esforços na adaptação ao ambiente distribuído. A *priori* poder-se-ia empregar qualquer técnica de robustez geométrica: computação simbólica [18], heurística de tolerância ε [7, 21], técnicas de perturbações [70], aritmética intervalar e afim [14], dentre outras [45, 1]. A escolha do método adequado depende dos requisitos da aplicação.

Já para explorar adequadamente os recursos computacionais locais, cada instância da aplicação interativa pode ajustar independentemente os parâmetros de renderização e visualização, de acordo com a capacidade de cada *hardware*. O modelo gráfico, gerado a partir do modelo geométrico, é replicado passivamente para as instâncias das aplicações interativas onde é renderizado e visualizado com os recursos locais. No caso de aplicações interativas críticas, os parâmetros de tempo de resposta exercem forte influência na escolha da qualidade das imagens finais.

Sob o ponto de vista funcional, a arquitetura pressupõe a construção de pelo menos duas aplicações para o ambiente de modelagem colaborativa: um servidor de modelagem e a aplicação interativa. A primeira é responsável pelo processamento geométrico,

enquanto a segunda renderiza os objetos gráficos e fornece a interface para os usuários do sistema.

2.3.1 Servidor de modelagem

O servidor de modelagem provê os serviços relativos à modelagem geométrica propriamente dita, com a manutenção dos modelos geométrico e gráfico. Além disso, sob o ponto de vista funcional, o servidor é responsável pela replicação do modelo gráfico e interação com os mecanismos de controle de concorrência. A Figura 2.6 destaca a estrutura básica do servidor de modelagem geométrica.

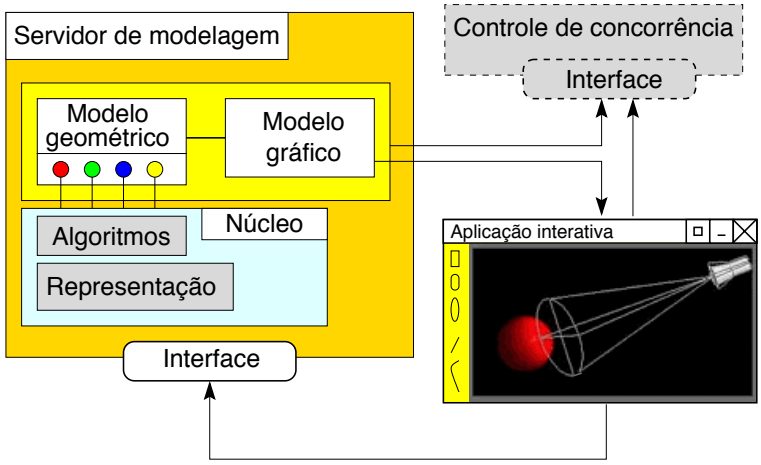


Figura 2.6: Servidor de modelagem geométrica.

Devido à grande variedade de esquemas de representações e algoritmos geométricos para diferentes aplicações, torna-se praticamente impossível contemplar todas possibilidades de representação em um único núcleo de modelagem. Desta forma, o foco deste trabalho não é prover um esquema de representação unificado (ou neutro), mas propor uma arquitetura para modelagem geométrica colaborativa com ênfase na garantia de consistência dos objetos geométricos em cada transação, utilizando algoritmos robustos e estratégias de controle já consolidados em aplicações mono-usuário, sem a necessidade de adaptá-los ao contexto colaborativo. Desta feita, o servidor dispõe de uma interface de acesso ao modelo geométrico, permitindo a inclusão de um núcleo de modelagem (esquemas de representação e respectivos algoritmos geométricos) de acordo com os requisitos e funcionalidades de cada aplicação.

Para distinguir os algoritmos dependentes dos não-dependentes das aplicações, classificamos os componentes da nossa arquitetura em aspectos imutáveis e aspectos configuráveis da arquitetura. Entende-se por aspectos imutáveis (*frozen spots*) aquelas funcionalidades providas pelos componentes do servidor de modelagem que não estão sujeitas à alterações quando se projeta uma aplicação. O modelo gráfico é um exemplo de ponto imutável por prover uma estrutura uniforme para serialização, deserialização e replicação para qualquer esquema de representação. Os aspectos configuráveis (*hot-spots*) são interfaces, segundo um conjunto de regras de reuso, para a inserção de funcionalidades de acordo com os requisitos de uma aplicação. O modelo geométrico é um aspecto configurável do servidor de modelagem, pois permite a integração de diferentes esquemas de representação.

2.3.1.1 Modelo gráfico

Os objetos enviados do modelador geométrico para as instâncias das aplicações interativas são cópias das representações alternativas do modelo geométrico, eliminadas as informações desnecessárias à renderização e interação. Estes objetos, denominados objetos gráficos, são representados através de um conjunto de primitivas básicas, com geometria simples e adequada à renderização. O conjunto destes objetos forma o modelo gráfico. Para que as aplicações interativas possam renderizar e exibir qualquer objeto geométrico é necessário que o desenvolvedor da aplicação implemente um método de conversão para a representação gráfica.

A coleção de primitivas, operações e algoritmos providas por um modelador geométrico é muito mais especializada do que a provida pelas bibliotecas gráficas 2D e 3D mais populares. Desta forma, a maioria das bibliotecas gráficas distinguem em torno de três classes de primitivas gráficas: pontos, segmentos de reta e polígonos. Algumas delas, como o padrão gráfico PHIGS [66], Java3D[67] e Open Inventor[77], permitem organizar estas primitivas numa estrutura hierárquica para facilitar a construção de objetos complexos a partir dos blocos elementares.

Propomos que o modelo gráfico seja um ponto imutável da plataforma, cuja estrutura hierárquica pode ser representada através de um padrão de projeto estrutural denominado *composite* [25]. A classe base (abstrata) *DigGraphics* guarda os atributos e uma interface comum às suas subclasses. Ao passo que as subclasses *Ponto*,

Segmento e *Polígono* representam as folhas da estrutura hierárquica, que preenche os requisitos da classe base através da implementação da sua interface. A classe *DigComposite* é um agregado recursivo da classe *DigGraphics* e suas subclasses, o que permite instanciar uma hierarquia complexa na descrição de objetos geométricos, utilizando primitivas simples com um tratamento uniforme proporcionado pela interface comum, Figura 2.7. A classe *DigGraphics* define as seguintes operações primitivas abstratas: método de notificação ampla (Seção 2.4.2) e o método de transferência de dados para máquinas participantes (Seção 2.5.3), denominado *streamming* e o método que reconstrói os dados recebidos chamado *unstreamming*. Os demais métodos públicos da interface são relativos ao padrão *Composite* e são empregados para manipulação das primitivas geométricos:

- virtual void Add(DigGraphics*) = 0;
- virtual void Remove(DigGraphics*) = 0;
- virtual void Remove(const int) = 0;
- virtual void Update(DigGraphics*) = 0;
- virtual DigGraphics *Get(const int) = 0;

O método *Get* retorna o apontador para um determinado objeto identificado por seu ID (inteiro). Note que todos os métodos são abstratos puros, o que exige usas implementações para cada nova primitiva adicionada.

2.3.1.2 Modelo geométrico

A premissa da nossa arquitetura é que o modelador geométrico no servidor de modelagem seja robusto, assegurando a consistência das operações geométricas sobre a única instância do modelo geométrico em uma sessão de trabalho, através de transações atômicas.

A separação entre o modelo geométrico (aspecto configurável) e o modelo gráfico (aspecto imutável) impõe a especificação de uma interface entre os dados geométricos, especializados para cada aplicação, e os objetos gráficos correspondentes. Para ser o menos dependente possível da aplicação, propomos utilizar o padrão de projeto *template*

method [25] para definir uma interface abstrata para o modelo gráfico (*DigModel*), deixando a cargo da aplicação prover adequados algoritmos de conversão (*convert()*) para realizar a conversão das entidades do modelo geométrico para as entidades do modelo gráfico, Figura 2.7. Outros métodos declarados e definidos na classe *DigModel* são mostrados na seção 2.5.1.

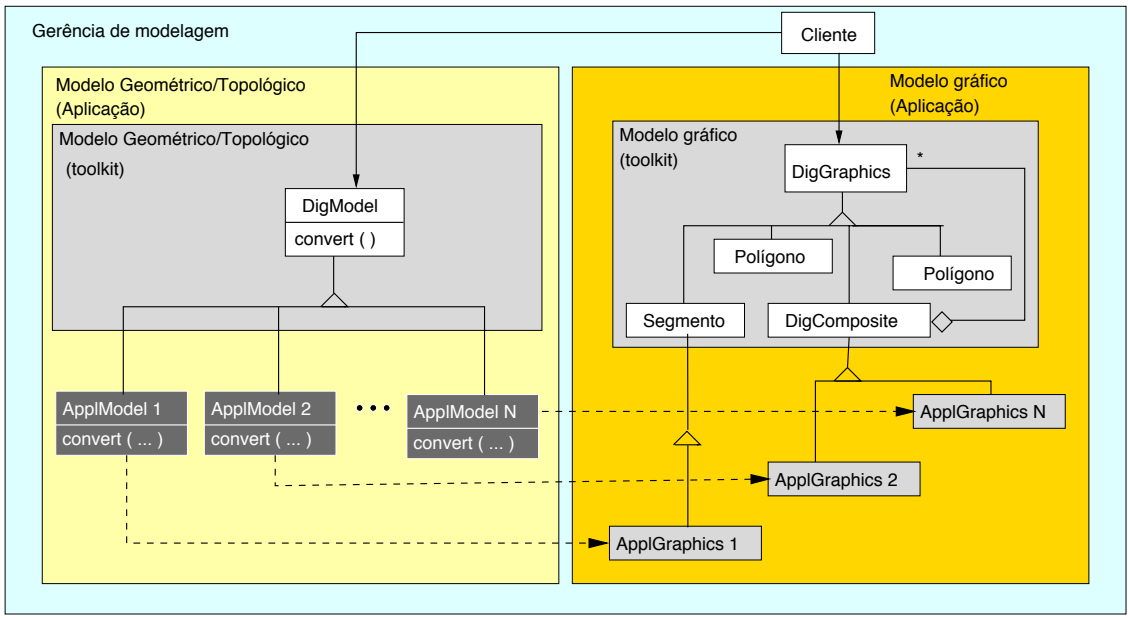


Figura 2.7: Relação entre os modelos geométrico e gráfico

2.3.2 Aplicação interativa 3D

Denominamos uma aplicação interativa 3D a ferramenta utilizada pelos usuários de uma sessão para visualizar e manipular diretamente o modelo geométrico 3D. Esta aplicação processa o modelo gráfico proveniente do servidor de modelagem e provê um conjunto de facilidades, que inclui a renderização da cena e os mecanismos de coleta de eventos externos e de suporte à comunicação com o servidor de modelagem (Figura 2.8).

É importante ressaltar que na nossa arquitetura os usuários nunca interagem, de fato, com o modelo geométrico, mas sim com uma cópia do seu representante – o modelo gráfico. Para que a semântica das interações diretas dos usuários seja captada corretamente pelo servidor de modelagem, proporcionando uma percepção de estar atuando diretamente sobre o modelo geométrico, propomos o uso de um conjunto de

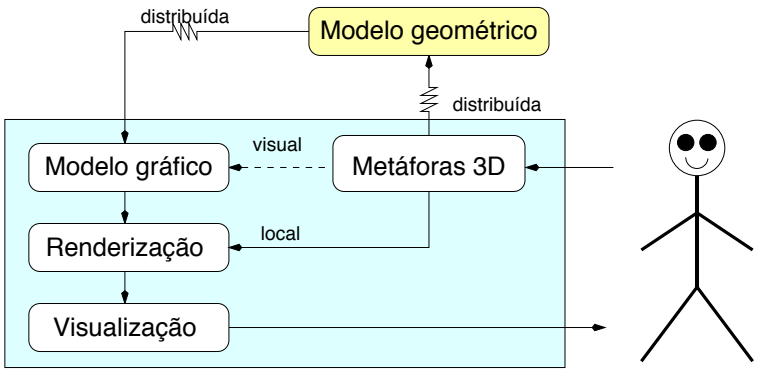


Figura 2.8: Aplicação interativa 3D.

metáforas de manipulação 3D. Tendo estas metáforas uma semântica pré-estabelecida, é importante ressaltar que o preço desta solução é, à primeira vista, a redução do escopo de interações dos usuários às metáforas. No entanto, no Capítulo 3, mostraremos um modelo de interação entre usuário, objeto gráfico e objeto geométrico, que, através do uso intensivo de metáforas de interação, consegue minimizar a granularidade de comunicação entre as aplicações 3D e o modelo geométrico, sem onerar em demasia a qualidade da realimentação visual.

2.4 Infra-estrutura de comunicação

A separação física entre o modelo geométrico residente num servidor e os modelos gráficos replicados para as diferentes instâncias da aplicação 3D, requer a definição de uma infra-estrutura de comunicação que possa assegurar

- acesso às funcionalidades do modelo geométrico remoto;
- manutenção da consistência entre os modelos e as réplicas.

2.4.1 Acesso remoto

Para que haja uma comunicação transparente entre os objetos gráficos e os respectivos objetos geométricos remotos, de modo a permitir a troca de informações entre as instâncias das aplicações colaborativas e o servidor de modelagem, empregamos os

padrões *proxy* e *broker*. Cada *proxy* fornece um objeto-representante da interface remota do modelador geométrico para a aplicação interativa, possibilitando o acesso aos serviços remotos, como criação, remoção e manipulação de qualquer objeto geométrico do modelador dedicado.

O *proxy* mantém sempre uma referência para o objeto real, além de se responsabilizar pela codificação das requisições e seus argumentos e pelo envio destes para o objeto real, em um espaço de endereço diferente, através de um *broker*. O *proxy* utiliza a interface comum, tanto para o acesso ao objeto quanto para prover os serviços ao cliente. Ou seja, os métodos do *proxy* são os mesmos do objeto real, tornando fácil a tarefa de manter transparente a comunicação entre a aplicação interativa e o modelador dedicado (Figura 2.9).

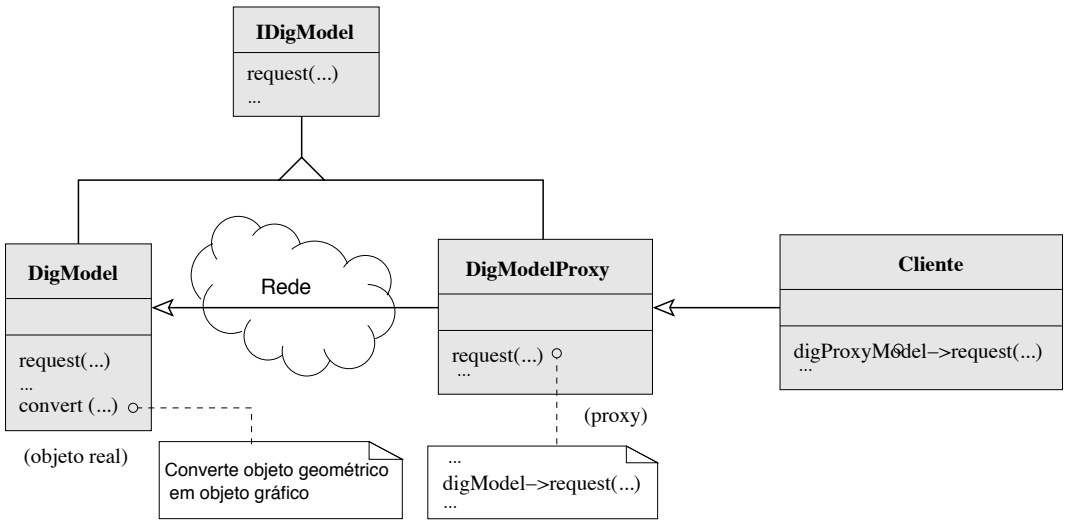


Figura 2.9: Estrutura de *proxy* remoto

O *broker* fornece uma infra-estrutura de comunicação sobre uma plataforma distribuída, estabelecendo uma conexão lógica, de forma transparente, entre os *proxies* e os correspondentes objetos reais no servidor de modelagem. Assim, as instâncias da aplicação interativa podem ter acessos ao núcleo de modelagem através da invocação usual dos métodos dos objetos, sem se preocupar com a implementação dos canais de comunicação e os protocolos de comunicação (Figura 2.10).

A propagação da requisição, serialização e deserialização dos parâmetros dos métodos dos objetos remotos são realizadas em conjunto pelo *broker* e pelo *proxy*. Resta,

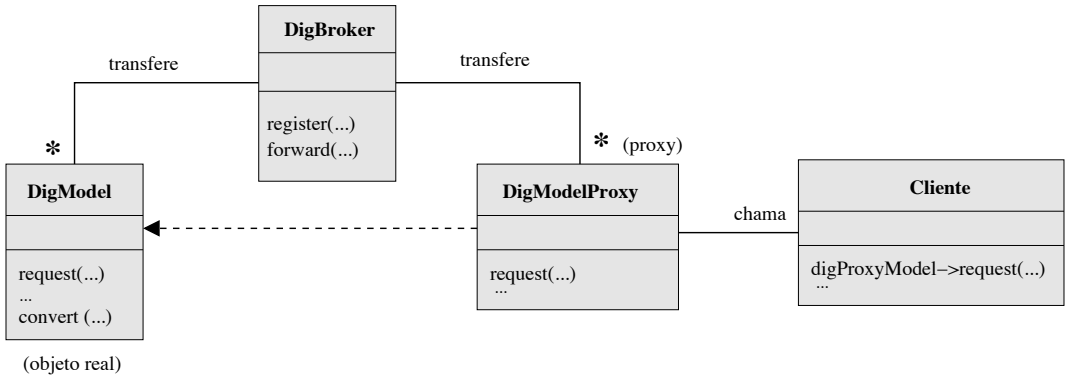


Figura 2.10: Padrão de projeto *broker*.

durante a fase de realização, utilizar alguma implementação de *broker* capaz de efetuar esta comunicação entre diferentes espaços de endereçamento de forma transparente.

2.4.2 Consistência entre os modelos

No processo colaborativo, admitimos a existência de uma instância da aplicação interativa para cada usuário participante de um grupo de trabalho. Desta feita, os objetos gráficos gerados no modelador geométrico precisam ser sincronizados com as instâncias da aplicação interativa. Este mecanismo é denominado replicação. A estratégia de replicação é de fundamental importância para a manutenção da consistência entre o modelo geométrico, o modelo gráfico e as suas cópias em cada transação.

Nossa estratégia de replicação foi elaborada a partir das seguintes premissas:

① maximizar a exploração dos recursos computacionais – em aplicações gráficas é muito comum utilizar-se de recursos de *hardware* dedicado para melhorar o desempenho do sistema. Cada instância da plataforma utiliza os recursos que dispõe e processa localmente a informação gráfica - renderização e visualização.

② permitir a independência na visualização – cada usuário pode manipular livremente o volume de visualização, utilizando transformações locais de visualização (acoplamento flexível de aplicações).

③ não onerar a rede – já que pode-se adotar uma comunicação baseada em eventos e

realizar algumas operações localmente, ao invés de cada operação ser processada pelo servidor e as réplicas da aplicação apenas exibir as informações.

④ propiciar um fraco acoplamento entre usuários – permitir o ingresso e saída de usuários de forma transparente durante uma sessão de trabalho.

O fato de replicarmos o modelo gráfico, ao invés de replicarmos o resultado da síntese do modelo (suas imagens), nos proporciona maior flexibilidade na especificação de "o quê deve ser visto" e "como ele deve ser visto" em cada máquina participante, atendendo os requisitos (1) e (2). Além disso, tendo cada máquina participante uma cópia do modelo gráfico, não é necessário transferí-lo integralmente do servidor para cada instância de aplicação em todas as atualização. Muitas vezes, o fluxo de dados pode se restringir ao fluxo de um objeto geométrico modificado e/ou às operações geométricas aplicadas. Com isso, o ponto (3) é também contemplado.

Para flexibilizar a mobilidade dos participantes de uma sessão de trabalho (ponto (4)), propomos adotar o padrão de projeto *Mediator* [25] para interrelacionar o servidor de modelagem e as instâncias de aplicação. Este padrão define o objeto FloorControl que encapsula a forma de interação entre um conjunto de elementos, promovendo um fraco acoplamento entre eles no sentido de que eles não precisam se referenciar explicitamente [25]. Ou seja, o número de instâncias da aplicação não precisa ser conhecido pelo servidor de modelagem. É o mediador quem conhece todas as instâncias e coordena as suas interações com o servidor.

A atualização de um mesmo modelo geométrico por mais de um usuário requer uma política de controle de concorrência para evitar contenção de recursos. O controle de concorrência poderia ser baseado na coordenação do acesso concorrente ao modelo geométrico através de alguma técnica de sincronização de processos, como por exemplo, semáforo ou monitores que impõem o acesso exclusivo ao modelo geométrico rotulando a parte "em uso" como região crítica [24]. No entanto, nós propomos utilizar um controle mais elaborado que se baseia em travas individuais usuário/objeto. Neste paradigma, cada objeto geométrico em manipulação é associado a um único usuário, sem que os outros usuários percam o acesso consciente de leitura. Este mecanismo é denominado de controle de bastão [15]. Dentre as políticas de controle mais comuns, adotamos, sem perda de generalidade, a política de requisição e posse. Esta política permite que um membro faça uma requisição e tome posse imediatamente do bastão de um objeto,

possivelmente preemptando o dono atual do bastão. O FloorControl, dentre as suas funções de coordenação das interações, intermedia o acesso dos objetos geométricos pelas distintas instâncias de aplicação, sem que estas precisem ter o conhecimento do estado de uso de todos os objetos.

Para manter a consistência entre o modelo geométrico, o modelo gráfico e suas cópias, propomos incluir na nossa arquitetura dois sistemas de replicação: ativa no sentido de aplicação para servidor, ou seja, cada aplicação pode requisitar a atualização do modelo geométrico, sob demanda; e passiva no sentido do servidor para aplicação, ou seja, qualquer atualização no modelo geométrico implica sempre em atualização no modelo gráfico. A nova versão do modelo gráfico é postada automaticamente no canal de comunicação entre o servidor e as máquinas participantes.

Sob o ponto de vista conceitual, a replicação passiva pode ser caracterizada como uma relação 1:N (lê-se: 'um para n') entre o modelo gráfico armazenado no modelador e suas N cópias[4] nas instâncias da aplicação interativa. Este relacionamento entre objetos, estabelecido pela dependência entre um objeto que muda o estado e um conjunto de objetos que necessita ser notificado automaticamente, pode ser descrito e modelado através do padrão de projeto *Observer* [25], Figura 2.11.

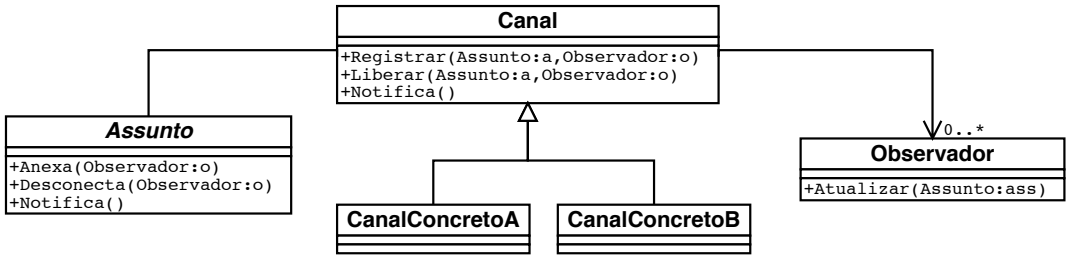


Figura 2.11: Padrão de projeto *Observer*

Dois objetos relevantes para o padrão *Observer* são: o *observador* e o *assunto* de interesse. Instâncias do objeto observador, que mantém interesse no assunto, são notificadas quando o estado do assunto se altera. O grau de acoplamento entre observadores e assunto é naturalmente fraco, pois geralmente o assunto não precisa conhecer os seus observadores. Desta forma, insere-se um elemento entre estes objetos para que seja implementada uma comunicação assíncrona entre ambos. Isto é obtido através da

[4] número de membros de uma sessão de trabalho.

introdução de um objeto denominado *canal de replicação* entre assunto e observadores. Assim, o objeto *canal* é, de fato, o mediador entre o assunto e os seus respectivos observadores. O canal, além de registrar assuntos e observadores para efetuar a troca de informações entre eles, repassa as notificações para os observadores. Desta forma, os padrões *Observer* e *Mediator* compõem o núcleo básico do componente de replicação.

Em nosso caso, o assunto é qualquer objeto do modelo gráfico no servidor de modelagem, enquanto os observadores são as suas réplicas nas instâncias da aplicação interativa. Qualquer mudança no modelo gráfico do servidor provoca a transferência dos objetos modificados para o canal do qual todas instâncias da aplicação ficam à escuta.

2.5 Uma visão de implementação

Figura 2.12 fornece uma visão da realização da arquitetura proposta neste capítulo, com destaque para o servidor de modelagem, que contém tanto o modelo geométrico como a cópia original do modelo gráfico. Três instâncias da aplicação interativa também são mostradas.

Computadores neste ambiente de rede podem diferir em arquitetura de *hardware*, sistemas operacionais e linguagens de programação usados para implementar os objetos. *Middlewares* facilitam a implementação entre os objetos neste cenário. Optamos por utilizar na nossa implementação o padrão CORBA (*Common Object Request Broker Architecture*) [50, 75] como *middleware* para a comunicação cliente/servidor. Ao padrão CORBA, agregamos o serviço de eventos CORBA, com o qual implementamos as facilidades de comunicação de grupo.

A biblioteca OpenGL é empregada para o acesso às funcionalidades relativas à renderização de cena 3D realizadas pelo *hardware* gráfico.

2.5.1 Comunicação cliente/servidor

Para realizar a comunicação entre os objetos distribuídos na arquitetura, utiliza-se um *middleware*. A escolha de um *middleware* adequado é uma tarefa consideravelmente complexa. Dentre os candidatos mais viáveis destacam-se RPC, RMI, DCOM e CORBA [12].

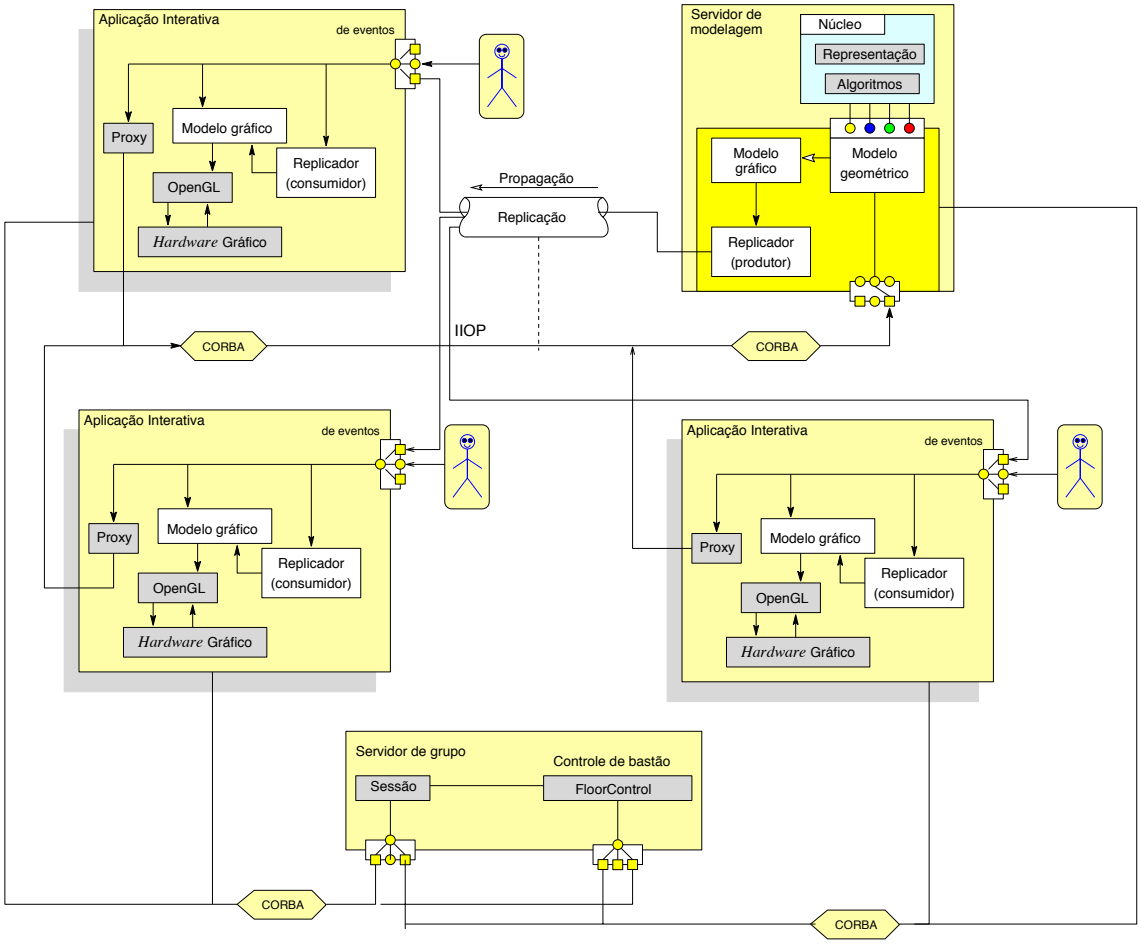


Figura 2.12: Visão geral da arquitetura fracamente acoplada.

O RPC (*remote procedure call*) possibilita a interconexão de processos através da chamada a procedimentos remotos, e também resolve o problema da formatação dos dados, pois realiza esta tarefa de forma transparente para a aplicação. Embora RPC seja bastante utilizado em sistemas operacionais Unix, não existe uma padronização para os demais sistemas operacionais. Esta restrição fere diretamente o requisito de heterogeneidade estabelecida para o ambiente proposto.

Semelhante ao RPC, o RMI realiza a interconexão entre processos em máquinas distintas através de invocação de métodos remotos. A diferença entre RPC e RMI é que no segundo a comunicação é realizada entre objeto e é implementado na linguagem Java. No entanto, por razões de baixa eficiência, Java não foi considerada no nosso projeto.

Dentre as tecnologias de interconexão de objetos, DCOM tem sido uma das mais

empregadas na comunicação entre objetos distribuídos. Ela permite a invocação de métodos remotos. No entanto, DCOM é uma tecnologia fechada, empregada na comunicação de objetos implementados sobre o sistema operacional Microsoft Windows. Com isso, não atende ao requisito básico de heterogeneidade de plataforma computacional.

Diante das desvantagens das alternativas já descritas, consideramos o padrão CORBA do consórcio OMG [50], como o canditato mais adequado para prover a comunicação na implementação da arquitetura fracamente acoplada. CORBA oferece a comunicação entre objeto remotos, interoperabilidade entre plataformas computacionais e linguagens de programação distintas, a formatação transparente dos dados e a possibilidade de tunelar requisições CORBA via HTTP para ultrapassar as barreiras de segurança na comunicação entre aplicações em diferentes redes. Além disso, um serviço de nomes é agregado ao padrão CORBA para resolver o problema de endereçamento e o serviço de eventos para a realização da comunicação de grupo.

A interoperabilidade é decorrente do uso de protocolos padronizados, no caso GIOP e sua versão para Internet - IIOP e da especificação de uma linguagem de definição de interface remota denominada CORBA/IDL (*Interface Definition Language*), que possui uma sintaxe muito próxima da linguagem C++. A escolha de uma implementação CORBA é feita de acordo com as necessidades de cada projeto, dado que cada implementação pode dispor de apenas um subconjunto das características especificadas pelo OMG. Para o nosso projeto, além do núcleo da plataforma CORBA, o serviço de eventos e o serviço de nomes foram determinantes para a escolha de uma implementação denominada MICO [55].

A linguagem IDL suporta os tipos básicos e os definidos pelo usuário, os quais são utilizados nas assinaturas dos métodos dos objetos remotos, definindo assim a interface distribuída dos serviços da plataforma. Os tipos e as interfaces IDL são mapeados, por um compilador IDL, para tipos e classes em qualquer linguagem de programação orientada a objetos, como C++, Java, entre outras. Por exemplo, a interface remota do modelador geométrico é especificada em IDL da seguinte forma:

```
interface DigModel {
  // object manipulation method
  void create  (in UserId uid, inout ObjId id, in ObjType objtype);
  void remove (in UserId uid,in ObjId id);
```

```
  void freetoken (in UserId uid,in ObjId id);
  void getToken (in UserId uid,in ObjId id);
  void setColor (in UserId uid,in ObjId id, in SColor color);
  void setMaterial (in UserId uid,in ObjId id, in SMaterial color);
  void setProperty (in UserId uid,in ObjId id, in string obj);
  //transformation
  void move (in UserId uid,in ObjId id, in SVector3 v);
  void scale (in ObjId id, in SPoint3 p, in SVector3 v, in double f);
  void rotate (uid,in ObjId id, in SPoint3 p, in SVector3 v,
               in double a);
  // generic geometric transformation
  void transform (in UserId uid,in ObjId id,in TSMatrix4 matrix);

  void extrude (in UserId uid, inout ObjId parid, inout ObjId id, in
                seqPoint3 seqpoints, in SVector3 v);

  // convert: geometric model --> graphics model
  void convert(in UserId uid);

  // boolean operations
  void booleanop (in UserId uid, in ObjId id1,
                  in ObjId id2, inout ObjId oid,
                  in  unsigned long boper);
  //file manipulation
  SeqOfString dirlist (in UserId uid, in string dirname);
  boolean load (in UserId uid, in string filename);
  boolean save (in UserId uid, in string filename);
...
};
```

Alguns métodos da interface foram omitidos por questões de clareza do texto. Note que *UserId*, *ObjId*, *ObjType* são tipos redefinidos a partir de tipos primitivos e os modificadores de parâmetros *in*, *out* e *inout* indicam que os parâmetros são de entrada,

saída ou ambos, respectivamente. Após processar a interface através do compilador IDL, são gerados as interface remotas e os respectivos *proxies*[5]. As interfaces são implementadas no servidor (*DigModelImpl*), enquanto o *DigModelProxy* é utilizado diretamente nas aplicações interativas, para acesso remoto ao servidor.

2.5.2 Replicação passiva baseada em CORBA

O serviço de replicação passiva foi implementado sobre o serviço de eventos CORBA [30]. Neste serviço distinguem-se 3 tipos básicos de objetos e dois modelos de comunicação. Os objetos são os produtores, consumidores e o canal de eventos. Os produtores fornecem os dados (eventos), enquanto os consumidores prcessam estes dados. O canal de eventos é um objeto CORBA, com interface especificada em IDL, que promove comunicação indireta e assíncrona entre produtores e consumidores, garantindo o baixo acoplamento entre essas entidades. Os dois modelos de comunicação são o modelo *push* e o modelo *pull*. No modelo *push* os produtores tomam a iniciativa na comunicação postando os eventos no canal para serem coletados pelos consumidores, enquanto no modelo *pull* os consumidores iniciam a comunicação requisitando os eventos dos produtores.

Na nossa implementação utilizamos o modelo *push* para realizar o serviço de replicação passiva[6]. O servidor de modelagem (produtor) serializa os objetos gráficos e os entrega para o replicador de eventos, que os posta no canal de eventos. As instâncias da aplicação interativa (consumidor) coletam as informações colocadas no canal ao qual estão conectadas, Figura 2.13.

Consideramos ainda que uma transação somente é considerada finalizada depois que a cópia mestre do modelo gráfico for postada no serviço de eventos. Com isso, a replicação passiva do modelo gráfico fornece suporte à garantia da consistência dos objetos modelados, de forma simples e eficiente.

[5] No jargão CORBA, utiliza-se a expressão *stub* para designar esta entidade.

[6] Denominamos replicação passiva quando a iniciativa da comunicação é do assunto de interesse, ou seja o produtor.

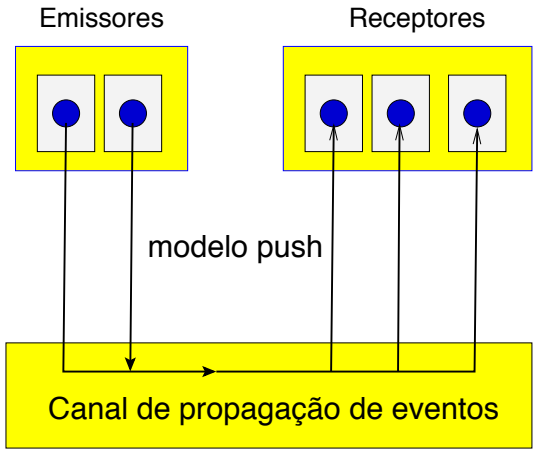


Figura 2.13: Replicação de objeto usando o serviço de eventos CORBA

2.5.3 Formatação e transmissão de dados

Os objetos gráficos são transmitidos na notificação de eventos. Para tanto, estes objetos devem ser colocados em um formato adequado para transmissão e recuperados no destino, através da serialização (ou externalização) e deserialização (ou internalização).

A realização da externalização é obtida através da gravação do estado (valores dos atributos) do objeto em um formato linear e contíguo, pronto para ser transmitido (ou salvo em memória, arquivos, etc.). O processo de internalização ocorre de forma inversa quando objetos são recuperados através da reconstrução do seu estado original, Figura 2.14.

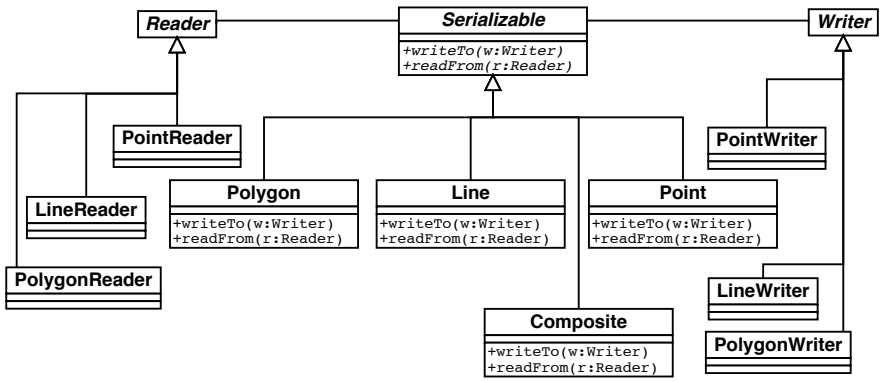


Figura 2.14: Serviço de externalização e internalização.

As primitivas do modelo gráfico herdam da classe *Serializable*. Assim, elas são obrigadas a implementar os métodos *writeTo* e *readFrom* que realizam, respectivamente, a serialização e deserialização. No entanto, o serviço de eventos CORBA não replica tipos primitivos nem objetos, mas instâncias de um tipo genérico denominado *Any*. Dessa forma, os objetos gráficos são convertidos para um objeto do tipo *Any* para serem transmitidos (*writeTo*) e recuperados no destino *readFrom*. Esta conversão é direta para os tipos nativos ou declarados pelo usuário na linguagem IDL. Por isso, as primitivas do modelo gráfico são declaradas em IDL e compiladas para a(s) linguagem(ns) de programação, no caso C++.

2.6 Experimentos

Implementamos no protótipo do modelador colaborativo, funcionalidades básicas de modelagem e visualização, com o intuito de ilustrar a adequação da arquitetura na manutenção da consistência geométrica e na exploração dos recursos individuais dos equipamentos utilizados na atividade colaborativa. O cenário de avaliação foi composto pelos seguintes equipamentos listados na Tabela 2.7.

Hardware	Processador	Placa 3D	Conexão de rede
UltraSPARC/1	UltraSPARC IIi/440MHz	Creator 3D	10 Mb/s
UltraSPARC/10	UltraSPARC IIi/440MHz	Elite 3D	100 Mb/s
UltraSPARC/10	UltraSPARC IIi/440MHz	Sem Placa	100 Mb/s
SunEnterprise 10000	2x Ultra Sparc/440MHz	Sem Placa	10/100 Mb/s
PC Intel	Pentium II/300MHZ	Rage 3D LT	10 Mb/s
PC Intel	Pentium IV/2.8MHz HT	GeForce 6600	100 Mb/s

Tabela 2.7: Equipamentos utilizados nos experimentos

As UltraSPARC/1, UltraSPARC/10 e os PCs Intel são utilizados pelos usuários para rodar as aplicações interativas, enquanto a SunEnterprise é utilizada para hospedar os servidores de modelagem e e o objeto FloorControl.

As estações UltraSPARC/10 e PC com adaptador gráfico Rage 3D estão conectadas à SunEnterprise por uma rede Ethernet de 10 Mb/s, enquanto as demais utilizam também uma rede Ethernet, mas de 100 Mb/s para se conectar ao servidor de modelagem. Esta configuração visa diversificar o cenário de avaliação, a fim de se obter maior

heterogeneidade possível. As estações UltraSPARC e SunEnterprise utilizam sistema operacional Solaris, enquanto o sistema operacional do ambiente PC/Intel é Linux.

O objetivo dos experimentos é mostrar que a arquitetura proposta consegue absorver adequadamente algoritmos geométricos robustos, projetados para aplicações mono-usuário, sem qualquer necessidade de adaptação para um ambiente multi-usuários.

2.6.1 Consistência geométrica

Para ilustrar a consistência geométrica, implementamos um conjunto de operadores booleanos robustos para objetos modelados por complexos celulares, projetados inicialmente para uma aplicação mono-usuário [13]. Para colocar em prática a combinação da arquitetura fracamente acoplada e algoritmos robustos, os operadores booleanos são baseados em métodos propostos por Bruderlin [6] e Segal [63]. Duas regiões de tolerâncias são definidas para todas as primitivas (ponto, segmento, polígono) que compõem os objetos geométricos, chamadas região de tolerância de inclusão, denotada por ϵ, e a região de tolerância de exclusão, denotada por δ. Dadas duas entidades fundamentais A e B, se suas regiões de inclusão se tocam, então garante-se que elas se interceptam. Se suas regiões de exclusão não se tocam, então garante-se que elas não se interceptam.

Dado um ponto $P_0 = (x_0, y_0, z_0)$ no espaço $\Re^3$, a sua região de tolerância de inclusão é definida como:

$$\epsilon_{P_0} = \{P \ | \ | \ P - P_0 \ | < \epsilon\}, \tag{2.1}$$

sendo $\epsilon > 0$.

Enquanto a região de tolerância de exclusão δ_{P_0} é definida como:

$$\delta_{P_0} = \{P \ | \ | \ P - P_0 \ | < \delta\}, \tag{2.2}$$

sendo $\delta >> \epsilon$.

Geometricamente, as regiões de tolerância de inclusão e exclusão para um ponto P podem ser vistas como esferas concêntricas de raios ϵ e δ, com centro no ponto P_0,

com a restrição de que a região de exclusão é sempre maior do que a região de inclusão. Para uma reta, as regiões de tolerância de inclusão e exclusão são cilindros com raios ϵ e δ, respectivamente. Já no caso do plano, os elementos de tolerância de inclusão e exclusão seriam placas com espessuras ϵ e δ, respectivamente.

A composição das regiões de tolerância de pontos e linha formam as regiões para o segmento de reta, já que os segmentos são limitados por pontos nos seus extremos. As regiões ϵ e δ para um polígono são obtidas pela composição das regiões para pontos, segmentos e plano, pois um polígono é limitado por um conjunto de segmentos e este por pontos.

A determinação de coincidências de pontos no $\Re^3$, a incidência de um ponto sobre um segmento, a identificação da posição de um ponto em relação a um plano e de um ponto em relação a um polígono no plano são as relações geométricas básicas implementadas pelos algoritmos de interseção. A partir destas relações as interseções: segmento/segmento, segmento/plano, segmento/polígono, polígono/plano, polígono/polígono são facilmente determinadas.

A cada ocorrência de interseção, as primitivas envolvidas e suas regiões de tolerância de inclusão são novamente avaliadas e atualizadas. Os testes de interseção já processados para aquelas primitivas dentro das regiões de exclusão das entidades originais são repetidos. Este procedimento garante a preservação da propriedade de transitividade na maioria das situações práticas. Detalhes sobre a implementação das interseções básicas, empregando as regiões de tolerâncias, podem ser obtidos em [13]. A Figura 2.15 mostra resultados de operações booleanas robustas entre dois objetos utilizando operadores robustos.

(a) *união* (b) *diferença* (c) *diferença* (d) *interseção*

Figura 2.15: Operações booleanas robustas.

Algoritmos geométricos desta natureza, pode ser acomodos diretamente no servidor de modelagem. Desta forma, uma única instância executa no ambiente colaborativo, como se fora em um única aplicação. As decisões geométricas robustas pro-

movem operações topológicas consistentes, grando assi modelos geométricos consistentes. Como consequência, modelos gráficos são gerados e replicados para cada usuário também de forma consistente e robusta.

2.6.2 Visualização independente

Para ilustrar a visualização independente entre as máquinas, utilizamos objetos mais complexos e configuramos os parâmetros de renderização e do volume de visualização para cada máquina. A Figura 2.16 ilustra este cenário, no qual três usuários visualizam a mesma cena com parâmetros de visualização (posição da câmera) e renderização (sombreamento facetado, suave e *wireframe*) distintos, da seguinte forma:

- sombreamento facetado: UltraSPARC/10 com placa gráfica acelerada Creator/3D;
- *wireframe*: PC Intel e UltraSPARC/10 sem placas aceleradoras;
- sombreamento suave: UltraSPARC/10 com placa gráfica acelerada Elite/3D;

Com esta combinação de parâmetros de renderização/equipamentos, realizamos medições do tempo de renderização e exibição da cena e obtivemos tempo de resposta muito semelhantes, algo da ordem de 1 a 10 milisegundos para a cena exibida na Figura 2.16. Com isso, verificamos que diferentes plataformas computacionais podem ser utilizadas de forma produtiva para o trabalho colaborativo, bastando a cada usuário ajustar os parâmetros de renderização de acordo com os recursos disponíveis localmente. Além disso, pode-se notar total independência na navegação da cena, através das modificações dos parâmetros de visualização, tais como posição, orientação e abertura da câmera e o volume de visualização, proporcionando alto grau de flexibilidade na interação usuário/computador.

2.7 Resumo e considerações

Neste capítulo apresentamos uma arquitetura fracamente acoplada para sistemas de modelagem geométrica colaborativa capaz de garantir em cada transação a consistência geométrica e topológica dos objetos modelados, através da separação e tratamento diferenciado das informações de modelagem e renderização. Além disso, a

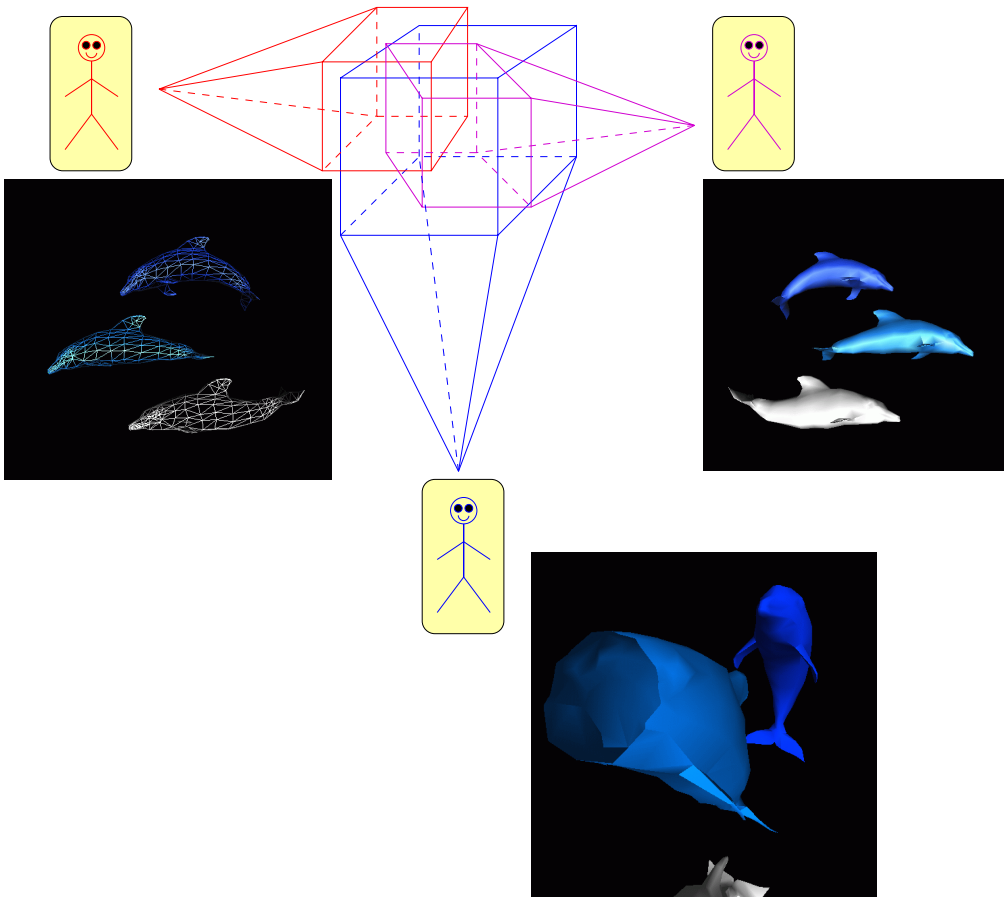


Figura 2.16: Diferentes volumes de visualização e parâmetros de renderização para mesma cena em distintos equipamentos.

arquitetura possibilita uma renderização independente em cada máquina utilizada na colaboração, permitindo explorar adequadamente os recursos individuais destes equipamentos e também uma flexibilização do processo interativo, dado que cada usuário pode ajustar seus parâmetros de renderização e visualização.

A utilização de um modelo geométrico em um modelador dedicado possiblita o emprego de algoritmos robustos como também estratégias de controle de versões de um mesmo projeto, consolidados em aplicações mono-usuário sem qualquer necessidade de modificação. Desta forma, garante-se a consistência da informação geométrica e topológica do modelo central. A estratégia de replicação passiva do modelo gráfico em cada transação assegura, adicionalmente, a consistência das cópias de modelo gráfico, de forma simples e eficiente.

A independência na renderização, visualização e manipulação do modelo gráfico traz excelente flexibilidade ao processo de interação (humano-computador) e no aproveitamento dos recursos computacionais locais. No entanto, estes fatores introduzem um alto grau de dispersão no processo de colaboração [52], que é um efeito negativo ao trabalho colaborativo, dadas as limitações impostas pela não-presencialidade, acarretando uma perda de consciência coletiva. Para minimizar, e em algumas situações resolver este problema de dispersão, apresentamos no Capítulo 4 mecanismos que promovem o aumento da consciência coletiva na arquitetura fracamente acoplada. Outro fator negativo desta arquitetura é o alto tempo de resposta provocado pela centralização do modelador geométrico. Para resolver este problema, apresentamos um modelo de interação no Capítulo 3.

Capítulo 3

Um modelo de interação distribuída

Difícil manter o meu propósito de sanidade.
Raquel Grabauska.

A arquitetura fracamente acoplada proposta no Capítulo 2 garante, com o apoio de algoritmos geométricos robustos, a consistência dos objetos geométricos e proporciona o melhor aproveitamento dos recursos de cada máquina no processo de renderização. No entanto, sob o ponto de vista de interação distribuída, esta solução baseada em um modelador dedicado gera maior tráfego na rede se comparada com uma arquitetura totalmente replicada, podendo comprometer o nível de interatividade do sistema.

A eficácia da arquitetura proposta depende de mecanismos que realizem o fraco acoplamento entre eles, para que seja possível o tratamento diferenciado da modelagem e renderização. Se por um lado, o serviço de replicação provê o fraco acoplamento no compartilhamento do modelo gráfico (comunicação no sentido servidor de modelagem $\rightarrow$ as instâncias da aplicação interativa), o mesmo não ocorre na requisição de operações de modelagem (comunicação no sentido aplicação interativa $\rightarrow$ servidor de modelagem). Pois, a dupla *Proxy* e *Broker* fornece uma infra-estrutura de comunicação síncrona com os objetos remotos do servidor de modelagem. Desta forma, qualquer manipulação realizada na aplicação interativa gera tráfego na rede, Figura 3.1.

A necessidade de um modelo de interação distribuída[1] nasce justamente devido aos problemas inerentes à comunicação entre as aplicações via rede de computadores,

[1]No decorrer do capítulo, utilizaremos apenas a expressão *modelo de interação* para designar o modelo de interação 3D distribuída.

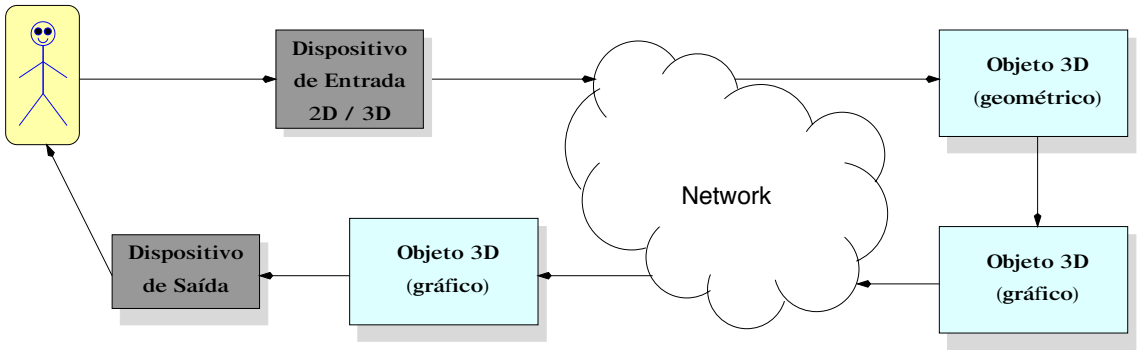


Figura 3.1: Visão geral da interação distribuída.

a qual pode se tornar um gargalo para o sistema colaborativo. Com isso, propomos o uso de metáforas 3D com representação pictorial simplificada para o suporte ao fraco acoplamento na interação distribuída e, também, para promover a realimentação visual (local) das ações dos usuários na arquitetura fracamente acoplada, como forma de manter o grau de interatividade em um nível aceitável. Segundo Aurélio [22], uma metáfora é um "tropo que consiste na transferência de uma palavra para um âmbito semântico que não é o do objeto que ela designa, e que se fundamenta numa relação de semelhança subentendida entre o sentido próprio e o figurado". A metáfora 3D consiste em uma representação pictorial associada a uma semântica de manipulação, que possibilita a "interação direta" 3D através dos dispositivos de entrada 2D e 3D.

A introdução das metáforas 3D, como base do modelo interação, resulta em duas fontes de eventos assíncronos para as instâncias da aplicação interativa: eventos locais gerados pelos dispositivos de entrada e eventos de rede, provenientes do modelador dedicado e das instâncias da aplicação interativa. Estes eventos devem ser manipulados consistentemente pela aplicação interativa em cada estação participante, com a ajuda do sistema de janelas e da plataforma de comunicação, para garantir a integridade da informação e o funcionamento correto do sistema colaborativo.

Na seção 3.1, destacamos o funcionamento e a estrutura de algumas metáforas de interação 3D mais comumente utilizadas com dispositivos de entrada 2D e 3D. Na seção 3.2 propomos um modelo de interação distribuída baseado em metáforas de interação 3D para a arquitetura fracamente acoplada, proposta no Capítulo 2. Na seção 3.3, apresentamos o mecanismo de tratamento de eventos proveniente de múltiplas fontes. Na seção 3.4, descrevemos uma visão de implementação do modelo de interação

proposto. Na seção 3.5 são realizados e apresentados experimentos para avaliação do tempo de resposta destas metáforas e do modelo de interação como um todo. Por fim, um resumo das deste capítulo destaca as nossas contribuições para modelo de interação proposto.

3.1 Trabalhos correlatos

No contexto de interação homem–máquina, a literatura acerca de metáforas para manipulação direta 3D é bastante vasta e destaca principalmente os mecanismos de manipulação 3D utilizando dispositivos 2D e 3D [8, 11, 49, 61, 79]. Estes trabalhos estão direcionados predominantemente para a investigação de aspectos de usabilidade na interação humano–computador em aplicações mono-usuário. Como conseqüência, metáforas de manipulação têm sido propostas e implementadas para facilitar o processo interativo 3D [40, 43]. No entanto, pouco tem sido feito no contexto de interação humano–computador–rede–computador–humano [27], principalmente no que diz respeito aos problemas em interação 3D distribuída em aplicações colaborativas, com tratamento adequado de realimentação visual.

Uma grande preocupação, ao se propor um modelo de interação distribuída, seria como validá-lo. A literatura de interação humano-computador revela trabalhos relacionados à avaliação de usabilidade de sistemas interativos mono-usuário [48, 54]. Acreditamos que algumas das métricas de usabilidade de sistemas interativos, como funções da latência do sistema, sejam particularmente interessantes para validar sistemas colaborativos interativos. Nielsen [48] estabeleceu empiricamente limites para os valores de latências, identificando-os como determinantes para as reações e comportamento dos usuários durante o trabalho interativo. Adaptamos este modelo para avaliar a usabilidade de metáforas 3D para modelagem geométrica colaborativa baseada na arquitetura fracamente acoplada.

3.2 Um modelo de interação

Em busca de uma solução para melhorar o tempo de resposta às ações dos usuários, propomos um modelo de interação para a arquitetura fracamente acoplada,

que diferencia duas classes de interações: uma que depende exclusivamente do comportamento do modelador geométrico e a outra que não necessita dos dados do modelo geométrico. Com isso, conseguimos identificar dois laços de realimentação: o laço local, o qual chamamos de *interação local* e o laço remoto, que denominamos *interação remota*, Figura 3.2. Veremos ainda no Capítulo 4 que há uma terceira classe de realimentação para aumentar a consciência coletiva, na qual a resposta da ação do usuário é replicada para todas as instâncias de aplicação 3D.

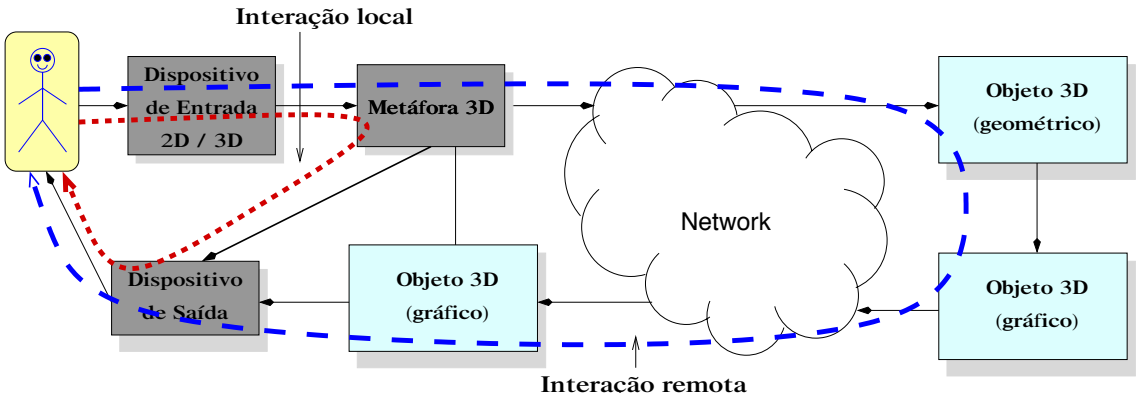


Figura 3.2: Visão geral do modelo de interação distribuída.

Nossa proposta consiste essencialmente na adequação dos mecanismos de manipulação direta 3D (originalmente empregados em aplicações mono-usuário) para um ambiente colaborativo, no qual as metáforas 3D definem a granularidade da comunicação com os objetos geométricos remotos e concomitantemente fornecem uma realimentação visual das manipulações diretas. Este controle de granularidade supre a necessidade de fraco acoplamento na interação com o modelo geométrico especificado na arquitetura, permitindo ajustar o tráfego na rede em cada transação.

Com o aumento da granularidade da comunicação entre a aplicação interativa e o servidor de modelagem (diminuição do tráfego na rede), a preservação da interatividade da aplicação é mantida através da realimentação local da metáfora de manipulação em tempo real. O intuito é fornecer um ferramental para a manipulação de objetos distribuídos de forma transparente para o usuário, preservando a usabilidade do sistema.

Os objetos gráficos formam uma ponte natural para a manipulação direta 3D dos objetos geométricos, pois são processados e renderizados nas instâncias da aplicação interativa e têm os mesmos identificadores dos respectivos objetos geométricos remotos.

Assim, as metáforas são associadas, via *proxy*, aos objetos geométricos através dos identificadores dos objetos gráficos. Com isso, a interação direta distribuída com os objetos geométricos pode ser realizada de forma transparente com dispositivos de entrada 2D e 3D, através das metáforas de interação.

Além dos modelos geométrico e gráfico, fazem parte do modelo de interação: metáforas de interação, infra-estrutura de comunicação e mecanismo de tratamento de eventos (Área hachurada na Figura 3.3). Os dispositivos de entrada e saída estão disponíveis em cada máquina participante, enquanto as metáforas de interação 3D formam o elo de ligação entre as ações dos usuários e os dois modelos de dados. A infra-estrutura de comunicação entre as instâncias da aplicação interativa e o modelador geométrico contempla três diferentes vias: a comunicação entre metáfora e objeto geométrico remoto, a replicação do modelo gráfico e a comunicação entre as instâncias da aplicação interativa para a sincronização das metáforas (representação visual). O mecanismo de tratamento de eventos é responsável por coletar eventos provenientes de várias fontes e entregá-los para a aplicação interativa. Essas fontes podem ser os servidores e as instâncias da aplicação interativa.

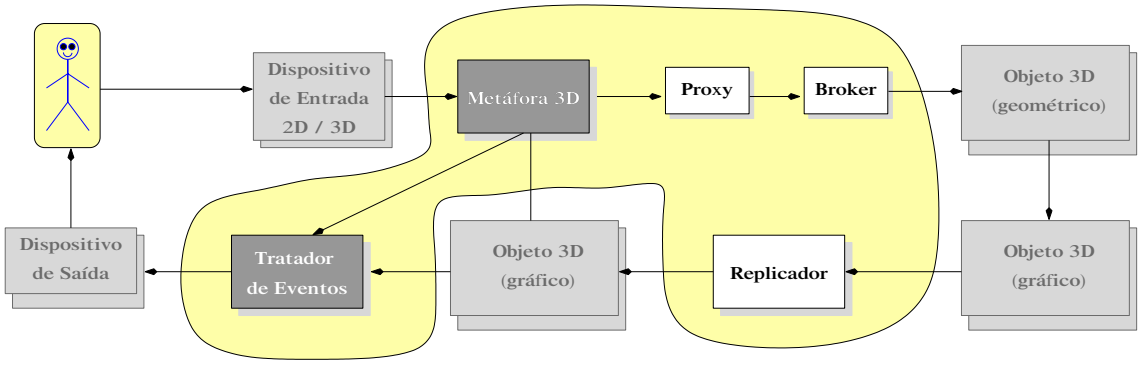


Figura 3.3: Componentes de interação distribuída.

Nota-se que o usuário interage com a cena renderizada em seu espaço de trabalho através das metáforas, tendo-se como realimentação visual a representação pictorial das próprias metáforas e o modelo gráfico renderizado localmente. Como as metáforas são processadas localmente, a resposta é praticamente simultânea às manipulações. O modelo gráfico é gerado a partir do modelo geométrico remoto, levando mais tempo para ser processado e transferido para os espaços de trabalho remotos.

Cada metáfora está associada ao objeto gráfico local, através da sua representação

pictorial simplificada, definindo uma semântica de manipulação para modelo geométrico remoto. Como o objeto e a representação pictorial da metáfora são entidades distintas, definimos duas semânticas para a interação: a *semântica local* de atualização da *metáfora* e a *semântica distribuída* de manipulação remota. Estas semânticas estabelecem a granularidade de atualização (renderização e visualização) das metáforas na aplicação interativa e da interação com o modelo geométrico, respectivamente.

3.2.1 Interação local

Apesar da literatura existente utilizar comumente o termo de metáforas 3D para denotar representações pictoriais que facilitam a especificação das operações espaciais, como rotação, mudança de escala e deslocamento, através dos dispositivos de entrada 2D, consideramos também a aplicação destas metáforas num modelo de interação distribuída com dispositivos de entrada 3D.

As metáforas buscam minimizar o tráfego na rede, mantendo-se o grau de interatividade confortável para o usuário localmente. O princípio básico é tornar independente as atualizações da representação pictorial das metáforas e dos objetos correspondentes no servidor de modelagem. Desta forma, transformações geométricas sucessivas podem ser realizadas localmente nas metáforas, com a atualização em tempo real. Enquanto, com maior granularidade, as transformações resultantes intermediárias são aplicadas ao modelo geométrico.

A semântica local neste modelo determina a reação imediata das *metáforas* às manipulações feitas por algum usuário em determinado objeto, atualizando-as simultaneamente. Por exemplo, eventos de seleção provocam a renderização e exibição "instantânea" da *metáfora* associada ao objeto a ser manipulado, se esta já não estiver exibida. Outro exemplo, em uma operação de translação, a metáfora é atualizada a cada deslocamento e somente após a transformação ser finalizada, a transformação resultante é enviada para o objeto geométrico correspondente, Figura 3.4.

A atualização imediata da informação pictorial em tempo real no espaço de trabalho do usuário fornece instantaneamente uma estimativa visual do próximo estado do objeto, sem que as operações tenham sido processadas no servidor de modelagem. Somente após a aplicação de cada transformação intermediária, o objeto gráfico gerado a partir do objeto geométrico transformado é replicado para outras instâncias da aplicação

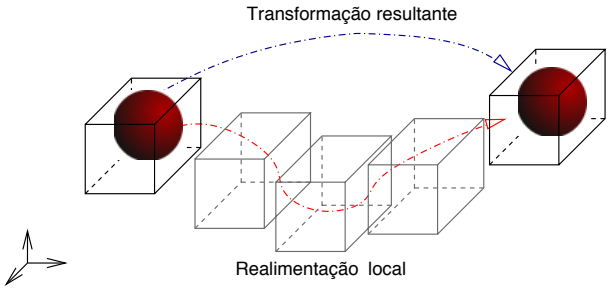


Figura 3.4: Manipulação 3D distribuída com realimentação local.

interativa, onde é renderizado. Com isso, pode-se conseguir uma realimentação visual adequada sem necessariamente onerar a rede com um tráfego freqüente, muito comum numa seqüência de interações.

A latência de uma metáfora representa a somatória dos tempos de resposta relativos ao processamento da metáfora (transformações e mapeamento entre espaços 2D e 3D) e o tempo de renderização e exibição. Estes valores de atrasos dependem somente da capacidade de processamento local de cada equipamento utilizado, já que a metáfora é processada apenas localmente, Figura 3.5.

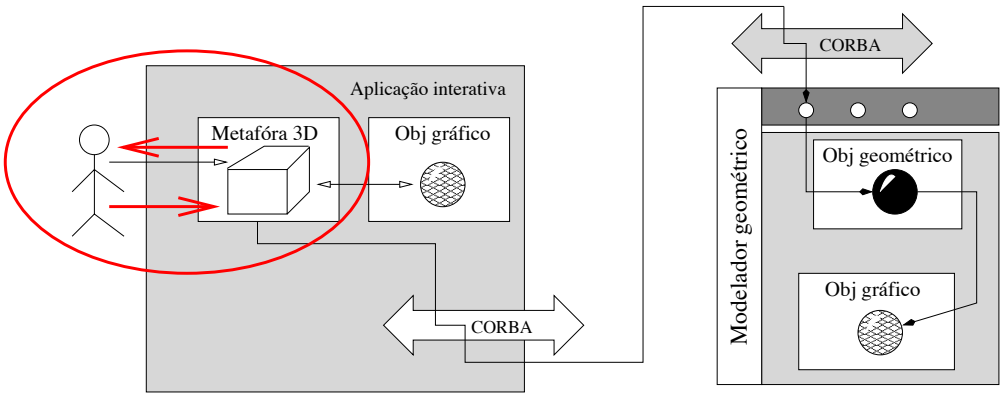


Figura 3.5: Destaque da latência de uma metáfora 3D

Em termos dos parâmetros identificados, podemos equacionar a latência de uma metáfora, manipulada localmente, através da equação:

$$tr_l = t_{pl} + t_r, \tag{3.1}$$

sendo t_{pl} e t_r, respectivamente, os tempo de processamento local e de renderização e exibição.

3.2.2 Interação remota

A interação distribuída é parte importante na definição da qualidade dos parâmetros de interatividade de um sistema colaborativo, pois o usuário visualiza o resultado de suas ações e dos demais usuários em tempo real [65]. O sistema deve reagir o mais rápido possível às ações dos usuários e atualizar as modificações para todo o grupo de cooperação, evitando gerar desconforto. No entanto, este objetivo é difícil de se atingir, pois existe um atraso inerente à comunicação via rede e a limitação quanto à largura de banda para transmissão de grande quantidade de informação repetidas vezes. Um agravante a esta situação é a possibilidade de diferentes atrasos entre usuários geograficamente dispersos conectados através de diferentes redes.

A semântica distribuída estabelece a relação entre metáforas e objetos geométricos remotos, bem como define, para cada par, um controle na granularidade no envio de informação entre a aplicação interativa e o servidor de modelagem. Através do controle de granularidade, pode-se estabelecer a freqüência de comunicação entre eles. O repasse dos dados para o modelador geométrico pode ser, por exemplo, em cada interação, ou somente após um determinado número de eventos de entrada.

Ao chegar no modelador geométrico, a operação resultante é aplicada ao objeto geométrico, que gera o respectivo objeto gráfico, o qual é replicado para as aplicações interativas, onde é renderizado. Sob o ponto de vista de interface com o usuário, as semânticas de manipulação podem, por exemplo, ser construídas com base nas ações de pressionar/liberar botões dos dispositivos de entrada. A Figura 3.6 mostra o caminho percorrido pelos dados necessários a uma ação sobre um objeto geométrico, ao se disparar um evento.

O tempo gasto desde o acionamento da ação na metáfora, transmissão através da rede, até a atualização do objeto gráfico, fornece a latência de manipulação, e é dado por:

$$tr_m = t_{pl} + t_{t1} + t_m + t_{t2} + t_r, \tag{3.2}$$

sendo t_{t1}, t_m, e t_{t2}, respectivamente, os tempos de requisição entre cliente e servidor, o tempo de modelagem (atualização do modelo geométrico) e o tempo de replicação, respectivamente. Note que estes valores podem ser alterados de acordo com a granularidade da

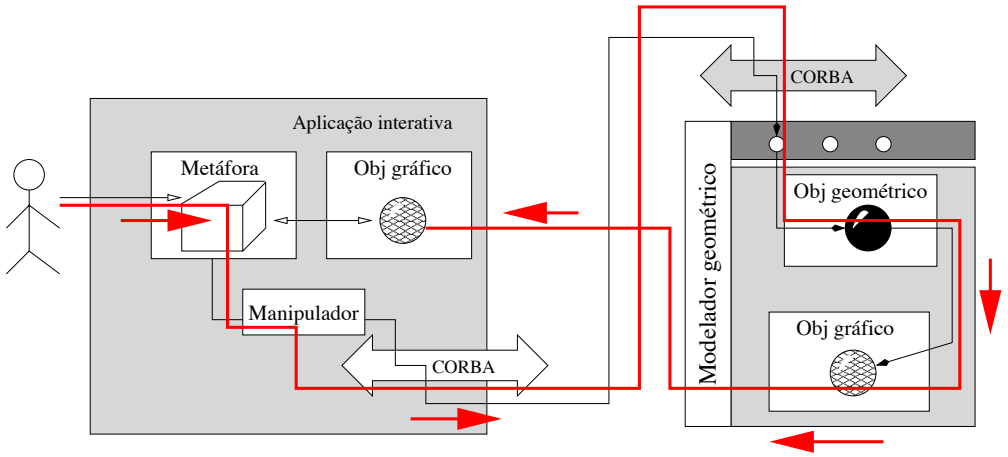


Figura 3.6: A latência de manipulação

comunicação entre metáforas e modelador geométrico. A equação (3.2) nos sugere que temos várias formas de melhorar a latência de uma aplicação em modelagem geométrica colaborativa:

- ajustar localmente os parâmetros de renderização (t_r) para compensar os atrasos na rede (t_1 e t_2);
- investir no poder de processamento do servidor de modelagem (t_m) e das estações de trabalho(t_r);
- investir em uma infra-estrutura de rede mais rápida (t_1 e t_2); e
- diminuir o tráfego na rede, através do controle de granularidade das metáforas.

A mudança de *hardware* está fora do nosso foco de análise, pois pressupomos que a infra-estrutura física já exista. No entanto, através da arquitetura proposta é possível influenciar diretamente no tempo de renderização, ajustando-se os parâmetros de renderização em cada equipamento (adaptabilidade prevista na arquitetura), bem como diminuindo a granularidade de postagem de informação na rede, através da definição da semântica das metáforas 3D.

3.3 Tratamento de classes de eventos

Sob o ponto de vista de desenvolvimento de aplicações colaborativas, uma preocupação adicional é a necessidade do tratamento de informações que chegam simultaneamente às instâncias da aplicação interativa provenientes dos dispositivos de entrada locais (*mouse*, teclado, *spaceball*, etc.) e da rede. Esta tarefa é geralmente realizada pelo sistema de janela e a plataforma de comunicação, respectivamente. Especificamente na arquitetura proposta, o mecanismo de recepção do serviço de replicação é responsável pelo recepção e repasse dos eventos remotos que chegam à aplicação interativa.

Um problema inerente a esta configuração consiste em capturar e processar adequadamente ambas as classes de eventos, pois a maioria dos sistemas de janela e os receptores do serviço de replicação possuem mecanismos próprios de coleta de eventos, baseados em sistemas de prontidão, que são disparados ao detectar a chegada de um evento. Colocados em uma mesma aplicação, ambos mecanismos funcionariam separadamente, implementados geralmente através de laços infinitos. Desta forma, ao se utilizar dois comandos desta natureza em uma mesma aplicação, somente o primeiro a ser alcançado iria acionar o mecanismo associado a ele. Este fenômeno de corrida faria com que parte das funcionalidades, seja do coletor de eventos locais, seja do coletor de eventos de rede, nunca fosse acionada, provocando problemas no funcionamento da aplicação interativa.

Duas soluções poderiam ser adotadas para resolver este problema: através de múltiplos fluxos de execução[2] ou da uniformização do mecanismo de tratamento de eventos sob um mesmo fluxo de execução. No primeiro caso, *threads* dedicadas seriam responsáveis pela coleta de cada classe de eventos individualmente. Na segunda solução, o fluxo de execução do processo principal faria este trabalho, coletando todos os eventos indistintamente e tratando-os adequadamente. Além da segunda ser uma solução mais elegante, mais fácil de implementar, OpenGL não é *thread safe*, o que pode provocar bastante problemas.

Propomos um mecanismo de coleta para eventos provenientes da replicação e dos dispositivos de entrada, baseado na segunda classe de soluções, ou seja, usando um único fluxo de controle. As fontes de eventos são registradas no mecanismo através da interface *DigReplicatorCB*. Desta interface deriva a classe de coletores (*DigDispatcher*),

[2] do inglês:*multi-threading*

que capturam os eventos das diversas fontes, através de um único fluxo de execução (*WMDispatcher*) e os despacha para um tratador de eventos (*EventHandler*). Este objeto identifica o tipo do evento e o trata coerentemente (deserialização), entregando-o em seguida para a aplicação, Figura 3.7. O *WDispatcher* disponibiliza ainda o mecanismo de prontidão para verificação e gerenciamento de dados de entrada.

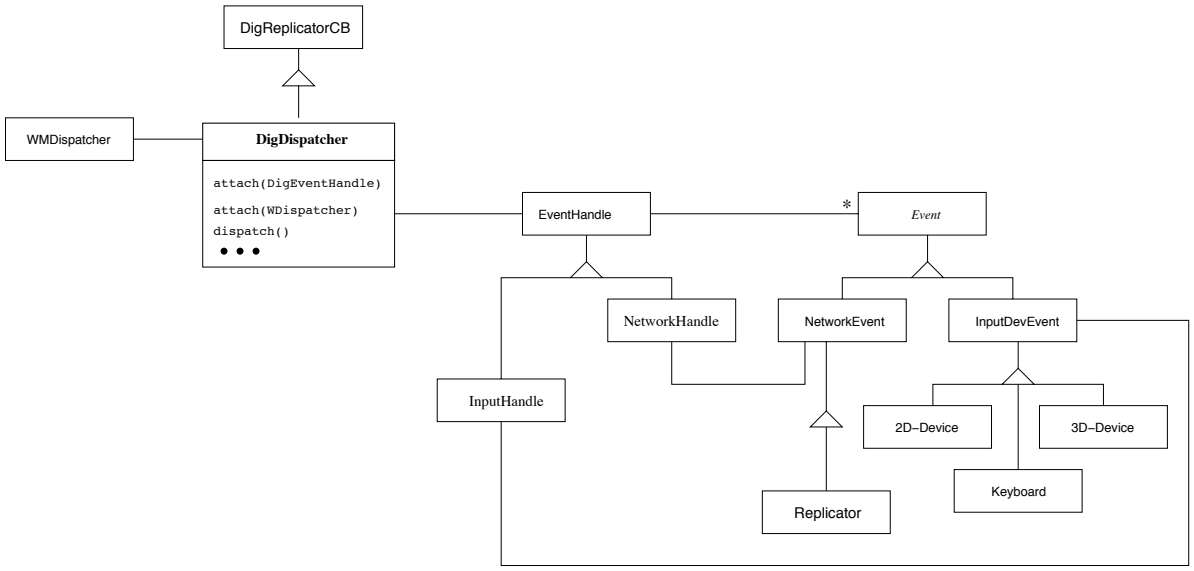


Figura 3.7: Mecanismo de tratamento de eventos.

As relações entre o *EventHandler* e as subclasses de *Event* podem ser implementadas com uso do padrão de projeto *Chain of Responsability*[3] [25]. Assim, os eventos que chegam ao tratador são passados para instâncias específicas de tratadores de cada tipo de evento, por exemplo *NetworkHandle* e *InputHandle*, até ser encontrado o adequado para tratá-lo.

Esta forma de tratar os eventos evita que o *DigDispatcher* necessite conhecer todos os tratadores de eventos explicitamente, além da possibilidade de se adicionar tratadores específicos dinamicamente.

Na seção 3.4.3 descrevemos uma implementação deste mecanismo de tratamento de eventos para a aplicação interativa no sistema de janelas *X/Window*, utilizando-se *dispatchers* fornecidos pela *toolkit GTK+*, encapsulados na subclasse *GtkDispatcher*.

[3]Preservamos a nomenclatura original em inglês, assim como foi feito nos demais padrões de projeto

3.4 Uma visão de implementação

A Figura 3.8 fornece uma visão global da realização da arquitetura fracamente acoplada, adicionando-se os elementos do modelo de interação proposto neste capítulo.

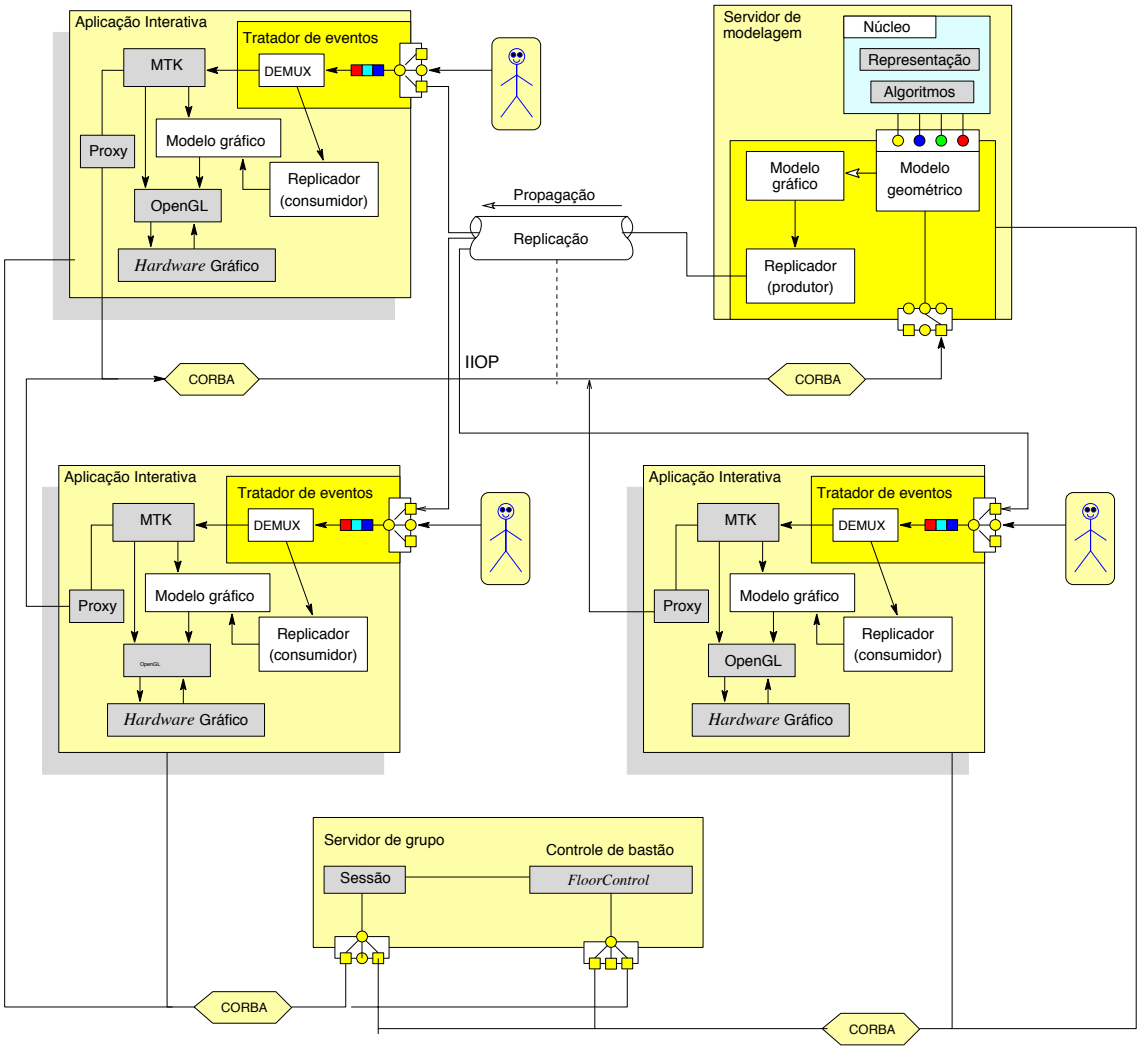


Figura 3.8: Visão geral da arquitetura fracamente com os componentes de interação.

Nesta seção faremos uma breve apresentação das principais tecnologias escolhidas para implementação do modelo de interação, com destaque para a biblioteca de desenvolvimento de interface gráfica com o usuário e manipulação direta 3D. Por se tratar de um padrão largamente difundido no âmbito da computação gráfica, um resumo acerca da biblioteca de renderização OpenGL é apresentada no Apêndice B, para leitura

complementar.

A escolha das tecnologias foi influenciada também pela abrangência de cada uma delas na implementação do protótipo, principalmente no que diz respeito à disponibilidade de implementações em código aberto para diversas plataformas computacionais[4]. Assim como o padrão CORBA fornece uma infra-estrutura de suporte à programação distribuída orientada a objetos, a biblioteca OpenGL é resultado de esforços na direção de padronização das funcionalidades relativas a renderização de cena 3D, através de uma camada de *software* entre o aplicativo e o *hardware* gráfico [47]. A biblioteca *MTK*, desenvolvida pelo nosso grupo, fornece os elementos essenciais para a manipulação direta 3D, incluindo as metáforas para suporte a manipulação 3D usando dispositivos 2D e 3D [43]. A biblioteca *GTK* provê as funcionalidades de interface gráfica e o mecanismo de coleta de eventos locais [31]. Como a arquitetura fracamente acoplada requer uma forma de interação assíncrona com os objetos geométricos, é providencial esta interrelação entre *MTK*, OpenGL, GTK, CORBA e seus serviços.

3.4.1 OpenGL

A renderização é um aspecto relevante e crítico sob o ponto de vista de eficiência em uma aplicação gráfica 3D, devendo as aplicações tirarem o máximo de proveito de cada equipamento, principalmente do *hardware* 3D. Para equipamentos que possuem placas de vídeo com aceleração 2D/3D é possível ter o acesso aos algoritmos de renderização implementados no *hardware*. Aqueles que não estão equipados com este tipo de placa necessitam que a biblioteca gráfica realize esta tarefa por *software*, ou seja, no processador. Em ambos os casos, bibliotecas gráficas largamente utilizadas conseguem realizar estas tarefas de forma transparente para o desenvolvedor da aplicação. O requisito de heterogeneidade nos motivou a utilizar OpenGL [47, 64] como biblioteca de renderização, já que possui implementação para diversos sistemas operacionais.

Desenvolvida pela Silicon Graphics e padronizada por um consórcio, a OpenGL, com característica de independência de *hardware*, é de grande utilização no desenvolvimento de aplicações gráficas 3D. Ela disponibiliza um controle simples e direto sobre um conjunto de rotinas, permitindo ao programador especificar os objetos e as operações

[4]Optar por código aberto é uma filosofia de trabalho do nosso grupo; no entanto, modelo de interação proposto poderia ser implementado utilizando-se tecnologias proprietárias também.

necessárias para a produção de imagens gráficas de alta qualidade em tempo real. Sobre ela foi desenvolvida *MTK*, sintetizada na Seção 3.4.2.

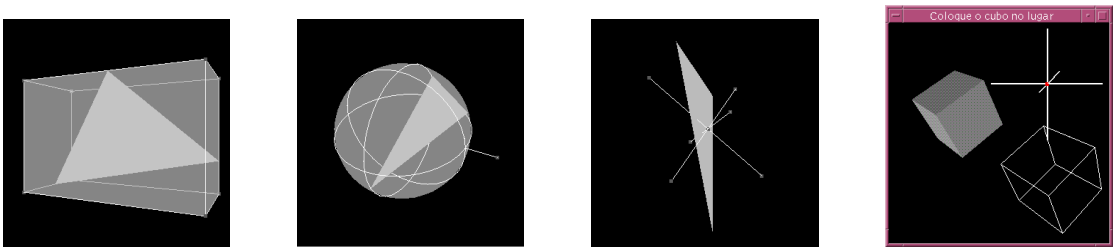
3.4.2 *MTK*: manipulação direta

MTK [43] é uma biblioteca de programação em C++, cujo principal objetivo é prover uma interface direta e simples para manipulação de objeto 3D em uma cena. Para tanto, ela implementa várias metáforas 3D para manipulação direta. O projeto da *MTK* difere essencialmente das demais bibliotecas com alto nível de abstração de programação para manipulação 3D no tocante ao acoplamento entre metáforas e objetos manipulados, tais como Open Inventor [77] e Java3D [67], tornando-a atrativa para implementação do modelo de interação distribuída proposto.

Ao invés de integrar aplicação e interface gráfica com o usuário dentro de um mesmo ambiente de desenvolvimento, a *MTK* fornece um ambiente de desenvolvimento para programação gráfica 3D, no qual o modelo a ser manipulado está separado dos elementos de manipulação. *MTK* inclui, além das metáforas 3D, um conjunto de funcionalidades gráficas 3D, tais como *display list*, multi-visualização através de várias câmeras, multi-iluminação com diversas fontes de luz e mecanismos de seleção para dispositivos 2D e 3D. No que concerne às facilidades de interação (metáforas 3D), a *MTK* fornece: um cursor para indicar visualmente um mouse-3D virtual, um conjunto de representações pictoriais para representação de eixos e planos no espaço, facilitando a percepção de profundidade e um conjunto de funcionalidades que suportam a implementação de metáforas de interação 3D. Estas funções foram organizadas de tal forma que nos ajudam a definir a granularidade das interações locais e remotas, discutidas na Seção 3.2.

As metáforas de manipulação são realizadas no *MTK* através de duas abstrações: a representação visual, denominada *dragger* e a semântica de manipulação, denominada *manipulador*. As metáforas mais comuns utilizadas neste trabalho são: *bounding box*, *bounding sphere*, *jack*, além do *cursor*-3D (Figura 3.9).

A Figura 3.9 mostra apenas a representação pictorial da metáfora, no entanto, cada metáfora possui uma semântica de manipulação configurável que determina quais operações são realizadas nos objetos, através das ações dos usuários. Apesar da semelhança visual, o *jack* e o cursor-3D são funcionalmente distintos. O cursor-3D indica visualmente a posição de um *mouse*-3D virtual, enquanto o *jack*, como as outras



(a) *bounding box* (b) *bounding sphere* (c) *jack* (d) *cursor*-3D

Figura 3.9: Metáforas para manipulação direta 3D.

metáforas, constitui uma representação pictorial (realimentação visual) das ações sobre o(s) objeto(s) slecionado(s).

Os *draggers* proporcionam diferentes formas de interação do usuário com a cena, através das funções de tratamento associadas a pontos sensíveis das suas representações gráficas. Por exemplo, um *bounding box* permite ao usuário alterar o tamanho do objeto através de manipulação direta nos seus vértices, rotações em torno de um eixo através da manipulação nas arestas e translações sobre um plano ao manipular alguma face da metáfora. Basta programar coerentemente a semântica de manipulação no manipulador.

A associação entre uma metáfora e os objetos a serem manipulados se dá através do registro de um ou mais objetos em um ou mais pares manipulador/*dragger*. Assim cada objeto dever possuir um identificador único que o correlaciona com uma ou mais metáforas. Em aplicações mono-usuário, o usuário age diretamente sobre o *dragger*, realizando uma operação com o *mouse* ou com o cursor-3D; em seguida esta operação é mapeada pelo manipulador para o objeto e metáforas. Como isso, tanto a metáfora quanto o objeto podem ser atualizados com o valor semântico gerado pelo manipulador.

Para prover uma semântica de interação distribuída fracamente acoplada, a nossa implementação se aproveita do modelo fracamente acoplado entre manipulador e objeto a ser manipulado da *MTK*. Assim para cada *dragger* é definida a semântica local de interação, através da qual cada evento de entrada local, que provoca uma mudança de estado no *dragger*, dispara sua renderização e exibição imediatas. Estes valores semânticos locais são acumulados e disparados para o modelo geométrico, via *Proxy*, após um determinado número de interações locais. Com isso, é possível realizar manipulações locais em um *dragger* e requisitar uma operação resultante sobre o objeto

geométrico no servidor remoto ao final de um conjunto de interações locais.

O servidor de modelagem, por sua vez, atende à requisição, realiza a operação, gera uma representação gráfica e replica os objetos alterados ou novos para as aplicações servidoras. Desta forma, pode-se manipular um objeto geométrico remoto, através do registro dos métodos remotos do serviço de modelagem, providos pelo *proxy* deste serviço no manipulador da metáfora.

3.4.3 GTK: interface gráfica com o usuário

A construção da interface gráfica com o usuário (GUI) de uma aplicação interativa, requer uma API específica para cada sistema de janelas. No entanto, a programação com a API do *X/Window*, apesar de flexível, é muito trabalhosa e pouco produtiva. O que se utiliza atualmente são *toolkits*, disponibilizando *widgets*. Esses componentes são providos de geometria e comportamento para um conjunto de ações do usuário.

Análises de pré-requisitos e funcional mostraram a viabilidade da *toolkit GTK* para desenvolvimento da GUI do aplicação interativa [31], pela possibilidade de integração com CORBA e A plataforma de desenvolvimento *MTK*. Mas sobretudo, a *GTK* possui *widgets* específicas para exibição da cena renderizada pela *OpenGL*, as mais populares são a *GtkGLArea* e a *GtkGLext*. No início da nossa implementação, existia apenas a *GtkGLArea*[5]. *GtkGLArea* é uma envoltória sobre a API GLX, ou seja, a *GtkGLArea* está para a *GTK* assim como a GLX está para a biblioteca X [41, 62].

A escolha da *GTK* e *GtkGLArea* tem impacto direto na implementação do mecanismo para tratamento de multieventos. A classe *WDispatcher* é especializada para *GtkDispatcher*. Assim, *DigDispatcher* é implementada utilizando o laço principal (*main loop*) da *GTK*, enquanto a coleta dos eventos da rede é feita pela *CORBADispatcher*, encapsulada na *DigReplicator*. Desta forma, ao se instanciar um objeto da classe *DigDispatcher* é criado um mecanismo de tratamento de eventos locais e de rede, que os processa uniformemente. Quando novos eventos chegam ao *dispatch* e o laço principal está processando um evento anterior, uma fila de novos eventos é criada. Assim que o laço principal fica ocioso, os eventos da fila são consumidos e repassados paulatinamente para o manipulador (*EventHandler*). O laço principal é responsável ainda

[5]Para futuras implementações é mais interessante utilizar a *GtkGLExt*, pois além de atualizada, se tornou uma *widget* padrão da *GTK* para renderização de cena com *OpenGL*

pelo pelo redesenho dos componentes da interface gráfica, incluindo-se a renderização e exibição da cena 3D (GUI e OpenGL).

3.5 Experimentos

O processo de avaliação de aplicações interativas pode tomar três caminhos distintos complementares: avaliação subjetiva, avaliação objetiva ou ambas. A primeira, é geralmente realizada através de testes com os usuários, que emitem pareceres de acordo com um conjunto de perguntas e critérios pré-estabelecidos. Já a segunda categoria de avaliação, também realizada com usuários, procura medir alguns parâmetros de desempenho do sistema relacionados a usabilidade do sistema. No terceiro caso, têm-se ambas informações.

Baseando-se na arquitetura fracamente acoplada e no modelo de interação distribuída, propomos uma avaliação objetiva de desempenho para a plataforma implementada. Este mecanismo serve também como verificador de tráfego. Mais especificamente, para avaliar a eficácia de metáforas no apoio à interação distribuída na arquitetura fracamente acoplada, propomos um modelo de avaliação que contempla alguns critérios, procedimentos de coleta de dados e análise dos resultados.

O processo de avaliação busca capturar aspectos de comunicação, renderização, visualização e avaliar, segundo critérios também por ele estabelecidos, a tolerância e o grau de interatividade de usuários. Este processo contempla três diferentes etapas: critérios de avaliação, coleta de dados e análise dos resultados com base nos critérios.

3.5.1 Critérios de avaliação

Para se tornar viável, qualquer sistema interativo deve ser projetado com aspectos de usabilidade em mente. Estes aspectos de usabilidade consistem em projetar sistemas fáceis de serem utilizados e que forneçam respostas rápidas aos usuários. Em uma interação homem-máquina, um sistema interativo deve fornecer uma resposta tão rápida quanto seja possível, pois o ser humano tem pouca tolerância a atrasos em respostas à ações decorrentes de eventos por ele disparados. Por outro lado, o processamento humano não é instatâneo como mostrado por Nielsen [48].

Dentre as métricas de desempenho de sistemas interativos estão o tempo para se completar uma tarefa, o tempo gasto com erros, a relação entre sucessos e falhas, dentre outras [54]. Para sistemas interativos, algumas das métricas de usabilidade são funções da latência do sistema. A latência está relacionada ao tempo gasto entre o disparo de um evento e a obtenção da resposta do sistema. Este parâmetro mede a fluidez das interações entre o usuário e as aplicações. A latência deve ser a menor possível. Três importantes limites para as latências foram identificados como determinantes para as reações e comportamento dos usuários durante o trabalho [48]:

- **0.1 segundos:** limite até o qual o usuário sente que o sistema está reagindo instantaneamente as suas ações. Com isso nenhuma realimentação é necessária.
- **1.0 segundos:** limite até o qual o pensamento do usuário mantém-se sem interrupções, apesar de notar algum atraso na resposta do sistema. Normalmente, nenhuma realimentação especial é necessária na faixa entre 0.1 e 1.0 segundo.
- **10.0 segundos:** limite para manter o foco de atenção do usuário na aplicação. No entanto, sob o ponto de vista de usabilidade, um retorno visual é necessário para a faixa de 1.0 a 10.0 segundos, já que o sistema levará um significativo espaço de tempo para realizar a tarefa, e com isso, o usuário precisa ter consciência de que a tarefa está em curso.

Extrapolando para o universo distribuído, onde a ação de um usuário deve ser propagada para um grupo que realiza várias tarefas em conjunto, observa-se mais ainda a necessidade de se avaliar o tempo de resposta do sistema. A utilização da rede para disseminação e compartilhamento de informação constitui-se naturalmente uma fonte geradora de atrasos. Desta forma, é necessário estabelecer quais os recursos mínimos são exigidos da estrutura de comunicação e quais os dispositivos de aceleração de 3D que suportam um ambiente 3D colaborativo. Acreditamos que, pela transparência do processo de modelagem criado pela arquitetura com o apoio do padrão CORBA, os critérios de avaliação de aplicações interativas mono-usuário possam ser utilizados para se medir o grau de interatividade de aplicações colaborativas, bastando serem adequadamente adaptados e configurados. A medição do atraso pode ser empregada *a priori* para decidir sobre a capacidade de uma determinada rede ou placa gráfica sobre a qual roda o sistema, ou ainda para fornecer subsídios à aplicação para a tomada de decisões visando a melhoria da interação no trabalho colaborativo.

Portanto, com base nos limites estabelecidos por Nielsen [48], propomos avaliar a arquitetura quanto ao aspecto de tempo de resposta, discutido na Seção 3.2. Dois cenários se apresentam para capturar os aspectos mais relevantes do modelo de interação distribuído:

- latência local das metáforas 3D, e
- latência de manipulação de um modelo geométrico.

3.5.2 Coleta de dados

A aferição dos tempos de resposta de manipulações das metáforas e de manipulação de modelos foi realizada para avaliar o desempenho das metáfora 3D em um ambiente colaborativo síncrono. Um componente de coleta de tempos foi projetado para esse fim. Para cômputo da latência local das metáforas, pontos de coletas são inseridos nas *callbacks* de seleção de objetos e na liberação dos mesmo. Em cada ponto de coleta, o valor do relógio é obtido e a diferença entre valores no momento da seleção e da liberação fornece a latência local das metáforas.

As requisições feitas entre a aplicação interativa e os servidores de grupo e de modelagem, além da replicação das metáforas e volume de visualização carregam informações relativamente pequenas (4 valores em ponto flutuante para cada transformação, ou no máximo 16 valores para a matriz de transformação para 2 ou mais transformações concatenadas) através dos métodos dos objetos CORBA remotos. No entanto, dependendo do tipo de modelo geométrico inserido na aplicação, os objetos gráficos podem variar de tamanho. Para o nosso teste, utilizamos os objetos gráficos implementados no protótipo. Desta forma, temos objetos como *cubo*, *esfera*, *cone*, *cilindro* e *tetraedro* e objetos com 300, 400 e 500 facetas, modelados externamente e importados para o ambiente.

O cálculo da latência de modelagem envolve um cenário mais complexo pelos diversos valores de tempo e principalmente pelo envolvimento de diferentes máquinas em uma rede. Para que os tempos dos relógios façam sentido e sejam coerentes, o ambiente de aferição exige que as máquinas tenham seus relógios sincronizados. Portanto a coleta foi feita em equipamentos com relógios sincronizados em relação a um servidor próximo.

No caso, configuramos um servidor no mesmo domínio para evitar erros entre os ajustes dos relógios.

Os atrasos no processamento, renderização e na exibição da metáfora 3D constituem a latência local, ou latência da metáfora, como mostra a equação (3.1), captura. Portanto, ocorre tão somente na aplicação interativa, sem qualquer tráfego em rede. Esta latência depende do suporte à aceleração gráfica 3D de cada equipamento. A Figura 3.10 resume a latência local média (10 repetições) em relação ao número de primitivas do modelo gráfico presentes na cena.

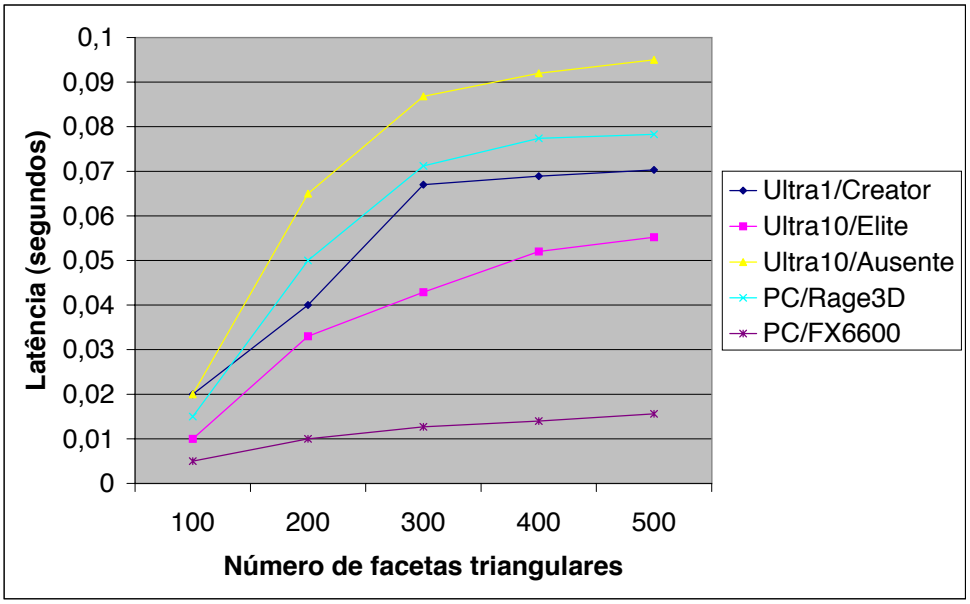


Figura 3.10: Latência local da metáfora.

A latência de manipulação e modelagem, dada pela equação (3.2), expressa o tempo de resposta a manipulações diretas 3D nos objetos geométricos. Nos experimentos realizados, estas manipulações são basicamente transformações geométricas (rotação, escala, translação). O gráfico na Figura 3.11 mostra a latência média de manipulação 3D distribuída.

As diferentes interligações entre as aplicações que compõem o sistema colabo-

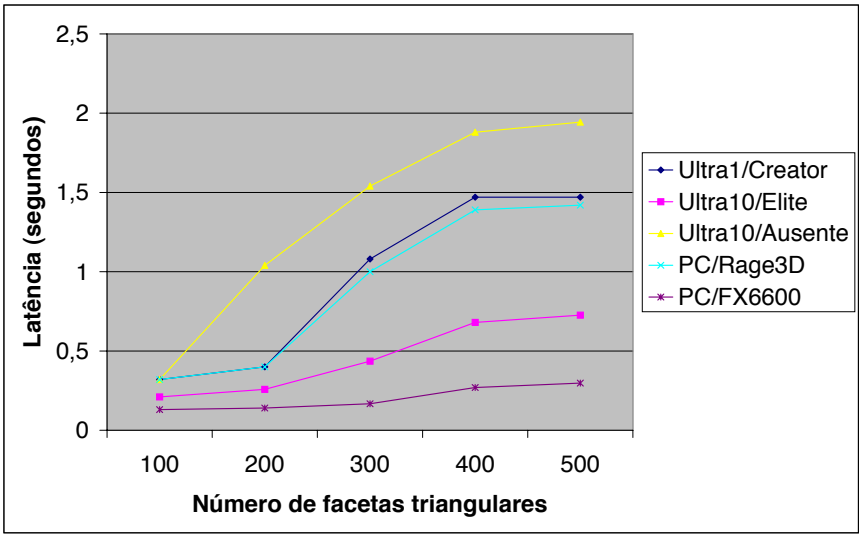


Figura 3.11: Latência de manipulação e modelagem.

rativo como um todo são descritas pelas equações (3.1) e (3.2). Os componentes de medição de latência obtém do sistema os valores dos dois tipos de latência descritos por estas equações. De posse destas informações de latências, podemos avaliar o tempo de resposta do sistema, bem como a eficácia do uso de metáforas na interação distribuída fracamente acoplada. As medidas de latência realizadas fornecem tempo de transferência de bloco de informação seja localmente ou no ambiente distribuído.

3.6 Resumo e considerações

O modelo de interação define diferentes graus de acoplamento entre os componentes do sistema de modelagem colaborativa, preservando a interatividade. A manipulação local das metáforas permite estimar, com razoável precisão, o estado futuro de um objeto manipulado, sem precisar efetivar as operações distribuídas simultaneamente.

De posse dos gráficos com os valores médios de latências das metáforas, verificamos que os valores ficam no intervalo entre 0.01 e 0.2 segundos para máquinas com aceleradores gráficos e entre 0.1 e 1.0 segundos para máquinas sem aceleração.

Este resultados mostram que as metáforas podem ser empregadas sem a necessidade de qualquer elemento extra para indicação de atividade na aplicação interativa. Além disso, é importante salientar que a utilização de metáforas para minimizar o tráfego e realimentar o processo interativo é adequada, dado que o baixo tempo de resposta faz com que o usuário tenha uma realimentação praticamente instantânea de suas ações. Vale salientar que atualmente quase não se usa mais equipamentos sem aceleração 3D em computação gráfica, dado o baixo custo destas placas aceleradoras gráficas para computadores pessoais. Cabe ainda uma ressalva feita aos equipamentos portáteis, como *palmtop*, que certamente ganharão maior poder computacional em um futuro bem próximo.

O gráfico da Figura 3.11 exibe os valores médios da latência de manipulação de modelos geométricos. Vemos que para uma cena simples e com poucos objetos, a diferença não é grande em relação a latência das metáforas. No entanto, para cenas mais complexas, o modelo gráfico tende a crescer e com isso, o parâmetro de tempo de replicação passa a ter um peso grande na equação de latência de manipulação. Outro aspecto que pode provocar um aumento considerável no tempo de resposta é a distância entre os colaboradores, que introduz um atraso natural no tráfego de informação.

É importante observar que resultados próximos das latências de manipulação e de metáforas, indicam que em uma rede rápida e usuários geograficamente próximos, a maior influência no tempo fica por conta dos processos de renderização e visualização. As estações com aceleração gráfica 3D demonstram melhor desempenho na interação 3D, por ocasião da manipulação direta das metáforas, como seria esperado. Isto se reflete diretamente na interação colaborativa, pois o processamento 3D mais rápido acarreta uma melhoria no recebimento da informação compartilhada, principalmente quando aumentamos o acoplamento da interação (metáfora/objeto geométrico).

Concluímos que nossa proposta de utilizar metáforas 3D para interação distribuída traz uma significativa contribuição à modelagem geométrica em ambiente colaborativo. Sob nosso ponto de vista a maior contribuição deste capítulo é ter mostrado conceitualmente que vários mecanismos de interação individual são de grande valor para interação distribuída. Sob o ponto de vista do desempenho, o uso das metáforas tem contribuído decisivamente para interatividade no contexto de modelagem colaborativa: minimiza-se o envio de informações para o servidor de modelagem e, por conseqüência, diminui-se o tráfego na rede. A manutenção da interatividade está, portanto, relacio-

nada com uma realimentação ágil, promovida pela atualização local das metáforas, e redução do envio de requisições de operações ao servidor de modelagem remoto.

Capítulo 4

Um modelo de consciência coletiva

Tudo é vago e incompleto! E o que mais pesa
É nada ser perfeito.
Florbela Espanca.

No Capítulo 3 propomos um modelo de interação distribuída para minimizar o tráfego mantendo-se a interatividade. Introduzimos o uso de metáforas 3D como elemento intermediário na interação entre usuário e o modelo geométrico. Além da capacidade de controlar a granularidade da comunicação, as metáforas 3D provêem a realimentação visual local das ações dos usuários em tempo real. Adicionalmente, o ajuste independente nos parâmetros de visualização e renderização locais possibilita a exploração individual dos recursos computacionais. No entanto, esta flexibilidade no ajuste dos parâmetros de visualização introduz um problema em relação à interação em grupo (humano–computador–computador–humano): a perda da consciência coletiva, também denominada *dispersão*.

Entendemos neste trabalho por *consciência coletiva* o conhecimento que um determinado usuário detém a respeito do espaço de trabalho dos demais usuários de um grupo, tais como localização (*consciência de proximidade*), atividades desenvolvidas (*consciência de atividade*), orientação e posição de visão (*consciência de perspectiva*). Quando se realizam atividades colaborativas, principalmente na manipulação simultânea de artefatos interdependentes, é desejável que esta consciência seja alta, facilitando o trabalho e a coordenação das atividades dentro do grupo e aumentando a produtividade do sistema como um todo. Apesar do modelo de interação distribuída propiciar um ganho quanto à usabilidade (interatividade) do usuário para com a aplicação,

identificamos que ele provoca uma diminuição da consciência coletiva, ou seja, diminui-se a interatividade humano–humano. Com isso, mesmo existindo outros meios de comunicação, seja verbalmente ou através de sinais, a limitação espacial do volume de visão e a oclusão de um objeto por outro podem restringir ou impedir que um determinado usuário perceba as atividades de algum outro.

A perda da consciência coletiva é indesejada em um trabalho cooperativo ou colaborativo. A solução mais imediata para evitá-la ou diminuir o seu impacto sobre a colaboração em um espaço de trabalho tridimensional seria forçando todos os usuários a terem a mesma visão da cena. No entanto, este enfoque contradiz os requisitos de usabilidade e adaptabilidade impostos ao modelo de interação, pois deseja-se que cada usuário possa, de forma independente, ajustar os parâmetros de renderização e visualização. Desta forma, acreditamos que qualquer iniciativa em aumentar a consciência coletiva não deve passar pela restrição dos aspectos de interatividade humano–computador, mas adicionar novos artefatos para aumentar a consciência sem restringir a interatividade local.

Para minimizar os problemas de dispersão, que podem decorrer da arquitetura fracamente acoplada e de um modelo de interação com dois laços de interação, propomos um modelo de consciência coletiva que reutiliza as metáforas 3D empregadas no modelo de interação e introduzimos novos componentes de consciência para grupo. Um servidor de grupo é integrado ao sistema para assegurar a integridade do trabalho coletivo e coordenar o processamento dos eventos provenientes de distintas instâncias de aplicação. Adicionalmente, uma interface multijanela é projetada para acomodar os componentes de consciência e a cena 3D nas instâncias de aplicação interativa.

Neste capítulo propomos ainda uma solução para concretização do modelo de consciência coletiva através da especificação de uma infra-estrutura de replicação das metáforas 3D. Na visão de implementação, a nossa solução se reduz a incorporação das funcionalidades de distribuição de eventos ao *toolkit MTK* para replicação das metáforas e a adição de um componente de serviço de gerência de grupo e de um mecanismo de sincronização dos estados. Por fim, são realizados alguns experimentos com usuários para avaliar a usabilidade do modelo de consciência coletiva.

4.1 Trabalhos correlatos

Existe uma grande quantidade de trabalhos que abordam os problemas de CSCW em variados aspectos, incluindo *hardware*, *software*, organização e gerência de grupos, serviços de suporte, dentre outros [2, 19, 29, 33, 32, 42, 54]. O foco principal destes artigos é a investigação de como tirar o maior proveito do espaço de trabalho compartilhado e como aumentar a consciência coletiva no espaço disponível. Alguns estudos baseados em observação têm mostrado que em atividades de coordenação de trabalho em grupo as pessoas utilizam suas habilidades perceptivas para ter noção das atividades dos outros colaboradores quando compartilham um mesmo espaço físico [73]. Gutwin e Greenberg [33] complementam que um aumento do suporte à consciência coletiva em sistemas colaborativos pode melhorar a usabilidade destes espaços de trabalho compartilhados.

A consciência coletiva em aplicações 2D utiliza-se de elementos auxiliares de interface para apoiar diferentes estilos de colaboração [2, 27]. Os componentes de interface para consciência coletiva mais comuns são: visão de radar, barra de rolagem multiusuário, tele-apontadores e visão WYSIWID (O que você vê é o que eu faço). Gutwin et al. [34] realizaram experimentos a fim de examinar a usabilidade destes componentes adicionados a um sistema colaborativo interativo para desenho, com funcionalidades de mover, agrupar e alinhar objetos em um plano. O experimento foi realizado por um grupo de estudantes de ciência da computação e algumas importantes conclusões foram tiradas:

- A consciência de espaço de trabalho deve cobrir mais do que o conhecimento acerca da interação dos demais usuários. Ela deve incluir o estado do espaço de trabalho e seus artefatos, fornecendo uma visão global do ambiente compartilhado;
- A dispersão pode ser causada não somente pela falta de percepção, muito mais pela dificuldade de interpretação. Com isso, a informação de consciência deve ser facilmente interpretável;
- A informação de consciência coletiva deve ser coletada passivamente e distribuída pelo sistema, ao invés de ser explicitamente gerada pelos usuários, pois no decorrer do tempo os usuários poderiam deixar de repassar estas informações.

Com a incursão do mundo 3D no contexto colaborativo, atualmente as pesquisas caminham no sentido de propor, além de metáforas para manipulação 3D, componentes de consciência coletiva em ambientes virtuais 3D. Dyck e Gutwin [16] têm estudado algumas formas de prover consciência coletiva num espaço compartilhado 3D e sugerem alguns componentes que podem ajudar a aumentar a consciência coletiva e propõem a representação de cada usuário no ambiente compartilhado, vistas alternativas do espaço de trabalho para fornecer perspectivas distintas sobre o meio em observação e uma lista de participantes com informações adicionais para destacar as habilidades individuais de cada um dentro do grupo.

4.2 Um modelo para consciência

Com base nos resultados dos experimentos reportados por Dyck et al. [16, 33], nossa hipótese é que uma boa realimentação visual (percepção) de quais ações (*consciência de atividade*), onde elas são realizadas (*consciência de proximidade* e *consciência de perspectiva*) e quem as realiza (*consciência de participante*) pode ser valorosa para o aumento da consciência em um espaço compartilhado, principalmente quando existe interesse em um objeto comum por mais de um usuário. A consciência de atividade, em especial, pode ser substancialmente relevante para a coordenação do trabalho colaborativo quando os usuários manipulam objetos interrelacionados geometricamente.

Propomos um modelo de consciência que compartilha o conhecimento entre os participantes de uma sessão de trabalho, através da replicação dos dados sobre os próprios participantes presentes (um participante por instância de aplicação 3D) e das suas ações realizadas através de metáforas 3D no espaço de trabalho comum. Acreditamos que o compartilhamento das ações individuais de cada participante seja eficaz para o aumento da consciência coletiva quando acompanha a dinâmica das atividades em si (onde, por quem e de que forma estão sendo realizadas), pois é feito de forma síncrona e não polui o espaço de trabalho comum.

O modelo de consciência coletiva é composto pela gerência de grupo, pelo compartilhamento de metáforas 3D, pelos componentes de consciência 3D, além de uma interface multijanela que agrega os componentes de forma adequada. A gerência de grupo é responsável por armazenar e oferecer mecanismos para compartilhar as informações sobre os participantes presentes numa sessão de trabalho colaborativo. O compartilha-

mento das metáforas de interação provê informações sobre as atividades individuais. Os componentes de consciência, dispostos em uma interface multijanela, são responsáveis por assegurar consciência coletiva, sem poluir a interface gráfica da aplicação interativa. Para não comprometer o desempenho do sistema e obter uma comunicação síncrona entre as distintas instâncias da aplicação, propomos um canal de comunicação adicional direto entre as instâncias de aplicação 3D, para compartilhamento das informações de consciência coletiva.

4.2.1 Gerência de grupo

Uma sessão de trabalho de um ambiente colaborativo caracteriza-se por um grupo de usuários trabalhando juntos e simultaneamente, através de um sistema computacional com a finalidade de resolver um problema específico. A gerência de grupo é responsável por cada sessão de trabalho e apoio à coordenação de atividades de cada usuário. Ela é o elemento articulador entre as instâncias de aplicação, que evita problemas como contenção de recursos ou violação da integridade do sistema. O objeto FloorControl, apresentado na seção 2.4.2, controla o acesso aos objetos geométricos em cada transação colaborativa, bem como permite verificar se um determinado objeto está registrado no servidor, se está livre ou qual usuário tem o direito de manipulá-lo, e ainda fornece a lista de objetos sendo processados de um determinado usuário.

Com a introdução do modelo de consciência coletiva, precisamos estender as funcionalidades da implementação do objeto FloorControl. Além da mediação entre o servidor de modelagem e as instâncias de aplicação, o gerente de grupo deve ser capaz de mediar diretamente as instâncias de aplicação para gerenciar as informações importantes na preservação da consciência coletiva. Propomos, portanto, introduzir em nossa arquitetura um servidor de grupo que integre os serviços de controle de concorrência descritos no Capítulo 2, com serviços de gerência de sessão. Sem perda de generalidade, consideramos que estes últimos serviços consistem na manutenção das informações sobre os membros, estabelecimento dos papéis de cada um, definição dos critérios de integridade e políticas de pertinência de sessão, e a autenticação dos membros em uma sessão de trabalho.

4.2.2 Metáforas 3D: consciências de proximidade, atividade e orientação

As metáforas no modelo de interação distribuída são desenvolvidas para minimizar o tráfego na rede e preservar a interatividade nas instâncias da aplicação interativa. No entanto, não provê diretamente qualquer funcionalidade de colaboração entre os participantes da sessão de trabalho. Acreditamos, contudo, que estas metáforas possam ser empregadas com êxito para aumentar a consciência coletiva, no que concerne principalmente à consciência de proximidade e de atividade.

Como já mencionado na seção 3.4.2, os *draggers* emulam dispositivos 3D de entrada, utilizando funções, que mapeiam um ponto 2D (do dispositivo de entrada) em um ponto 3D (na cena), e são redesenhados quando seus atributos são alterados. O cursor-3D pode realizar tarefa de mapeamento semelhante aos *draggers* para dispositivos 2D e também pode representar graficamente dispositivos de entrada 3D, tratando eventos provenientes destes. Para prover consciência coletiva através destas metáforas, propomos capturar sua posição e orientação espaciais e replicá-la juntamente com a identificação do usuário daquela metáfora para as demais instâncias da aplicação interativa. Através da visualização e da correlação entre as metáforas, tem-se a noção do grau de proximidade espacial entre os usuários. Conhecendo a semântica associada a cada metáfora, o usuário pode inferir que tipos de atividades estão sendo desenvolvidas pelos demais usuários da sessão, ou seja o usuário pode adquirir a consciência de atividade. A representação remota de uma metáfora numa instância da aplicação interativa é denominada de tele-metáfora, especificamente tele-*draggers* e tele-cursor.

Além das tele-metáforas, acreditamos que a relação entre os parâmetros de visualização dos usuários pode ajudar substancialmente a atividade colaborativa que requer um alto grau de percepção comum, como no estágio final do ciclo de vida do produto onde existe maior correlação entre as atividades dos usuários, exigindo maior coordenação do grupo de trabalho. Portanto, mais que a consciência de orientação descrita por Dyck e Gutwin [16], nós propomos a *consciência de perspectiva*, ou simplesmente, *consciência de visão*, através do compartilhamento da exibição dos volumes de visualização de todos os usuários. Adicionalmente, propomos incluir a funcionalidade de visualizar a cena com a mesma perspectiva de visão de um outro usuário. Para isso, coletamos informações sobre o volume de visualização e de câmera de cada usuário e os replicamos para as demais instâncias da aplicação interativa. Denominamos de tele-

visão os volumes de visualização replicados e exibidos remotamente e tele-câmera os parâmetros de câmera replicados.

4.3 Inteface multijanela

Acreditamos que de nada adianta prover elementos que aumentem a consciência coletiva em um ambiente colaborativo 3D e, com a quantidade de informação visual, sem que se busque formas adequadas de exibir as informações para os usuários. Com isso, nossa sugestão é de se evitar poluir o ambiente de trabalho compartilhado 3D, que consiste de uma interface com um componente onde o usuário visualiza o modelo gráfico renderizado, como vimos nos Capítulos 2 e 3. Além da visualização, pode-se interagir individual e diretamente com o modelo geométrico através das metáforas de interação 3D e *cursor*-3D, e navegar pelo espaço compartilhado sem a interferência no trabalho dos demais usuários. Sendo desacoplados os espaços e visualização dos participantes, qualquer usuário pode controlar os parâmetros de renderização e promover mudanças no modo de operação do sistema de modelagem, inclusive utilizar os parâmetros correntes de qualquer outro usuário, como ilustra a Figura 2.16. Para facilitar a identificação deste componente, nós a denominamos o componente de *visão de cena*.

Realizamos experimentos com elementos de consciência 3D e constatamos que a presença de tele-cursores e tele-*draggers* provoca muita confusão aos usuários do sistema, causando uma maior dispersão e, portano, menor produtividade. Diante disso, para resolver o dilema entre introduzir mais informação e não sobrecarregar visualmente a interface gráfica da aplicação, propomos integrar, além do componente de *visão de cena*, mais dois componentes de interface distintos para compor o modelo de consciência coletiva: *lista de participantes* e *visão global*. A lista de participantes fornece a relação dos membros de uma sessão de trabalho.Apesar de já largamente utilizada em aplicações de tele-conferências, neste contexto a lista está relacionada aos outros dois componentes para identificação de usuários no espaço de trabalho compartilhado.

4.3.1 Lista de participantes

A identidade, localização e outros atributos dos usuários ativos de um grupo de trabalho são conhecimentos essenciais quando se trabalha em grupo. Para que outras

informações possam ser correlacionadas com algum usuário e torne a percepção desta informação mais fácil, associamos a cada usuário uma cor que o identifica e personaliza. Desta forma, pode-se mostrar uma lista de usuários, cada um com sua respectiva cor, bem como associar as cores aos outros componentes de consciência, a fim de facilitar a percepção no grupo. Por exemplo, as tele-visões e as tele-metáforas podem ser renderizadas com a cor do usuário.

O controle do fluxo de usuários de uma sessão é realizado pelo serviço de sessão, que passivamente atualiza as informações das aplicações interativas com as informações contidas na sua base de dados. Diferente das listas de participantes tradicionais, que contém imagens ou modelo de cada participante, a lista de participantes proposta divulga apenas os nomes dos usuário, eventualmente as máquinas que estão utilizando e os atributos de cor de cada participante. A idéia é tirar proveito da grande quantidade de informação pictorial disponível, evitando-se onerar o sistema com informações pouco relevantes, no escopo de uma aplicação específica. Além disso, sob o ponto de vista da aplicação interativa, reduz-se o espaço de ocupação da tela, sem perda da informação básica de consciência coletiva.

4.3.2 Visão global

A principal funcionalidade do componente visão global é fornecer uma vista inteira do espaço de trabalho, que engloba todos os usuários da cena. Ao invés de mostrar todos os objetos da cena, a visão global exibe os volumes de visualização de todos os membros, ou eventualmente de um sub-grupo, conforme ilustra a Figura 4.1. As informações dos volumes são coletadas em cada aplicação interativa e repassadas para o grupo. Cada usuário pode habilitar a visualização dos volumes dos demais usuários no componente de visão global. Este tipo de informação realça a consciência de perspectiva, exibindo os volumes no mesmo componente e, com isso, pode-se perceber as diferenças entre orientação, ângulo e profundidade de visão de cada participante.

4.4 Uma infra-estrutura para replicação de metáforas

A principal diferença do nosso modelo de consciência coletiva em comparação com os demais é a possibilidade de se ajustar parâmetros de visualização por demanda,

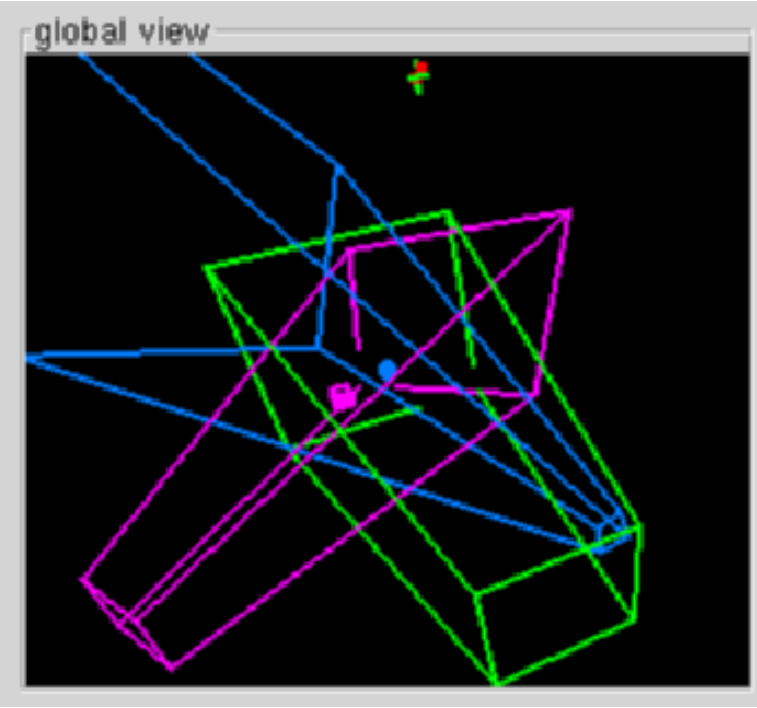


Figura 4.1: Componente visão global.

sob a supervisão do grupo inteiro. E para preservar a consciência seguimos as recomendações de Ellis et al [19]: "uma boa interface de grupo deve fornecer uma visão geral das atividades do grupo e ao mesmo tempo não provocar a distração". Portanto, não utilizamos o componente *visão de cena* para exibir os recursos de reforço à consciência de forma a não poluí-la com elementos que poderão causar dispersão.

Ao detalharmos o nosso modelo de consciência, identificamos dois problemas: replicação das metáforas e dos volumes de visualização, bem como a sincronização da renderização das subjanelas de visão de cena (*scene view*) e de visão global (*global view*).

4.4.1 Replicação

Para que as informações de consciência, providas pelas metáforas 3D e volume de visualização, sejam compartilhadas entre as aplicações interativas propomos um modelo de comunicação entre as aplicações interativas. Este modelo difere do modelo de comunicação entre as aplicações interativas e o servidor de modelagem, que é composto por uma arquitetura cliente/servidor em conjunto com um sistema de replicação origi-

nada sempre no servidor de modelagem. A comunicação entre as aplicações interativas, para provimento de consciência coletiva, é realizada através de um sistema de replicação direta, no qual todas as instâncias da aplicação interativa são capazes de originar um evento para as demais instâncias. Pois, ao contrário do modelo de interação distribuída (Capítulo 3), as informações de consciência ocorrem de participante para participantes, enquanto as informações de interação ocorrem de participante para servidor geométrico.

A necessidade individual de se ter informações de consciência atualizadas exige que cada ação do usuário seja capturada e replicada aos demais membros sincronamente. Desta forma, cada ocorrência de evento que carrega informação de consciência dispara um processo de coleta e replicação da metáfora e/ou parâmetros de visualização para as demais instâncias das aplicações interativas. Por se tratar de operações que ocorrem em alta freqüência e não ter impacto direto sobre o trabalho corrente, a coleta dos dados de consciência, especificamente metáforas e volume de visualização, deve ocorrer de forma independente da coleta dos dados de modelagem e sem a intervenção do usuário.

Para assegurar a usabilidade das tele-metáforas e tele-visões, é imprescindível avaliar a latência de consciência. A latência de consciência tr_d refere-se o tempo gasto para se processar localmente um elemento de suporte à consciência, replicá-lo para as outras instâncias da aplicação interativa e em seguida renderizá-lo nos seus destinos (Figura 4.2), ou seja,

$$tr_d = t_{lp} + t_{rl} + t_r, \tag{4.1}$$

sendo t_{lp}, t_{rl}, e t_r, o tempo de processamento local, o tempo de transmissão e o tempo de renderização, respectivamente. Comparando-se as equações de latências de consciência com a equação 3.1, nota-se que foi adicionada à latência das metáforas o tempo de transmissão dos dados dos elementos de suporte à consciência coletiva.

Para reduzir a latência tr_d e manter o elevado grau de acoplamento entre os elementos, propomos ainda, como uma tentativa de minimizar a parcela de tempo t_r, utilizar um modo de renderização mais simples e mais eficiente para os tele-objetos: modo *wireframe*. Vale ressaltar ainda que a parcela de tempo de processamento local t_{lp} que aparece na equação 4.1 é comum à parcela t_{lp} que aparece na Equação 3.1.

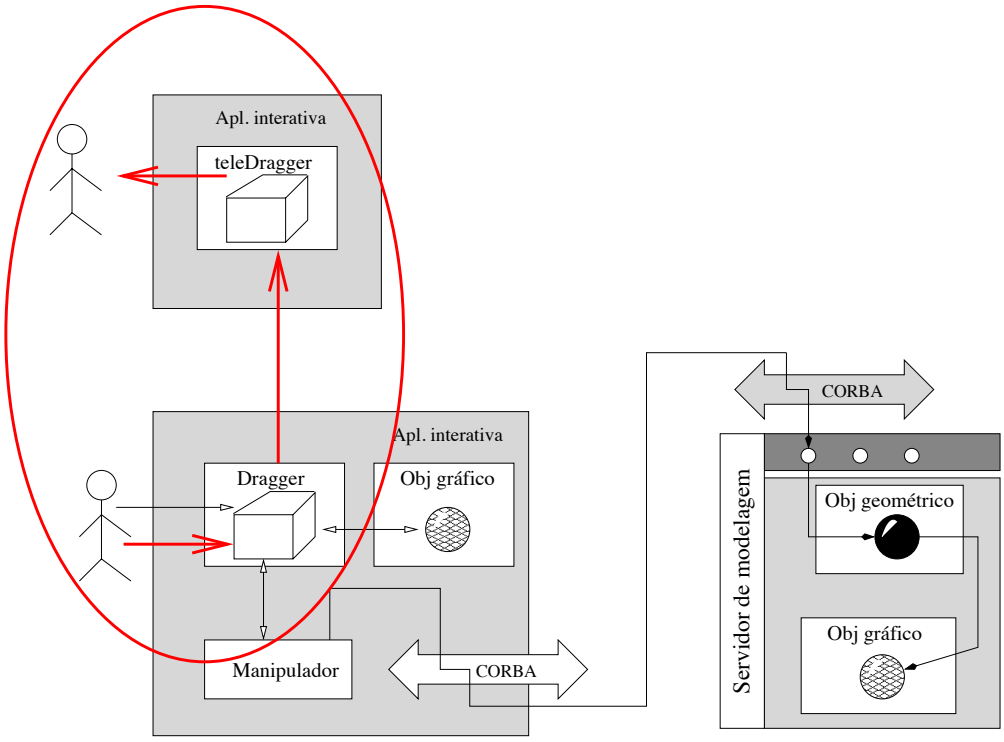


Figura 4.2: Destaque da latência de consciência

4.4.2 Sincronização

Para que a *OpenGL* "desenhe" corretamente a cena nas *widgets* global e *scene view*, uma série de procedimentos deve ser realizada, inclusive indicar o local da renderização, denominado contexto gráfico[1]. O contexto corrente pode ser determinado através de uma função da *GtkGLArea*. No entanto, durante a implementação da aplicação interativa nos deparamos com um problema intrigante relacionado ao contexto e às múltiplas fontes de eventos. A *global view* e a *scene view* tem contextos gráficos próprios, exibindo objetos de naturezas distintas. A primeira é reservada para exibir os elementos de consciência coletiva e a segunda, objetos gráficos e espaço de manipulação dos objetos gráficos. Os objetos gráficos são diretamente manipulados na *scene view* através dos *draggers* e cursor-3D, providos pelo *MTK*. Ao se realizar uma manipulação, informações sobre o *dragger* são replicadas em tempo real para todas as instâncias da aplicação interativa, inclusive para aquela que replicou os dados, sendo renderizadas na *global view*, como

[1]OpenGL não tem qualquer relação com o sistema de janela, assim o contexto gráfico é uma estrutura que determina onde a cena renderizada será exibida, bem como outros parâmetros de configuração[41].

ilustra a Figura 4.3. Neste instante, o mecanismo de despacho das janelas perde a referência sobre o contexto gráfico corrente. Por exemplo, podendo desenhar objetos na *scene view* com os parâmetros da *global view*. Isso gera ambigüidades sobre qual janela se capturou as informações, levando a efeitos efeitos visuais incorretos.

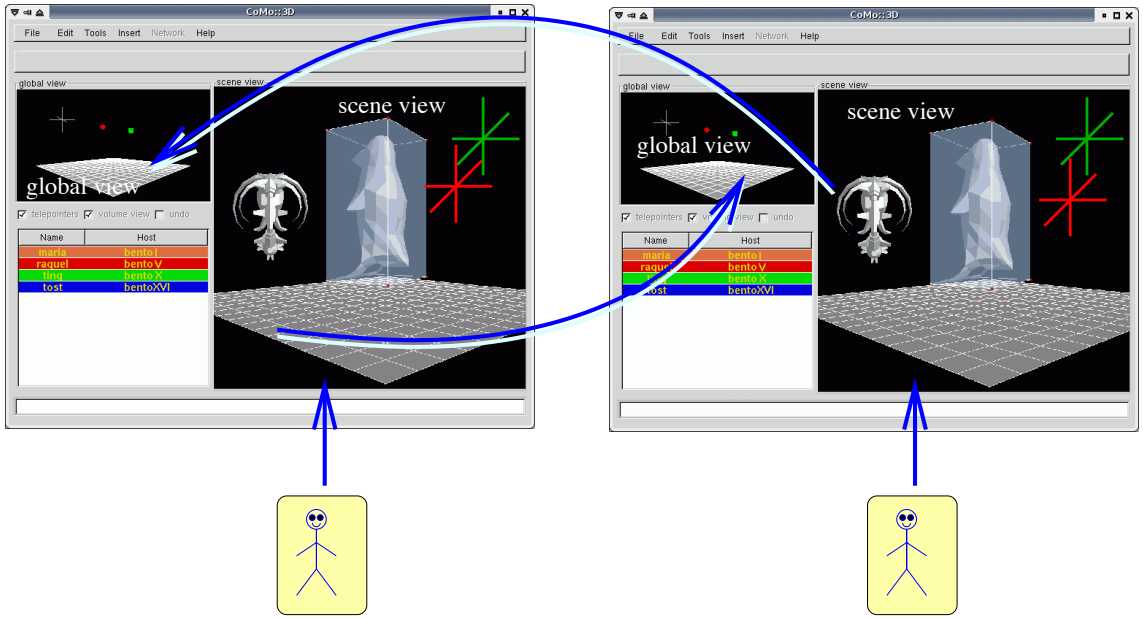


Figura 4.3: Visão geral da interface gráfica e dos componentes *scene view* e *global view*

A movimentação correta do Cursor-3D depende da obtenção das coordenadas e dimensões da *scene view*, que o contexto gráfico corrente seja o da *scene view*. Para assegurar isso, projetamos semáforos que impedem que a troca de contexto seja realizada antes do término da descarga dos comandos *OpenGL* no contexto corrente. Esta solução apresentou-se muito satisfatória para o propósito do nosso projeto, apesar de conhecermos limitações do X neste sentido. Como trabalho futuro, uma solução mais interessante, requereria uma árdua tarefa de reestruturação do X e GLX para processar adequadamente múltiplas *widgets* de renderização e manipulação 3D.

4.5 Uma Visão de implementação

A Figura 4.4 fornece uma visão global da realização da arquitetura fracamente acoplada (Capítulo 2), adicionando-se os elementos do modelo de interação (Capítulo 3)

e os componentes de consciência propostos neste capítulo.

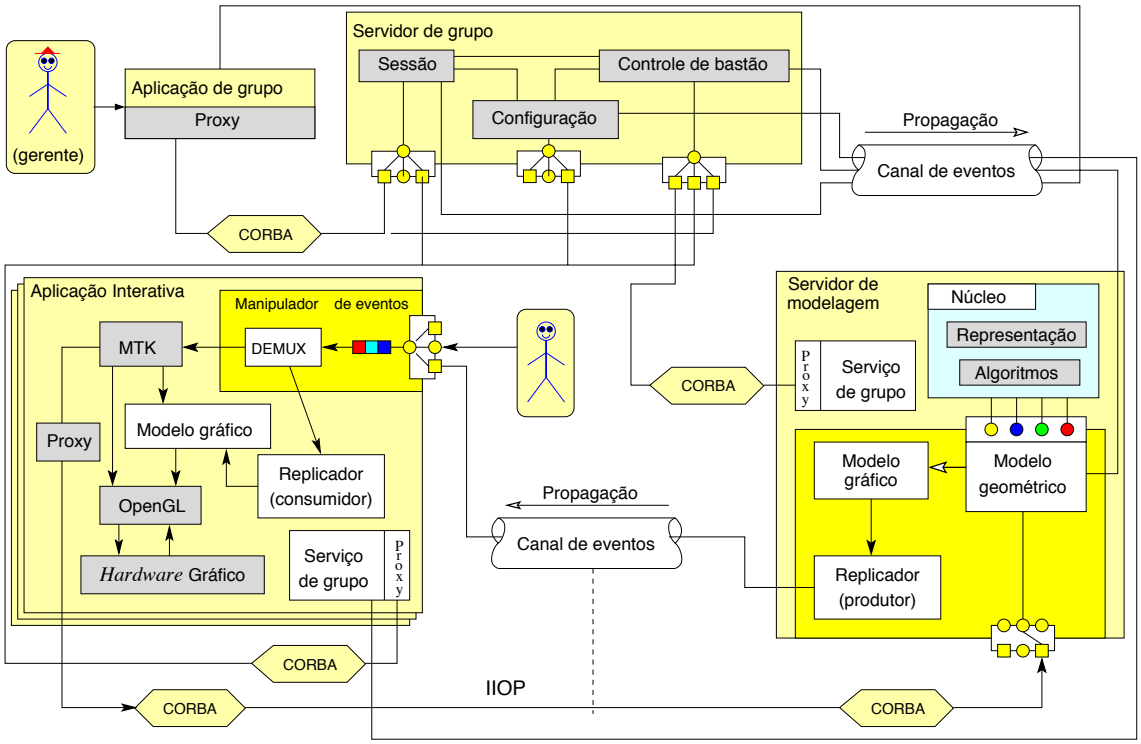


Figura 4.4: Visão geral da arquitetura fracamente acoplada com os componentes de interação e consciência coletiva.

Para realizar o componente lista de participantes e o suporte para a coordenação do trabalho colaborativo, implementamos o serviço de grupo descrito na seção 4.2.1. Já para utilizar as próprias metáforas de manipulação para prover informações de consciência coletiva, estendemos as funcionalidades do *MTK*. Chamamos a versão da *MTK* estendido de *NetMTK*, a ser detalhada na seção 4.5.2.

4.5.1 Gerência de grupo

A configuração do serviço de grupo (sessão e controle de bastão) é realizada pelo gerente de sessão remota ou localmente. Para tanto, o serviço de grupo fornece uma interface (com *proxy* remoto) que pode ser utilizada pelo gerente de sessão em outro espaço de endereço e de forma transparente.

A gerência de grupo é provida pelos serviços de sessão e de controle de con-

corrência. Além disso, uma interface de configuração do serviço de grupo é fornecida pela plataforma. A especificação CORBA/IDL da interface de acesso ao serviço de sessão é a seguinte:

```
interface Session {
  ...
  boolean  join ( inout Member acc );
  boolean  leave ( in Member acc );
  boolean isJoined (in Member acc);
  MemberSeq list();
  void notification();
  ...
};
```

Os tipos *Member*, *MemberSeq* representam abstrações para membros e lista dos membros de uma sessão, respectivamente. Os métodos *join*,*leave*,*isJoined* são utilizados por qualquer usuário para entrar, sair ou verificar se algum outro usuário está em uma determinada sessão de trabalho, respectivamente. O método *list* retorna a lista dos usuários de uma sessão, enquanto o método *notification* permite a qualquer usuário requisitar do serviço de sessão notificações de mudanças no estado da sessão, através do envio de informações alteradas para os membros do grupo, como por exemplo, avisos de entrada e saída da sessão de algum usuário.

A configuração de uma sessão de trabalho, é realizada pelo gerente da sessão remotamente. Para tanto, o serviço de grupo disponibiliza uma interface para acesso às funcionalidades oferecidas pelo servidor de grupo. A especificação da interface CORBA/IDL é a seguinte:

```
interface GroupConfig {
  ...
  void chairman (in Member acc);
  void setStatus (in SESSION_STATUS status);
  SESSION_STATUS  getStatus ();
  void setIntegrity (in INTEGRITY_CRITERIA criteria);
  INTEGRITY_CRITERIA getIntegrity ();
```

```
  void limitOfMembers (in short lower, in short upper);
  unsigned short  getLowerLimit ();
  unsigned short  getUpperLimit ();
  void setInfo(in SessionInfo sinfo);
  SessionInfo getInfo();
  boolean isJoinable();
 ...
};
```

O gerente pode ingressar na sessão antes de realizar qualquer configuração. Assim, ele estará apto a realizar as devidas configurações da sessão, tais como definir a quantidade de usuários, políticas de ingresso, dentre outras. Por fim, o gerente pode disponibilizar a sessão para os demais usuários. Os critérios adotados para regulamentação da sessão dizem respeito à disponibilidade de uma referida sessão de trabalho (*status*), integridade quanto à dinâmica de ingresso e limites de usuários na sessão. Métodos de consulta a estes critérios estão disponíveis na interface distribuída. Os métodos *setInfo* e *getInfo* são utilizados para a consulta ou ajuste dos critérios, através de uma única requisição. Já o método *isJoinable* retorna verdadeiro se a sessão estiver pronta para receber os usuários e falso, caso contrário. Se todos os critérios forem satisfeitos, o gerente de sessão autoriza o início dos trabalhos; caso contrário, o gerente pode suspender ou até mesmo encerrar os trabalhos.

Por questões de generalidade da plataforma, extensibilidade através da acomodação de diferentes políticas e escalabilidade quanto ao número de usuários e modelos, é conveniente o serviço de controle de concorrência baseado em passagem de bastão (*token*) ficar no servidor de grupo. Assim projetamos um servidor de grupo, que agrega o serviço de sessão e o controle de concorrência (classe FloorControl). O acesso ao servidor de grupo é feito através de *proxies* dos serviços de sessão e do controle de concorrência nas aplicações interativas, no servidor de modelagem e na aplicação de grupo. Nesta última aplicação há ainda o *proxy* de configuração de sessão. A Figura 4.5 esquematiza a relação entre estes componentes.

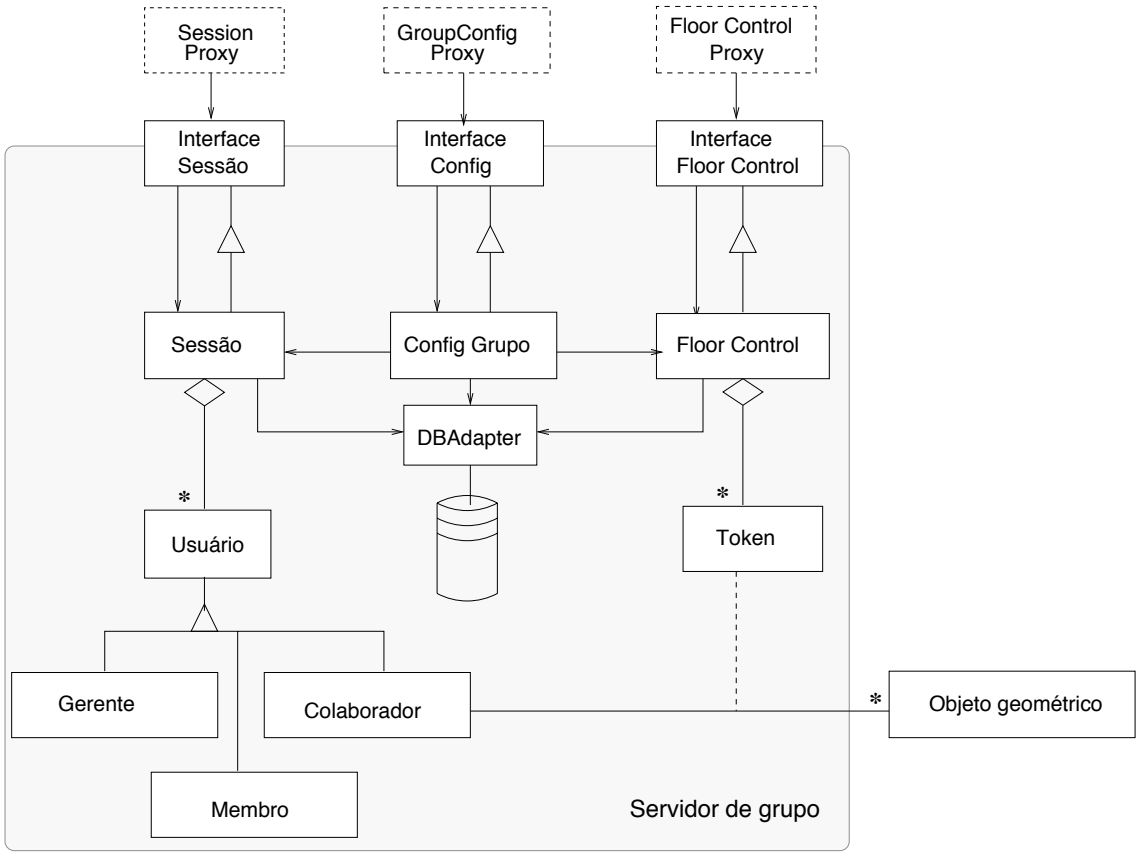


Figura 4.5: Arquitetura do serviço de grupo (servidor).

4.5.2 netMTK

A arquitetura do *MTK*, com a separação explícita entre modelo e metáforas, permite a sua utilização direta para suporte ao modelo de interação distribuída na arquitetura fracamente acoplada, bastando definir a granularidade do acoplamento nos manipuladores. Na maioria dos casos, atualiza-se o *dragger* a cada evento de manipulação e realiza-se uma requisição de operações ao servidor de modelagem. No entanto, para implementação dos componentes de consciência coletiva, implementamos uma extensão no *MTK*, adicionando funcionalidades de coleta de informações de consciência (posição, orientação de *draggers* ativos, de *cursor*-3D ativos e também os parâmetros do volume de visualização) e um serviço de replicação imediata para distribuir as informações coletadas para as demais instâncias da aplicação interativa. Denominamos esta extensão do *MTK* de *NetMTK*, Figura 4.6.

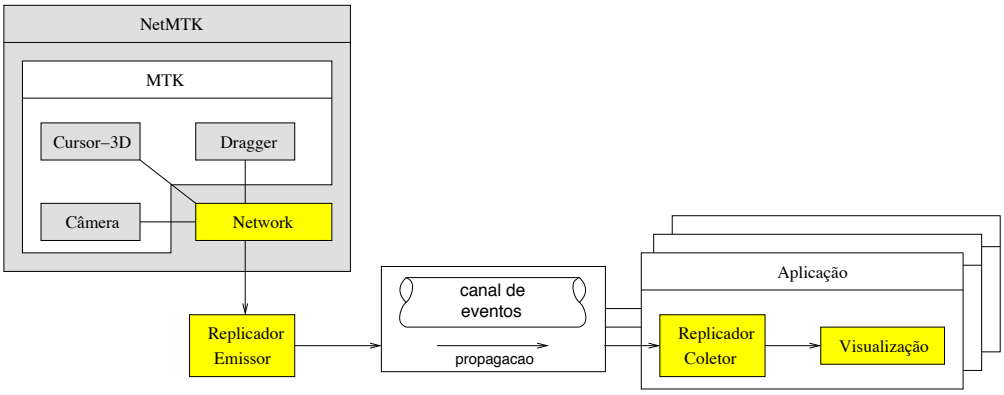


Figura 4.6: Extensão do *MTK* para prover informações de consciência

O módulo *network* é responsável pela coleta de informações de consciência coletiva, sem a necessidade de ação ativa dos usuários. Três classes compõem o compõem: *mtkNetCamera*, *mtkNetDragger* e *mtkNetCursor*, Figura 4.7.

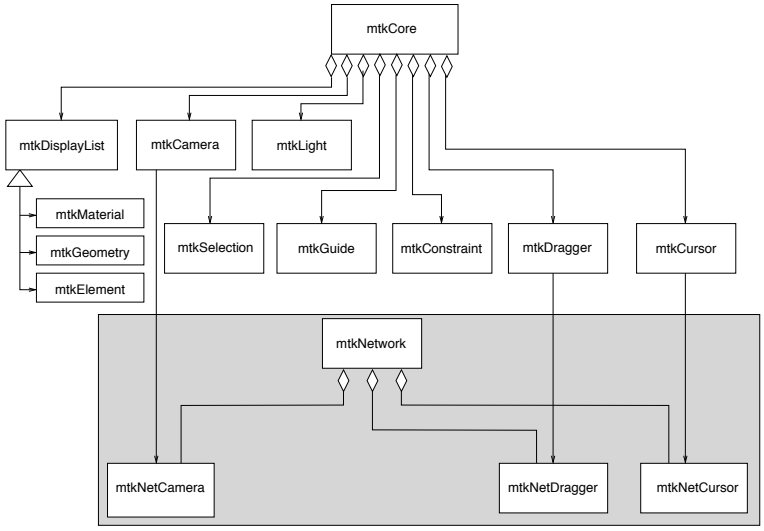


Figura 4.7: Diagrama de classe no *NetMTK*

A classe *mtkNetCamera* captura a posição e orientação da câmera e o volume de visualização, enquanto a *mtkNetDragger* e *mtkNetCursor* capturam as posições do *dragger* e do Cursor-3D ativos, respectivamente.

O mesmo replicador projetado para enviar os objetos gráficos do modelador geo-

métrico para as aplicações interativas é utilizado para a replicação das informações de consciência. As subclasses da *mtkNetwork* formatam as respectivas informações de consciência e as entregam para o replicador, que as distribui através do canal de eventos para as demais instâncias da aplicação interativa. A recepção destas informações é feita pelo replicador/coletor, que as entrega para a aplicação. Para promover maior transparência no processamento da informação recebida, incluímos três outras classes para recuperar informações enviadas e colocá-las em um formato adequado para a aplicação poder visualizá-las corretamente. Estas classes são: *TeleCamera*, *TeleDragger* e *TeleCursor*. Ao identificar o tipo de informação que fora recebida, o replicador/coletor entrega esta informação para uma instância adequada de uma destas classes para que ela seja processada e recuperada corretamente.

4.6 Experimentos

Nesta seção, nós apresentaremos resultados quantitativos da latência de consciência, expressa pela equação (4.1). Para não incorrer em avaliações e comparações equivocadas, os experimentos realizados neste capítulo, utilizaram os mesmos equipamentos dos experimentos e a mesma metodologia do capítulo 3.

Conforme o trabalho de Ellis e Wainer [20], é importante avaliar o impacto gerado pela inclusão de componentes de consciência propostos neste modelo, no que se refere às possíveis melhorias no processo colaborativo. Apresentamos na seção 4.6.2, portanto, uma avaliação subjetiva realizada com grupos de estudantes de engenheria de computação, de acordo com dois questionários.

4.6.1 Latência de tele-metáforas

Qualquer manipulação 3D ou mudança no volume de visualização realizada por algum usuário é coletada e replicada diretamente para as demais instâncias da aplicação interativa. Neste caso, o tempo de resposta é calculado desde o ação do usuário até a visualização pelos demais. O gráfico na Figura 4.8 mostra a latência média para os componentes de consciência.

O baixo tempo de resposta das metáforas de consciência é substancialmente re-

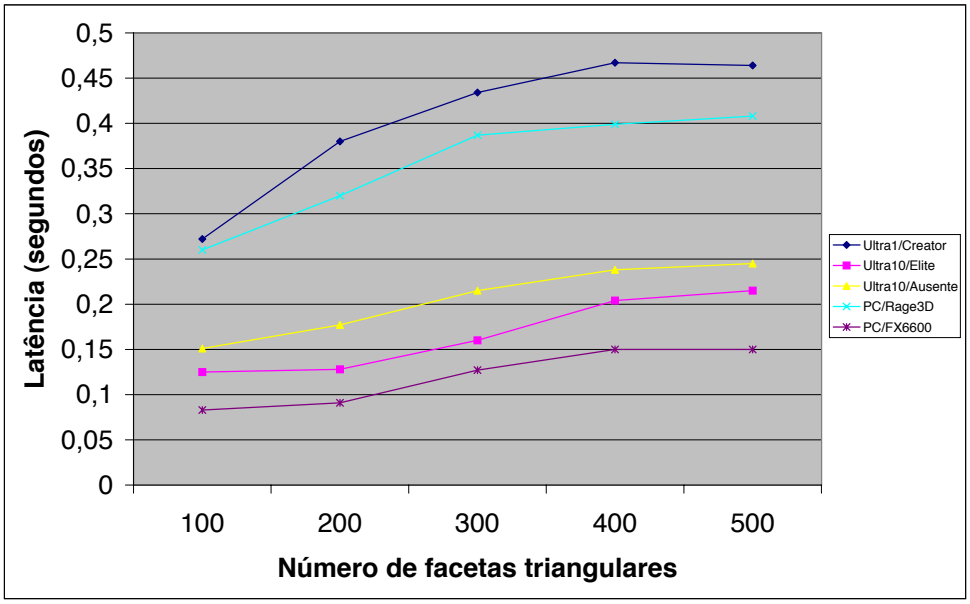


Figura 4.8: Latência de consciência (em segundos).

levante para o modelo proposto, pois está diretamente ligado à dinâmica do grupo e à usabilidade dos componentes de consciência.

4.6.2 Avaliação subjetiva

As informações de consciência mais valiosas são aquelas que ajudam o usuário a perceber as atividades de algum outro usuário, principalmente quando as ações deste usuário depende do que o outro está realizando. No entanto, é necessário que as informações de consciência, além de fáceis de serem percebidas, não provoquem dispersão. A tarefa seria criar uma cena-3D formada por objetos primitivos, e objetos mais complexos ao realizar operações booleanas entre estes.

Os experimentos realizados tomaram como base dois cenários distintos: uma versão da aplicação interativa, sem os componentes de consciência e sem as tele-metáforas; e uma outra versão do protótipo com a inclusão dos componentes e tele-metáforas. Por se tratar de experimentos com usuários para a aferição de parâmetros subjetivos de usabilidade, elaboramos dois questionários para serem respondidos durante e imediatamente após as sessões de colaboração utilizando o *CoMo* (*Collaborative Modeling*) nas duas configurações. Antes de cada avaliação, os usuários passaram por sessões de treinamento e utilização livre. As seguintes perguntas foram colocadas no questionário, quando a aplicação não disponibilizou os mecanismos de consciência coletiva:

1. Sente a presença de outros usuários no processo?
2. Sente a necessidade de saber das ações dos demais usuários?
3. Sabe o que os demais usuários estão fazendo?
4. Consegue mostrar objetos a outros usuários?
5. Consegue visualizar objetos que outro usuário mostra?

As seguintes perguntas foram colocadas no questionário, quando a aplicação disponibilizou os mecanismos de consciência coletiva:

1. Sente a presença de outros usuários no processo?

2. Sente a necessidade de saber das ações dos demais usuários?

3. Sabe o que os demais usuários estão fazendo?

4. Consegue mostrar objetos a outros usuários?

5. Consegue visualizar objetos que outro usuário mostra?

6. Sente maior conforto com a presença dos mecanismos de consciência?

7. A presença dos mecanismos de consciência ajudaria na modelagem colaborativa?

8. A inclusão de tele-metáforas no componente de visão de cena provoca dispersão?

Essas questões são baseadas fortemente no trabalho do Professor Nielsen [48]. As possíveis respostas às perguntas são: sempre, quase sempre, às vezes, nunca. Foram realizados quatro experimentos com cinco grupos formados por cinco usuários, em um mesmo ambiente de trabalho, porém sem a possibilidade de nenhum usuário ter contato direto com o que o outro está fazendo. O perfil médio dos usuários é formado por estudantes de computação, familiarizados com o uso de computadores, mas em sua maioria sem habilidades para manipulação de ferramentas 3D. As tabelas 4.1 e 4.2 mostram os percentuais médios de respostas às perguntas elaboradas no questionário, ou seja, após os experimentos calculou-se as médias de cada resposta.

Pergunta	Sempre	Quase sempre	Às vezes	Nunca
1	31,2	20,1	34,8	13,9
2	12,3	30	24,9	32,8
3	3,5	4,6	72,4	19,5
4	2,5	4,6	23,5	69,5
5	3,0	10,9	45,6	40,5

Tabela 4.1: Resultados do experimentos *sem* mecanismos de consciência

Os dados colhidos demonstram a contribuição dos componentes de consciência e metáforas 3D propostos neste capítulo para o aumento da consciência coletiva do sistema de modelagem geométrico.

Pergunta	Sempre	Quase sempre	Às vezes	Nunca
1	98,7	1,3	0,0	0,0
2	62,3	23,2	11,7	2,8
3	65,3	33,2	0,7	0,8
4	64,7	31,4	2,1	1,8
5	66,1	29,4	2,3	2,2
6	52,1	22,2	16,3	9,4
7	78,9	20,3	0,5	0,3
8	40,5	48,3	9,8	1,4

Tabela 4.2: Resultados do experimentos *com* a presença de mecanismos de consciência

4.7 Resumo e considerações

Com base em estudos recentes sobre consciência coletiva 3D, vislumbramos que nossa proposta de novos componentes de consciência e de utilização de metáforas 3D para aumento de consciência coletiva traz uma significativa contribuição à modelagem geométrica em ambiente colaborativo. Partindo de nossas observações, as tele-metáforas e as tele-visões têm se mostrado efetivas no suporte à consciência de tarefa, bem como a adequação do volume de visualização para a consciência de perspectiva.

A concepção de componente fora da visão de cena tem sido muito significativa, pois torna possível a utilização de uma grande quantidade de informações para apoio ao trabalho em grupo, sem gerar dispersão. Finalmente, sob nosso ponto de vista, a maior contribuição deste capítulo é mostrar conceitualmente que vários mecanismos de interação individual são de grande valor no suporte a mecanismos de consciência coletiva.

A sinergia de outros canais de comunicação possibilita, juntamente com os componentes apresentados, aumentar ainda mais a consciência coletiva. Vale comentar aqui que na literatura especializada há um consenso de que a comunicação por voz seja vital para qualquer sistema colaborativo. Por decisão de projeto, resolvemos não incluir em nossa proposta componentes de comunicação via áudio ou vídeo conferências [17, 69] para não abandonarmos nosso foco principal: a exploração do uso dos recursos gráficos 3D num ambiente colaborativo, logrando a importância de se adotar quaisquer outros recursos durante o trabalho colaborativo.

Capítulo 5

Um protótipo

Neste capítulo, apresentamos a implementação de um protótipo do modelador colaborativo baseado na arquitetura esquematizada nas Figuras 2.12, 3.8 e 4.4. Foram utilizadas as mesmas funcionalidades de modelagem geométrica descritas nas seções 2.5, 3.4 e 4.5. Este protótipo é denominado *CoMo* (*Modelador Colaborativo*), disponilizado no *site* do grupo `http://www.dca.fee.unicamp.br/projects/como/`.

O *CoMo* é formado por 4 aplicações: o servidor de modelagem, o servidor de grupo, a aplicação (cliente) de grupo e a aplicação interativa. Para o funcionamento do *CoMo* somente uma instância do servidor e outra do servidor de grupo são executadas, podendo compartilhar uma mesma máquina ou estar em máquinas diferentes. Observe que colocar ambos servidores em uma máquina é uma questão de conveniência, nada impede que residam em máquinas distintas. As interconexões entre as máquinas com aplicações interativas e o servidor de modelagem foram feitas utilizando uma rede Ethernet de 100 Mb/s e outra de 10 Mb/s, como descrito na seção 2.6.

Revisitando o cenário estabelecido na seção 1.1 e os experimentos relatados nos Capítulos 2, 3 e 4, exibimos uma disposição das aplicações *CoMo* na Figura 5.1.

5.1 Servidor de grupo

O servidor de grupo dispõe de três interfaces de acesso remoto aos seus serviços, uma para configuração dos serviços (GroupConfig) e outras duas para o acesso aos serviços providos pelos módulos de sessão (Session) e de controle de concorrência (FloorControl).

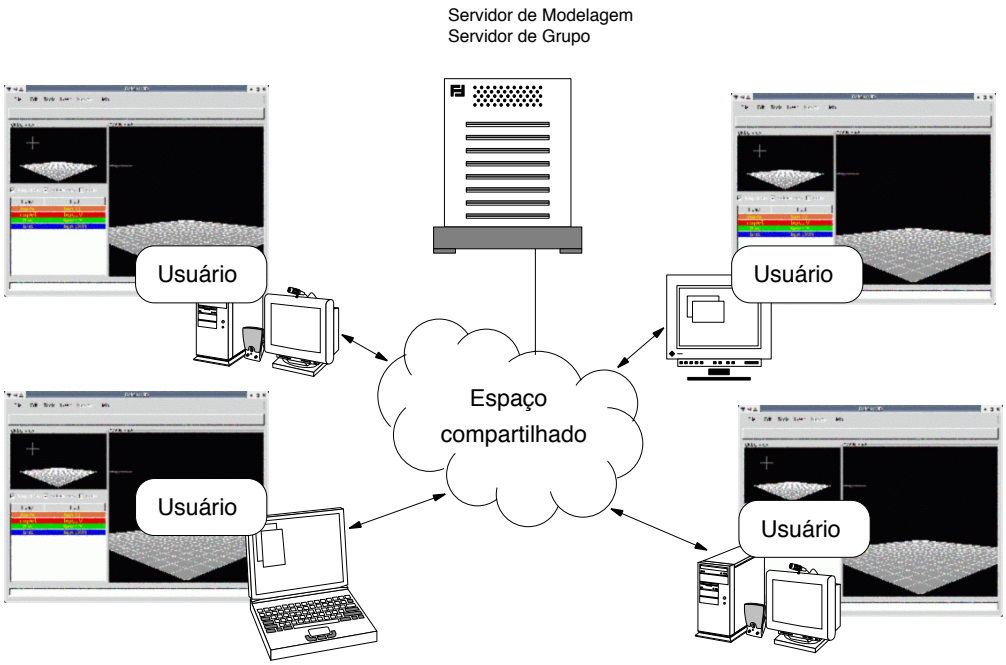


Figura 5.1: Um cenário para modelagem geométrica colaborativa usando o *CoMo*.

A Figura 4.5 mostra os componentes do servidor de grupo, destacando os *proxies* (pontilhado), que são objetos responsáveis pelo acesso remoto aos componentes do servidor de grupo a partir das aplicações interativas e servidor de modelagem.

Para assegurar a integridade do sistema, distinguimos em cada sessão (`Session`) os papéis dos usuários em três grupos, de acordo com os direitos de acesso e manipulação de um determinado objeto:

Membro: usuário pertencente a uma sessão de trabalho, com direitos à visualização da cena e manipulação de sua câmera e os parâmetros do volume da cena a ser visualizada.

Colaborador: membro com direto também à manipulação dos objetos em cena. A manipulação fica condicionada ao colaborador possuir o direito explícito, através da posse de um 'bastão' que correlaciona o objeto e o seu colaborador.

Gerente: membro responsável pela coordenação da sessão e suas atividades internas. Cada sessão de trabalho possui um único gerente. O gerente pode ser um membro ativo da sessão, assumindo os papéis de usuário e colaborador.

Além disso, diferenciamos a forma que um determinado usuário ingressou em cada sessão. Um usuário pode negociar um convite para uma sessão de trabalho, considerando três políticas distintas:

estática : cada usuário deve juntar-se à sessão através de uma negociação prévia e antes da sessão de seu início.

dinâmica e fechada : cada usuário deve ser explicitamente convidado a participar da sessão, podendo ingressar a qualquer momento.

dinâmica e aberta : cada usuário pode ingressar nesta sessão a qualquer momento, por convite ou por iniciativa própria.

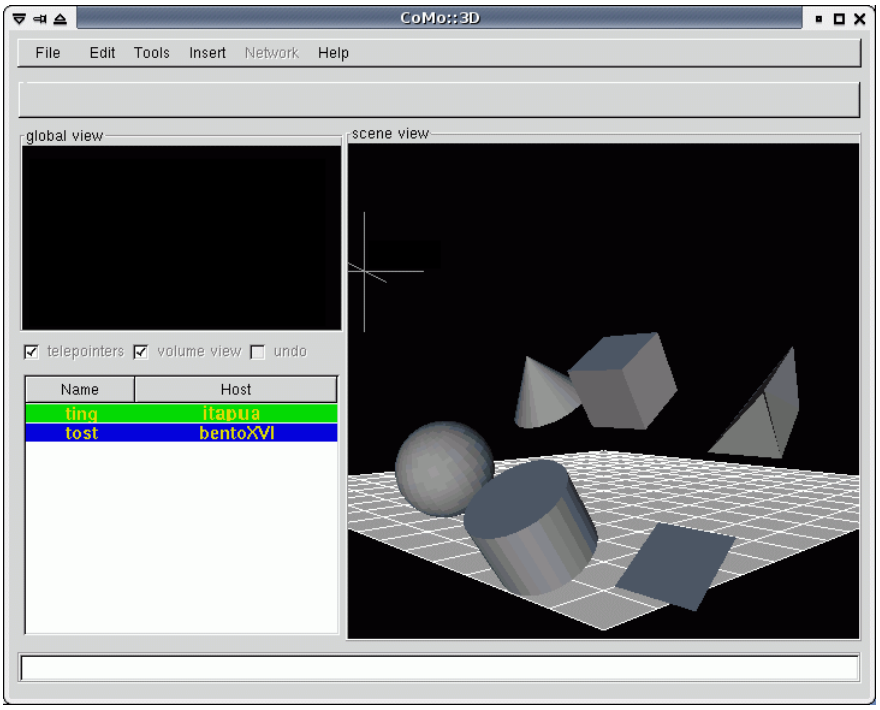
A aplicação de grupo, que permite ao gerente configurar a sessão, tem acesso privilegiado ao servidor do grupo. O Gerente da sessão pode configurar remotamente o acesso aos serviços de sessão através da interface *GroupConfigProxy*. Dentre as tarefas que podem ser realizadas estão o ajuste dos números máximo e mínimo de membros de uma sessão, a política de acesso dos usuários e a forma de ingresso de cada usuário em uma sessão.

5.2 Servidor de modelagem

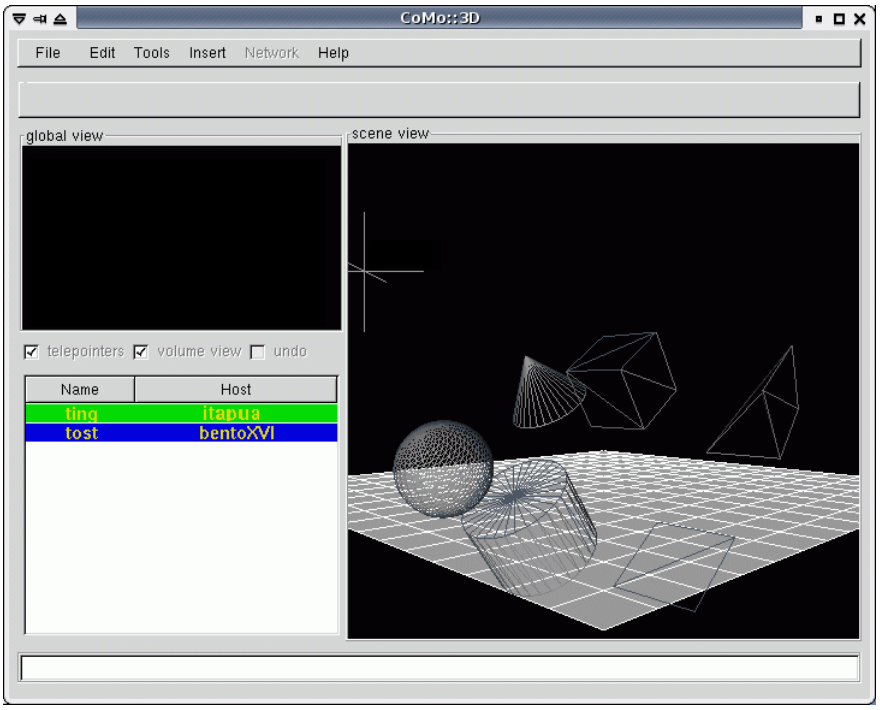
O servidor de modelagem implementado no *CoMo*, possui um conjunto de primitivas e operações capazes de modificá-las, de forma a se obter objetos mais complexos. As primitivas do *CoMo* são: cunha, cubo, cone, cilindro, esfera e polígonos simples[1], Figura 5.2. A partir de um polígono simples, pode-se gerar objetos mais complexos através da operação de extrusão. As primitivas são armazenadas conforme esquema de esquema de representação por fronteira [36]. A definição e manipulação das primitivas é realizada por um conjunto de operadores de Euler [78].

Operadores booleanos robustos, desenvolvidos originalmente para aplicações mono-usuário [13], foram incorporados ao servidor, para assegurar a consistência dos modelos gerados a partir das primitivas e modificados durante uma sessão de trabalho colaborativo, Figura 2.15.

[1]Um polígono é dito ser simples se não possui auto-interseções.



(a) Sombreamento facetado



(b) *Wireframe*

Figura 5.2: Primitivas do *CoMo*.

Além disso, operações como transformações afim, adição e remoção de objetos podem ser requisitadas remotamente ao servidor de modelagem, através da uma interface de acesso remoto.

5.3 Aplicação interativa: interface gráfica

Mostramos na Figura 5.3 a interface da aplicação interativa, na qual destacam-se as subjanelas *global view*, *scene view*, lista de participantes e a janela *pop-up* para autenticação.

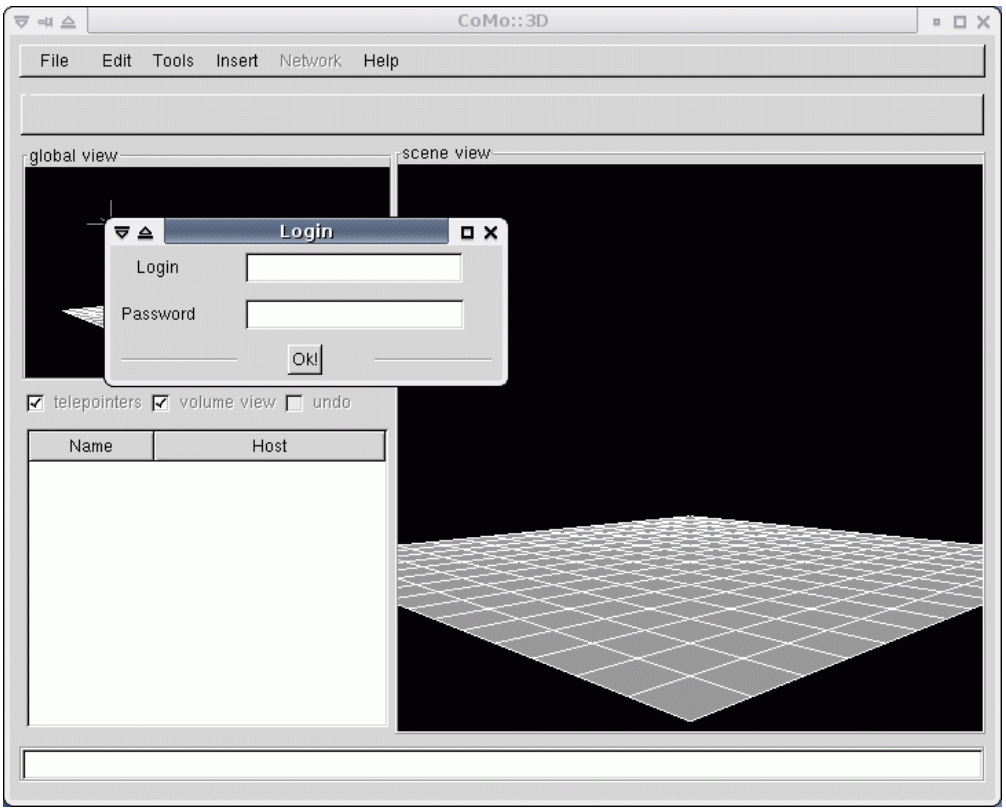


Figura 5.3: Interface do modelador colaborativo.

Através da barra de *menu*, tem-se acesso às seguintes funcionalidades da aplicação interativa:

File carregamento de modelos nos formatos OBJ/Wavefront ou LWO e o término da execução da aplicação (desconexão da sessão de trabalho);

Edit seleção e liberação de objetos, remoção de objetos selecionados, alteração dos modos de renderização e sombreamento, alteração de cores e materiais utilizados para o sombreamento dos objetos, Figura 5.4;

Tools permutação entre o cursor 2D (uso de *draggers*) e o cursor-3D, ativação e desativação *grid* e eixos, seleção do tipo de *dragger* e operadores booleanos, Figura 5.5;

Insert inserção de primitivas (cunha, cubo, esfera, cilindro, cone e polígono) na cena e operação de extrusão (produzir objetos mais complexos geometricamente a partir de primitivas planares), Figura 5.6;

Help descrição sucinta da aplicação; créditos.

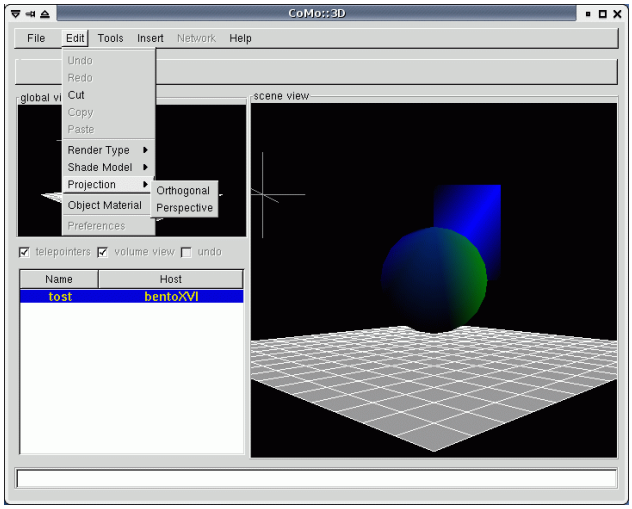


Figura 5.4: Menu *Edit* da aplicação interativa

Observar na Figura 5.6, que a opção *polygon* está desabilitada, pois o cursor-3D que marca os pontos do polígono, está desativado. Da mesma forma, a opção *extrude* é habilitada quando algum polígono é selecionado para a extrusão. Já escolha do tipo de *dragger* é somente habilitada quando cursor-2D está ativo, enquanto os operadores booleanos são disponibilizados quando dois objetos são selecionados. A ordem da seleção influencia o resultado da operação de subtração, pois o primeiro objeto a ser selecionado será o primeiro argumento da operação.

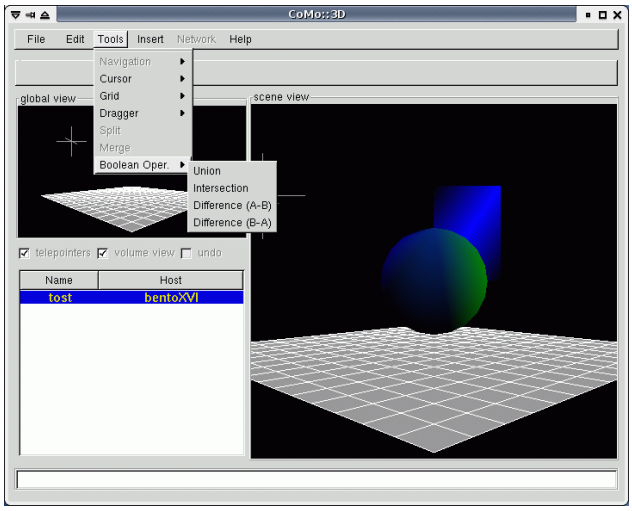


Figura 5.5: Menu *Tools* da aplicação interativa

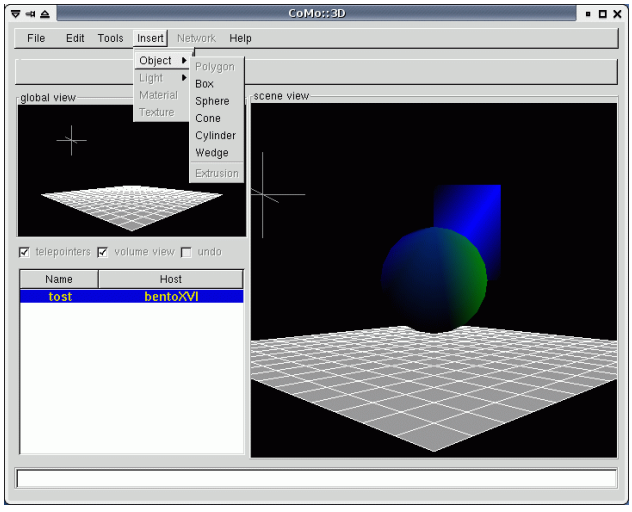


Figura 5.6: Menu *Insert* da aplicação interativa

5.4 A dinâmica da colaboração

A dinâmica de uma sessão de trabalho inicia-se com a ativação dos servidores de grupo e de modelagem, seguida pela configuração da sessão pelo gerente do grupo, a conexão dos participantes e por fim o início dos trabalhos propriamente dito. A seguir mostramos a saída do *script* de ativação dos servidores:

```
./runit.server

starting CORBA
  core daemon
  naming service
  event service
starting CoMo
  replication service
  group services
  group setup
  modeling server
  system waiting for conection...
```

O *script runit.server*, além de iniciar os servidores, realiza a configuração da sessão, estabelecendo o número mínimo e máximo de usuários na sessão. Esta configuração é feita remotamente através do proxy *GroupConfigProxy*. A partir deste instante, os usuários se conectam à sessão, sem poder trabalhar ainda, pois a sessão está configurada apenas para a autenticação dos usuários. A Figura 5.3 mostra a subjanela de autenticação, através da qual o usuário entra com o nome e a sua senha. Estes dados são validados no banco de dados do servidor de grupo.

Após conectar-se a uma sessão, o usuário deve aguardar o início da mesma, que será dado pelo gerente através da aplicação da sessão. Se o critério de integridade não é estático, ele pode iniciar o trabalho imediatamente. É importante notar que o gerente somente poderá iniciar a sessão quando a quantidade mínima de usuários for atingida e suspendê-la caso a saída de algum usuário provoque a violação desta condição. O servidor de grupo não permite ingresso de usuários além do limite máximo estabelecido

na configuração da sessão. Com o início da sessão, a aplicação interativa, mostrada já com o ingresso de dois usuários na Figura 5.7, fica disponível para o trabalho.

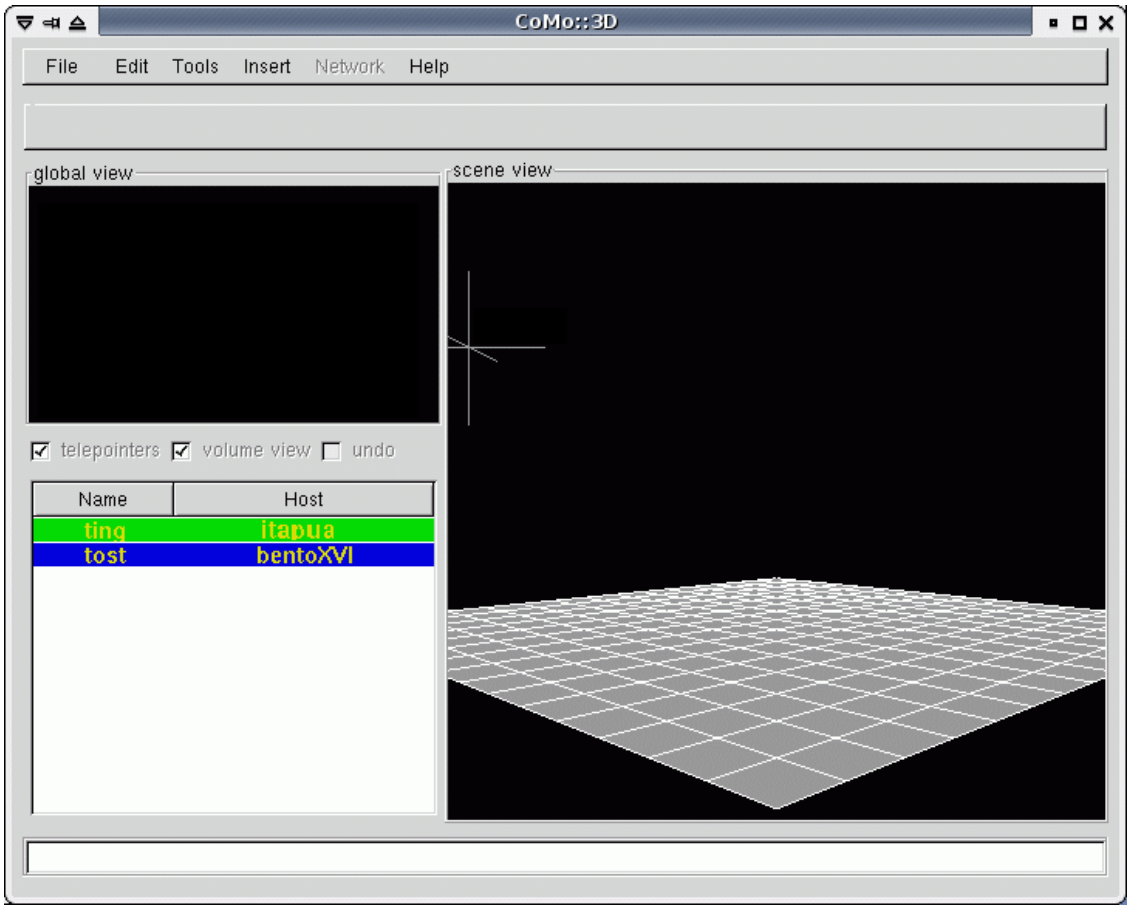


Figura 5.7: Aplicação interativa, com dois usuários conectados.

O sistema de eixos que aparece na Figura 5.7 representa o sistema de referência da *visão de cena*. Nem os cursores nem os volumes são mostrados, porque a sessão não foi iniciada ou os usuários não estão trabalhando no instante da captura da interface da aplicação interativa.

Outro cenário, agora com quatro usuários, é exibido nas Figuras 5.8, 5.9 e 5.10, porém com diferentes modos de renderização e iluminação. Para ilustrar as funcionalidades da aplicação, um objeto mais complexo é lido de um arquivo OBJ/*WaveFront* e compartilhado entre os participantes da sessão. Observar que os volumes de visualização e os teleapontadores não estão visíveis na *global view*, pois não existe naquele momento qualquer atividade de manipulação dos objetos.

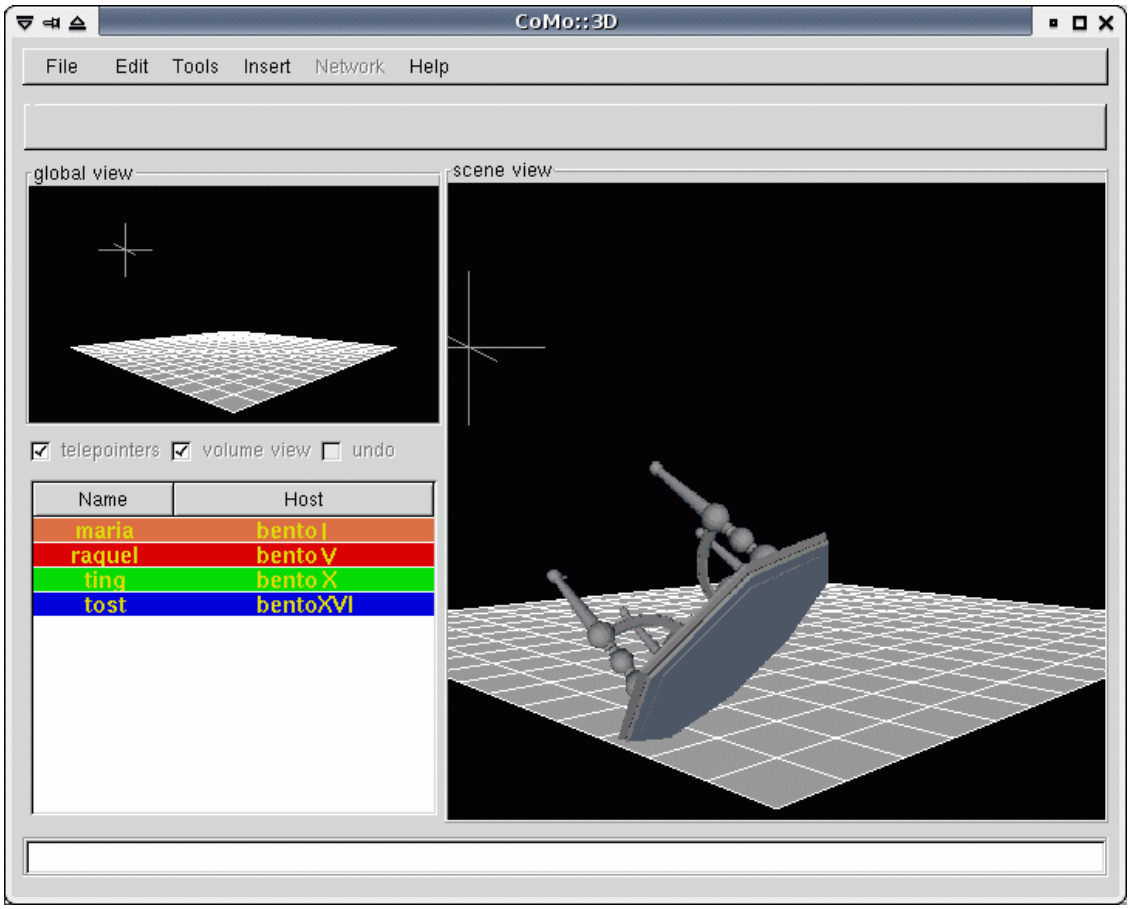


Figura 5.8: Aplicação interativa, com 4 usuários conectados, compartilhando um objeto complexo

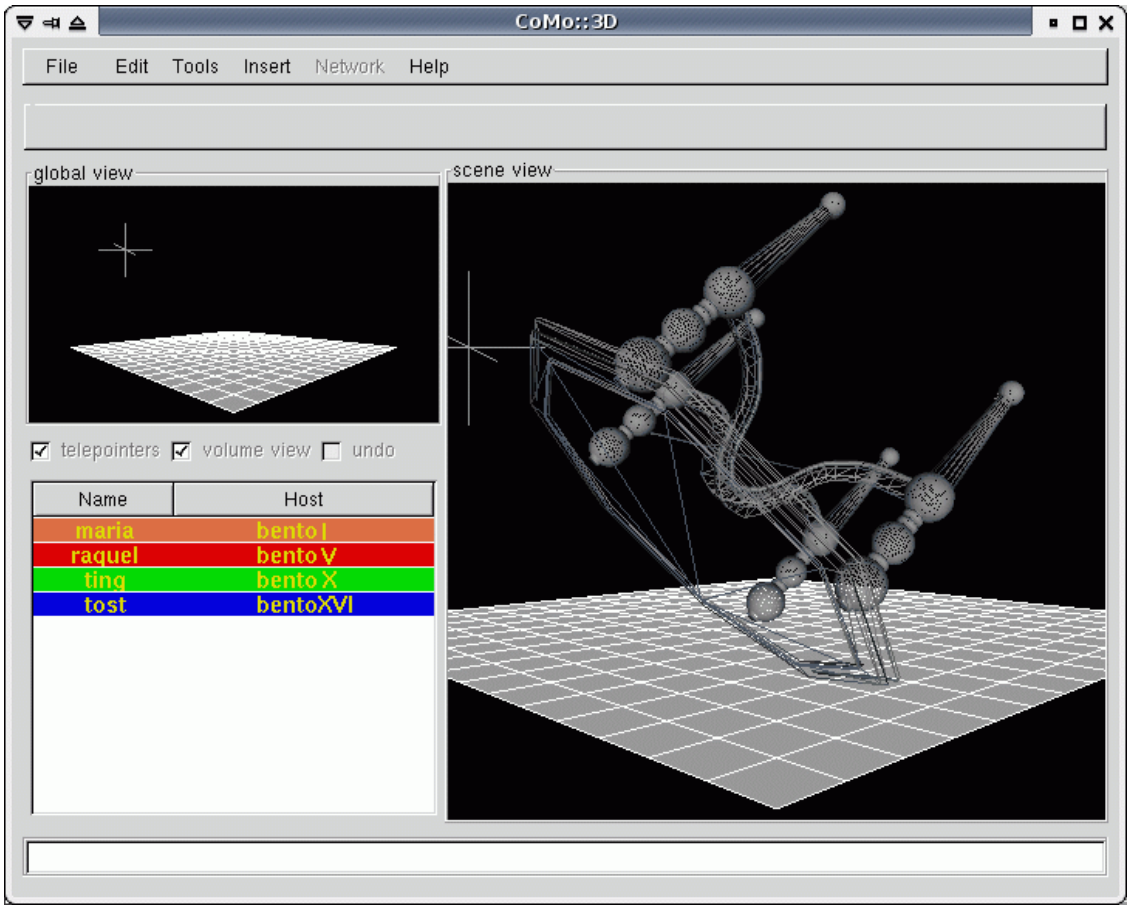


Figura 5.9: Cena da Figura 5.8, em *wireframe*.

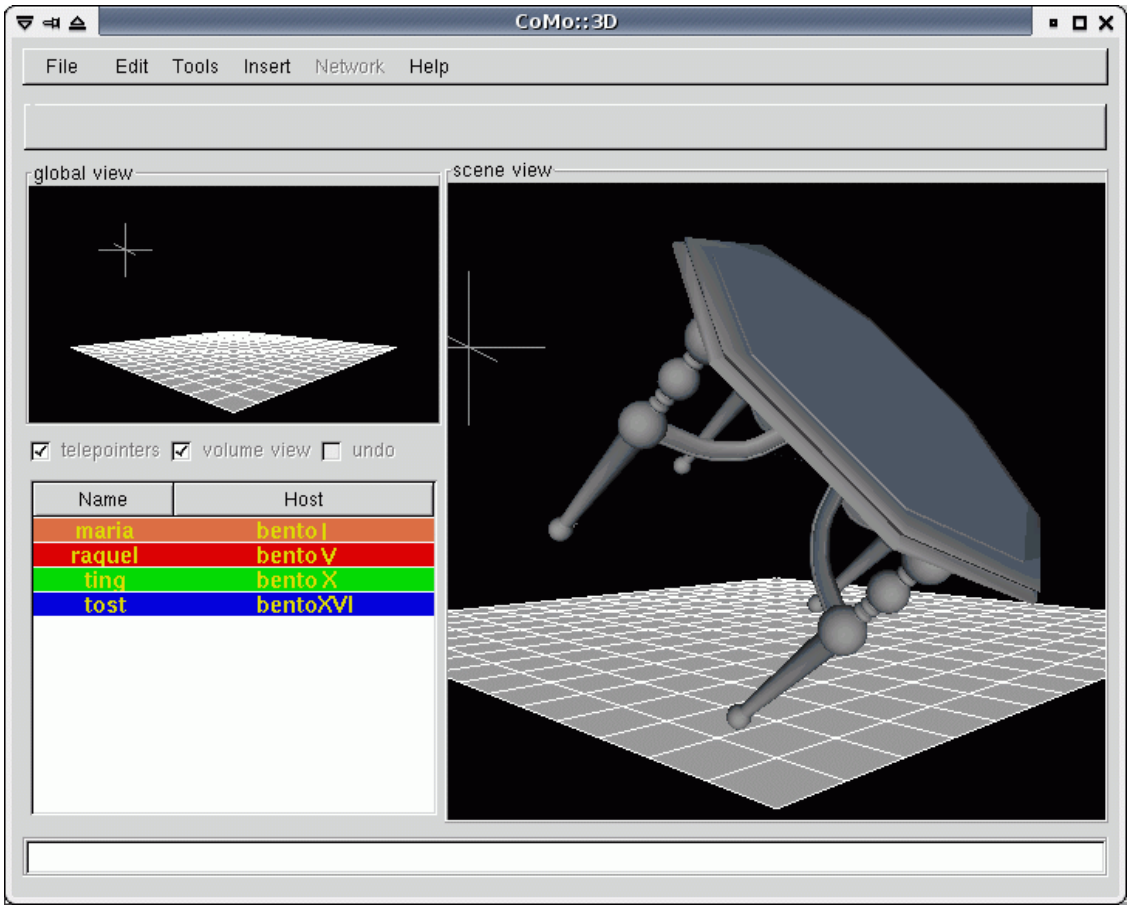


Figura 5.10: Cena da Figura 5.8, com sombreamento suave.

Além de instanciação, leitura e renderização de objetos com geometrias simples e complexas, o *CoMo* oferece aos usuários metáforas de manipulação e os componentes de consciência de grupo. As Figuras 5.11 e 5.12 mostram as metáforas em ação e as respectivas tele-metáforas, enquanto a Figura 5.13 exibe o volume de visualização. Enquanto um objeto é selecionado por um usuário, ele é envolvido pela representação gráfica da tele-metáfora, proporcionando ao usuário não só a realimentação visual das suas ações, como indicando aos demais usuários um bloqueio para acessos. Para não restringir a aplicabilidade do sistema, as tele-metáforas podem ser habilitadas/desabilitadas na *global view* e a sua visualização depende de atividade de manipulação de objetos em um determinada sessão.

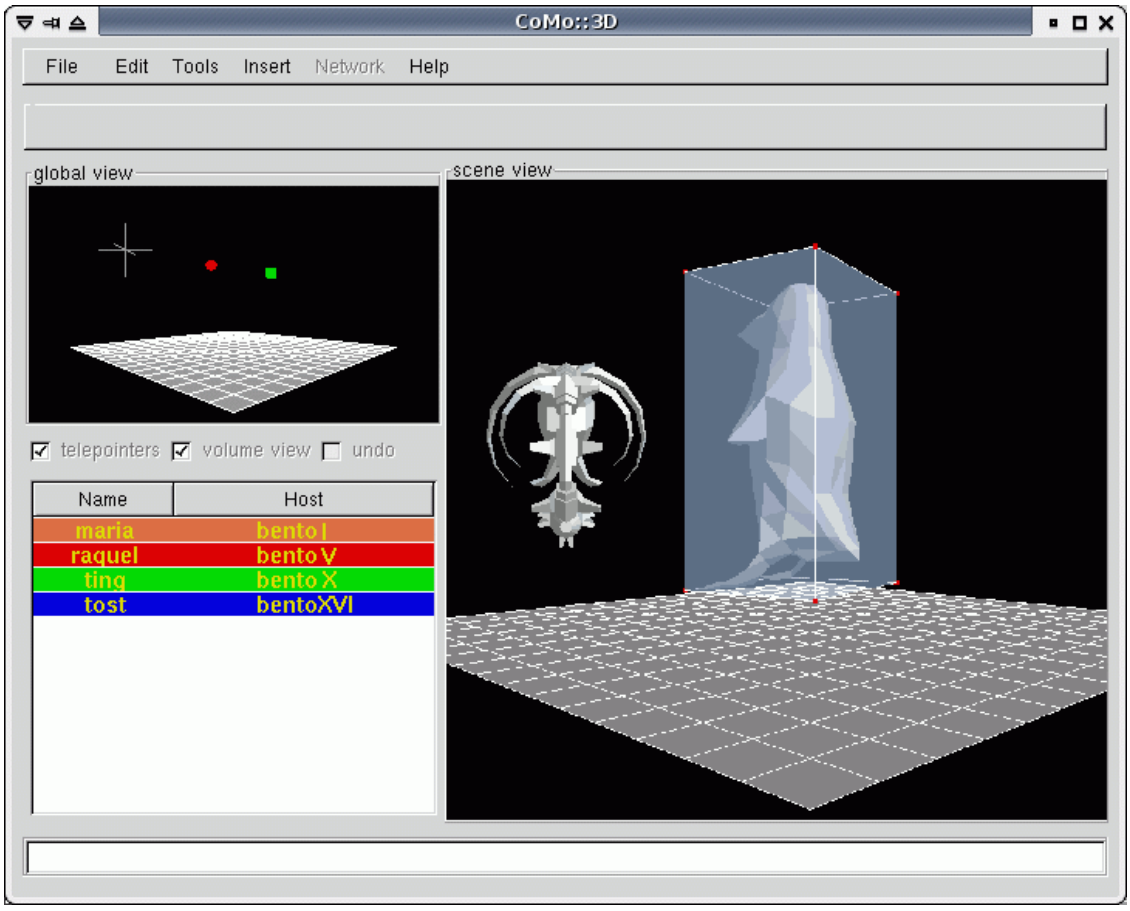


Figura 5.11: Metáforas em ação (em torno do pingüim), com tele-metáforas (*global view*

Observar que o volume de visualização somente aparece na Figura 5.13, quando cada usuário muda os suas respectivas câmeras. A escolha desta forma de implementação

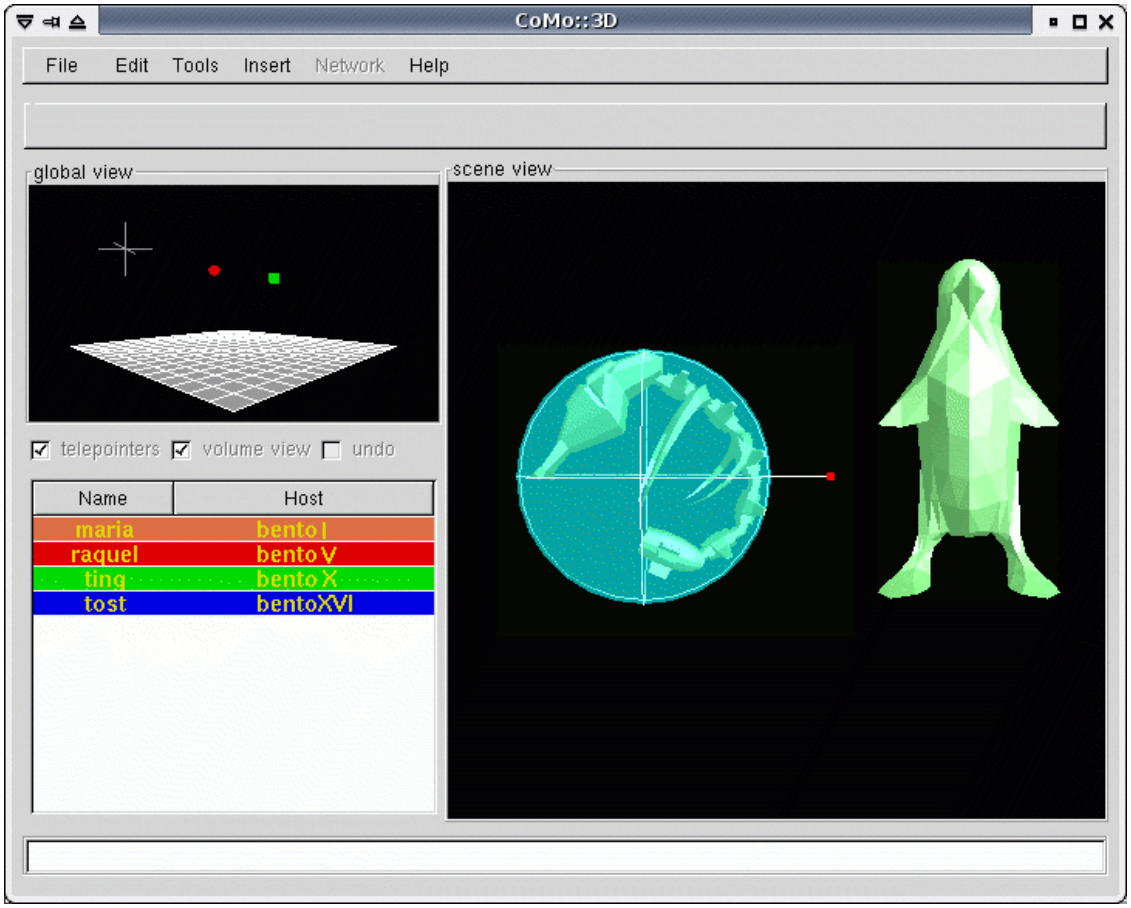


Figura 5.12: Metáforas em ação (em torno do *alien*), mesma cena da Figura 5.11.

se baseia no fato de que a necessidade de diferentes volumes de visualização ocorre quando cada um tem seu volume diferente dos demais. Assim, o fato de estar habilitado, não significa que aparecerá na cena imediatamente.

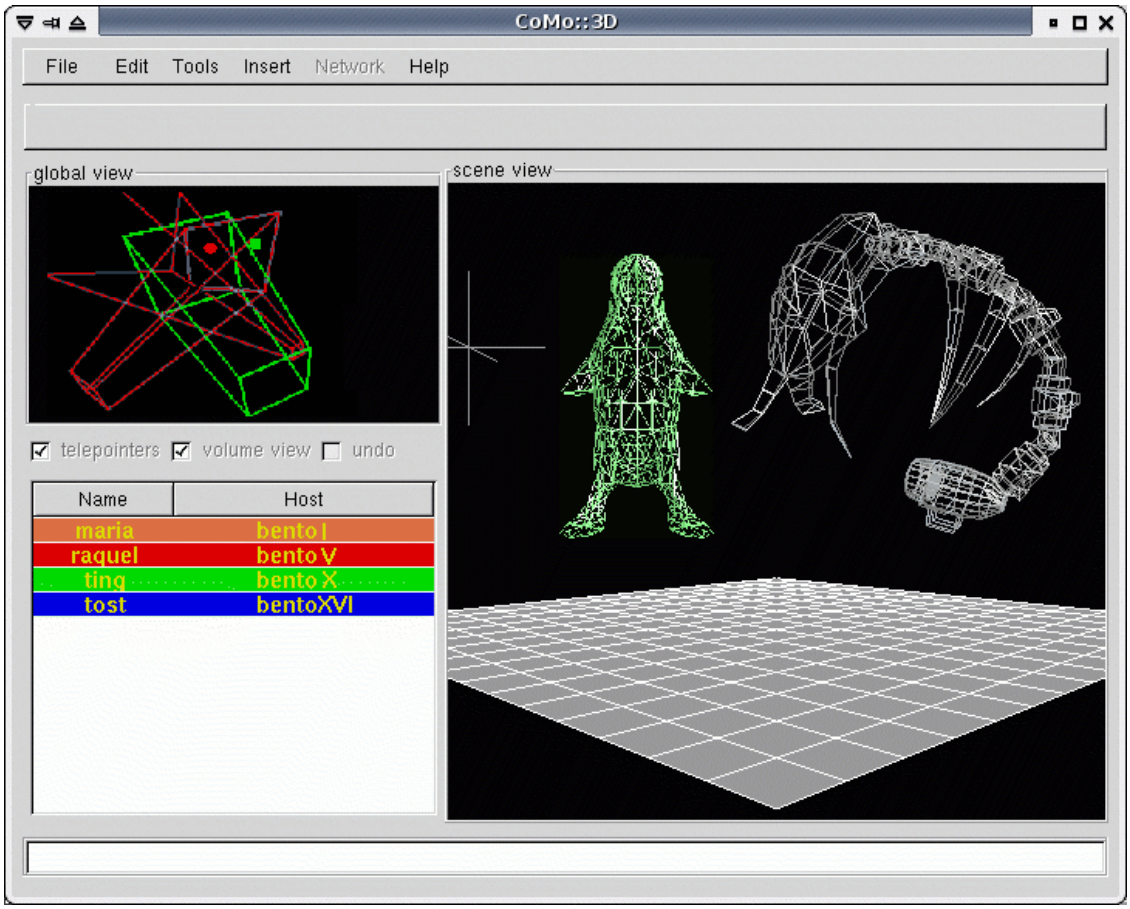


Figura 5.13: Metáforas em ação, com exibição dos volumes de visualização.

Capítulo 6

Conclusão

> "... tenho ido de tarde à biblioteca pública. E por estranho que pareça, estou estudando cálculo de probabilidades. Não só porque o abstrato me interessa, como porque eu posso renovar minha incompreensão e concretizar minhas dificuldades gerais." ... Clarice Lispector

Nesta tese estudamos aspectos fundamentais de modelagem geométrica colaborativa em uma plataforma computacional heterogênea, identificando alguns problemas relacionados ao contexto e propondo um conjunto de soluções para estes problemas. Mais especificamente, abordamos aspectos da robustez em operações geométricas, adaptabilidade à heterogeneidade da plataforma computacional (Capítulo 2), interatividade (Capítulo 3) e consciência de grupo (Capítulo 4).

Os problemas inerentes à robustez em modelagem geométrica foram tratados através de uma solução de arquitetura, que caracteriza-se pela separação dos processos de modelagem e renderização. Esta solução evita a necessidade de qualquer adaptação de algoritmos de modelagem para o contexto colaborativo, permitindo o emprego de algoritmos geométricos robustos utilizados em aplicações mono-usuário. Realizamos experimentos para comprovar os benefícios da arquitetura, utilizando algoritmos robustos para operações booleanas, desenvolvidos para uma aplicação mono-usuário sem qualquer modificação na sua concepção e implementação. Como conseqüência desta separação, uma representação do modelo geométrico é utilizada na renderização de forma independente em cada instância da aplicação interativa. Desta forma, diferentes equipamentos podem ser utilizados para o trabalho colaborativo, com a exploração individualizada da capacidade gráfica de cada um, bastando cada participante ajustar os

parâmetros de renderização de acordo com o poder computacional de seu equipamento.

A separação dos modelos com a distribuição de aplicações em diferentes máquinas, gerou a demanda por algum *middleware* capaz realizar a comunicação entre as aplicações. A arquitetura fracamente acoplada possui características peculiares, pois tanto o servidor de modelagem quanto as aplicações interativas tem comportamento de clientes e servidores concomitantemente. Além disso, as aplicações interativas recebem dados de dispositivos de entrada locais. Dentre as diversas opções disponíveis, optamos por utilizar o padrão aberto CORBA como *middleware* para comunicação cliente/servidor e o serviço de eventos CORBA para comunicação de grupo.

O serviço de eventos oferece transparência na comunicação de grupo. Os padrões *observer* e *mediator* foram adotados para prover o fraco acoplamento entre as entidades das aplicações distribuídas. No entanto, enfrentamos diversos desafios para integração das tecnologias de comunicação com o sistema de janela e a biblioteca de renderização durante desenvolvimento do protótipo do *CoMo*, devido à concorrência no tratamento de eventos provenientes de diversas fontes. Desenvolvemos um mecanismo de sincronização capaz de uniformizar a captura e tratamentos dos eventos na aplicação interativa. Para ilustrar a visualização independente entre diversas máquinas, experimentamos a visualização de modelos, ajustando os parâmetros de renderização para diferentes modos e o volume de visualização para diferentes valores.

O tempo de resposta do sistema e a percepção da presença de outros participantes são também fatores relevantes para a produtividade dos sistemas colaborativos. No entanto, a arquitetura proposta demanda um maior tempo de resposta devido à centralização do modelador geométrico. Este fato foi comprovado através dos tempos de resposta (latência) de modelagem. Em muitos casos, os atrasos ultrapassam valores limites, além dos quais os usuários perdem a noção de interatividade. Como forma de contornar este problema, propomos um modelo de interação distribuída baseado nas metáforas 3D de manipulação direta, que reduzem a comunicação na rede, através controle da granularidade da comunicação entre as interfaces gráficas (aplicação cliente) e modelador geométrico.

Quando objetos são manipulados sucessivamente, operações intermediárias são localmente acumuladas e somente o resultado da concatenação das operações executadas numa transação é enviada ao servidor de modelagem. Ao mesmo tempo, as metáforas 3D oferecem realimentação visual da ações dos usuários localmente através

de sua representação pictorial, que é atualizada em tempo real. Para comprovar a eficácia do emprego das metáforas adequamos métricas de avaliação de aplicações interativas mono-usuário para o contexto colaborativo e avaliamos a latência das metáforas (local) e a latência de modelagem (remota). Com isso, mostramos que a as metáforas podem ser atualizadas em tempo real, sem a realização de operações no servidor de modelagem para cada interação. Desta forma, acreditamos que o modelo de interação proposto mostra a viabilidade do emprego de mecanismos de interação individual em um ambiente interativo distribuído.

A renderização independente em cada intância da aplicação interativa provoca uma diminuição da consciência coletiva. Para minimizar este fenômeno, propomos um modelo de consciência coletiva. Este modelo reutiliza as metáforas 3D e adiciona novos componentes de consciência 3D. Particularmente, neste modelo são compartilhadas informações de atividade, proximidade e perspectiva entre os usuários do sistema colaborativo, com o objetivo de suprir perdas de consciência devido a não-presencialidade, pontos de vistas distintos da cena e modos de visualização diferenciados. Uma interface multijanelas foi projetada para acomodar os componentes de consciência 3D, lista de participantes, visão global (*global view*) e a visão de cena (*scene view*) na aplicação interativa. Os usuários podem trabalhar na *visão de cena*, concentrados nos seus objetos de interesse. Especificamente, a *visão global* permite a um usuário observar as atividades dos demais, bem como compartilhar parâmetros de visualização com o intuito de obter o mesmo ponto de vista da cena de um parceiro.

Para avaliar o modelo de consciência coletiva, medimos a latência das tele-metáforas a fim de identificar a viabilidade de empregá-las como mecanismo de aumento da consciência coletiva, bem como elaboramos uma avaliação subjetiva, para medir o grau de usabilidade do sistema. Através dos resultados destas avaliações, mostramos que mecanismos de interação individual são de grande utilidade no apoio aos mecanismos de consciência coletiva, pois através das tele-metáforas e tele-visões os usuários tem consciência da atividade do seu grupo.

A utilização de padrões de projetos nos ajudou substancialmente na concepção e desenvolvimento da arquitetura e dos modelos de interação e consciência coletiva, bem como na implementação de um protótipo de um modelador colaborativo, utilizado para avaliar as soluções propostas ao longo do trabalho. Além de oferecer uma nomenclatura que facilita interrelacionar diferentes áreas, os padrões fornecem soluções adequadas

para problemas específicos dentro de cada contexto.

6.1 Trabalhos futuros

A escolha do serviço de eventos CORBA como base para a implementação do serviço de replicação é muito oportuna, pela escolha do padrão CORBA para a comunicação cliente/servidor. Pois, além da nomenclatura semelhante, o serviço de eventos pode trabalhar em conjunto com o núcleo CORBA e interoperar com outros serviços, como o serviço de nomes empregado nesse projeto. Além disso, o serviço de eventos oferece funcionalidades adequadas para implementação dos padrões de projeto *Observer* e *Mediator*, que são a base do serviço de replicação da arquitetura. No entanto, o serviço de eventos do CORBA/Mico, utilizado no protótipo, é implementado sobre um protocolo orietado a conexão e é não-tipado. Desta forma, um certo *overhead* é gerado na replicação dos objetos gráfico e também exige a serialização e deserialização explicitas de todos os dados transportados.

Acreditamos que um estudo de viabilidade do emprego de novas tecnologias de redes para realizar, de forma mais eficiente, a comunicação distribuída na arquitetura fracamente acoplada seria muito oportuno. Principalmente, protocolos orientados a datagrama, mas com confiabilidade de entrega.

Acreditamos que a utilização de um formato padrão aberto, como X3D, BIFS e COLLADA, possa viabilizar o desenvolvimento de aplicações em modelagem geométrica para Web, bem como a interoperabilidade destas com aplicações já existentes. Visumbramos com isso o desenvolvimento de ferramentas de auxílio a educação presencial e à distância, visualização distribuída, planejamento urbano, entretenimento, dentre outras. No entanto, para tornar possível a prototipação de diferentes aplicações colaborativas, seria necessária a organização de um *framework* a partir das experiências adquiridas com o desenvolvimento de aplicações do ambiente *CoMo*.

Exploramos e avaliamos o protótipo do modelador colaborativo em uma rede local. No entanto, pretendemos ampliar a análise de desempenho através de novos cenários, utilizando diferentes tipos de redes, particularmente redes de longa distância. Neste contexto, planejamos explorar mecanismos para monitoramento das condições da rede e o ajuste automático da granularidade dos dados de comunicação com o servidor de mo-

delagem. Acreditamos, também, que algoritmos de compressão de dados podem melhorar a interatividade e a eficiência do sistema de modelagem colaborativa. Neste mesmo horizonte, continuamos as investigações relativas aos novos artefatos que promovam o aumento da consciência coletiva, incluindo a incorporação de diferentes formas de comunicação visual e interrelação entre os participantes.

Um servidor dedicado diminui consideravelmente a confiabilidade do sistema como um todo. Podemos estender o conceito do servidor de modelagem para multi-servidores, bem como investigar novas formas de distribuição e replicação da topologia e operadores topológicos, com o intuito de aumentar a confiabilidade e o desempenho da arquitetura proposta.

Outro aspecto pouco explorado na tese diz respeito ao tamanho dos modelos, pois utilizamos modelos geométricos relativamente pequenos. Dependendo da aplicação prática há necessidade de se trabalhar com modelos geométricos e gráficos muito maiores. O avanço das tecnologias de comunicação, dispositivos de renderização e processadores, darão suporte para a investigação do impacto de modelos geométricos com grande volume de informações para o *CoMo*, tanto no cenário estudado na tese quanto para os novos cenários.

6.2 Trabalhos publicados

Autores: L.G. Silveira Jr e Shin-Ting Wu
Título:*Towards Consistency in a Heterogeneous Collaborative Geometric Modeling Environment.*
Publicação: Proceedings of SIACG 2002, pp 139-148, ACM-Eurographics
Local: Guimarães, Portugal

Autores: L.G. Silveira Jr e Shin-Ting Wu
Título: *Workspace Awareness in Relaxed WYSIWIS Systems.*
Publicação: Proceedings of SIBGRAPI 2002, pp. 171–178, IEEE Computer Society.
Local: Fortaleza, Ceará.

Autores: L.G. Silveira Jr e Shin-Ting Wu
Título: *An Object-Oriented Groupware Framework for Developing Collaborative 3D-Modelers.*

Publicação: Eurographics Workshop on Parallel Graphics & Visualisation, 2001, pp. 103–114
Local:Girona, Espanha

Autores: L.G. Silveira **Jr** e Shin-Ting Wu
Título: *Towards a Direct 3D Interactive Groupware Framework*
Publicação: Poster SIBGRAPI 1999
Local: Campinas, SP

6.3 Outros trabalhos publicados

A seguir, alguns trabalhos publicados durante o período do doutorado, porém não relacionados diretamente ao tema de tese.

Autor: L.G. Silveira **Jr**
Título: *A Client/Server Architecture for the Visualization of Proprietary 3D-formats on the Web.*
Publicação: Short paper in SRV'2002
Local: Gramado, RS.

Autores: L.G. Silveira **Jr**, A. B. Raposo, A. L. Bicho e A.**J**.A Cruz
Título: *Programação Gráfica 3D com OpenGL, Open Inventor e Java 3D.*
Publicação: REIC (Revista Eletrônica de Iniciação Científica), Vol. II, no. I, 2002
Local: Porto Alegre, RS.

Autores: L.G. Silveira **Jr**, L. Conceição e R.A. Barra
Título: *Presenting STEP on Web.*
Publicação: ISPE CE, 1999.
Local: Oakland, EUA.

Apêndice A

OpenGL

No que se refere à renderização, a escolha natural é a biblioteca OpenGL [64, 46]. OpenGL ("GL" significa *Graphics Library*) é uma API de grande utilização no desenvolvimento de aplicações gráficas 3D. A OpenGL foi desenvolvida pela Silicon Graphics como uma interface de programação independente de hardware e dos sistemas de janelas [41]. Ela consiste de um conjunto de funções para a produção imagens a partir de cenas 3-D, com capacidade de remoção de linhas escondidas, iluminação, tonalização, desenho no *framebuffer*, modelo de câmera virtual, etc. No entanto, por ser independente dos sistemas de janelas e, portanto, não inclui o gerenciamento de janelas, nem a manipulação de eventos de entrada do usuário. Biblioteca gráficas 2-D, como GLUT, GTK (com GTK-GlArea) facilitam a comunicação da OpenGL e o sistema de janelas.

Por ser um padrão destinado somente à renderização [64], a OpenGL pode ser utilizada em qualquer sistema de janelas (por exemplo, X ou Windows), aproveitando-se dos recursos disponibilizados pelos diversos hardwares gráficos existentes, Figura A.

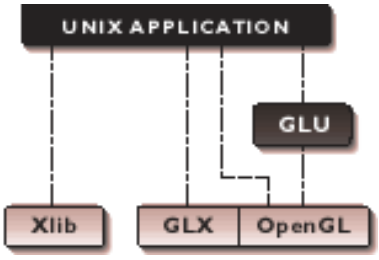


Figura A.1: OpenGL com X-Window

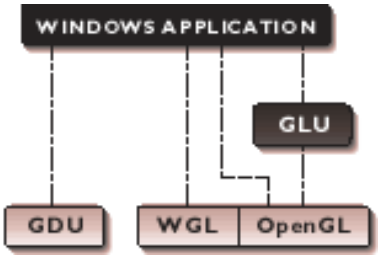


Figura A.2: OpenGL com Windows

Atualmente OpenGL é suportada por praticamente todas as placas gráficas e

está disponível para diversos sistemas computacionais, desde dispositivos móveis, PCs e estações de trabalho com hardware dedicado. Dessa forma, migrar uma aplicação 3D de uma plataforma computacional para outra, gera pouco ônus para o desenvolvedor da aplicação. Tudo fica transparente, dado que a API é única para qualquer tipo de implementação.

OpenGL é uma interface para aplicações gráficas que não possui rotinas de alto nível de abstração. Sendo assim, as primitivas geométricas são construídas a partir de seus vértices. Um vértice pode ser representado em coordenadas homogêneas (x, y, z, w). Se w for diferente de zero, estas coordenadas correspondem a um ponto tridimensional euclidiano (x/w, y/w, z/w). Além disso, a cada vértice pode ser associado um conjunto de atributos, como vetor normal, coordenadas de texturas e cor, que pode ser utilizados no processo de renderização. Objetos mais complexos como segmentos de reta e polígonos são representados por seus pontos extremos e os polígonos são áreas definidas por uma sequência de segmentos. OpenGL suporta algumas primitivas especiais, como triângulos, quadriláteros e variantes destes como sequência de triângulos com arestas compartilhadas. O mesmo conjunto pode ser obtido para quadriláteros. Objetos mais complexos são modelados e processados por conjuntos de pontos, linhas, triângulos, quadriláteros e polígonos.

As primitivas definidas com OpenGL são transportadas da memória do computador para a memória da placa gráfica e processada pelo *pipeline* gráfico. Em caso, de necessidade de melhorar o desempenho pode-se empregar a *display list*, que é uma maneira eficiente de armazenar um grupo de primitivas na memória da placa gráfica, que serão executadas posteriormente pela OpenGL, sem a necessidade de transportar os dados da memória do computador para a memória da placa gráfica a cada atualização. Quando uma *display list* é chamada, as rotinas são executadas na ordem em que elas foram originalmente armazenadas. A maioria das rotinas da OpenGL pode ser armazenada em uma *display list*, exceptuando mudanças de estados, e alguns parâmetros de controle.

Organizar as operações necessárias para converter objetos definidos em um espaço tridimensional para um espaço bidimensional (tela do computador) é uma das principais dificuldades no desenvolvimento de aplicações 3D. Para isso, transformações projetivas, *clipping* e definição das dimensões de *viewport* (porção da janela onde a imagem é desenhada) são providas pela OpenGL. Transformações são funções que mapeiam um ponto

(ou vetor) em um outro ponto (ou vetor). Na OpenGL, elas são implementadas através de matrizes e uma estrutura de pilha, que permitem um ganho considerável de desempenho do sistema gráfico.

O volume de visualização é definido através de seis planos que formam um paralelepípedo, sendo os quatro planos que formam a janela (*left*, *right*, *top* e *bottom*), mais os planos *near* e *far*. Sendo possíveis dois modos de projeção: ortogonal e perspectiva. No modo de projeção ortogonal, os planos são alinhados, de maneira que as dimensões do objeto não é afetado pela sua distância em relação à câmera. Esta projeção é fornecida na OpenGL através da rotina `glOrtho()`. Na projeção perspectiva os planos laterais são dispostos em forma piramidal, promovendo uma deformação nos objetos e obtendo-se uma sensação de profundidade. Assim, as dimensões dos objetos são reduzidas a medida que ele é afastado da câmera. Esta projeção é fornecida na OpenGL através da rotina `glFrustum()`.

OpenGL suporta o modo *true-color* (RGB, RGBA) para especificação de cor de cada vértice ou primitiva [46]. Cada cor é definida pelas componentes vermelho, verde, azul e alfa, respectivamente. Os três primeiros representam as cores primárias e quando normalizadas variam entre 0.0 entre 1.0, sendo muito úteis para renderizar cenas realísticas. A componente alfa é utilizada, por exemplo, em operações de *blending* (mistura) e transparência. Esta componente representa a opacidade da cor, variando de 0.0, onde a cor é totalmente transparente, até 1.0, onde a cor é totalmente opaca.

OpenGL utiliza o modelo de Gouraud para a tonalização, provendo realismo à cena. Uma cena é renderizada levando-se em consideração alguns aspectos como, por exemplo, o tipo de fonte de iluminação que está sendo usada na cena e as propriedades do material para cada superfície. Alguns efeitos complexos como a reflexão da luz e sombra não são diretamente suportados, embora estejam disponíveis códigos-fonte na rede para simular tais efeitos.

Na OpenGL, quando um mapeamento de textura é habilitado, cada pixel da imagem de textura pode ser mapeado no objeto geométrico pela OpenGL e denomina-se *texel*. Assim, o *texel* é um elemento de textura que representa a cor que será aplicada em um determinado fragmento.

Em uma aplicação, a área utilizada para armazenar temporariamente os dados é denominada *buffer*. Um conjunto de *buffers* de uma determinada janela de visualização

ou de um determinado contexto é denominado *framebuffer*. Na OpenGL, há um conjunto de *buffers* que podem ser manipulados conforme a necessidade: de cor, de profundidade, de seleção e de acumulação. Estes *buffers* são utilizados para armazenar a imagem de saída, armazenamento do valor de profundidade de cada *pixel*, separar regiões na cena e acumulação de diversas imagens de uma cena.

A OpenGL é um padrão em constante evolução, para prover uma interface de programação adequada para diferentes *hardwares* gráficas, sem que o programador tenha que conhecer detalhes de cada um. Para acompanhar esta evolução, sugerimos utilizar o site `http://www.opengl.org`.

Apêndice B

CORBA e serviços

CORBA é o acrônimo para *Common Object Request Broker Architecture*. É uma especificação, independente de fabricante de uma arquitetura e infra-estrutura para aplicações baseada em objetos que se comunicam através de uma rede de computadores. Mais especificamente é um *middleware* que habilita a comunicação entre objetos em uma plataforma heterogênea, através da invocação de operações remotas, Figura B.1.

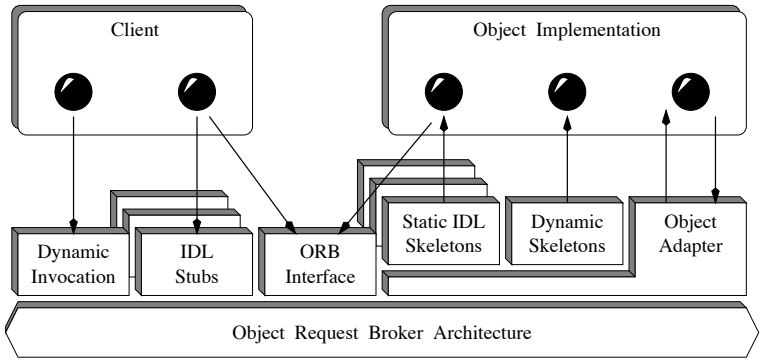


Figura B.1: Arquitetura CORBA

A interoperabilidade é decorrente da especificação de uma linguagem de definição de interface remota denominada OMG/IDL (*Interface Definition Language*) e protocolos padronizados, no caso GIOP e sua versão para Internet, o IIOP. Isto permite que aplicações baseadas em implementações CORBA de diversas procedências, em diferentes plataformas computacionais, sistemas operacionais, linguagens de programação e redes, se interoperem. A especificação CORBA oferece um conjunto de características

importante para facilitar o desenvolvimento de aplicações distribuídas baseada em objetos:

① Orientação a objetos: os objetos são as construções básicas das aplicações CORBA.

② Transparência na distribuição: um programador utiliza os mesmos mecanismos para invocar um objeto se ele estiver no mesmo espaço de endereço, na mesma máquina ou em uma máquina remota

③ Independência de *hardware*, linguagem e sistema operacional: componentes CORBA podem ser implementados usando diferentes linguagens de programação em diferentes arquiteturas de hardware e diferentes sistemas operacionais.

④ Independência de fabricante - implementações do CORBA de diferentes fabricantes, mas em conformidade com a especificação da OMG [51], são interoperáveis. Esta característica nos permite escolher qualquer linguagem de programação para desenvolvimento das aplicações.

CORBA é um padrão aberto na medida que qualquer pessoa pode obter a sua especificação e implementá-lo. Atualmente existem várias implementações do CORBA: Orbix, Visibroker, JavaORB, MICO, OmniORB, ILU e tantos outros. A escolha de qual implementação utilizar é feita de acordo com as necessidades de cada projeto, dado que cada implementação pode dispor de apenas um subconjunto das características especificadas pela OMG. Para o nosso projeto, além do núcleo da plataforma CORBA, o serviço de eventos e o serviço de nomes são pré-requisitos para a escolha de qual implementação do CORBA utilizar. Dentre aquelas que nos atenderam, escolhemos o MICO.

Além do núcleo da plataforma CORBA, a OMG especifica um conjunto de serviços conhecidos como *Common Object Services* (COSS) ou mais comumente chamados de Serviços CORBA. Dentre os mais usados estão os serviços de: Nomes, Eventos, Notificação, Transação, *Trading*, *Stream*, Tempo, Propriedades, dentre outros [51]. Alguns destes serviços oferecem um conjunto de ferramentas de apoio para à programação distribuída, e um grande auxílio na construção da camada de serviços específicos, como por exemplo, replicação do modelo gráfico baseado no serviço de eventos.

O acesso remoto aos objetos distribuídos são especificados através de uma interface linguagem IDL, que possui uma sintaxe muito próxima de C++. Ela suporta

tipos básicos e definidos pelo usuário, os quais são utilizados nos nas assinaturas dos métodos dos objetos remotos, definindo assim a interface distribuída dos serviços da plataforma. Os tipos e as interfaces CORBA/IDL são mapeadas, por um compilador IDL, para tipos e classes em qualquer linguagem de programação orientada a objetos, como C++, Java, dentre outras.

Pode-se ter acesso aos objetos CORBA localmente ou através de clientes remotos via invocação de operações, usando a linguagem IDL. Os clientes precisam conhecer a interface publicada pelo objeto servidor[1] e sua localização. Os clientes utilizam uma referência especial de objetos, denominada IOR, para invocar as operações definidas pelos objetos remotos. Esta referência possibilita a identificação dos objetos entre diferentes espaços de endereçamento, são válidas para o sistema inteiro e podem ser passadas de um nó para o outro. Após compiladas, *stubs* IDL definem como os clientes devem invocar serviços em máquina remota, enquanto do lado servidor, esqueletos IDL fornecem interface entre o CORBA e os objetos (serviços) implementado, através de adaptadores providos pela plataforma CORBA. Além disso, mecanismos de extensibilidade em tempo de execução oferecem a possibilidade troca de componentes sem recompilar o código.

Serviço de Nomes

O serviço de nomes, especificado pela OMG, relaciona objetos servidores (incluindo-se a sua localização) a nomes (*string*). Com isso, um servidor de nomes pode conter referências a diversos objetos servidores em diferentes máquinas. Para ter acesso a qualquer objeto remoto, basta consultar o servidor de nomes através de uma *string* e, caso o objeto esteja registrado, uma referência (IOR) é obtida.

Serviço de Eventos

O serviço de eventos [30], especificados pela OMG, incorpora a noção de produtor/consumidor, cuja comunicação entre eles é feita via um canal de eventos, Figura B.2. Este serviço permite múltiplas entidades produtoras e consumidoras se comunicarem

[1] ou objetos de implementação

de forma assíncrona[2]. Duas configurações são possíveis na conexão entre estas entidades: os produtores tomam a iniciativa na comunicação postando os eventos no canal de eventos para serem coletados pelos os consumidores (modelo *push*), ou os consumidores iniciam a comunicação requisitando os eventos dos produtores (modelo *pull*).

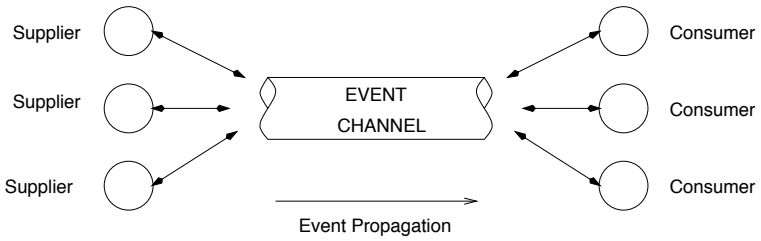


Figura B.2: Comunicação baseada em eventos via canal de eventos.

O canal de eventos é um objeto CORBA, com interface especificada em IDL, que promove comunicação indireta e assíncrona entre produtores e consumidores, dado que os produtores não precisam conhecer os respectivos consumidores e vice-versa, os consumidoras não sabem quais são os seus produtores. Desde que o canal de eventos é um objeto CORBA, M ($M > 0$) produtores e N ($N > 0$) consumidores podem utilizá-lo simultaneamente, através de sua interface distribuída. Ambos os componentes podem ser adicionados ou removidos do canal fácil e dinamicamente, utilizando métodos pré-definidos na interface do canal de eventos.

Os eventos em si não são objetos, porque o modelo do CORBA não suporta passagem de objetos por valor. O serviço de eventos permite o envio de dados do tipo *Any*[3]. Desta forma, para que um objeto seja transferido pela rede, deve ser externalizado no produtor e internalizado no consumidor, no caso uma variável do tipo *Any* é a portadora da informação a ser transferida.

[2]existe a possibilidade de comunicação direta entre produtores e consumidores, mas é uma configuração raramente utilizada

[3]O tipo *Any* do CORBA pode conter um valor de qualquer tipo

Bibliografia

[1] A. Agrawal and Aristides A. G. Requicha. A paradigm for the robust design of algorithms for geometric modeling. *Computer Graphics Forum*, 13(3):33–44, 1994.

[2] R. M. Baecker. *Readings in Groupware and Computer-Supported Cooperative Work*. Morgan Kaufmann Inc., San Francisco, Califórnia, 1993.

[3] C. Bajaj and F. Bernardini. Distributed and collaborative synthetic environments. Technical report, Department of Computer Science - Purdue University, [on-line] http://www.cs.purdue.

[4] Rafael Bidarra, Eelco van den Berg, and Willem Bronsvoort. Collaborative modeling with features. In *Proceedings of the 2001 ASME Computers and Information in Engineering Conference*, Pittsburgh, Pennsylvania, Sept 2001. ASME.

[5] L. S. Blackford, A. Cleary, A. Petitet, R. C. Whaley, J. Demmel, I. Dhillon, H. Ren, K. Stanley, J. Dongarra, and S. Hammarling. Practical experience in the numerical dangers of heterogeneous computing. *ACM Transactions on Mathematical Software (TOMS)*, 23(2):133–147, 1997.

[6] B. Bruderlin. Detecting ambiguities: An optimistic approach to robustness problems in computation geometry. Technical Report UUCS-90-003, Computer Science Department, University of Utah, Salt Lake City, Apr. 1990.

[7] B. Bruderlin. Robust regularized set operations on polyhedra. Technical Report UUCS-90-004, Computer Science Department, University of Utah, Salt Lake City, Apr. 1990.

[8] B. Castier, L. F. Martha, and M. Gattass. A taxonomy for interactive manipulation and visualization of 3d objects. In *Sibgrapi94*, pages 149–156, 1994. In Portuguese.

[9] "K. Chase and A. Parkinson. A survey of research in the application of tolerance analysis to the design of mechanical assemblie. *Research in Engineering Design*, 3:23–37, 1991.

[10] G. Chung, K. Jeffay, and H. Abdel-Wahab. Dynamic participation in computer-based conferencing system. *Journal of Computer Communications*, 17(1):7–16, Jan. 1994.

[11] D. Brookshire Conner, Scott S. Snibbe, Kenneth P. Herndon, Daniel C. Robbins, Robert C. Zeleznik, and Andries van Dam. Three-dimensional widgets. In *ACM Symposium on Interactive 3D Graphics, Special Issue of Computer Graphics*, volume 26, pages 183–188, 1992.

[12] G. Coulouris, J. Dollimore, and T. Kindberg. *Distributed System: Concepts and Design*. Addison Weley, 2001.

[13] L. G. da Silveira Júnior. Operadores booleanos para objetos modelados por complexos celulares. Tese de Mestrado, Universidade Estadual de Campinas, Dep. de Eng. da Comput. e Aut. Industrial - Fac. de Eng. Elétrica e de Computação, ago. 1996.

[14] Luiz H. de Figueiredo and Jorge Stolfi. *Self-Validated Numerical Methods and Applications*. Brazilian Mathematics Colloquium Monograph, IMPA, July 1997.

[15] P. Dourish and V. Bellotti. Awareness and coordination in shared workspaces. In *Conference on Computer Supported Cooperative Work (CSCW'92)*, pages 107–114, Toronto, Ontario, 1992. ACM Press.

[16] Jeff Dyck and Carl Gutwin. Awareness in 3d collaborative workspaces. In *ACM Graphics Interface (GI'2002)*, `http://hci.usask.ca/publications/2002/groupspace-gi/index.xml`, 2002. to appear.

[17] R. Earnshaw. 3d and multimedia on the information superhighway. *IEEECGA*, pages 30–31, 1997.

[18] Herbert Edelsbrunner and Ernst P. Mucke. Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithms. *ACM Transaction on Graphics*, 9(1):66–104, 1990.

[19] C. Ellis, S. Gibbs, and G. Rein. Groupware: Some issues and experiences. *Communications of the ACM*, 34(1):38–58, 91.

[20] C. Ellis and J. Wainer. A conceptual model of groupware. In ACM Press, editor, *CSCW'94*, pages 79–88, 1994.

[21] S. Fang, B. Bruderlin, and X. Zhu. Robustness in solid modeling: a tolerance-based intuitionistic approach. *Computer-Aided Design*, 25(9):567–576, 1993.

[22] Aurélio B. Holanda Ferreira. *Novo Aurelio - Dicionário da Língua Portuguesa*. Nova Fronteira, 1999.

[23] Steven Fortune. Robustness issues in geometric algorithms. In *WACG: 1st Workshop on Applied Computational Geometry: Towards Geometric Engineering*, *WACG*. LNCS, 1996.

[24] Greg Gagne, Abraham Silberschatz, and Peter Baer Galvin. *Sistemas Operacionais: Conceitos e Aplicações*. Campus, 2001.

[25] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Desin Patterns – Elements of Reusable Object-Oriented Software*. Addisson Wesley, 1995.

[26] D. Garfinkel, B.C. Wleti, and T.W. Yip. Hp sharedx: A toll for real-time collaboration. *HP Journal*, April 1994.

[27] S. Greenberg, S. Hayne, and R. Rada. *Designing Groupware for Real-Time Drawing*. McGraw Hill, 1995.

[28] Saul Greenberg and David Marwood. Real time groupware as a distributed system: Concurrency control and its effect on the interface. In ACM, editor, *CSCW'94*, pages 207–217, Chapel Hill,NC, 1994.

[29] Saul Greenberg and Mark Roseman. *Computer-Supported Cooperative Work (Trends in Software 7)*, chapter Groupware Toolkits for Synchronous Work, pages 135–168. M. Beaudouin-Lafon (editor). John Wiley & Sons Ltd, 1998.

[30] OMG Object Management Group. Corba event service specification - v1.0. `http://www.omg.org`, 1993.

[31] GTK+. Gtk+ - the gimp toolkit. `http://www.gtk.org/`.

[32] Carl Gutwin and Saul Greenberg. Support for group awareness in real-time desktop conferences. In *Proceedings of The Second New Zealand Computer Science Research Students Conference*, University of Waikato, Hamilton, New Zealand, April 1995.

[33] Carl Gutwin and Saul Greenberg. The effects of workspace awareness support on the usability of real-time distributed groupware. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 6(3):243–281, Sept. 1999.

[34] Carl Gutwin, Mark Roseman, and Saul Greenberg. A usability study of awareness widgets in a shared workspace groupware system. In *Computer Supported Cooperative Work*, pages 258–267, 1996.

[35] C.M. Hoffmann. The problems of accuracy and robustness in geometric computation. *IEEE Computer*, pages 31–41, Mar. 1989.

[36] C. Hoffmann; J. Hopcroft. Geometric ambiguities in boundary representations. Technical Report TR 86-725, Dep. of Computer Science - Cornell University, Ithaca, New York, Jan. 1986.

[37] IEEE. Ieee standard for binary floating point arithmetic: Standard 754. In *IEEE Press*. IEEE, Piscataway,N.J., standard 754-1987 edition, 1985.

[38] IEEE. Ieee standard for radix-independent floating-point arithmetic: Standard 854. In *IEEE Press*. IEEE, Piscataway,N.J., standard 854-1987 edition, 1987.

[39] IEEE. Ieee standard for shared-data formats optimized for scalable coherent interface (sci) processors. In *IEEE Press*. IEEE, Piscataway,N.J., standard 1596.5-1993 edition, 1994.

[40] Lutz Kettner. Theoretical foundations of 3d-metaphors. In *Workshop on the Challenges of 3D Interaction at the CHI'94*, february 1994.

[41] M. J. Kilgard. *Programming OpenGL for the X Window System*. Addison-Wesley, 1996.

[42] E. Koerner. Group management for a multimedia collaboration service. In *EUNICE '96 - Summer School on Telecommunications Services*, Lausanne, 23–27 Sept. 1996.

[43] M. de G. Malheiros, F. N. Fernandes, and S. T. Wu. Mtk: A direct 3d manipulation toolkit. In *SCCG'98 Proceedings*, pages 81–88, Brastilava, april 1998.

[44] D. Michelucci. Arithmetic issues in geometric computations. In *Second Real Numbers and Computer Conference*, 1996.

[45] D. Michelucci. An introduction to the robustness issue. In *Swiss Conference of CAD/CAM*, pages 214–221, Neuchâtel, Switzerland, Feb. 1999.

[46] J. Neider, T. Davis, and M. Woo. *OpenGL - Programming Guide - Release 1*. Addison Wesley Co., 1993.

[47] Jackie Neider, Tom Davis, and Mason Woo. *OpenGL Programming Guide: The Official Guide to Learning OpenGL, release 1*. Addison-Wesely, 1993. ISBN 0-201-63274-8.

[48] Jakob Nielsen. *Usability Engineering*. Morgan Kaufmann Publishers, 1994.

[49] G. M. Nielson and D.R. Olsen Jr. Direct manipulation techniques of 3D objects using 2D locator devices. In *Proceedings 1986 Workshop on Interactive 3D Graphics*, pages 175–182, Chapel Hill, North Carolina, New York, october 1986.

[50] Object Management Group (OMG). The common object request broker: Architecture and specification, Dez 2001. Version 2.6.

[51] Object management group. `http://www.omg.org`.

[52] N. Guimarães P. Antunes. Multiuser interface design in cscw systems. Technical University of Lisboa - INESC, Oct. 1994.

[53] John F. Patterson. A Taxonomy of Architetures for Synchronous Groupware Applications. In *ACM*, volume 15, pages 27–29. SIGOIS Bulletin, April 1995.

[54] Jenny Preece, Yvonne Rogers, Helen Sharp, and David Benyon. *Human-Computer Interaction*. Addison-Wesley Pub Co, 1994.

[55] Arno Puder and Kay Roemer. *MICO: An Open Source CORBA Implementation*. Morgan Kaufmann Publishers, Mar 2000.

[56] A.A.G. Requicha. Toward a theory of geometric tolerancing. *International Journal of Robotics Research*, 2(4), 1983.

[57] M. Ressel, D. Nitsche-Ruhland, and R. Gunzenhäuser. An integrating, transformation-oriented approach to concurrency control and undo in group editors. In *CSCW'96*, pages 288–296, Cambridge MA, USA, 1996. ACM.

[58] Tom Rodden. A survey of cscw systems. Technical report, Department of Computer Science - Lancaster University, 1992.

[59] M. Roseman and S. Greenberg. Building Real Time Groupware with GroupKit - A Groupware Toolkit. *ACM Trans. Computer-Human Interaction*, 3(1):66–106, Mar. 1996.

[60] Jörg Roth and Claus Unger. An extensible classification model for distribution architectures of synchronous groupware. In *4th. Int. Conference on the Design of Cooperative Systems (COOP 2000)*, pages 113–127, Sophia Antipolis, May 2000.

[61] Dean Rubine. Combining gestures and direct manipulation. In *Proceedings of ACM CHI'92 Conference on Human Factors in Computing Systems*, pages 659–660, Monterrey - California, may 1992. Addison Wesley.

[62] Robert W. Scheifler and Jim Gettys. The x-window system. *ACM Transactions on Graphics*, 5(2):79–109, april 1986.

[63] M. Segal. Using tolerances to guarantee valid polyhedral modeling results. In *Computer Graphics (Proc. Siggraph)*, volume 24. ACM Computer Graphics, Aug. 1990.

[64] M. Segal and K. Akeley. The opengl graphics interface. Technical report, Silicon Graphics Computer Systems, [on-line] http://www.sgi.com.

[65] Sandeep Singhal and Michael Zyda. *Networked Virtual Environments: Design and Implementation*. Addison-Wesley, 1999.

[66] GPHIGS (Enhanced PHIGS+ Solution). TGS - Template Graphics Software, Inc, nov. 1999.

[67] H. Sowizral and K. Rushfor. *The Java 3D API Specification*. Addison Wesley, Reading, MA, 1998.

[68] M. Stefik, G. Foster, D. Bobrow, K. Kahn, S. Lanning, and L. Suchman. Beyond the chalkboard: Computer support for collaboration and problem solving in meetings. *CACM*, 30(1), 1987.

[69] R. Steinmetz and K. Nahrstedt. *Multimedia: Computing, Communications & Applications*. Prentice Hall, 1995.

[70] A.J.K. Stewart. *The Theory and Practice of Robust Geometric Computation, or, How to Build Robust Solid Modelers*. PhD thesis, Cornell Universtiy, Ithaca, New York, Aug. 1991.

[71] R. Strom, G. Banavar, K. Miller, A. Prakash, and M. Ward. Concurrency control and view notification algorithms for collaborative replicated objects. *IEEE Transactions on Computers*, 47(4):458–471, April 1998.

[72] G. Blair T. Rodden. Cscw and distributed systems: the problem of control. In *Second European Conference on Computer Support Cooperative Work - ECSCW'91*, Amsterdam, 1991.

[73] J.C. Tang. Findings from observational studies of collaborative work. *International Journal of Man-Machine Studies*, 34(2):143–160, 1991.

[74] Eelco van den Berg. Collaborative modelling systems. Technical report, Faculty of Information Technology and Systems, Delft University of Technology, Oct. 1999.

[75] Steve Vinoski. CORBA: integrating diverse applications within distributed heterogeneous environments. *IEEE Communications Magazine*, 14(2), 1997.

[76] U. von Lukas and A. Stork. Introducing tools for collaborative modelling. In *ISATA '98*, 1998.

[77] Josie Wernecke. *The Inventor Mentor: Programming Object-Oriented 3D Graphics with Open Inventor, release 2*. Addison-Wesley Publishing Company, 1994. ISBN 0-201-62495-8.

[78] S.T. Wu. Considerations about minimal set of non-manifold operators. In *Workshop on Geometric Modeling*, pages 17–21, Rensselaerville, New York, June 1990.

[79] S.T. Wu, M. de G. Malheiros, and F. N. Fernandes. Widgets for constructing 3d graphic interfaces. Technical Report RT-DCA 001/98, DCA - FEEC - Unicamp, Campinas, january 1998. In Portuguese.