FUNDAÇÃO DE ENSINO “EURÍPIDES SOARES DA ROCHA”
CENTRO UNIVERSITÁRIO “EURÍPIDES DE MARÍLIA” - UNIVEM
PROGRAMA DE MESTRADO EM CIÊNCIA DA COMPUTAÇÃO

LEILA HAGA

IMPLEMENTAÇÃO DE UM SUPORTE AO GERENCIAMENTO DE GRUPOS EM AMBIENTES VIRTUAIS COMPARTILHADOS

MARÍLIA - SP
2005

Livros Grátis

http://www.livrosgratis.com.br

Milhares de livros grátis para download.

LEILA HAGA

IMPLEMENTAÇÃO DE UM SUPORTE AO GERENCIAMENTO DE GRUPOS EM AMBIENTES VIRTUAIS COMPARTILHADOS

Dissertação apresentada ao Programa de Mestrado do Centro Universitário Eurípides de Marília, mantido pela Fundação de Ensino Eurípides Soares da Rocha, para obtenção do Título de Mestre em Ciência da Computação. (Área de Concentração: Realidade Virtual).

Orientador*:*
Prof. Dr. Ildeberto Aparecido Rodello

MARÍLIA
2005

AGRADECIMENTOS

A Deus, que nos protegeu em nossas longas viagens de Dourados a Marília, e nos permitiu alcançar mais este objetivo.

Aos meus pais, dos quais o apoio foi incondicional e irrestrito.

Ao Prof. Dr. Ildeberto Aparecido Rodello, pela prestativa orientação.

Aos colegas e aos professores do mestrado, que tornaram nosso curso uma grande alegria e contribuíram para que esta jornada fosse muito mais agradável.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

3D – Tridimensional

API – *Aplication Program Interface*

AVC – Ambiente Virtual Colaborativo

AVD – Ambiente Virtual Distribuído

CPK – *Corey Pauling Kultin*

CSCW – *Computer Supported Cooperative Work*

DIVE – *Distributed Interactive Virtual Environment*

DMSO – *Defense and Modeling Simulation Office*

FOM – *Federation Object Model*

HLA – *High Level Architecture*

HMD – *Head Mounted Display*

Hz - *Hertz*

IEEE – *Institute of Electrical and Eletronics Engineers*

IMK - *Institut fùr Medienkommunikation*

JVRMol – Ambiente para Visualização e Análise de Moléculas de Proteínas

OMT – *Object Model Template*

PDB – *Protein Data Bank*

RMI – *Remote Method Invocation*

RPC – *Remote Procedure Call*

RTI – *Runtime Intrastructure*

RV – Realidade Virtual

SGI - *Silicon Graphics*

SICS – *Swedish Institute of Computer Science*

SOM – *Simulation Object Model*

UDP – *User Datagram Protocol*

HAGA, Leila. Implementação de um Suporte ao Gerenciamento de Grupos em Ambientes Virtuais Compartilhados. 2005. 92 f. Dissertação (Mestrado em Ciência da Computação) – Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares da Rocha, Marília, 2005.

RESUMO

O foco deste trabalho é pesquisar técnicas necessárias para dar suporte ao trabalho colaborativo em ambientes virtuais compartilhados e implementá-las no JVRMol, um ambiente virtual para visualização de moléculas de proteínas. Essas técnicas incluem o fornecimento de mecanismos de gerenciamento de grupos, além de mecanismos de comunicação e armazenamento dos dados. O trabalho colaborativo pode produzir melhores resultados do que se os membros do grupo atuassem individualmente. Em um grupo pode ocorrer à complementação de capacidades, de conhecimentos e de esforços individuais, e a interação entre pessoas com entendimentos, pontos de vista e habilidades complementares. Dessa forma, a implementação do suporte a grupos permitirá o desenvolvimento de projetos em que é possível a união dos conhecimentos dos usuários, o que contribui para a co-realização de projetos. Neste sentido, os requisitos de *groupware* objetivam contribuir para o aumento da produtividade de equipes de trabalho ou pesquisa por meio do acesso à informação compartilhada. Para tanto, são necessários os estudos dos requisitos dos ambientes virtuais distribuídos e colaborativos de *groupware* e da metáfora social na interação entre os usuários, para fornecer um ambiente propício a eles trocarem e a compartilharem informações. Assim, foram acrescentados métodos para a criação e gerenciamento de grupos no JVRMol. Isto permite que pesquisadores dispersos geograficamente troquem informações com um ou mais grupos e, portanto, participem da análise de várias estruturas de moléculas de proteínas simultaneamente, possibilitando chegar a resultados com maior eficiência e em menos tempo. Adicionalmente, este trabalho, apresenta-se como uma proposta de biblioteca para criação e gerenciamento de grupos.

Palavras-chave: *Groupware*. Ambientes Virtuais Distribuídos. Ambientes Virtuais Compartilhados.

HAGA, Leila. Implementação de um Suporte ao Gerenciamento de Grupos em Ambientes Virtuais Compartilhados. 2005. 92 f. Dissertação (Mestrado em Ciência da Computação) – Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares da Rocha, Marília, 2005.

ABSTRACT

The focus of this work is to research necessary techniques to give support to the collaborative work in networked virtual environments and put them into the JVRMol, a virtual environment that visualize molecules of proteins. These techniques include the catering of mechanism of group management, apart from mechanism of communication and data stockpile. The collaborative work can produce better results than if the members of the group work individually. In a group there is the complementation of skills, of knowledgeand, of personal efforts, and the interaction between person with understandings, points of view and complementary skills. By this way, the implementation of a support to a group will allow the development of projects in which is possible the fusion of knowledge of the users, which contributes to the co-realization of projects. In this meaning, the requirements of *groupware* want to contribute to increase the productivity in work staff or research by accessing the shared information. So, the studies of the requirements of the collaborative and shared virtual environments of *groupware* and of the social metaphor in the interaction between the users are needed to give them an appropriate environment to change and share information. Thus, methods of creation and management of groups were added to the JVRMol. It allows geographically scattered researchers change informations with one or more groups and, by this way join in the analysis of several structures of molecules of proteins simultaneously, reaching to results with more efficiency in a shorter time. In addition to this, this work show itself as a proposal of a library for creation and management of groups.

Keywords: *Groupware*. *Distributed Virtual Environment. Networked Virtual Environment*

HAGA, Leila. Implementação de um Suporte ao Gerenciamento de Grupos em Ambientes Virtuais Compartilhados. 2005. 92 f. Dissertação (Mestrado em Ciência da Computação) – Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares da Rocha, Marília, 2005.

RESUMEN

El objetivo de este trabajo es pesquisar técnicas necesarias para dar suporte al trabajo colaborativo en ambientes virtuales compartillados y ponelas en el JVRMol, un ambiente virtual para la visualización de moléculas de proteínas. Esas tecnicas incluen el fornecimiento de mecanismos de gerenciamiento de grupos, y también mecanismos de commucación y armazenamiento de informaciones. El trabajo colaborativo puede producir mejores resultados que cuando los miembros del grupo atuan solos. En un grupo puede ocurrir la complementación de capacidades, de conocimientos y de esforzos indibiduales, y la interación entre personas con entendimientos, puntos de vista y habilidades complementares. De esta manera, la inplementación del suporte a grupos les permitirá el desarrollo de projectos en que es posible la unión de conocimientos de los usuários, lo que contribuye para la co-realización de projectos. En este sentido los requisitos del *groupware* objetivan ayudar en el aumento de productividad de equipos de trabajo o pesquisa accesando la información compartillada. Para que eso ocurra son necesarios los estudios de los requisitos de los ambientes virtuales distribuydos y colaborativos de *groupware* y de la metaforia social en la interación entre los usuários, para darles un ambiente apropriado para que ellos puedan cambiar y compartillar informaciones. Así, métodos para la creación y gerenciamiento de grupos en el JVRMol fueron aumentados. Eso permite que pesquisadores que esteén lejos geograficamente cambien informaciones con un o mas grupos y, así participen de las analisis de varias estruturas de moléculas de proteinas al mismo tiempo. Además, este trabajo apresentase como uma propuesta de biblioteca para la creación y gerenciamiento de grupos.

Palabras-Clave: *Groupware*. Ambientes Virtuales Distribuydos. Ambientes Virtuales Compartillados.

INTRODUÇÃO

O aumento da complexidade e o tamanho das tarefas, aliados à necessidade de aumento da produtividade e da capacidade de processamento das máquinas exigem maior interação entre as pessoas que possuem conhecimento em diversas áreas. Além disso, muitas vezes as organizações se encontram geograficamente distribuídas, de forma que seus profissionais precisam trabalhar com colegas distantes e necessitam de resultados rápidos sem, no entanto, onerar os custos com viagens. Nota-se, dessa maneira, a necessidade de pesquisas sobre várias tecnologias que permitam o desenvolvimento de suportes eficazes para o trabalho em grupo, assim como a análise de aspectos cognitivos e sociais de processos colaborativos (MOECKEL, 2003).

Esta foi uma das razões para o surgimento do Trabalho Cooperativo Suportado por Computador, ou *Computer Supported Cooperative Work* – CSCW (GRUDIN, 1994). O CSCW é uma área da computação que descreve como desenvolver aplicações colaborativas que proporcionam maior eficiência nos requisitos: comunicação, coordenação e cooperação entre usuários geograficamente dispersos (GRUDIN, 1994).

Quanto aos Ambientes Virtuais Distribuídos (AVDs), estes permitem que, por meio de um suporte de redes de computadores, usuários dispersos geograficamente possam interagir compartilhando recursos computacionais e, também, visões do mundo virtual em tempo real (ZYDA; SINGHAL, 1999, BENFORT et al., 2001).

Com relação ao Ambiente Virtual Colaborativo (AVC em inglês *Collaborative Virtual Environment* – CVE), verifica-se que este se trata de um sistema específico de Realidade Virtual (RV) em que a ênfase é dada na colaboração entre os usuários e não na simulação (OLIVEIRA, 2000; BENFORD et al., 1994b), uma vez que procura aliar as características de representação tridimensional da RV às facilidades de compartilhamento e de troca de informações oferecidas por sistemas em rede.

Nesse sentido, a contribuição de pessoas com diferentes níveis de entendimentos, pontos de vista alternativos e habilidades complementares, geram resultados que dificilmente seriam encontrados no trabalho individual. Assim, resolver uma tarefa em um grupo de trabalho colaborativo é potencialmente mais eficaz do que resolvê-la individualmente, pois os membros do grupo podem auxiliar o pensamento individual, prover *feedback* e, ainda, buscar um conjunto de idéias, de informações e de referências para auxiliar na resolução de

problemas. Percebe-se que desse modo, na maioria das vezes, o grupo tem mais capacidade de gerar alternativas criativas, de levantar vantagens e desvantagens de cada uma, de selecionar as vantagens viáveis e de tomar decisões do que os indivíduos separadamente (GEROSA et al., 2002).

Portanto, o estudo de *groupware* objetiva compreender a formação de grupos e como as pessoas se comportam nesses grupos. Visa também ter uma compreensão da tecnologia de redes de computadores e como os aspectos dessa tecnologia (por exemplo, atraso na sincronização das visões) afetam a experiência de um usuário (TANENBAUM, 1995).

O JVRMol é um AVD para visualização e análise de moléculas de proteínas, implementado com a linguagem Java, incluindo APIs Java 3D e Java RMI. Possui uma interface gráfica implementada com Java 3D, e um conjunto de métodos para troca de mensagens baseado no modelo de comunicação cliente/servidor, desenvolvidas com Java RMI (RODELLO, 2003).

Com os avanços da tecnologia, o trabalho colaborativo vem se expandindo e sendo introduzido em aplicações de diversas áreas do conhecimento. Neste trabalho, especificamente, o interesse é a criação e gerenciamento de grupos tendo como ambiente o JVRMol, habilitando o trabalho compartilhado para que grupos de pesquisadores possam interagir e chegar a conclusões mais rapidamente utilizando o compartilhamento dos conhecimentos.

Para isso, foram implementados mecanismos para criar grupos, gerenciar a entrada e saída de participantes nos grupos, bem como a troca de informações entre os integrantes de cada grupo.

Com isso, cria-se a possibilidade dos participantes integrarem grupos diferentes simultaneamente, e que permite seu acesso a equipes com conhecimentos e visões distintas sobre um determinado assunto, e então acelerar a troca de informações, que pode ser de grande auxílio no alcance de resultados mais rápidos e eficientes.

Objetivos

Nesse contexto foram desenvolvidos mecanismos para o gerenciamento de grupos no ambiente do JVRMol, de forma a permitir que grupos de pesquisadores, dispersos

geograficamente, compartilhassem a mesma visão de uma estrutura tridimensional de uma proteína, com a possibilidade de participar de vários grupos simultaneamente.

Esses grupos podem ser dinâmicos e novos grupos podem ser criados ou destruídos. Um participante, por exemplo, pode se juntar a um grupo ou deixá-lo, e um outro pode pertencer a vários grupos ao mesmo tempo. Dessa maneira, são necessários mecanismos para gerenciar os grupos e seus membros. Para isso, se fez necessário um estudo para compreender a formação de grupos e como as pessoas se comportam em grupos. Outro aspecto relevante é a compreensão da tecnologia de redes de computadores e como os aspectos dessa tecnologia (por exemplo, atraso na sincronização das visões) afetam a experiência de um usuário.

Organização do Trabalho

Este trabalho está dividido em cinco capítulos: o primeiro discute as características dos AVDs e AVCs, os requisitos para estes ambientes e, também, apresenta um estudo dos ambientes virtuais compartilhados, em que as aplicações fornecem mecanismos de interações modeladas a partir das interações sociais do mundo real.

A seguir, o segundo capítulo apresenta a definição de *groupware*, que se trata de uma tecnologia desenvolvida para facilitar o trabalho em grupo, fornecendo suporte para que um grupo de usuários trabalhe de maneira colaborativa sobre a informação compartilhada.

No terceiro capítulo são visualizados três ambientes relacionados à AVCs: DIVE, HLA e Avocado.

O quarto capítulo aponta a descrição das bibliotecas de comunicação do JVRMol com tabelas que identificam as primitivas do projeto original e, posteriormente, as primitivas adicionadas durante este trabalho.

Finalmente, o quinto capítulo apresenta um estudo de caso sobre o gerenciamento de grupos no JVRMol.

Logo após, apresenta-se às considerações finais, em que são apontadas algumas sugestões para trabalhos futuros.

CAPÍTULO 1

AMBIENTES VIRTUAIS DISTRIBUÍDOS E COLABORATIVOS

Este capítulo aborda o conceito e as características de AVDs e AVCs. Nos AVDs, também chamados mundos virtuais geograficamente dispersos, os integrantes devem permitir interagir uns com os outros e com o ambiente como no mundo real. Além da manutenção da consistência e do controle de concorrência para objetos do mundo virtual, AVDs devem permitir algum tipo de comunicação entre participantes por múltiplos meios, como, áudio e vídeo. Os meios de comunicação aumentam o senso de realismo no ambiente virtual (TEIXEIRA; DUARTE, 2004).

Já o estudo de AVCs é importante no sentido de prover embasamento a este trabalho no suporte ao compartilhamento.

O reconhecimento e a percepção das atividades dos outros usuários é uma questão muito importante para a interação entre os usuários.

1.1 Definição de AVDs

O desenvolvimento do poder computacional aliado ao aumento da comunicação (aumento da capacidade de transmissão das redes), tornou possível a concepção e a expansão de AVDs.

O barateamento dos componentes de hardware, tais como placas de aceleração gráfica, permite que qualquer usuário comum tenha acesso a computadores capazes de executar aplicações gráficas. A disseminação da Internet possibilitou a conectividade de pessoas em escala global viabilizando a criação de AVDs, por meio do qual, vários sistemas computacionais ligados em rede podem compartilhar o mesmo ambiente virtual (BATISTA, 2000).

Os AVDs permitem que por meio de um suporte de redes de computadores, usuários dispersos geograficamente possam interagir compartilhando recursos computacionais e visões do mundo virtual em tempo real (ZYDA; SINGHAL, 1999, BENFORT et al., 2001).

A habilidade de ter múltiplos usuários que compartilham informações manipulam objetos e interagem socialmente no ambiente fazem dos sistemas de RV uma poderosa ferramenta. Porém, todas essas características tornam os AVDs complexos e os processos para suportá-los difíceis de implementar. Verifica-se que os AVDs necessitam também resolver problemas de consistência de dados, de limitação da largura de banda da rede e da demora nas transmissões das informações. (TEIXEIRA; DUARTE, 2004).

1.2 Definição de AVCs

Na década de 80, Paul Cashman e Irene Grief organizaram um *workshop* com pessoas de diversas áreas que compartilhavam o interesse em como pessoas trabalhavam em grupo e como a tecnologia poderia auxiliá-las. Criaram, então, o termo "*Computer Supported Cooperative Work*" (CSCW), que seria empregado em uma área da computação engajada no desenvolvimento de aplicações que permitiriam grupos de interesses comuns trabalharem conjuntamente em ambientes colaborativos (GRUDIN, 1994).

Em um sentido mais amplo, CSCW é a aplicação de tecnologias da computação e comunicação para facilitar cooperação e colaboração entre pessoas (CHANG et al., 2001). Mais especificamente, Zotto (1997) define CSCW como uma área da computação que descreve como desenvolver aplicações colaborativas que proporcionam maior eficiência nos requisitos comunicação, coordenação e cooperação de atividades.

O AVC representa uma importante categoria de sistemas CSCW, que explora as características de modelagem e interatividade da RV e, geralmente, faz uso do espaço tridimensional (3D) compartilhado para fornecer facilidades de colaboração por meio da

implementação de um espaço virtual distribuído.

Nestes ambientes, os usuários devem se mover livremente dentro do espaço compartilhado, encontrando outros usuários e também objetos e informações de interesse comum (ASSIS, 2003).

Benford et al. (1994b) definem AVC como uma tecnologia que visa transformar as redes de computadores atuais em um espaço 3D povoado e navegável, que suporte o trabalho colaborativo e a interação social. Os participantes compartilham mundos virtuais por meio de uma rede de computadores e podem ser expressos, dentro deste mundo, por representações gráficas chamadas avatares, que transportam suas identidades, presenças, localizações e atividades para os outros participantes. Esses avatares permitem aos usuários interagirem com o conteúdo do mundo e a se comunicarem com os outros usuários usando diferentes meios incluindo áudio, vídeo, ações gráficas e texto, entre outros.

O grande desafio no gerenciamento e na coordenação para controlar trabalho colaborativo consiste em se adequar ao dinamismo da interação entre os participantes e as atividades interdependentes realizadas no AVC, que por ser um ambiente dinâmico requer renegociação contínua durante a colaboração. Isto significa que a política de coordenação varia entre as colaborações e pode variar até mesmo durante a evolução de uma mesma colaboração. Nesse sentido, é essencial que os sistemas de suporte ao trabalho colaborativo sejam suficientemente flexíveis para suportar estas variações (FUKS et al., 2002).

A realização de tarefas colaborativas que exigem algum grau de planejamento e coordenação ainda é bastante difícil, devido ao fato de exigir um grande esforço de programação por parte do projetista do ambiente (FUKS et al., 2002).

Assim sendo, a coordenação de atividades interdependentes em ambientes colaborativos é um problema que deve ser tratado para garantir a eficiência da colaboração. A separação entre atividades e dependências, e a utilização de mecanismos de coordenação é uma maneira de lidar com este problema, que traz a vantagem da reutilização dos componentes em outras situações de colaboração.

Em resumo, os AVCs devem proporcionar: sensação de espaço compartilhado; sensação de presença; negociação e comunicação; flexibilidade e múltiplos pontos de vista (SNOWDON et al., 2001; ZYDA, 1999).

Isto significa que um grupo de usuários separados geograficamente pode interagir em tempo real em um ambiente tridimensional de forma que entre e saía do ambiente, mova-se, e mude os estados dos objetos. Assim, seus movimentos no ambiente transformam a

perspectiva visual e auditiva do usuário. Estes são representados como objetos do ambiente, e além de interagir entre si, podem interagir com simulações computacionais. AVCs, portanto, podem ir além da imitação da realidade, permitindo situações que não existem no mundo real. O grande potencial deste tipo de aplicação tem motivado o crescimento de sua utilização. O objetivo de muitas pesquisas é como suportar centenas ou milhares de participantes interagindo em tempo real, o que possibilita a manipulação dos objetos no ambiente e a comunicação com os outros, por meio de gráficos, áudios, vídeos ou textos. Nota-se que os AVCs podem ser aplicados a diversas áreas de conhecimento tais como: jogos, treinamento militar e industrial, simulações, educação e entretenimento (BENFORD et al., 1994b; ASSIS, 2003; VALIN et al., 2003).

1.3 Dimensões da Complexidade

Segundo Greenhalgh (1996), antes que tais ambientes se tornem realidade, um número de problemas devem ser superados. A relação para o usuário deve ser aceitável, nos termos do custo, do conforto, da facilidade de utilização e da potencialidade. As pessoas, futuramente, poderão utilizar os mundos virtuais rotineiramente, e para isso a infra-estrutura de comunicações deve ser adequada de forma a permitir o acesso de usuários comuns a um mundo virtual. A tecnologia para suportar a criação e a animação de mundos virtuais deverá lidar suficientemente com a complexidade que segundo Greenhalg (1996) incluem:

1.3.1 Número de Usuários

A quantidade de usuários que participam de um único mundo virtual influencia, proporcionalmente, na complexidade deste mundo. De modo que quanto maior o número de usuários maior a complexidade. Quanto aos sistemas atuais observa-se que estes podem ser divididos em três classes em relação ao número dos usuários que suportam simultaneamente: monousuário; multi-usuário em pequena escala e multi-usuário em grande escala.

1.3.2 Número de Aplicações ou Usos Diferentes

Esta dimensão indica como o mundo virtual pode suportar atividades diferentes simultaneamente. Isto pode estar no contexto de um usuário que executa múltiplas tarefas, ou que usa várias ferramentas, ou no contexto de múltiplos usuários, que pode cada um estar associado a uma atividade diferente, e usar ferramentas diferentes dentro de um espaço compartilhado.

1.3.3 Número de Mundos

Esta dimensão indica a quantidade de mundos que pode existir simultaneamente. As quantidades maiores indicam uma complexidade maior. Nota-se que essa complexidade de múltiplos mundos deriva dos problemas de nomear e de ligar um grande número de mundos.

1.3.4 Interconexão entre os Mundos

Relaciona-se ao item anterior, já que um sistema deve suportar múltiplos mundos antes que a interconexão seja possível. Analisa-se que uma interconexão mais rica entre números maiores de mundos reflete uma complexidade maior.

1.3.5 Tamanho do Mundo

Descreve a extensão espacial de um ambiente. Assim, mundos maiores são considerados mais complexos.

1.3.6 Número dos Objetos

Refere-se à quantidade de objetos existentes simultaneamente dentro de um mundo. Observa-se que números maiores de objetos tendem a uma complexidade maior.

1.3.7 Complexidade do Objeto

Expressa a flexibilidade e a potencialidade de objetos individuais. Dessa forma, objetos mais flexíveis e mais capazes correspondem a uma complexidade maior. Em alguns casos, um objeto mais complicado pode ser equivalente a uma quantidade de outros menos complicados. Já em outros casos, um limite na complexidade do objeto pode ser um limite na potencialidade do sistema. Assim, a complexidade do objeto pode incluir complexidade gráfica, de áudio ou de texto que representam limites na aparência do objeto, ou complexidade da conduta.

1.3.8 Interação com o Mundo

Esta dimensão reflete a complexidade da interação com um mundo virtual. Visualiza-se que uma complexidade maior corresponde a mais modalidades de interação, maior quantidade de operações, mais flexíveis e eficientes.

1.3.9 Interação com os Usuários

Trata-se da complexidade da interação disponível entre usuários. Observa-se que mais modalidades da interação, por exemplo, meios diferentes, resulta em renderização mais

detalhada e os controles mais flexíveis e expressivos correspondem a uma maior complexidade.

1.3.10 Comportamento do Objeto

Verifica-se que é o aspecto da complexidade do objeto que trata da dinâmica dos objetos. O comportamento do objeto pode ser expresso, por exemplo, pelo código em uma linguagem de programação, pelas regras, ou por uma rede neural. Dessa maneira, quanto mais rico possível o comportamento – nos termos de operações disponíveis, no tamanho da especificação do comportamento e na escala da interação do objeto –, maior poderá ser a complexidade.

1.3.11 Escopo do Espaço

Este fator está relacionado ao número dos objetos em um mundo e é particularmente importante no contexto do modelo espacial da interação. Freqüentemente um objeto ou um usuário não necessita estar ciente de todos os objetos em um mundo; pode ser suficiente para ele estar ciente daqueles que estão "próximos" a ele. Nota-se que a complexidade é maior quando não há nenhum escopo espacial - todos os objetos no mundo têm a consciência mútua.

1.3.12 Construção do Mundo

Esta dimensão mede a flexibilidade e a facilidade com que os mundos podem ser criados e modificados. Os aspectos que aumentam a complexidade incluem a modificação "imersiva", pois modifica um mundo quando estiver interagindo com ele, e a modificação simultânea realiza-se por múltiplos usuários.

1.3.13 Customização

A customização expressa a extensão e a flexibilidade com que cada usuário pode excepcionalmente customizar suas interações com a percepção do mundo virtual. A maioria dos sistemas permite que alguma customização reflita a relação de equipamento disponível. Isso é visualizado, por exemplo, no desktop ou no tipo de rastreador imersivo.

1.3.14 Interoperação

Em outros sistemas isto mede a habilidade dos mesmos de interoperar com outros, sejam eles outros sistemas de realidade virtual ou outros sistemas mais comuns (por exemplo: banco de dados, planilhas, processadores de texto). Nota-se que as aplicações específicas podem ser criadas para fornecer uma interface virtual às aplicações tradicionais.

1.3.15 Utilização do Equipamento Terminal

Mede a habilidade dos sistemas ao fazerem o uso eficaz do equipamento que o usuário tem. Desse modo, nota-se que fazer o uso mais eficaz em uma escala maior de equipamentos corresponde a uma complexidade maior.

1.4 Aspectos de Escalabilidade

Os itens citados aqui apresentam requisitos em termos técnicos de complexidade e escalabilidade que atendem aos requisitos de AVDs. A escalabilidade refere-se à capacidade de crescimento de um sistema em termos de quantidade de usuários conectados ao mundo

(TANENBAUM, 1995; BENFORD et al., 2001). Verifica-se que as dificuldades de gerenciamento aumentam proporcionalmente ao número de usuários participantes.

No sentido de conseguir a escalabilidade para suportar a complexidade, observa-se que alguns tópicos são apresentados (GREENHALGH, 1996):

1.4.1 Distribuição

A computação distribuída possibilita um desempenho maior com a execução paralela do código e permite que mais periféricos sejam usados. Percebe-se que a distribuição é requerida também para suportar usuários geograficamente dispersos. Assim, observa-se que o custo de distribuição está nas comunicações. A distribuição também pode tornar disponível mais memória (memória adicional que está associada aos processadores adicionais). Entretanto, em um sistema inteiramente replicado isto não trará quase nenhuma contribuição a escalabilidade.

1.4.2 Balanceamento da Carga

Trata-se de uma técnica adicional para fazer melhor uso da maioria de recursos computacionais, e, ainda, é o balanceamento da carga, suportado pela migração do objeto, pois os objetos podem ser movidos dos processadores mais carregados para os menos carregados, o que torna melhor o uso do poder de processamento disponível.

1.4.3 Declarações Explícitas dos Interesses

Todos os objetos e processos podem explicitamente declarar o interesse em outros objetos e eventos. Este item suporta duas técnicas em tipos diferentes de aplicação. Na primeira, em uma base de dados distribuída, a replicação e a distribuição de itens da base de

dados podem ser minimizadas; isto evita a necessidade de replicar inteiramente a base de dados. Na segunda, os sistemas que permitem a declaração explícita de interesse podem ser usados para limitar a distribuição das mensagens e dos eventos somente àqueles objetos que os usarão. Estas técnicas devem resultar em redução nos *overheads* das comunicações, em menor exigência da memória e na subseqüente redução da computação nos nós de recepção ou nos objetos. Porém, os objetos de emissão e os componentes do sistema intermediário consumirão memória e processamento adicional para suportar o registro do interesse e filtrar as mensagens.

1.4.4 Localização Espacial de Interação

O conceito de modelo espacial é utilizado para limitar as interações do objeto baseado na proximidade espacial. Esta técnica é baseada na observação de que o objeto e os eventos que estão próximos no mundo virtual demandam maior interesse e atenção, que os outros que estão longe. Nesse sentido, a interação do objeto e a propagação dos eventos são confinadas para ocorrer somente dentro da região crítica, definida para cada objeto. Assim, o objeto tem suas comunicações reduzidas e exigências mais baixas da memória e da computação, tendo em vista que está ciente que pode interagir com somente um subconjunto dos objetos no mundo.

1.4.5 Subdivisão Espacial Adaptável

Relaciona-se à localização espacial descrita anteriormente. Percebe-se que ele é mais aplicado aos componentes do sistema que requerem alguma visão geral do mundo, como, por exemplo, para a detecção da colisão. Quanto à subdivisão espacial adaptável, observa-se que esta se trata de uma técnica pela qual o espaço total é dividido em subespaços, que são geralmente não sobrepostos e adjacentes. Cada subespaço pode ser suportado por um processo separado (em um processador separado), ou a divisão pode ser puramente interna. Na maioria das situações, a eficácia desta técnica depende das interações do objeto que são geralmente

locais. Isto é, a maioria das interações deve cair dentro de um subespaço, já que as interações entre subespaços resultam em comunicações adicionais para coordenação entre os processos que suportam aqueles subespaços.

1.4.6 Nível de Detalhes

Trata-se da técnica de explorar limitações perceptivas relacionadas à distância. Isto se relaciona ao fato de que os objetos distantes podem ser vistos (ou ouvidos) em menos detalhe e, é, na maioria das vezes, mais significativo para a interações com os objetos próximos. A base deste método deve explorar o conhecimento de limitações da interação (por exemplo, tamanho de renderização) para fornecer somente informação suficiente do que a informação completa aos observadores distantes. Como na localização espacial, o interesse explícito do distanciamento tenta reduzir as comunicações e, também, a computação e as exigências da memória sistematicamente de forma que limita a interação e a intervisibilidade do objeto.

1.4.7 *Dead Reckonin*

Esta técnica leva em consideração o uso da informação histórica sobre a posição e o movimento do objeto para predizer a posição e os movimento futuros. Isto é utilizado para compensar taxas diferentes do quadro em sistemas colaborativos, como, por exemplo, nivelaR o movimento aparente de um objeto remoto que executa em um processador lento. Esta técnica pode reduzir o número de mensagens da posição, isto é, reduzir as exigências das comunicações, mas isso ocorre à custa da computação adicional no que diz respeito aos cálculos da extrapolação e compensação do erro.

1.5 Taxonomia

O desenvolvimento de mundos virtuais multi-usuários deve fornecer uma estrutura coerente para a compreensão de AVDs. Observa-se que as comunicações, visões, dados e processos distribuídos são aspectos que devem ser analisados para escalar ambientes. Além disso, deve-se discutir as exigências, tais como, os sistemas que demandam forte consistência dos dados, a comunicação de confiança, que, ao mesmo tempo, necessitam suportar interação em tempo real e, por isso, provavelmente, não são muito bem escalares. Quanto aos sistemas dispersos geograficamente, nota-se que estes requerem alta velocidade e um suporte para a comunicação.

Verifica-se que os problemas são superados devido ao ligeiro crescimento de interconexões de rede de alta-velocidade, que resulta da disponibilidade de *workstations* gráficas de baixo-custo e do desenvolvimento de ferramentas gráficas e bibliotecas padronizadas. Entretanto, poucos AVDs tentam envolver todo esses componentes. Dessa forma, percebe-se que as experiências com estes são limitadas e, assim, os resultados publicados são raros (MACEDONIA; ZYDA, 1997).

Ainda segundo Macedônia e Zyda (1997), os aspectos críticos para a escala de AVDs são:

1.5.1 Comunicação

O recurso de comunicação mede a habilidade de transferir a informação entre os elementos distribuídos do sistema. Os parâmetros para avaliação da comunicação incluem:

1.5.1.1 Largura de Banda

A largura de banda é um fator limitante que representa a medida da capacidade de trafegar informações em um meio físico. Conceitualmente, é a diferença entre a maior e a menor freqüência que compõem a banda passante, determinando a capacidade de

transferência de dados. Esse fator tem grande importância, uma vez que a largura da banda da rede disponível pode determinar o tamanho e a complexidade de um ambiente virtual. A necessidade de largura de banda cresce com o aumento do número de participantes. Dessa maneira, os ambientes distribuídos requerem uma largura da banda que possa suportar os múltiplos usuários, os vídeos, os áudios e a troca de modelos e gráficos primitivos 3D em tempo-real. Assim, a mistura de dados requer novos protocolos e técnicas para tratar dados de forma apropriada acima de um *link* de rede (MACEDONIA; ZYDA, 1997).

1.5.1.2 Distribuição

Verifica-se que alguns esquemas de distribuição escalonam melhor que outros, de forma que a troca de mensagens entre as entidades participantes depende do tipo de implementação do sistema. Shirmohammadi e Georganas (2000) definem as seguintes categorias:

***Unicast*:** trata-se de uma abordagem distribuída, em que cada usuário tem em sua máquina uma cópia do mundo e a mudança local é enviada ao restante do grupo em um modo ponto-a-ponto. Esta abordagem torna-se ineficiente com o aumento de entidades e *hosts*, uma vez que aumenta o tráfego de mensagens;

***Broadcast*:** a mensagem do usuário é replicada para todos os usuários da rede. Nesse sentido, por questão de segurança a maioria dos roteadores da Internet bloqueia esse tipo de comunicação;

***Multicast*:** as mensagens são enviadas a todos os integrantes de um determinado grupo previamente definido. É a mais utilizada nos AVCs e pode ser do tipo *UDP (User Datagram Protocol) Multicast, multicast* sem garantia de entrega ou *Reliable Multicast, multicast* com garantia de entrega.

1.5.1.3 Latência

A latência controla a natureza dinâmica e interativa do ambiente virtual, bem como o comportamento interativo dos usuários participantes. Se um ambiente distribuído emula o mundo real, isto deve operar em tempo-real em termos de percepção humana. Sublinha-se que um desafio chave trata-se de que os sistemas apropriados envolvendo operadores humanos devem entregar pacotes com a latência mínima (menos que 100 ms) e gerar gráficos 3D texturizados em 30-60 Hz para garantir a ilusão de realidade. O ápice disso é a necessidade de fornecer em tempo-real: áudio, vídeo, e serviços de imagem para simulação de serviços de comunicação (MACEDONIA; ZYDA, 1997).

1.5.2 Visões

As Visões são as janelas dentro do ambiente virtual na perspectiva das pessoas ou no processo que as utilizam. Nota-se que são definidas duas espécies de visões úteis para ambientes distribuídos, a primeira visão apontada é a síncrona. Nessa visão se requer tanto alta realidade quanto baixa latência. Entretanto, ambientes virtuais que pedem visões síncronas, devido aos custos da sincronização, são restritos a redes locais.

As visões síncronas também são importantes para os projetos auxiliados por computadores e sistemas utilizados para projetos concorrentes.

O segundo conceito, considerado mais geral, é a visão assíncrona. Neste paradigma, os usuários múltiplos têm controle individual sobre quando e o que eles podem ver no ambiente virtual compartilhado. Assim, os AVDs de grande escala usarão visões assíncronas por causa do custo de sincronização em redes de longa distância.

1.5.3 Armazenamento de Dados

Talvez a decisão mais difícil em um ambiente distribuído está em determinar onde colocar os dados relevantes ao estado do mundo virtual e seus objetos. Essas decisões afetam a escala, oS requerimentos de comunicação, e a confiança dos dados.

Tendo em vista o armazenamento de dados, Macedonia e Zyda (1997) apontam uma classificação para estes:

A primeira classificação indica os **bancos de dados replicados de mundo homogêneos** em que se nota que cada *host* conectado na aplicação possui uma réplica completa de todos os dados que descrevem o ambiente virtual. Sempre que algum participante altera algo no ambiente, percebe-se que mensagens de atualização são enviadas para informar seu estado a todos os clientes da aplicação para manter a consistência de todos os mundos.

A vantagem é que, estas mensagens têm o tamanho reduzido, pois contém apenas as atualizações no estado do objeto. Trata-se do modelo mais simples de ser implementado, porém se a aplicação necessitar de um alto grau de interatividade e a rede de comunicação não permitir o suporte adequado, o desempenho poderá ser prejudicado, o que dificulta a manutenção da consistência dos dados entre os participantes da aplicação.

Já nos **bancos de dados centralizados e compartilhados** os dados ficam centralizados em uma única máquina, em que o servidor recebe requisições de atualização do estado do ambiente virtual. Isto facilita o controle de inconsistência no ambiente virtual. Dessa forma cada participante mantém somente as informações pertinentes à cena na memória e somente um participante pode modificar o banco de dados em um determinado momento.

Nesse modelo, o servidor pode se tornar um gargalo, limitando o número de participantes da aplicação.

Quanto aos **bancos de dados distribuídos e compartilhados com alteração *peer-to-peer***, observa-se que neste os objetos são replicados nos participantes da aplicação de acordo com a necessidade do colaborador. A cada momento, a replicação de dados se dá somente naqueles participantes que de fato necessitam de uma mesma informação.

Este modelo é adequado para construir ambientes distribuídos em grande escala. Sua desvantagem está na dificuldade de escalabilidade, devido ao custo da comunicação associada à confiabilidade e à consistência de dados por meio de redes de comunicação.

Nos **bancos de dados distribuídos e compartilhados cliente/servidor,** cada parte

do ambiente virtual é de responsabilidade de um dos participantes. Neste caso, um participante recebe todas as requisições referentes aos objetos que armazena e encarrega-se de manter a consistência desta parte do ambiente. Um agente chamado *broker* é responsável por manter o servidor atualizado.

Esses bancos de dados replicados do mundo são mais eficientes que os esquemas de banco de dados compartilhados centralizados ou distribuídos. Entretanto, eles, geralmente, carecem de um modo para manter a consistência do mundo e, ainda, precisam da capacidade para atualizar o ambiente virtual com novos objetos ou comportamentos. Porém, AVDs com muitos participantes podem usar um modelo misto – inicialização do participante com pequenos dados replicados e um modelo cliente-servidor distribuído. Isto permite mais consistência de dados e persistência caso um mecanismo ou heurística for utilizado para reduzir a latência de transferência (MACEDONIA; ZYDA, 1997).

1.6 Modelos de Comunicação de Comunicação

Visualiza-se que um aspecto relevante relacionado à concepção de AVCs envolve o suporte à comunicação. Existem vários modelos que podem ser seguidos para o desenvolvimento da comunicação em AVCs, e os modelos de comunicação apresentados nesta seção serão: Cliente/Servidor, *Peer-to-Peer* e Híbrido.

1.6.1 Avaliação das Arquiteturas de Comunicação

Segundo Batista et al. (2000), não existe uma arquitetura de comunicação melhor que a outra para AVCs. A escolha deve ser feita levando-se em conta os recursos existentes e as aplicações para as quais está mais vocacionada. A avaliação das arquiteturas tem como base os seguintes pontos:

- **Atualização da posição de objetos compartilhados:** A alteração de um objeto presente no mundo deve ser comunicada a todos os outros participantes, informando-lhes a nova

posição do objeto. A freqüência das atualizações se dá de forma a manter um ritmo suave, isto é, sem quebra de movimento.

- **Aquisição de acesso exclusivo a um objeto:** Refere-se ao gerenciamento do acesso exclusivo da manipulação de um dado objeto. Para isso, são utilizados mecanismos de trava (*lock*), que consiste em adquirir/fechar antes de usar e liberar/abrir após o uso. Deste modo, assegura-se que só quem tem a posse da trava poderá alterar o referido objeto.

- **Invocação distribuída de funções:** É necessário comunicar a todos os participantes o desejo de invocar uma função em todos os sistemas de RV e passar os parâmetros para essa função.

- **Carregamento de objetos:** Para carregar um novo objeto no mundo virtual é preciso avisar todas as máquinas para fazerem o mesmo e passar a geometria do mesmo para todas as máquinas.

- **Admissão e saída dos participantes:** Quando um novo usuário entra no mundo, os outros participantes devem ser avisados e o usuário deve obter o estado atual do mundo. Ao sair, dependendo da arquitetura, deverá avisar os outros participantes.

Os principais modelos de comunicação propostos por Macedonia e Zyda (1997) são:

1.6.1.1 Arquitetura Cliente Servidor

Nesta arquitetura, há um servidor central responsável por gerenciar o estado da aplicação. Assim, os clientes devem consultá-lo para poder fazer qualquer tipo de operação. O estado refere-se à posição dos objetos compartilhados e localização dos participantes e quem está com o direito de manipular o objeto.

Antes de gerar uma imagem o participante deve consultar o servidor sobre a posição dos objetos comuns a todos os participantes e, ainda, verificar como deve fazer um pedido ao servidor para poder atualizar sua posição.

Observa-se que quando alguém deseja manipular um objeto, deve antes solicitar ao servidor que visualiza se o objeto não está no poder de outro participante, caso esteja livre, há o bloqueio de seu acesso e, então, há a liberação.

A invocação às funções distribuídas deve ser pedida ao servidor, juntamente com os parâmetros necessários Os objetos compartilhados também devem ser carregados da mesma forma.

Quanto às entradas e às saídas dos participantes, estas devem ser solicitadas ao servidor, que se encarrega de avisar os demais participantes e fazer as devidas atualizações do sistema, tais como, a liberação de eventuais travas de acesso que estiverem em poder do mesmo.

Esse modelo é de simples utilização, tendo em vista que todo o estado do sistema está concentrado no servidor que apenas recebe as solicitações dos clientes e os passa as respostas. Dessa forma, ele facilita a tomada de decisões e torna mais fácil a construção de sistemas distribuídos. Além disso, cada participante pode ter uma percepção bem próxima do estado atual do sistema. A otimização também se dá pelo fato do servidor precisar informar as posições dos objetos que o usuário visualizar no momento.

Observa-se que a concentração do sistema no servidor central pode se tornar um fator limitante da escalabilidade do sistema, uma vez que restringe a expansão do número de usuários pela capacidade de processamento no servidor. Nesse sentido a tolerância às falhas também fica comprometida, pois se o servidor falhar, todos os usuários serão afetados. Outro fator de interferência é o desperdício da largura de banda, pois todas as mensagens necessariamente passam pelo servidor, mesmo as que poderiam ser passadas diretamente entre os participantes.

Este tipo de arquitetura é mais indicado para aplicações em que os participantes necessitam de uma visão consistente do mundo e independente do tipo de rede.

1.6.1.2 Arquitetura *Pee-to-Peer*

Na arquitetura *peer-to-peer,* as aplicações rodam na máquina de cada participante, sem um servidor central. O estado da aplicação é distribuído/replicado em todos os participantes, sendo cada um o responsável pela comunicação dos demais participantes e pelas

modificações que fizerem, tanto de objetos quanto de quem os está modificando. Para alterar a posição de um objeto compartilhado ou sua posição no ambiente, é necessário que o participante envie um pedido de atualização, por difusão, a todos os participantes do grupo. Assim, a comunicação entre eles é feita por difusão simples ou seletiva.

Para poder gerenciar o controle das alterações dos objetos, cada participante deve possuir o estado das travas de acesso dos objetos, e com quais participantes estão. Ao conhecer o estado da trava do objeto desejado, caso esta esteja liberada, pode-se fazer as solicitações de alteração do mesmo. Após a liberação do objeto, deve-se avisar a todos os participantes desse evento. Quanto ao carregamento e à execução de funções, observa-se que estes também são feitos por meio da difusão do pedido para todos os participantes fazerem o mesmo.

A entrada e a saída de um novo usuário no ambiente devem ser avisadas a todos, para que estes atualizem o sistema com a adição ou exclusão do usuário. Também será preciso pedir a um dos participantes que envie por difusão o estado atual do mundo.

Nesta arquitetura, a falha de algum participante pode não ser percebida pelos outros. Um modo de controlar este problema seria, de tempo em tempo, os participantes enviarem pacotes confirmando que ainda estão ativos. Caso algum participante ultrapasse o limite de tempo estabelecido, este terá as informações associadas a ele eliminadas do grupo.

Verifica-se que a vantagem desta arquitetura está na segurança de que caso uma máquina falhe, somente ela será afetada. Nota-se que a inexistência de um ponto central, além de não prejudicar a performance do sistema, também não limita a escalabilidade do mesmo, isso revela uma diminuição do tráfego de mensagens, que não necessitam passar pelo servidor central, o que melhora o desempenho do sistema e reduz o consumo de largura de banda.

Em contrapartida, a sincronização no acesso aos objetos compartilhados é mais complexa, tendo em vista que a visão do mundo na máquina do usuário pode estar defasada em relação ao seu estado atual. Assim, é mais complexa de se realizar do que a abordagem cliente-servidor, pois cada máquina tem que ser responsável pelo seu processamento e pelos pedidos da rede.

Dessa forma, observa-se que este tipo de arquitetura é adequado para as aplicações que necessitem suportar um grande número de usuários e eles não necessitam compartilhar a mesma visão idêntica do mundo.

1.6.1.3 Arquitetura Híbrida

A arquitetura híbrida visa aproveitar as vantagens das duas arquiteturas anteriores e fornecer uma opção mais robusta.

Assim, utilizam-se as características mais eficientes da arquitetura cliente-servidor, tais como, o controle de acesso exclusivo ao objeto compartilhado e o controle de entrada e saída dos participantes. Neste caso, ao entrar no mundo, o participante deve informar o servidor central e pedir o estado atual do mundo, além das referências dos objetos e suas localizações. O servidor central apenas guarda os estados das travas de acesso e não suas posições e gerencia o acesso às travas.

Percebe-se que as atualizações de posição, bem como, as invocações de funções são feitas pela arquitetura *peer-to-peer*, que se mostra mais adequada para estas atividades. Essa arquitetura agrega as vantagens das arquiteturas anteriores. Assim, os problemas de processamento e de largura de banda são eliminados. Já o número de participantes não é restringido. Quanto à sincronização de objetos compartilhados, esta é fácil e tolerante à falhas dos participantes.

Porém, como não poderia deixar de ser, traz consigo algumas das desvantagens das duas arquiteturas tais como, o servidor central continua a ser um ponto vulnerável à falhas, de modo que a visão do mundo para cada participante, em um dado instante, pode estar defasada em relação ao seu estado atual.

1.7 Metáfora Colaborativa

Tendo em vista que a construção de tecnologias não é suficiente para definir o trabalho colaborativo, é necessário aprender mais sobre como as pessoas trabalham em grupo e como a tecnologia afeta isto.

Nota-se que muitos projetos de interfaces monousuário exploram características cognitivas espaciais do indivíduo, tal como, a habilidade para classificar espacialmente e navegar. Nesse sentido, deve-se levar em conta como as considerações espaciais se relacionam às interações sociais e como são relevantes para projetos de sistemas

colaborativos.

O argumento utilizado é que a provisão do espaço virtual como ambiente para trabalho colaborativo permite as pessoas empregarem um rico conjunto de habilidades espaciais sociais para gerenciar interação, compartilhar e, também, trocar informações.

1.7.1. Espaço Social

No trabalho de Benford et al. (1994b apud GIDDENS, 1984) são discutidos alguns dos modos em que o espaço e o tempo podem ser considerados como recursos para atividade de interação. O espaço constitui um recurso chave para estabelecer e habilitar uma atividade. Quanto à introdução das noções de “modos de regionalização”, verifica-se que esta tenta capturar a interdependência do espaço geográfico como prática social.

Nesse sentido, conforme Benford et al. (1994b apud GIDDENS, 1984), os limites são construídos para segmentar o espaço em diferentes regiões. Esses limites podem ser simbólicos ou físicos, como, por exemplo, paredes entre salas, telas ou marcas no mapa, estes marcam um limite e a partir daí constroem uma região. Verifica-se que eles além de estabelecerem os limites de espaços diferenciados também servem para identificar locais para atividades específicas e, ainda, incluem alguns elementos, como, pessoas, recursos, objetos dentre outros, enquanto excluem outros.

Sublinha-se que a marcação do espaço externo que limita e constitui as regiões é importante para a prática social e define o que as pessoas podem ou não fazer. Dessa forma, os limites no espaço permitem diferentes modelos de participação em uma prática social ao longo de diferentes exemplos de consciência. De acordo com Benford et al. (1994b), os limites reforçam diferenças entre modelos de consciência.

Já o espaço consente a percepção implícita da presença e da atividade de outros espaços e fornece uma faixa da sutil negociação entre os habitantes. Observa-se que a percepção contínua de outros espaços permite às pessoas uma visão prévia de suas prováveis ações e modificam flexivelmente suas próprias atividades em situações sociais.

Nota-se que durante a conversação, usam-se propriedades do espaço para mostrar quem, naquele momento, fala e quem deseja falar a seguir. Verifica-se que, em particular, orientação e direção do olhar são usadas como mecanismos de interação flexíveis. Quando

alguém fala, pode-se julgar a reação de outros usuários do espaço, como, por exemplo, a movimentação inquieta ao redor pode significar aborrecimento, o que sugere que a palavra deve ser passada para outro.

Visualiza-se que o espaço também fornece negociação do acesso aos recursos compartilhados pelo fornecimento da percepção de quem o está usando e quem pretende usá-lo. Isto permite desenvolver convenções sociais para controlar o acesso do recurso. Todas estas observações, quando associadas, fornecem motivações convincentes para adotar um espaço aproximado do CSCW.

1.7.2 Modelo Espacial de Interação

O modelo espacial de interação é usado para gerenciar as interações entre um número de usuários e artefatos dentro do mundo. Seu objetivo é fornecer um pequeno, mas poderoso conjunto de mecanismos para suportar a negociação da interação social por meio de um escopo de espaço virtual.

O modelo espacial usa a propriedade de espaço como a base para mediar à interação com o objetivo de suportar grupos de pessoas usando suas habilidades naturais de comunicação em ambientes virtuais distribuídos (BENFORD et al., 1994a).

Benford et al. (1994a) apontam um conjunto de conceitos que são abstraídos para definir este modelo:

Espaço e Objetos: Espaço trata-sede métricas espaciais, ou seja, modos bem definidos de medir a posição e direção por meio de um conjunto de dimensões. Observa-se que o espaço é habitado por objetos que podem representar pessoas, informação ou outros artefatos computacionais. Assim, qualquer interação entre objetos ocorre por meio de algum método que deve representar um típico meio de comunicação, tal como, áudio, visual, texto ou talvez algum outro tipo de objeto específico de interface. Cada objeto, dessa forma, deve ser capaz de interagir por meio de uma combinação de meio/interface e negociar o meio compatível toda vez que eles se encontrarem no espaço.

Aura: Define um subespaço em que a interação é possível, ou seja, a aura de um objeto é um

limitador das possibilidades de suas interações em um mundo virtual povoado (GREENHALG, 1994). Assim, a aura determina quais objetos são capazes de interagir com outros objetos num dado tempo. Quanto aos objetos, estes carregam suas auras onde se movem. Já quando duas auras se colidem, a interação torna-se uma possibilidade. Nesse sentido, verifica-se que a aura atua como uma tecnologia fundamental habilitadora da interação e trata-se do caminho mais elementar para identificar um subespaço associado com outro objeto. Sublinha-se que uma aura pode ter qualquer forma e tamanho e não necessita estar ao redor do objeto ao qual pertence e nem necessita ser contínua no espaço. Além disso, cada objeto tipicamente possuirá diferentes auras para diferentes meios – com diferentes tamanhos e formas. Dessa maneira, nota-se que quando um participante R se aproxima do participante T por meio do espaço, R deve ser capaz de ver T antes de poder ouvi-lo, uma vez que a aura visual é maior do que aura auditiva.

Focus: Refere-se a um subconjunto do espaço, em que o usuário tem sua atenção focalizada (Figura 1.1). Tendo em vista o familiar conceito de foco visual, o focus espacial é um meio de direcionar a atenção e, então, filtrar informação (GREENHALG, 1994).

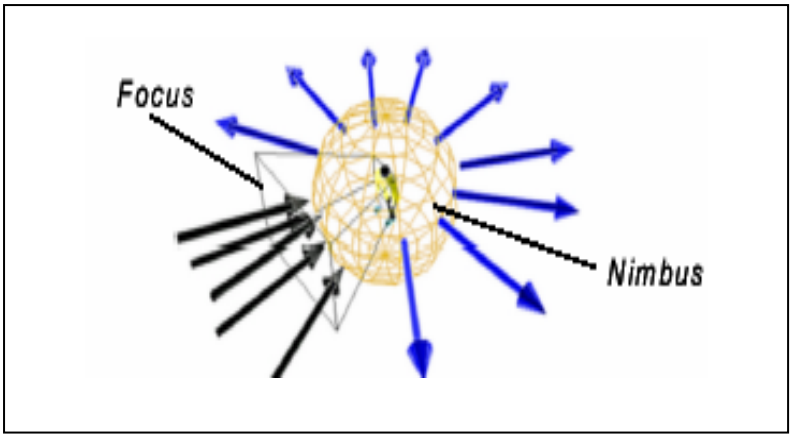

Figura 1.1: Focus e Nimbus (CHEVERST; SMITH, 2001).

Nimbus: Define-se como a projeção do usuário sobre o espaço. Ou seja, trata-se de um conjunto de recursos próprios que o usuário interessa compartilhar com outros e que podem ser informados sobre as atividades do usuário. Em termos gerais, um nimbus é um subespaço em que um objeto faz alguns de seus aspectos disponíveis para outros. Isto pode ser sua presença, identidade, atividade ou alguma combinação deles (Figura 11). Observa-se que o

nimbus permite aos objetos tentarem influenciar outros – projetar eles mesmos ou suas atividades para serem ouvidos ou vistos. Dessa maneira, percebe-se que há ocasiões em que o transmissor de informações necessita atrair a atenção do receptor (interromper alguém). O nimbus fornece ao transmissor a habilidade para fazer isto. O focus e o nimbus são mecanismos pelos quais o transmissor e o receptor controlam o fluxo de informação ao longo do canal uma vez que ele foi estabelecido. Nota-se que o focus representa o controle do receptor e já o nimbus é o controle do transmissor (GREENHALG, 1994).

Percepção: responsável por quantificar o grau, a natureza ou a qualidade de comunicação entre dois objetos. Sabe-se que esta pode se dar por meio de várias formas, maneiras e níveis. Quanto á unidade de informação que é comunicada verifica-se que esta pode ser atingida apenas se houver entendimento entre o comunicador e o receptor. Isso evidencia que a informação tem que ser percebida no contexto em que foi inserida e com base nas regras e nas linguagens estabelecidas para a comunicação. Percebe-se que a percepção quantifica o significado do objeto para outro objeto em um meio particular; assim, ela corresponde ao volume do canal de áudio ou ao nível de detalhes gráficos de uma renderização. As percepções são negociadas pela combinação do focus do observador e do nimbus do observado (GREENHALG, 1994).

A mensuração da percepção entre dois objetos não necessita ser mutuamente simétrica (a percepção de R sobre T não necessita ser igual à percepção de T sobre R), como pode ser percebido na Figura 1.2. Da mesma maneira que a aura, os níveis de percepção são específicos do meio. Quanto aos objetos, estes negociam o nível de percepção pelo uso de seu focus e de seu nimbus para tentarem atrair a atenção para eles ou fazerem deles mesmos mais percebidos pelos outros. Mais especificamente (LOGAN et al., 2002):

- **Quanto mais um objeto R está dentro do *focus de T*, mais percepção R tem de T.**
- **Quanto mais um objeto R está dentro do *nimbus* de T, mais percepção T tem de R.**

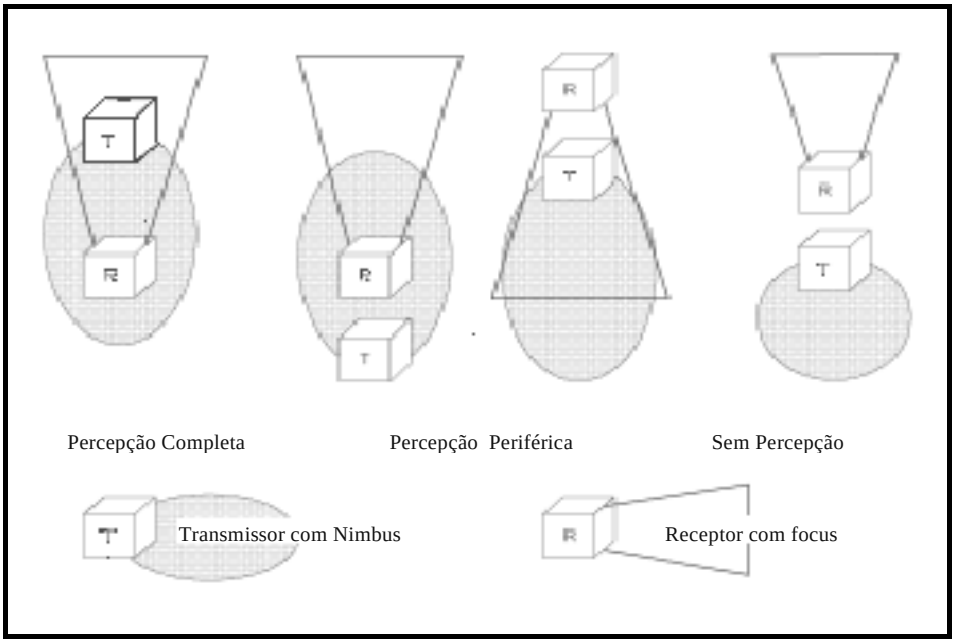

Figura 1.2: Como o focus de R e nimbus de T determinam a percepção de R sobre T. (LOGAN et al., 2002).

- **Meio:** trata-se do tipo de comunicação, tal como, áudio, vídeo ou texto. (GREEHALGH; BENFORD, 1997).

- **Limites:** os limites no espaço também influenciam a aura, o focus e o nimbus, de maneira que divide o espaço em áreas e regiões diferentes, fornecendo mecanismos para marcar território, controlar movimento e influenciar a propriedade de interação do espaço.

- **Manipulação:** ao se mover um objeto, automaticamente, carrega consigo sua aura, seu focus e seu nimbus, que podem ser implicitamente manipulados por meio de ações como movimento e orientação.

- **Adaptadores:** o focus e o nimbus de um objeto podem ser modificados pelos chamados objetos adaptadores, tais como, lanternas, telescópios, alto-falantes, paredes, dentre outros. Todos esses objetos podem ativar amplificar ou atenuar a aura, o focus e o nimbus de algum modo. (GREENHALGH, 1994; BENFORD et al., 1994a).

CAPÍTULO 2

Groupware

Este capítulo descreve o *groupware*, que se trata de uma tecnologia desenvolvida para facilitar o trabalho em grupo. Esta tecnologia pode ser usada para comunicar, cooperar, coordenar, resolver problemas, competir ou negociar (FUKS et al., 2002). O termo *groupware* é usado para se referir a uma classe de tecnologia utilizada para comunicação entre duas ou mais pessoas; essa comunicação relaciona-se com as modernas redes de computadores, como, por exemplo, e-mail, grupo de notícias, vídeo conferência ou *chat* (DIX et al., 2004).

2.1 Definição de *Groupware*

Desde o seu surgimento, o homem se organiza em grupos, uma vez que percebeu que só assim conseguiria atingir seus objetivos de forma mais rápida e com maior facilidade (FUKS et al., 2002). Para isso, começou a buscar parcerias, dividiu tarefas e cooperou na execução das mesmas. Dessa forma, vários modelos para auxiliar a cooperação apareceram.

Com o surgimento dos computadores e a sua ligação em rede, o homem passou a dispor de um auxiliar poderoso para o trabalho em grupo e para realizá-lo de uma forma mais eficaz.

Este pressuposto constitui a motivação para o aparecimento dos conceitos de *Groupware* e CSCW. Observa-se que a massificação do uso da Internet tornou possível ao

homem, com o recurso do computador, se manter em contato com um número crescente de outros indivíduos dispersos geograficamente por, praticamente, todo o planeta, constituindo-se como um sistema de informação global (ZYDA; SINGHAL 1999, BENFORT et al., 2001).

A área de pesquisa em CSCW visa estudar o trabalho colaborativo por meio de sistemas de computação, que apóiem a comunicação, a coordenação e a cooperação entre os membros da equipe envolvida no trabalho, mesmo que estes estejam distribuídos no tempo e no espaço (FUKS et al., 2002). Percebe-se que o campo atrai, principalmente, àqueles interessados no projeto do software e no comportamento social e organizacional. Esses interessados incluem pessoas de negócios, cientistas da computação, psicólogos organizacionais, pesquisadores das comunicações, antropólogos dentre outros especialistas.

O estudo de *groupware* objetiva compreender a formação de grupos e como as pessoas se comportam nesses grupos. Visa também ter uma compreensão da tecnologia de redes de computadores e como os aspectos dessa tecnologia (por exemplo, atraso na sincronização das visões) afetam a experiência de um usuário. Observa-se que todas as características relacionadas ao projeto tradicional de interface com o usuário permanecem relevantes, visto que a tecnologia ainda envolve pessoas. Entretanto, muitos aspectos dos grupos requerem consideração especial. Por exemplo, visualiza-se que grupos com milhares de pessoas se comportam de forma diferente de um grupo de poucas pessoas, mas também se nota que os parâmetros de desempenho das tecnologias para suportar grupos diferentes variam. Dessa forma, a resposta e a confiabilidade do sistema transformam-se em umas das questões mais significativas (TANENBAUM, 1995).

O trabalho cooperativo pode ter muitas formas, isso depende de fatores como: a tarefa a ser executada, os grupos formados, a duração e o contexto. Estas formas freqüentemente mudam com o tempo, mesmo dentro do confinamento de um único projeto. Para suportar contextos dinâmicos, os serviços de *groupware* devem ser compreensíveis e flexíveis. Segundo Hofte (1998) e Dix (2004), as tecnologias *groupware* podem variar quanto aos requisitos entre os suportes para:

1 – usuários que trabalham ao mesmo tempo (“tempo real” ou “síncrono”) ou em tempos diferentes (“assíncrono”).

2 – usuários que trabalham no mesmo local (“mesmo local” ou “face-a-face”) ou em diferentes locais (“locais diferentes” ou “à distância”).

3 – único meio discreto para comunicação como um texto ou gráfico ou meios múltiplos que incluem meios contínuos como áudio e vídeo.

4 – liberdade de ação do usuário (“suporte permissivo”) ou coordenação da ação do usuário (“suporte restritivo”).

A Figura 2.1 mostra a Matriz Tempo X Espaço.

	Mesmo Tempo	Tempos Diferentes
Mesmo Local	Interação Síncrona (face-a-face) ex.: sala de reuniões virtuais, salas de aula virtuais	Interação Assíncrona ex: mural de avisos, ferramentas de coordenação, programas de projeto, agenda eletrônica
Locais Diferentes	Interação Síncrona Distribuída Ex.: telefone, editores compartilhados, janelas de vídeo compartilhadas	Interação Assíncrona Ex.: carta, e-mails, *newsgroups*

Figura 2. 1: Matriz Tempo X Espaço. Adaptada de Dix (2004).

O *groupware* oferece significativas vantagens sobre sistemas monousuário. Visualiza-se que algumas das razões mais comuns, pelas quais as pessoas querem usá-lo são (BRINCK, 2005):

- facilitar uma comunicação;
- permitir uma comunicação em que não há outra maneira possível;
- permitir a telecomunicação;
- reduzir nos custos de viagens;
- juntar múltiplas perspectivas e especialidades;
- formar grupos com interesses comuns mesmo que estejam localizados remotamente;
- economizar tempo e custos na coordenação de trabalho em grupo;
- facilitar a solução de problemas em grupo;

- permitir novas modalidades de comunicação, como intercâmbios ou interações estruturadas.

Nesse sentido, a propriedade principal, em relação à interação, que todos os grupos devem apresentar, trata-se do envio de uma a mensagem para todo grupo, ou seja, um membro envia a mesma mensagem para todos os outros integrantes do grupo ao mesmo tempo. É uma forma de comunicação um-para-muitos (um envia, muitos recebem); assim, apresenta-se de maneira contrária à comunicação ponto-a-ponto, também denominada um-para-um.

Sublinha-se que os grupos são dinâmicos e que novos grupos podem ser criados e grupos velhos podem ser destruídos e, ainda, um processo pode se juntar a um grupo ou deixá-lo. Também se verifica que um processo pode ser um membro de vários grupos ao mesmo tempo. Conseqüentemente, para isso são necessários mecanismos para gerenciar grupos e os membros do grupo (TANENBAUM, 1995).

Observa-se que o *Groupware* é análogo a uma organização social. Assim, um modelo de integração pode suportar atividades síncronas assíncronas, que habilita um grupo de pessoas ou time virtual trocar informações e interagir com outros. Por um lado, em uma sessão síncrona, todos os usuários compartilham uma visão de uma discussão e informações são trocadas tão logo são feitas avaliações. Já, por outro lado, em uma sessão assíncrona, as informações somente são trocadas sob demanda (IQBAL et al., 2003).

2.2 Características do Projeto

Segundo Tanenbaum (1995), a comunicação em grupo tem características de projeto importantes que devem ser analisadas, tais como:

2.2.1 Grupos Fechados ou Grupos Abertos

Tendo em vista os sistemas que suportam a comunicação em grupo, verifica-se que estes podem ser divididos em duas categorias, dependendo de como eles enviam as

mensagens e para quem as enviam. Alguns sistemas suportam grupos fechados, em que somente os membros de um grupo podem enviar para o mesmo grupo. Dessa forma, os membros externos ao grupo não podem enviar mensagens para o grupo como um todo, embora eles possam ser capazes de enviar mensagens para membros individuais. Porém, quando os grupos abertos são usados, qualquer processo no sistema pode enviar para qualquer grupo (TANEMBAUM, 1995).

2.2.2 Grupos Simétricos ou Grupos Hierárquicos

A distinção entre grupos fechados e abertos aponta quem pode comunicar com o grupo. Outra distinção importante está relacionada à estrutura interna do grupo. Em alguns grupos, todos os processos são iguais, uma vez que não há diferença na hierarquia dos processos e todas as decisões são feitas coletivamente.

Percebe-se que pode haver grupos em que exista hierarquia. Nesse sentido há um processo que é o coordenador e todos os outros são trabalhadores. Neste modelo, quando uma requisição para um trabalho é gerada, ou por um cliente externo ou por um dos trabalhadores, ele é enviado para o coordenador, que decide qual trabalhador está qualificado para executá-lo.

Nesse sentido, cada uma destas organizações tem suas próprias vantagens e desvantagens. No grupo simétrico, por exemplo, caso um dos processos falhem, o grupo simplesmente se torna menor, mas pode continuar a existir. Uma das desvantagens evidenciadas é que a decisão se torna mais complicada. Assim, para decidir algo, uma votação deverá ser feita, incorrendo em demora ou despesa.

Observa-se que os grupos hierárquicos têm propriedades contrárias aos grupos simétricos. Isso é revelado quando a perda de um coordenador causa a parada do grupo inteiro, porém quando está em execução, pode tomar as decisões sem a necessidade de consultar outro processo (TANEMBAUM, 1995).

2.2.3 Membros do grupo

Ressalta-se que quando há comunicação em grupo, é necessário algum método para criar e destruir grupos, como, também, para permitir processos de se juntar aos grupos e deixá-los. Uma abordagem possível é ter servidores de grupo para onde todos estes requisitos possam ser enviados. O servidor de grupos pode manter uma completa base de dados de todos os grupos e de seus membros. Este método é eficiente e fácil de implementar. Porém, ele compartilha uma desvantagem de todas as técnicas centralizadas, pois se o servidor de grupos falhar, o grupo deixa de existir. Provavelmente, os grupos têm de ser reestruturados do princípio, e a partir daí encerrar qualquer trabalho que estiver sendo feito.

A abordagem oposta é o gerenciado de membros do grupo de modo distribuído, que indica que em um grupo aberto, um processo pode enviar mensagens para todos os membros dos grupos que estão presentes. Observa-se que em um grupo fechado, algumas particularidades são necessárias, como, por exemplo, para deixar um grupo, um membro deve enviar uma mensagem informando a todos a sua desconexão.

Entretanto, se um membro falhar, ele efetivamente deixará o grupo. O problema visualizado está na questão de que não há um anúncio deste fato como em um processo voluntário de desconexão, tendo em vista que o outro membro precisa descobrir isso, experimentalmente, pela notificação de que o membro que falhou não está respondendo, e assim removê-lo do grupo.

A outra questão complicada é que o ato de deixar ou de se conectar a um grupo só acontece com o envio de mensagens síncronas. Ou seja, no instante que o processo se juntar ao grupo, deve receber todas as mensagens enviadas ao grupo. Similarmente, tão logo um processo deixe o grupo, ele não deve mais receber mensagens do grupo. A partir disso, uma mensagem deve ser enviada para todos os integrantes do grupo para avisá-los quando há uma conexão ou uma desconexão de um processo do grupo (TANEMBAUM, 1995).

2.2.4 Endereçamento do grupo

Para enviar uma mensagem para um grupo, um processo deve ter um modo de especificar a qual grupo ele pertence, isto é, os grupos necessitam ser endereçados. Desse

modo, é necessário dar a cada grupo um endereço único, da mesma forma que o endereço do processo. Caso a rede suporte *multicast*, o endereço do grupo pode ser associado com um endereço *multicast,* de modo que toda mensagem enviada para o endereço do grupo, possa ser *multicast*.

Se o hardware suporta broadcast e não *multicast*, a mensagem pode ser *broadcast.* O *kernel*, então, extrai da mensagem o endereço do grupo. Se nenhum dos processos na máquina é membro do grupo, a mensagem é simplesmente descartada. Caso contrário, ela é passada para todos os membros do grupo.

Porém, se nem o *multicast* nem o *broadcast* são suportados, o *kernel* na máquina remetente terá que ter uma lista de máquinas que têm processos pertencentes ao grupo e, a partir daí, enviara a cada um, a mensagem *unicast* (ponto-a-ponto). Nota-se que nos três casos, um processo só enviará uma mensagem para um endereço de grupo e esta será entregue para todos os membros. Nota-se que o remetente não tem consciência do tamanho do grupo ou se a comunicação é implementada por *multicasting*, *broadcasting*, ou *unicasting*.

Um segundo método de endereçamento de grupo é requerer que o remetente forneça uma lista explícita de todos os destinatários. Quando este método é usado, um ponteiro para uma lista de endereços deve ser utilizado como parâmetro para chamar a rotina e enviar a mensagem. Este método apresenta o inconveniente, que se trata de forçar processos usuários para conhecer quem é membro de qual grupo. Ou seja, ele não é transparente. Além disso, quando um membro muda de grupo, o processo usuário deve atualizar a lista de membros.

Verifica-se que a comunicação em grupo também permite um terceiro método de endereçamento denominado de predicado de endereçamento. Com este sistema, cada mensagem é enviada para todos os membros do grupo (ou possivelmente o sistema inteiro) usando um dos métodos descritos anteriormente, mas com um novo desdobramento. Cada mensagem contém um predicado (expressão booleana) para ser avaliada. O predicado pode envolver o número da máquina receptora, sua variável local, ou outros fatores. Se o predicado é avaliado como verdadeiro, a mensagem é aceita. Caso seja avaliada como falsa, a mensagem é descartada (TANENBAUM, 1995).

2.2.5 Primitivas *Send* e *Receive*

Preferivelmente, a comunicação em grupo e a ponto-a-ponto deveriam ser unidas em um único conjunto de primitivas.

Porém, o envio e o recebimento de mensagens em um grupo não podem ser modelados como uma chamada a procedimento. A primeira dificuldade encontrada é que com *Remote Procedure Call* (RPC – chamada de procedimento remoto), o cliente envia uma mensagem ao servidor e recebe de volta a resposta. Já na comunicação em grupo, há a possibilidade de **n** respostas diferentes. Dessa maneira, como é inviável a um procedimento controlar **n** respostas, uma tentativa é abandonar o modelo RPC e voltar para chamadas explícitas, uma para *send* e outra para *receive*.

A biblioteca de procedimentos, que processa chamadas para invocar a comunicação em grupo, pode ser a mesma comunicação ponto-a-ponto. Se o sistema é baseado no RPC, processos do usuário nunca chamam *send* e *receive* diretamente. Caso o programa do usuário chamar diretamente *send* e *receive,* há algumas coisas que precisam ser observadas para fazer a comunicação em grupo com estas primitivas existentes, ao invés de inventar um novo conjunto.

Ao supor que se deseje juntar as duas formas de comunicação para enviar uma mensagem, um dos parâmetros do *send* indica o destinatário. Caso for um endereço de processo, uma única mensagem é enviada para aquele processo. Já se for um endereço de grupo (ou um apontador para uma lista de destinatário), uma mensagem é enviada para todos os membros do grupo. Um segundo parâmetro para *send* aponta para a mensagem.

Quanto à chamada, esta pode ser armazenada ou não armazenada, bloqueada ou desbloqueada, confiável ou não confiável, para ambos, ponto-a-ponto e em grupo. Genericamente, estas escolhas são feitas pelos desenvolvedores de sistema e são fixos.

Estas duas formas de comunicação são freqüentemente usadas para propósitos diferentes, portanto, alguns sistemas introduzem uma nova biblioteca de procedimentos, ou seja, *group_send* e *group_receive* e, assim, um processo pode indicar se quer uma mensagem ponto-a-ponto ou uma mensagem em grupo (TANEMBAUM, 1995).

2.2.6 Atomicidade

Uma das características da comunicação em grupo é a propriedade todos-ou-nenhum. Muitos sistemas de comunicação em grupo são projetados para que quando uma mensagem seja enviada para o grupo, ela chegue corretamente para todos os membros do grupo ou não chegue para nenhum deles. Percebe-se que as situações em que apenas alguns membros recebam uma mensagem e outros não, não são permitidas. A propriedade de entregar para todos-ou-nenhum é chamado atomicidade ou *brodcast* atômico.

A atomicidade é desejável tendo em vista que facilita a programação dos sistemas distribuídos. Assim, quando um processo envia uma mensagem para o grupo, ele não tem que se preocupar sobre o que fazer se algum deles não receber.

Sabe-se que implementar *broadcast* atômico não é uma tarefa simples como parece. O único modo de estar certo que cada destino recebeu a mensagem é requisitando-o a enviar de volta um reconhecimento sobre a mensagem recebida. Deve-se contar que nenhuma máquina nunca irá quebrar, para que este método funcione.

Entretanto, muitos sistemas distribuídos suportam tolerância à falha. Para eles, é essencial que a atomicidade seja assegurada, mesmo na presença de falhas de máquinas (TANEMBAUM, 1995).

2.2.7 Ordenação das Mensagens

Para tornar a comunicação em grupo fácil de usar e entender, duas propriedades são requeridas. A primeira é o *broadcast* atômico. Ela assegura que uma mensagem enviada para o grupo chegue a todos os membros ou a nenhum deles. A segunda propriedade diz respeito à ordem das mensagens.

A melhor garantia seria ter todas as mensagens entregues, e na ordem em que foram enviadas. O sistema deve, pelo menos, assegurar que as mensagens cheguem a todos os membros do grupo na mesma ordem, mesmo que a ordem não seja a ordem real em que eles foram enviados (TANENBAUM, 1995).

2.2.8 Sobreposição de Grupos

Um processo pode ser membro de múltiplos grupos ao mesmo tempo. Este fato pode trazer um novo tipo de inconsistência. Observa-se que o problema aqui é que, embora, haja uma ordenação de tempo global com cada grupo, não há, necessariamente, alguma coordenação entre múltiplos grupos, e isto pode afetar a ordem da entrega das mensagens. Alguns sistemas suportam ordenação de tempo bem-definido entre grupos sobrepostos e já outros não. Implementar a ordenação de tempo entre diferentes grupos é freqüentemente difícil de se fazer (TANEMBAUM, 1995).

2.2.9 Escalabilidade

Outra característica do projeto a ser analisada é a escalabilidade. Muitos algoritmos trabalham considerando grupos contendo poucos membros, mas o que deve ser feito quando há dezenas, centenas, ou milhares de membros no grupo é a observação das características, tais como: se há somente uma rede local ou várias redes locais interligas por *gateways*, ou se é uma rede transcontinental. A presença de *gateways* pode afetar as características de implementação. Para começar, o *multicasting* torna-se mais complicado (TANENBAUM, 1995).

2.3 Tecnologia versus Interações Sociais

Groupware é significativamente mais difícil de implementar do que o software tradicional. O desenvolvimento de sistemas *groupware* envolve características técnicas da área de desenvolvimento de sistemas distribuídos, como, por exemplo, a replicação, a consistência, a concorrência e os protocolos de comunicação (HOFTE, 1998).

Os testes dos usuários também são freqüentemente mais difíceis do que com sistemas monousuário, pelas seguintes razões (BRINCK, 2005):

- Organizar e programar para grupos é mais difícil do que para indivíduos;
- O estilo da interação que ocorrerá entre os membros do grupo é difícil de selecionar previamente;
- Os grupos pré-estabelecidos variam no estilo da interação, e a distância onde estão os membros de um grupo, afeta seus padrões de comunicação;
- Os grupos novos mudam rapidamente durante o processo da formação do grupo;
- Os grupos são dinâmicos; os papéis mudam.

A comunicação entre pessoas é, geralmente, altamente estruturada. Quando alguém faz uma pergunta, espera-se uma resposta ou um pedido de esclarecimento. Depois do pedido, uma resposta típica deve cumprir o pedido ou especificar uma razão para não cumpri-lo. A maioria das ações tem uma escala conhecida de respostas e as pessoas para assegurá-las, ou seja, uma comunicação é realizada a partir de uma estrutura.

Quando o sistema determina exatamente como a conversação está estruturada, esta é conhecida como uma estrutura de comunicação mediada tecnologicamente. Nota-se que a alternativa é uma comunicação mediada socialmente. Quando alguém quer fazer um pedido, emite, por exemplo, uma mensagem para o e-mail de outra pessoa e essa pessoa decide se irá responder, como fará a resposta e a quem responderá (BRINCK, 2005).

CAPÍTULO 3

TRABALHOS RELACIONADOS

Atualmente, muitos projetos têm desenvolvido aplicações de ambientes virtuais colaborativos ou cooperativos. Este capítulo relaciona três destes ambientes: DIVE, HLA e AVOCADO.

3.1 DIVE

O DIVE (*Distributed Interactive Virtual Environment*) é uma plataforma de software para AVDs multi-usuários, que tem sido usada como ferramenta para a criação de várias aplicações. Foi desenvolvido pelo *Swedish Institute of Computer Science* (SICS) e fornece um ambiente com dados compartilhados distribuídos, com atualização ponto-a-ponto. Cada participante possui uma réplica do mundo compartilhado – nas primeiras versões esta réplica era total, nas versões mais recentes, cada participante tem uma réplica parcial do mundo. As mudanças são propagadas aos outros participantes por meio de protocolo *multicast* confiável.

Observa-se que a base de dados é particionada em mundos, em que cada mundo representa um conjunto de objetos e parâmetros específicos, completamente distintos de outros mundos e, ainda, cada mundo está associado a um endereço *multicast* e, assim, o participante só pode estar em um mundo de cada vez, embora possa mudar de mundo

dinamicamente.

Quanto ao problema de modificações concorrentes de objetos, este é solucionado com um algoritmo de passagem de *token* (objeto de dados estruturados ou mensagem que regula o direito de acesso do objeto), de modo que se algum participante quiser transformar um objeto que está sendo modificado, ele fica bloqueado até receber o *token*. Assim, os participantes podem se comunicar por textos e áudios.

Quando um novo participante deseja entrar no mundo, ele requisita o endereço *multicast* daquele mundo e recebe réplica atualizada do mundo, vinda do participante que estiver mais perto.

A questão da percepção dos usuários é tratada no DIVE. Estes podem ser representados de várias maneiras, desde formas mais simples, que apenas informam a presença, a localização e a orientação, até formas mais complexas, que mapeiam uma foto estática do usuário na cabeça do avatar. A atividade do usuário também é identificada por meio de uma linha que liga o avatar ao ponto de manipulação e, também, pela mudança de cor dos objetos manipulados. A Figura 3.1 mostra o ambiente DIVE.

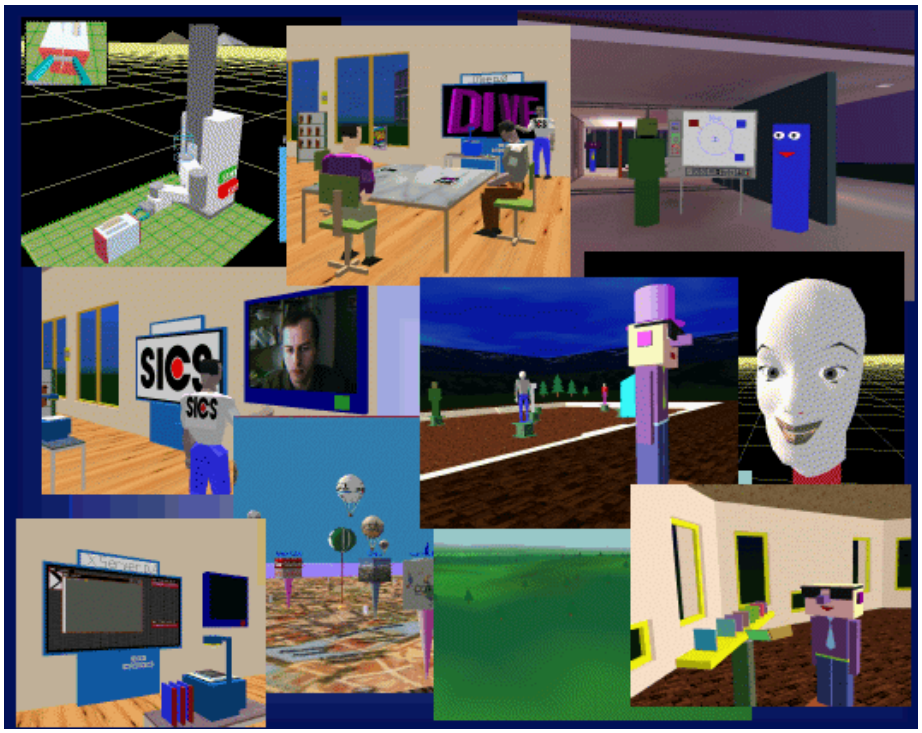

Figura 3.1: Ambiente DIVE (DIVE, 2005).

O DIVE suporta muitos usuários simultaneamente, em que cada usuário pode ter uma configuração diferente de hardware. Dessa maneira, cada usuário controla sua visão

individualmente e pode navegar livremente através do ambiente. A incorporação única permite ao participante ter a percepção da presença de cada um dos outros e suas ações. Esta incorporação pode incluir uma simples figura como um bloco, ou um modelo mais complexo, como, por exemplo, um corpo humano. Já, as representações mais simples, como um bloco são menos expressivos, limitando-se a mostrar a presença, a posição e a orientação. Enquanto, uma figura humanóide, além desses atributos, pode-se também rastrear a direção da cabeça, indicando a visão certa, e os membros podem representar gestos. Além disso, um participante de videoconferência pode se unir ao mundo através de uma janela de vídeo, isto mostra a identidade e a expressão facial, mas não a posição e a orientação, o ponto de vista e o ponto de ação.

Quanto aos objetos e às aplicações no ambiente DIVE, estes podem usar sinais para controlar um comportamento para especificação de um número de ações para serem executadas quando certos eventos ocorrem. Visualiza-se que um comportamento do objeto é definido como uma simples máquina de estado finito, consistindo de um conjunto de estados e arcos direcionados. Cada arco especifica para qual sinal ou evento a transição é permitida, e qual operação executará quando isto ocorrer. Este tipo de operação inclui translação, rotação, generalização de som e mudança de propriedades dos objetos, como cor ou giro. A máquina de estado é descrita em um arquivo de dados DIVE, que é lido pelo sistema quando o objeto está para ser inserido no sistema. Uma vez que o objeto existe, seu comportamento pode ser mudado e manipulado dinamicamente, assim como qualquer outra propriedade do objeto (DIVE, 2005).

3.2 HLA

High Level Architecture (HLA) foi definida pelo *Defense and Modeling Simulation Office* (DMSO), dos Estados Unidos, como um padrão de interoperabilidade e, hoje, também é um padrão *Institute of Electrical and Eletronics Engineers* (IEEE). Uma das razões que contribuíram para a sua criação é que não é possível antecipar todas as aplicações de um modelo de simulação, assim como, as maneiras de combinar o modelo desenvolvido com outras simulações no futuro.

Quando é necessário integrar modelos que foram desenvolvidos em plataformas ou

linguagens de programação diferentes, nota-se que esforços devem ser realizados para construir protocolos comuns, pelo qual os modelos podem então se comunicar. Com o padrão HLA isso não ocorre, pois a especificação do padrão define serviços e interfaces que devem ser usadas no desenvolvimento dos modelos. HLA busca a alta reutilização de modelos e componentes de modelos de simulação.

Alguns conceitos importantes sobre a arquitetura são descritos em (DAHMANN; FUJIMOTO; WEATHERLY, 1997). São eles:

- Federação: conjunto de simulações com o mesmo FOM (*Federation Object Model*);
- Federado: o membro da federação representa um ponto de conexão com a infra-estrutura. Pode ser um modelo simples (automóvel), como, também, agregar uma simulação maior (controle de tráfego aéreo);
- SOM (*Simulation Object Model*): descreve o que cada federado pode produzir ou consumir;
- FOM (*Federation Object Model*): define o que existe de comum nos federados (para ser usado na simulação).

Verifica-se que o HLA é uma arquitetura voltada para a reutilização e interoperabilidade de simulações, constituída basicamente de três partes:

Regras da HLA: define princípios e convenções que devem ser seguidos pelos federados e pelas federações para que se tenha uma interação adequada;

Object Model Template (OMT): descreve as entidades que serão simuladas e as interações entre estas entidades na federação. O OMT é um meta-modelo para SOMs e FOMs;

Especificação da interface: trata-se da especificação da interface entre a RTI (*Runtime Intrastructure*) e os federados.

Dentro do processo de especificação da interface RTI são descritos o conjunto de classes de serviços que são fornecidos para os federados pela RTI. Percebe-se que dentre os serviços existentes tem-se: (1) Gerenciamento de Federação: funcionalidades básicas necessárias para criar e operar a federação; (2) Gerenciamento de Objetos: criação, destruição e identificação de objetos; (3) Gerenciamento de Tempo: sincronização e troca de dados da simulação em execução; (4) Gerenciamento de Declaração: meios de o federado declarar que informações ele vai exportar (publicar), permitindo que outros federados tenham acesso. Além disso, é aqui que é realizada a inscrição do federado para atualizações e interações produzidas por outras simulações; (5) Gerenciamento de Propriedade: transferir

dinamicamente os direitos de propriedades de objetos/atributos durante a execução da simulação. Este serviço é muito importante, pois, apenas o federado com a identificação de "propriedade" de atributos e/ou objetos tem o direito de modificar ou atualizar valores de um atributo em particular; (6) Gerenciamento de Distribuição de Dados: roteamento eficiente dos dados entre os federados durante o processo de execução da federação.

O processo de desenvolvimento do modelo de objetos envolve, em sua grande parte, a execução de processo de documentação. O ambiente de simulação pode gerar esta documentação, o que facilita o trabalho do usuário e desta forma simplifica o processo, já que, de outra forma, o usuário teria que além de definir o modelo no ambiente, também utilizar outras ferramentas para realizar o processo de documentação de informações que em sua totalidade podem estar agregadas ao modelo.

Os documentos necessários para a geração do modelo de objetos da simulação (SOM) são os seguintes: tabela com a estrutura de objetos, tabela de interação entre objetos, tabela de parâmetros, tabela de atributos, tabela de tipos de dados enumerados, tabela de tipos de dados complexos, tabela léxica e tabela de roteamento usada no gerenciamento da distribuição dos dados.

No processo de adaptação do ambiente para a RTI, têm-se três questões principais:

- Sincronização - Na abordagem HLA, a RTI provê facilidades para que um federado não precise conhecer o esquema de gerenciamento de tempo de outros federados;

- Representação de dados - De acordo com a representação de dados, unidades, formatos dependentes de plataforma de execução, formatos de *strings* e traduções de SOM e FOM são questões importantes de serem consideradas;

- Troca de dados - É importante que a implementação consista de funções para aumentar a interoperabilidade e simplificar a interface. Estas são questões importantes quando se implementa as facilidades para atualizar atributos e enviar/receber interações de outros federados (DAHMANN; FUJIMOTO; WEATHERLY, 1997).

3.3 AVOCADO

O Avocado foi criado pelo instituto IMK/*Fraunhofer* e consiste de um *framework*

para o desenvolvimento de AVDs interativos. Usa-se a linguagem de programação C++ para definir duas categorias de classes de objetos. A primeira diz respeito aos nos, responsáveis por fornecem uma API de grafo de cena orientada ao objeto e permitem a representação e renderização de geometrias complexas. Já os sensores fornecem ao Avocado sua interface para o mundo real, e eles são usados para importar dispositivos externos de dados em uma aplicação. A Figura 3.2 mostra a estrutura do Avocado.

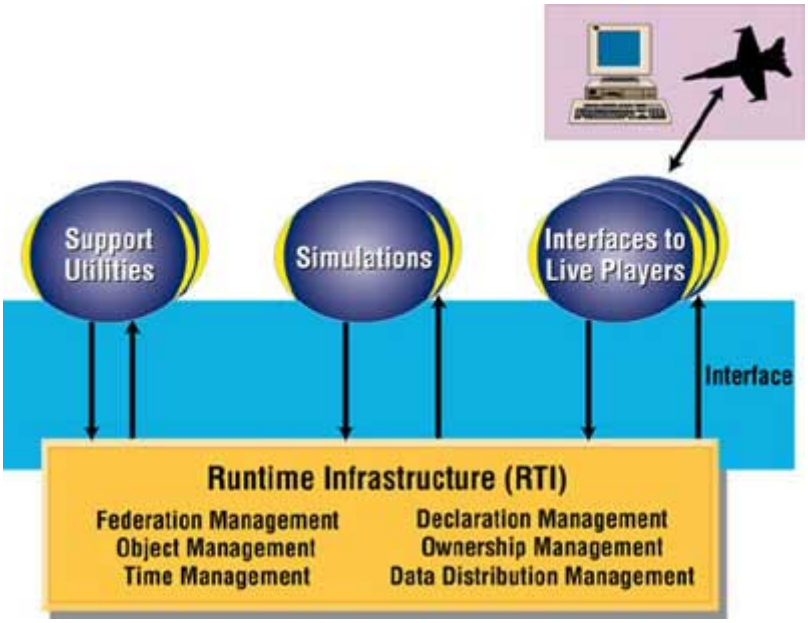

Figura 3.2: Visão funcional de um HLA *Federation* ((DAHMANN; FUJIMOTO; WEATHERLY, 1997).

Todos os objetos do Avocado são *fieldconteiners*, representando informações do estado do objeto com uma coleção de campos. Eles suportam interface *streaming* genérica, que permite objetos e suas informações de estado serem escritas em um *stream*, e a subseqüente reconstrução do objeto daquele *stream*. Esta interface é um dos blocos básico de construção, usado para a implementação de distribuição de objetos.

Observa-se que o Avocado usa conexão entre campos para construir grafo de fluxo de dados que é conceitualmente ortogonal ao grafo de cena e, também, é usado para especificar o relacionamento ortogonal entre nos, que não podem ser expressos em termos de grafo de cena. Dessa forma, há a facilitação da implementação do comportamento de interação e a importação de dados do mundo real no grafo de cena.

Em adição a API C++, verifica-se que o Avocado caracteriza uma linguagem

completa construída para a linguagem interpretada *Scheme*. *Scheme,* que se trata de uma linguagem de programação de propósito geral descendente do Algol e Lisp. Essa linguagem é de alto nível e, ainda, suporta operações em estruturas de dados como *strings*, listas e vetores. Todos os objetos de alto nível do Avocado podem ser criados e manipulados pelo *Scheme*.

O Avocado funciona como um grafo de cena distribuído de forma semitransparente, e provê formas básicas de interação com o mundo virtual (clicar, mover objetos, etc.). Percebe-se que tratar as formas de interação com o ambiente virtual faz com que o Avocado envolva restrições sérias, como, padronizar a representação do mundo virtual, porém não faz sentido para todas as aplicações.

O próprio Avocado é baseado no SGI *Performer* para alcançar o máximo possível de performance para uma aplicação e habilitar necessidades especiais envolvidas no desenvolvimento de aplicações para ambientes virtuais. Além da alta performance de renderização este *framework* permite recursos de vídeo tanto como animações pré-fabricadas para ser importadas no mundo virtual. As tarefas de renderizações avançadas, como, nível de detalhe e comunicação com hardware gráfico, são habilitados pelo *Performer* (TRAMBEREND, 2005; NIKITINA; NIKITIN, 2005).

CAPÍTULO 4

DESCRIÇÃO DA BIBLIOTECA DE COMUNICAÇÃO

Neste capítulo são apresentadas as bibliotecas originais do JVRMol e, também, as implementações que foram acrescentadas para dar suporte a grupos, que se trata do objetivo deste trabalho.

O principal objetivo do JVRMol é permitir que pesquisadores geograficamente dispersos possam, em tempo real, cooperar no estudo de uma determinada molécula de proteína visualizada de forma tridimensional. Também é um objetivo pelo fato de existirem usuários remotos e com possibilidade de estarem usando arquiteturas diferentes (tanto de *hardware* quanto de *software*) e, ainda, que o sistema seja portável e heterogêneo (RODELLO, 2003).

4.1 Arquitetura

O JVRMol foi implementado utilizando a arquitetura cliente/servidor para ser suporte à comunicação e um modelo de armazenamento de banco de dados replicados de mundos heterogêneos. A linguagem utilizada para implementação foi Java com suas extensões Java 3D (JAVA 3D, 2003), como interface gráfica 3D, e o Java RMI (JAVA RMI, 2003) para realizar e gerenciar a troca de mensagens. Quanto às estruturas das moléculas, estas são montadas a partir de informações extraídas dos registros do arquivo do PDB

(Protein Data Bank) (PDB, 2004) e visualizadas no modelo esfera CPK (*Corey-Pauling-Kultin*). A arquitetura do JVRMol é visualizada na Figura 4.1.

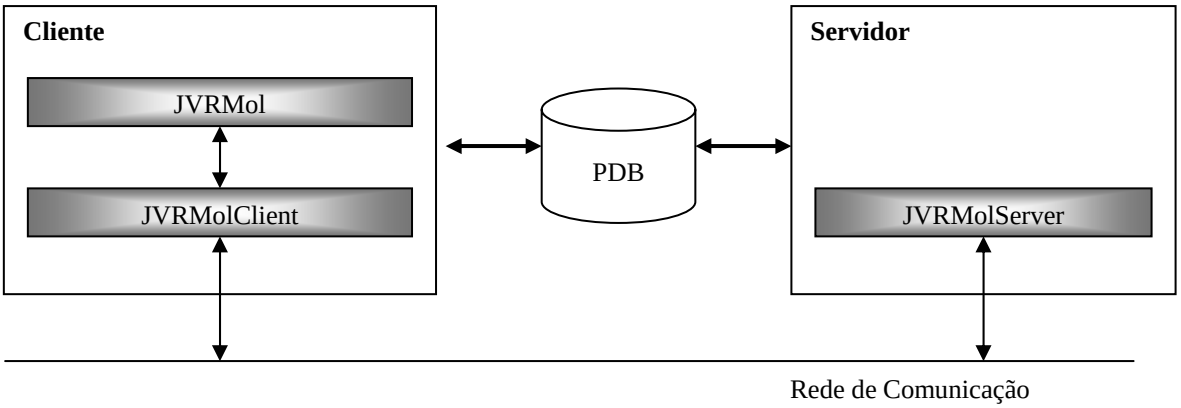

Figura 4.1: Arquitetura do JVRMol (RODELLO, 2005).

O JVRMol é composto por três classes principais: JVRMol, JVRMol*Client* e JVRMol*Server* (Figura 4.2.).

A classe JVRMol fornece uma interface para a visualização da molécula, implementada com Java 3D (JAVA 3D, 2003) e oferece suporte aos meios de interação como: *mouse*, *joystick*, luvas (*gloves*), óculos estereoscópicos *(stereo glasses*) e capacetes (HMD – *Head Mounted Display*).

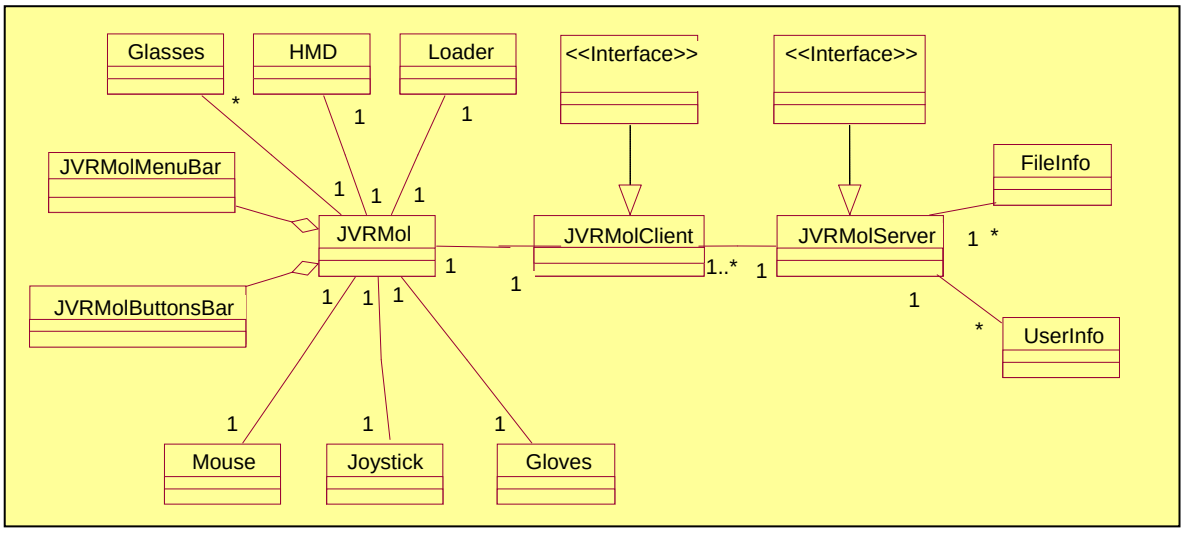

Figura 4.2: Diagrama de classes do JVRMol (RODELLO, 2003).

A classe *JVRMolClient* implementa a interface *Client* e é a responsável por realizar e gerenciar a troca de mensagens com o módulo servidor. A *JVRMolClient* foi implementada

com suporte em Java RMI.

Observa-se que os eventos gerados pelo usuário na interface JVRMol são capturados pelo módulo *JVRMolClient* e transferidos para o módulo servidor representado pela classe *JVRMolServer*, que transfere o evento gerado para todos os participantes.

Por fim, a classe *JVRMolServer* implementa a interface *Server*, responsável por gerenciar, além da troca de mensagens, também, a entrada e a saída dos usuários do sistema. Toda e qualquer ação realizada pelo usuário passa pelo servidor que reflete a mesma para todos os participantes.

Ressalta-se que o carregamento da molécula de proteína é feito como por meio da classe *loader*, para a conversão das coordenadas em um tipo de *scene* da biblioteca Java 3D. Estas coordenadas são obtidas no respectivo PDB da molécula.

Já as classes *FileInfo* e *UserInfo* contêm atributos para auxiliar à classe VRMol a controlar os arquivos e usuários.

4.2 Considerações do Projeto Original

Como pode ser observado na Figura 4.3, a arquitetura JVRMol é composta por três módulos: dois do lado do cliente e um do lado do servidor.

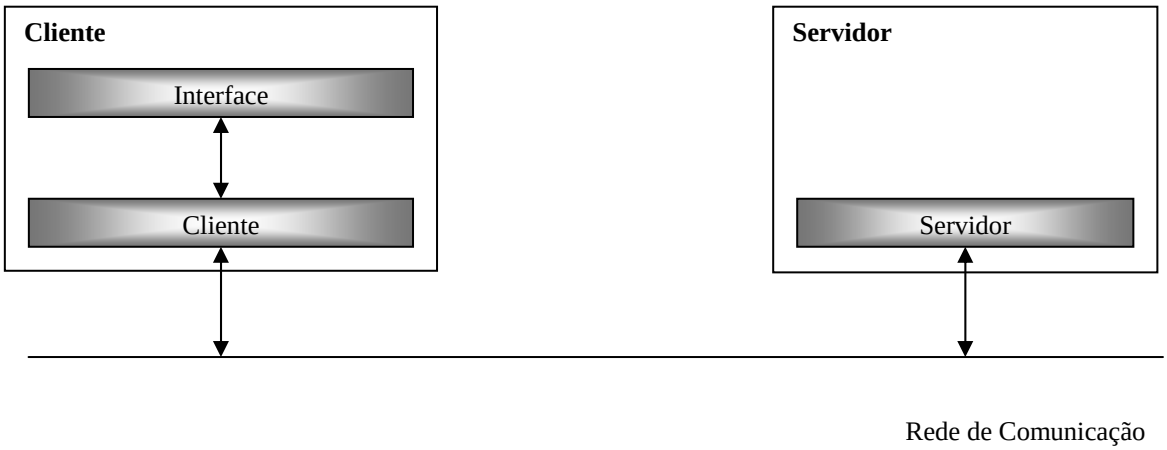

Figura 4.3: Módulos do JVRMol (RODELLO, 2005).

O lado do cliente é composto pelos módulos *Interface* e *Cliente* que se relacionam. O

módulo *Interface* representa uma classe que fornece a interface gráfica para a visualização tridimensional, enquanto o módulo *Cliente* realiza e gerencia a troca de mensagens com o módulo *Servidor*, com suporte para Java RMI.

A adoção de módulos independentes para interface e para cliente justifica-se pela necessidade de renderização. Com a separação, a troca de mensagens teoricamente não influenciará na renderização da cena pela interface.

4.3 Descrição das Primitivas

As descrições das bibliotecas do Projeto Original foram transcritas do trabalho de RODELLO (2005).

Verifica-se que a biblioteca foi desenvolvida em Java RMI e fornece base para troca de mensagens que, na realidade, ocorre por invocação de métodos das classes que representam os clientes e o servidor. Esses métodos estão divididos em três tipos: métodos para conexão/desconexão, métodos para troca de mensagens textuais e métodos para manipulação de controle de objetos.

Na Tabela 4.1 são apresentadas as primitivas. Procurou-se estabelecer um pequeno conjunto de primitivas que pudessem satisfazer a grande maioria dos requisitos necessários de um AVD. No Java RMI, esses métodos compõem uma interface remota que deverá ser utilizada na implementação dos clientes e do servidor.

Os métodos para conexão/desconexão abrangem aqueles responsáveis por estabelecer e terminar as conexões entre os clientes e o servidor. Correspondem às primitivas *login* e *logout* no lado servidor e à *ReceiveEnter* e *ReceiveExit* no lado Cliente.

Os métodos para manipulação e controle de objetos compreendem as primitivas que tratam do carregamento e remoção de arquivos (objetos virtuais) e da difusão de interações. As primitivas desse tipo são do lado servidor: *StoreOpenedFile, removeObject, requestFile, broadCoordinates multCoordinates*; do lado cliente: *receiveFile, receiveRemove, requestFile, receiveCoordinates* e *receiveMCoordinates*.

Por fim, os métodos para troca de mensagens textuais são os que transmitem as mensagens textuais entre os integrantes do sistema. São as primitivas o *chat* (servidor) e o *receiveChat* (cliente).

Tabela 4.1: Primitivas de comunicação.

Métodos para conexão/desconexão	
Servidor	**Cliente**
login(String name, Chatter chatter)	receiveEnter(String name, Chatter chatter)
logout(String name)	receiveExit(String name)
Métodos para troca de mensagens textuais	
Servidor	**Cliente**
chat(String name, String message)	receiveChat(String name, String message)
Métodos para manipulação e controle de objetos	
Servidor	**Cliente**
StoreOpenedFile(String file, String textString)	ReceiveFile(String file, String textString)
RemoveObject(String file, String textString)	ReceiveRemove(String file, String textString)
requestFile(String name, String filename)	RequestFile(String file)
broadCoordinates(double Trans[], double Rotat[])	ReceiveCoordinates(double Trans[], double Rotat[])

4.3.1. Métodos para conexão e desconexão

Estes métodos abrangem aqueles responsáveis por estabelecer e terminar as conexões entre os clientes e o servidor. Os métodos reúnem-se em duas operações: *login* e *logout*.

A operação de *login* cuida da inclusão de um novo participante no ambiente. Quando o módulo servidor recebe uma invocação do método *login*, terá o nome do participante como parâmetro. Tal informação é armazenada como um objeto da classe *UserInfo*, que armazena o nome e uma referência para a troca de mensagens textuais. Em seguida, o servidor avisa todos os outros participantes da entrada de um novo integrante do grupo.

A Figura 4.4 mostra a seqüência de ativações para *login*. Os métodos utilizados são (Tabela 4.2):

Tabela 4.2: Métodos para Login.

Primitiva	Descrição
Init()	Estimula o módulo Cliente a conectar-se com o módulo Servidor.
login(name)	Solicita ao módulo servidor, a inclusão de um novo participante (name).
receiveEnter(name)	Propaga a entrada de um novo participante.
add()	Recebimento da adesão de um novo participante.
receiveFile(filename, file)	Recebimento do arquivo solicitado.
requestFile(filename, file)	Requisita o arquivo aberto para ser carregado.
loadFile(filename)	Arquivo é carregado na interface.

Percebe-se que, individualmente, o novo integrante recebe uma notificação pela qual solicita todos os arquivos que já estavam abertos antes da sua entrada.

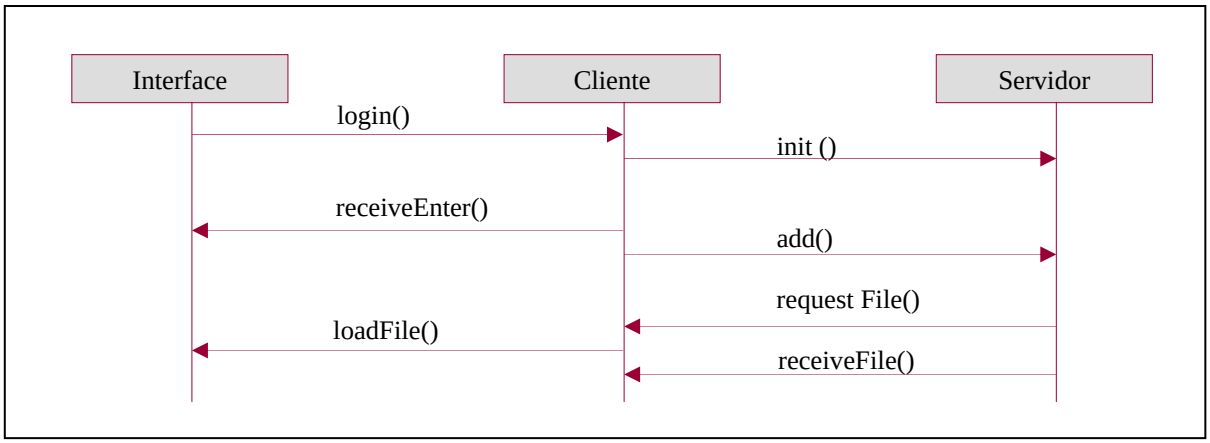

Figura 4.4: Diagrama de seqüência de mensagem: login.

Para cada participante nota-se que o servidor cria um objeto *UserInfo* com a finalidade de armazenar todos os participantes. Individualmente cada cliente também mantém um objeto *UserInfo*.

No método add(), caso existam arquivos abertos, ou seja, objetos na cena, o novo cliente é estimulado a receber os mesmos para que não haja inconsistências no conteúdo do ambiente em questão. Os métodos com essa finalidade são:

A operação *logout*, por sua vez, trata da saída de um participante, eliminando o respectivo objeto criado (com base na classe *UserInfo)* para que mensagens não sejam enviadas para um destino que não exista mais. A Figura 4.5 mostra a seqüência de mensagens da operação *logout*, cujos métodos são (Tabela 4.3):

Tabela 4.4: Métodos para *Logout*

Primitiva	**Descrição**
disconnect()	Estimula o módulo Cliente a desconectar-se do módulo Servidor.
logout(name)	Solicita ao módulo Servidor a exclusão do participante. A referência ao objeto *UserInfo* que representava o participante é excluída.
receiveExit(name)	Propaga a saída de um participante.
remove()	Realiza a remoção do participante localmente.

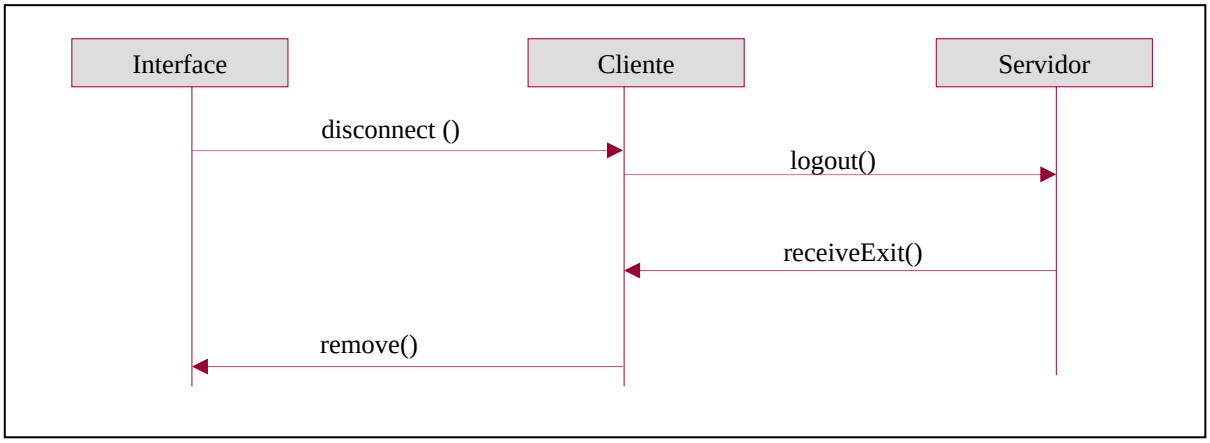

Figura 4.5: Diagrama de seqüência de mensagem: logout.

4.3.2. Métodos para manipulação e controle de objetos

Englobam todas as mensagens utilizadas para gerenciar os objetos. Tratam das operações de carregamento de um arquivo, remoção de objetos do ambiente, transferência de coordenadas (translação, rotação e escala). Os métodos são: *StoreOpenedFile*, *removeObject*, *requestFile*, *broadCoordinates* e *multCoordinates*; do lado servidor e *receiveFile*, *removeObject*, *requestFile*, *receiveCoordinates* e R*eceiveMCoordinates*, do lado cliente.

Quando algum arquivo é carregado por algum participante, ele é enviado para o servidor que instancia um objeto da classe *FileInfo* para armazenar sua referência. Em seguida, comunica a todos os participantes da existência de um novo objeto no ambiente. Assim, todo integrante recebe uma cópia do arquivo. A Figura 4.6 demonstra a seqüência de métodos usada para carregar um arquivo. Os métodos usados são (Tabela 4.4):

Tabela 4.4: Métodos para carregamento de um arquivo de proteína.

Primitiva	Descrição
loadFileToServer(nome do arquivo, arquivo)	Método que ativa o módulo Cliente, comunicando o carregamento de um novo arquivo (objeto virtual).
StoreOpenedFile(nome do arquivo, arquivo)	Solicita ao módulo Servidor o armazenamento do arquivo que foi aberto.
RequestFile(nome do arquivo)	Avisa os outros módulos Clientes do carregamento de um novo objeto.
requestFile(nome do arquivo, arquivo)	Os módulos Clientes solicitam o carregamento do arquivo.
ReceiveFile (nome do arquivo, arquivo)	Módulo Servidor envia o arquivo para os participantes.
loadFile(nome do arquivo)	O arquivo recebido é carregado pela interface tridimensional.

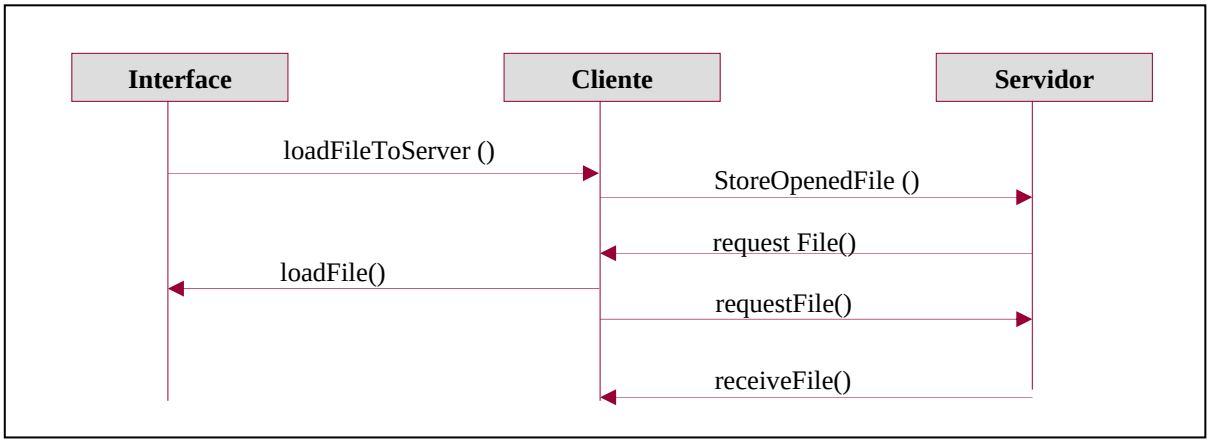

Figura 4. 6: Diagrama de seqüência de mensagens: carregamento de um arquivo.

O objeto *FileInfo* também é responsável por armazenar o posicionamento do objeto e pela sincronização do acesso a um objeto. Para que não haja conflitos entre as ações dos participantes, um mecanismo de travamento (*lock*) é ativado na seleção do objeto, garantindo que somente um dos participantes irá interagir até que o mesmo seja liberado.

A remoção de objetos trata da exclusão dos mesmos do ambiente virtual. Por questões de consistência, quando algum objeto é removido, sua retirada deve ser comunicada a todos os participantes, não mais sendo representado dentro do ambiente. A Figura 4.7 apresenta a seqüência. Os métodos utilizados são (Tabela 4.5):

Tabela 4.5: Métodos para exclusão de um arquivo de proteína

Primitiva	Descrição
removeObject(nome do arquivo, arquivo)	Estimula módulo cliente sobre a exclusão de um determinado objeto do ambiente virtual.
removeObject(nome do arquivo, arquivo)	Comunica o servidor sobre a remoção. A referência ao objeto *FileInfo* é excluída.
receiveObjRemove(nome do arquivo, arquivo)	Estimula os clientes a receberem a mensagem de exclusão proveniente do módulo servidor.
removeObjfromScene(nome do arquivo, arquivo)	Implementa a retirada da representação do objeto do ambiente virtual.

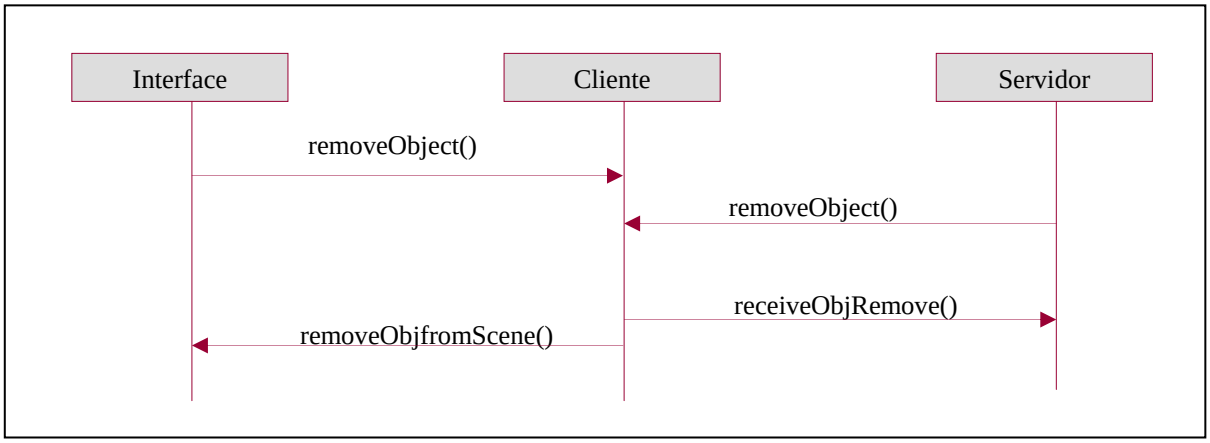

Figura 4.7: Diagrama de seqüência de mensagens: exclusão de objetos.

Os métodos englobados nas transferências de coordenadas tratam do envio das coordenadas do objeto assim que algum evento de modificação, tanto de translação quanto de rotação é executado. Destaca-se que foram projetados métodos para *broadcast* e para *multicast*. A Figura 4.8 indica a seqüência para o *broadcast*. Os métodos utilizados são (Tabela 4.6):

Tabela 4.6: Métodos para b*roadcast*.

Primitiva	**Descrição**
sendTranslation(Trans[], Rotat[])	Método que envia os vetores contendo os valores de translação e rotação para o módulo cliente.
broadCoordinates(Trans[], Rotat[])	Envia os vetores contendo os valores para o servidor armazenar (em *FileInfo*) e difundir as modificações para os participantes.
receiveCoordinates(Trans[], Rotat[])	Estimula os clientes a receberem as modificações provenientes do módulo Servidor.
TranslateObj(Trans[], Rotat[])	Transfere o vetor contendo valores de translação e rotação para serem aplicados no objeto, reposicionando-o.

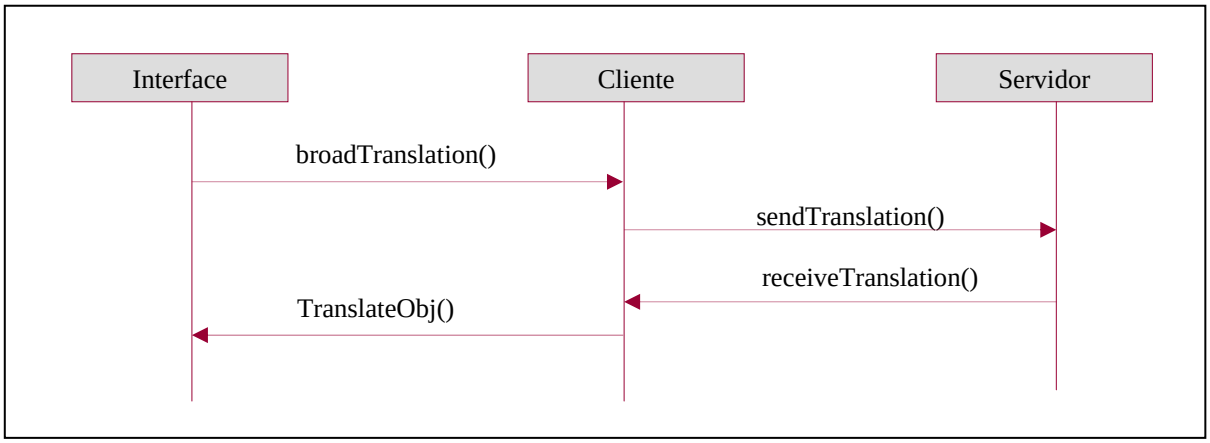

Figura 4.8: Diagrama de seqüência de mensagens: transferência de coordenadas.

No *multicast*, a primitiva utilizada (*multCoordinates*) estimula o servidor para direcionar a mensagem para um determinado grupo. A seqüência adotada é similar àquela descrita na Figura 4.8.

4.3.3. Métodos para troca de mensagens textuais

Corresponde às mensagens para troca de mensagens via *chat* textual. Refere-se aos métodos *Chat*, (lado servidor) e *receiveChat* (lado cliente), originando uma operação denominada *Chat*.

O *chat* é um tipo de operação que tem por objetivo difundir uma mensagem de texto enviada por algum dos participantes. A Figura 4.9 apresenta a seqüência dos métodos utilizados. Eles são (Tabela 4.7):

Tabela 4.7: Métodos para troca de mensagens via *chat* textual.

Primitiva	Descrição
sendmessage(mensagem)	Invoca o módulo Cliente para enviar uma mensagem textual.
Chat(mensagem)	A mensagem é passada ao módulo Servidor para que seja difundida a todos os participantes.
receiveChat(mensagem)	Clientes recebem mensagem do Servidor.
display(mensagem)	Método que exibe a mensagem recebida.

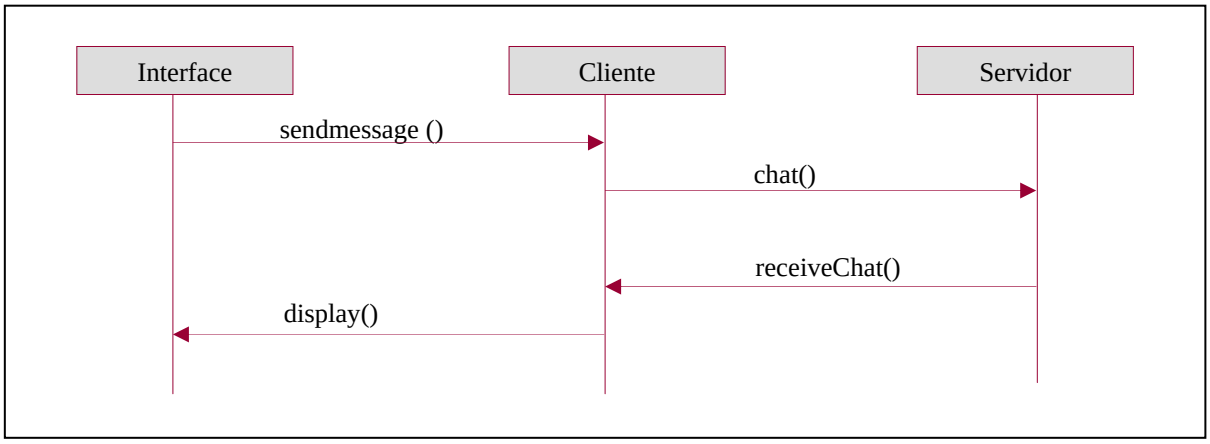

Figura 4.9: Diagrama de seqüência de mensagens: chat.

4.4 Descrição das Primitivas com suporte a grupos

A Tabela 4.8 apresenta as primitivas após extensão para suporte a grupos, e as novas primitivas criadas para o gerenciamento dos grupos.

Os métodos para conexão/desconexão, troca de mensagens textuais, manipulação e controle de objetos, receberam o parâmetro *group*, responsável por identificar a qual grupo o cliente pertence e, assim, somente receber mensagens textuais e de manipulação de objetos de outros elementos do mesmo grupo.

Tabela 4.8: Primitivas de comunicação com suporte para grupos.

Métodos para troca de mensagens textuais	
Servidor	**Cliente**
Chat(String group, String name, String message)	ReceiveChat(String group, String name, String message)
Métodos para manipulação e controle de objetos	
Servidor	**Cliente**
StoreOpenedFile(String group, String file, String textString)	ReceiveFile(String group, String file, String textString)
RemoveObject(String group, String file, String textString)	ReceiveRemove(String group, String file, String textString)
RequestFile(String group, String name, String filename)	RequestFile(String group, String file)
Métodos para gerenciamento de grupos	
Servidor	**Cliente**
CreateGroup(String group, String name, Chatter chatter)	ConnectGroup(String group, String name, Chatter chatter)
CloseGroup(String group, String name)	DisconectGroup(String group, String name)
ListGroup()	ShowGroup()
Join(String group, String name)	EnterToGroup(String group, String name)
Disjoin (String group, String name)	ExitGroup(String group, String name)
ListMembersGroup(String group)	ShowMembersGroup(String group)

Inicialmente pode-se integrar a aplicação sem se conectar a nenhum grupo, o que corresponde a pertencer ao grupo *null*.

4.4.1 Métodos para gerenciamento de grupos

Estes métodos abrangem aqueles responsáveis por criar e excluir grupos conectados ao servidor. Os métodos reúnem-se em duas operações: c*reateGroup* e *closeGroup*.

A operação de *createGroup* é a responsável pela inclusão de um novo grupo no ambiente. Quando o módulo servidor recebe uma invocação do método c*reateGroup*, receberá o nome do grupo como parâmetro e o nome do cliente que o está criando. Tal informação é armazenada como um objeto da classe *GroupInfo*, que armazena o nome do grupo, o nome do cliente que o está criando e uma referência para a troca de mensagens textuais.

A classe *GroupInfo* é utilizada para auxiliar a classe VRMolServer e também, para gerenciar os grupos (Figura 4.10).

```
Public class GroupInfo {
  public String group;
  public Chatter chatter;
  public GroupInfo(String group,Chatter chatter)
  {
     this.group = group;
     this.chatter = chatter;
  }
}
```

Figura 4.10: Classe *GroupInfo*

A Figura 4.11 aponta a seqüência de ativações para *createGroup*. Os métodos utilizados são (Tabela 4.9):

Tabela 4.9: Métodos para criação de grupos.

Primitiva	**Descrição**
GroupList()	Chama o módulo do grupo.
createGroup(group,name,chatter	Solicita ao módulo Servidor, a inclusão de um novo grupo (*group*) e identifica quem é o responsável pela criação do grupo (*name*).
connectGroup(group)	Propaga a criação de um novo grupo
addGroup(group)	adiciona o novo grupo na lista de grupos.

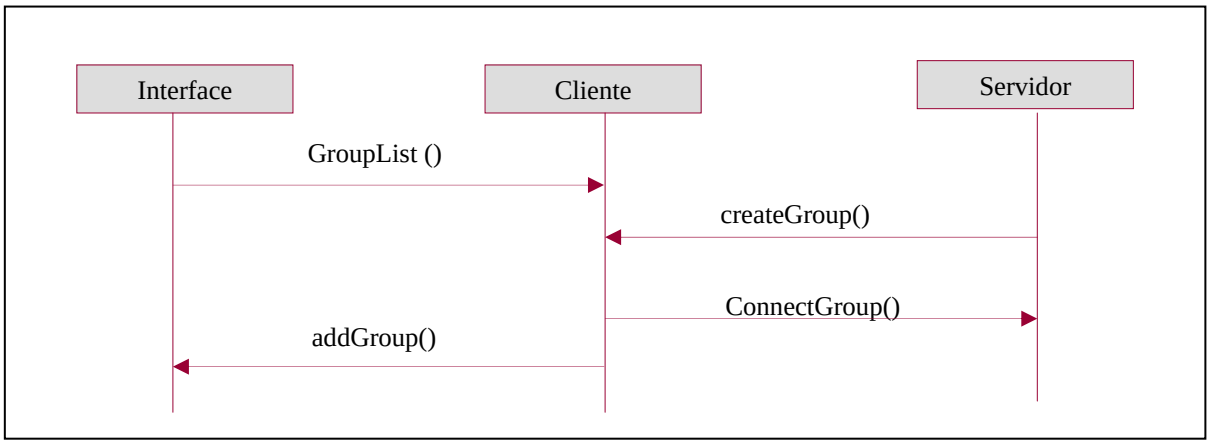

Figura 4.11: Diagrama de seqüência de mensagem: criar grupo.

Verifica-se que para cada grupo, o servidor cria um objeto *GroupInfo* com a finalidade de armazenar todos os grupos.

A operação *closeGroup* trata da exclusão de um grupo e elimina o objeto da classe *GroupInfo*. A Figura 4.12 demonstra a seqüência de mensagens da operação *closeGroup*, cujos métodos são (Tabela 4.10):

Tabela 4.10: Métodos para exclusão de grupos.

Primitiva	**Descrição**
disconnectGroup(Group,name)	Solicita ao módulo Servidor a exclusão de um grupo.
closeGroup(group,name)	Solicita ao módulo Servidor a exclusão do grupo. O parâmetro *name* é utilizado para verificar se o requisitante da exclusão é o mesmo que o criou. Caso seja verdadeiro, a referência ao objeto *GroupInfo* que representava o grupo é excluída. Caso contrário, uma mensagem informará que o participante não tem permissão para excluí-lo.
exitGroup(group)	Propaga a exclusão de um grupo.
removeGroup(group)	Realiza a remoção do grupo da lista de grupos.

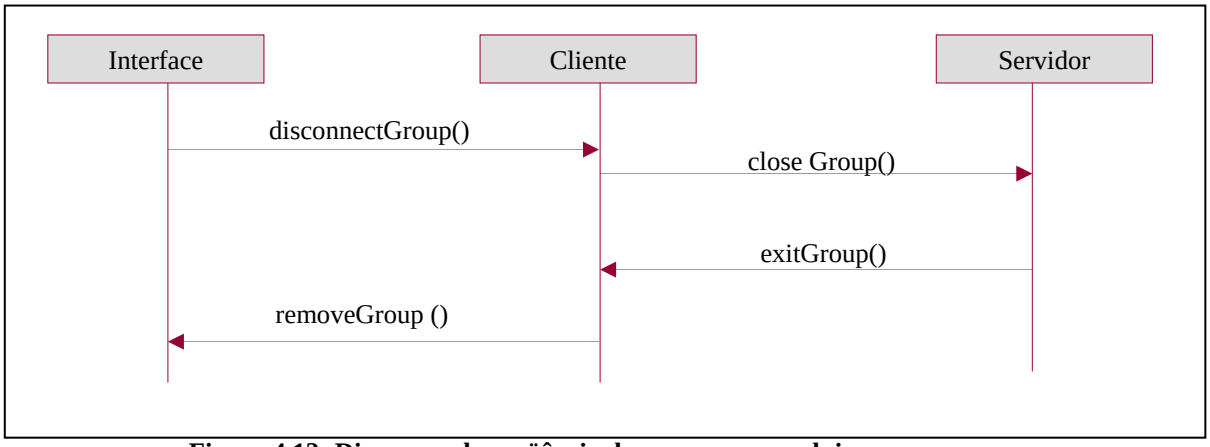

Figura 4.12: Diagrama de seqüência de mensagem: excluir grupo.

Quanto à operação *listGroup*, verifica-se que esta exibe os grupos existentes, com base na classe *GroupInfo*. A Figura 4.13 mostra a seqüência de mensagens da operação *listGroup*, cujos métodos são (Tabela 4.11):

Tabela 4.11: Métodos para visualização dos grupos existentes.

Primitiva	**Descrição**
groupList()	Chama o módulo do grupo.
listGroup()	Solicita ao Servidor a lista de todos os grupos existentes.
showGroup()	Lista para o Cliente os grupos existentes.
showListGroup()	Mostra os grupos existentes.

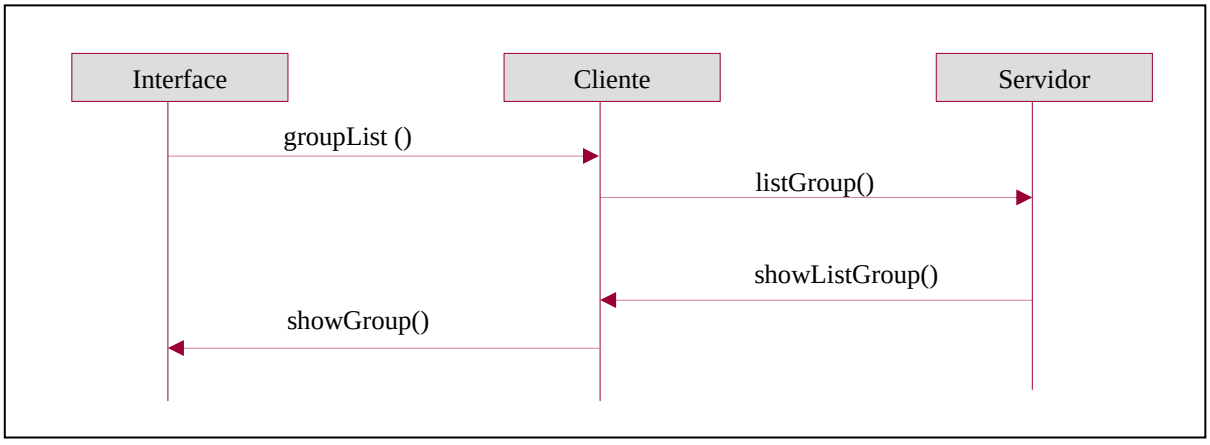

Figura 4.13: Diagrama de seqüência de mensagem: listar grupos.

A operação de *join* é responsável pela inclusão de um novo participante em um grupo existente. Quando o módulo servidor recebe uma invocação do método *join*, recebe o nome do grupo e o nome do participante como parâmetro. Tal informação é armazenada como um objeto da classe *MembersGroupInfo*, que armazena o grupo a qual o participante está se conectando, o nome do participante e uma referência para a troca de mensagens textuais (Figura 4.14). Em seguida, o servidor avisa todos os outros participantes do grupo da entrada de um novo integrante no grupo.

```
Public class MembersGroupInfo {
  public String group;
  public String name:
  public Chatter chatter;
  public MembersGroupInfo (String group, String name, Chatter chatter)
  {
     this.group = group;
     this.name = name;
     this.chatter = chatter;
   }
}
```

Figura 4.14: Classe *MembersGroupInfo*

Nota-se que individualmente o novo integrante recebe uma notificação pela qual solicita todos os arquivos que já estavam abertos pelo grupo antes da sua entrada. Estas mensagens são abordadas adiante. A Figura 4.15 indica a seqüência de ativações para *join*. Os métodos utilizados são (Tabela 4.12):

Tabela 4.12: Métodos para conexão do participante ao grupo.

Primitiva	Descrição
addToGroup()	Chama o módulo *MembersGroup* para poder solicitar a entrada em um grupo.
Join(group,name)	Solicita ao módulo Servidor, a inclusão de um novo participante (*name*) em um determinado grupo (*group*).
ConnectToGroup(group,name)	Propaga a entrada de um novo membro no grupo.
requestFile(group,filename)	Quando se é um novo participante do grupo e já existe algum arquivo sendo tratado pelos outros participantes do grupo, este é requisitado.
receiveFile(group,filename, file)	Recebimento do arquivo do grupo solicitado.
loadFile(group,filename)	Arquivo do grupo é carregado na interface.

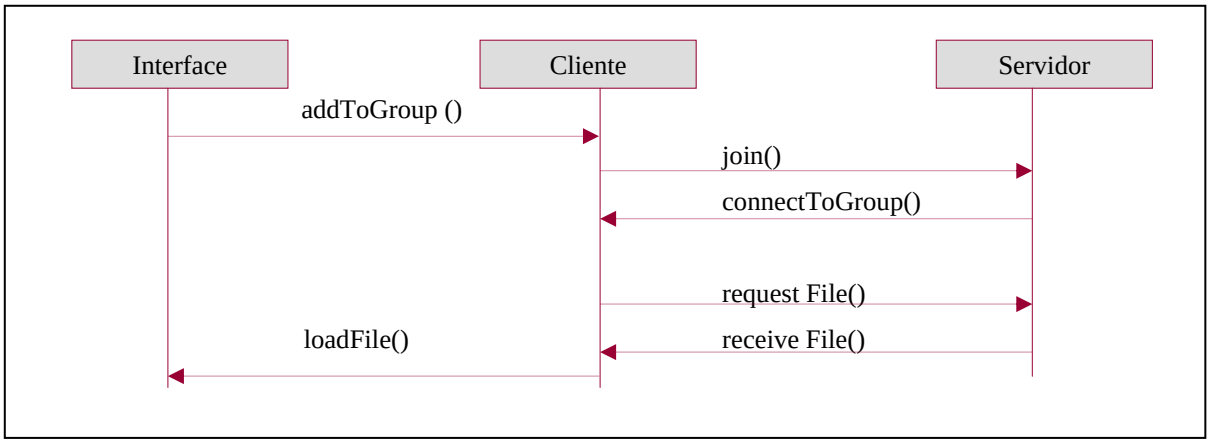

Figura 4.15: Diagrama de seqüência de mensagem: conectar usuário ao grupo

A operação *disjoin*, por sua vez, trata da saída de um participante de um determinado grupo, para que as mensagens do grupo não sejam enviadas para um destino que não exista mais. A Figura 4.16 aponta a seqüência de mensagens da operação *disjoin*, em que os métodos são (Tabela 4.13):

Tabela 4.13: Métodos para saída do participante de um grupo a qual pertence.

Primitiva	Descrição
leaveGroup(group,name)	Solicita ao módulo *MembersGroup* a saída do grupo.
Disjoin(group,name)	Solicita ao módulo Servidor a saída do grupo. A referência ao objeto *MembersGroupInfo* que representava o participante é excluída do grupo.
exitFromGroup(group,name)	Propaga a saída de um participante do grupo.
removeFromGroup(group,name)	Realiza a remoção do participante do grupo localmente.

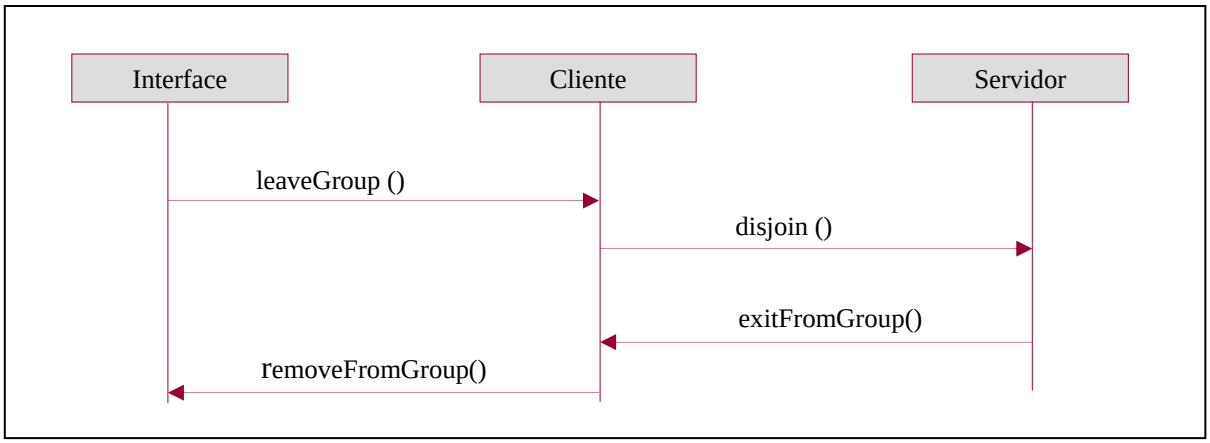

Figura 4.16: Diagrama de seqüência de mensagem: desconectar usuário do grupo

A operação *listMembersGroup* exibe os membros do grupo selecionado, com base na classe *MembersGroupInfo*. A Figura 4.17 sinaliza a seqüência de mensagens da operação *listMembersGroup*, cujos métodos apresentados são (Tabela 4.14):

Tabela 4.14: Métodos para visualizar os membros de um determinado grupo.

Primitiva	**Descrição**
listMembersGroup(group)	Chama o módulo do grupo.
listGroup(group)	Solicita ao Servidor a lista de todos os membros do grupo solicitado.
showGroup(group)	Lista para o Cliente os membros do grupo solicitado.
ShowMembers(group)	Mostra os membros do grupo selecionado.

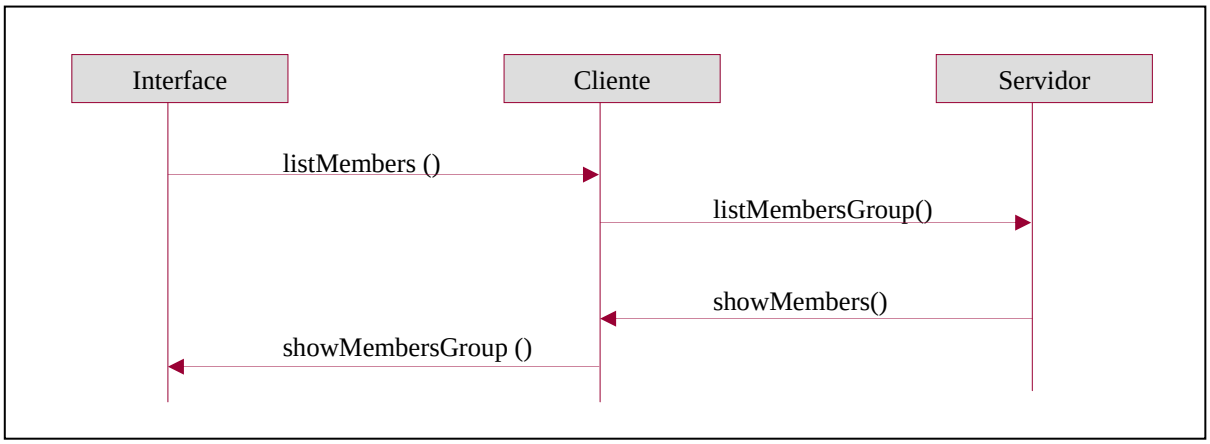

Figura 4.17: Diagrama de seqüência de mensagem: listar membros do grupo.

CAPÍTULO 5

ESTUDO DE CASO

Este capítulo mostra o funcionamento do JVRMol no gerenciamento de grupos. Isto é demonstrado por meio da criação de grupos e posteriormente, da troca de mensagens e alteração das coordenadas da molécula de proteína entre membros do mesmo grupo e os reflexos dessas mudanças nos membros dos grupos.

5.1 Sobre o JVRMol

Para utilizar o JVRMol, o usuário precisa se conectar fazendo o *login*, que permite a inclusão de um novo participante no Ambiente. Para isso, é necessário fornecer a identificação do usuário e o IP do servidor, quando, então, será criada uma instância da classe *UserInfo*, responsável pela difusão das mensagens aos participantes. Figura 5.1.

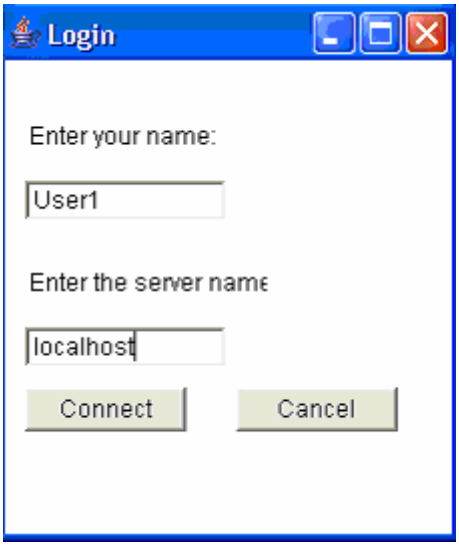

Figura 5.1: Tela de Login

A Figura 5.2 aponta a tela principal do ambiento do JVRMol. Nela se visualiza uma janela por meio da qual o usuário pode se comunicar com os outros participantes na forma de *Chat* textual.

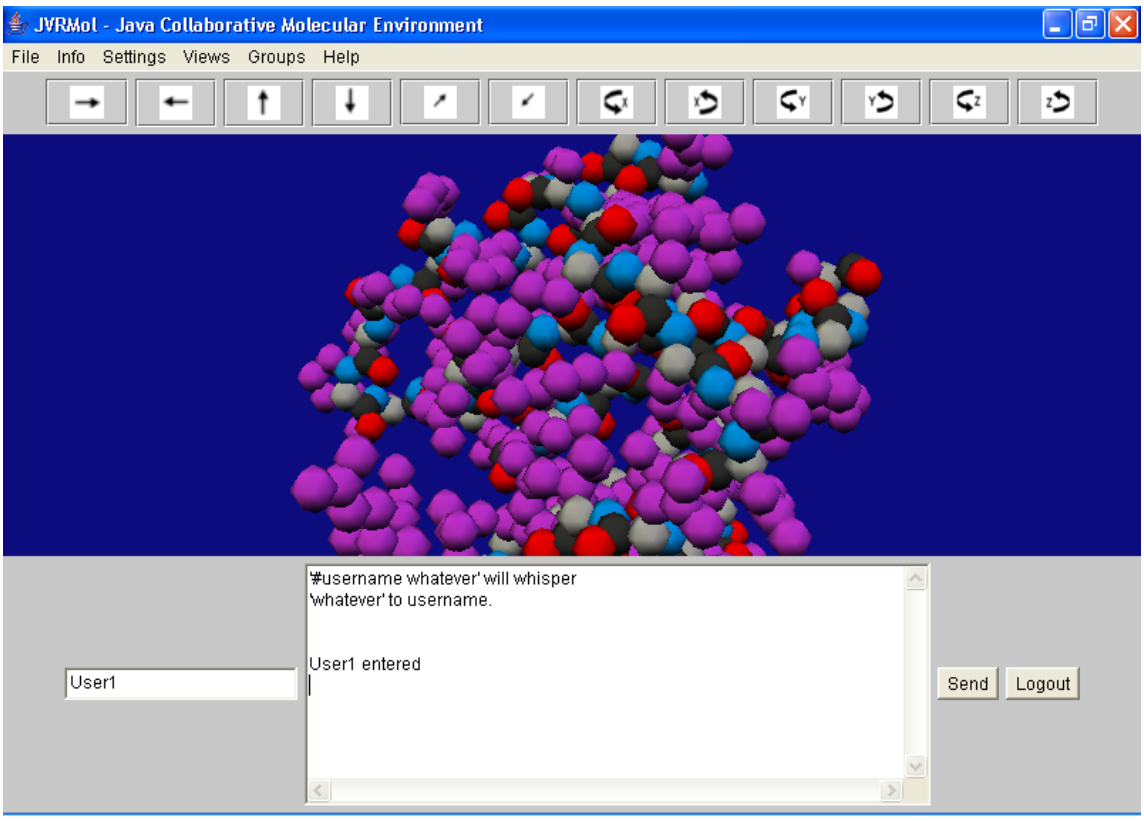

Figura 5.2: Visualização compartilhada de uma molécula de proteína (modelo CPK)

Verifica-se que os integrantes do grupo podem participar de modo interativo compartilhando a mesma visão da molécula e, ainda, trocar idéias e conceitos por meio do recurso da *Chat* textual. A troca de mensagens é feita digitando-se a mensagem na caixa de texto e em seguida clicando-se no botão *Send.* O botão Logout permite a saída de um participante do Ambiente, eliminando-o da classe UserInfo, para que as mensagens não sejam enviadas para um destino inexistente.

A Figura 5.2 demonstra uma molécula de proteína carregada a partir de informações obtidas do arquivo PDB (PDB, 2004). Observa-se que quando um participante seleciona a molécula, o mecanismo de travamento (*lock*) é ativado para garantir a integridade de que terá acesso exclusivo, até que o mesmo a libere.

Para cada novo arquivo PDB aberto é criada uma instância da classe *FileInfo.* Essa classe contém os campos Rotat[] e Trans[], que armazenam as coordenadas de rotação e de translação da molécula, o que auxilia na difusão das alterações feitas na molécula de proteína para todos os participantes. O campo l*ocked* garante que um único participante interaja com a molécula de proteína em um dado instante. Quando um participante inicia uma interação com a molécula, seu valor é alterado para 1 e nenhum participante poderá interagir com esta molécula até que seu valor assuma o valor 0 novamente (RODELLO, 2003).

Observando a barra de menus, identificamos o menu *file,* que contém as opções para abrir e fechar arquivos de moléculas de proteínas, além da opção para fechar o Ambiente.

Quanto ao menu *info,* esse fornece informações sobre a molécula, como, por exemplo: nome da molécula, autor, informações sobre publicações, e, também, os comentários contidos nos campos *remark* de um registro PDB (RODELLO, 2003).

Já o menu *settings* é responsável pelas configurações dos dispositivos de entrada para interação com o Ambiente, tais como: *mouse*, *joystick*, luvas (*gloves*), óculos estereoscópios e capacetes (HMD).

O menu *view* permite a opção para a escolha da abertura de uma visão local de uma molécula de proteína, como mostra a Figura 5.3. Esta opção permite que o participante possa abrir uma visão não compartilhada entre o grupo, permitindo sua análise individual, sem que isto afete ou prejudique o trabalho simultâneo em grupo.

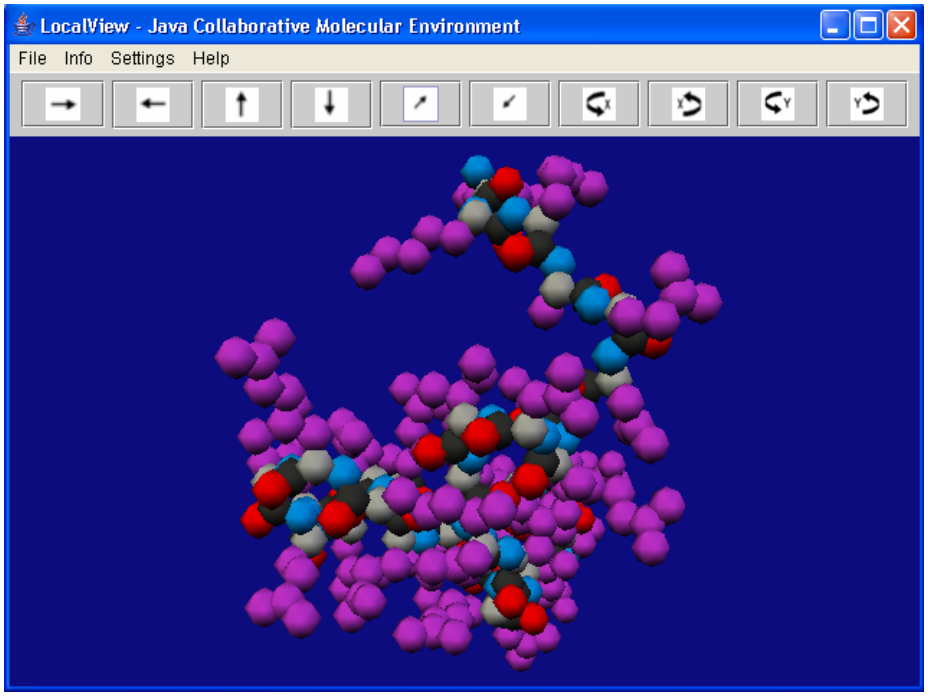

Figura 5.3: Visualização local uma molécula de proteína.

Em seguida encontra-se o menu *groups*, que fornece as opções para a o gerenciamento de grupos.

Finalmente, o menu *help* fornece informações sobre o JVRMol.

5.2 Grupos

Para permitir a criação e o gerenciamento de grupos no JVRMol, foram implementadas as seguintes alterações:

Inicialmente, é necessário que para cada grupo criado, o módulo servidor crie uma instância da classe *GroupInfo*, responsável por identificar os grupos existentes.

Observa-se que a criação de grupos dentro do ambiente do JVRMol é opcional, dessa maneira, a entrada do usuário permanece a mesma do projeto original. Nota-se que, inicialmente, o participante se conecta ao ambiente sem pertencer a nenhum grupo (grupo null). O participante pode criar um novo grupo do qual passa a ser o proprietário, e somente ele terá permissão para encerrá-lo, ou poderá se juntar a um grupo pré-existente.

Quando a janela *Groups* é aberta, invoca-se a operação *listGroup*, que exibe os grupos existentes, com base na classe *GroupInfo*.

O botão *Add Group* permite a inserção de um novo grupo por meio da entrada do nome do grupo na caixa de texto, quando então será invocada a operação *createGroup*, responsável pela inclusão do mesmo no ambiente. O nome do grupo é então armazenado como um objeto da classe *GroupInfo*.

Já o botão *join* permite ao participante se conectar a um grupo por meio da operação do mesmo nome, *join*, responsável pela inclusão de um novo participante em um grupo existente. Isto é feito por meio da seleção de um dos grupos existentes e clicando-se no botão *join*. São passados como parâmetros o nome do grupo e o nome do participante para serem armazenados na classe *MembersGroupInfo*. O servidor é responsável por avisar todos os outros participantes do grupo da entrada de um novo integrante no grupo.

Em seguida, o botão *disjoin* permite o participante deixar um determinado grupo, por meio da operação do mesmo nome, *disjoin*, quando então devem ser removidas as suas referências na classe *MembersGroupInfo*.

Analogamente, o botão *delete* corresponde à exclusão do grupo selecionado por meio da operação *closeGroup*, que trata da exclusão de um grupo e elimina o objeto da classe *GroupInfo*.

Finalmente, o botão *members* permite visualizar os membros de um determinado grupo por meio da operação *listMembersGroup*, *que* exibe os membros do grupo selecionado, com base na classe *MembersGroupInfo*.

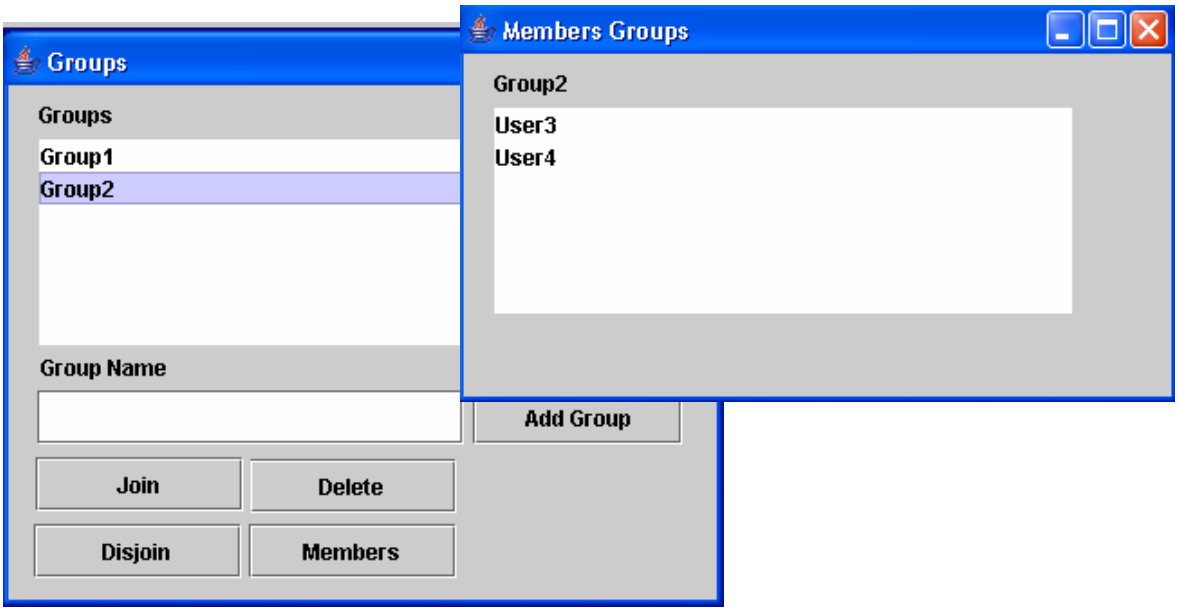

Figura 5.4: Grupos e membros dos grupos

A Figura 5.4 apresenta a tela para criação de grupos e conexão do usuário a um determinado grupo, bem como os membros pertencentes a cada um deles. A classe *MembersGroup* guarda as informações que permitem identificar os participantes de cada grupo.

A Figura 5.5 mostra a instância do participante User1, pertencente ao Group1, após a entrada do segundo participante, User2, mostrado na Figura 5.6. Quando um novo participante se junta ao grupo, sua entrada é notificada aos demais elementos do grupo. Assim, o novo participante deve receber os parâmetros para o carregamento da imagem da molécula de proteína compartilhada pelo grupo.

Em relação ao gerenciamento dos objetos, são utilizadas as operações de carregamento de um arquivo, remoção de objetos do ambiente, transferência de coordenadas (translação, rotação e escala).

Quando algum arquivo é carregado por algum participante, ele é enviado para o servidor que instancia um objeto da classe *FileInfo* para armazenar sua referência. Em seguida, comunica a todos os participantes, pertencentes ao mesmo grupo, da existência de um novo objeto no ambiente. Estes então recebem uma cópia do arquivo.

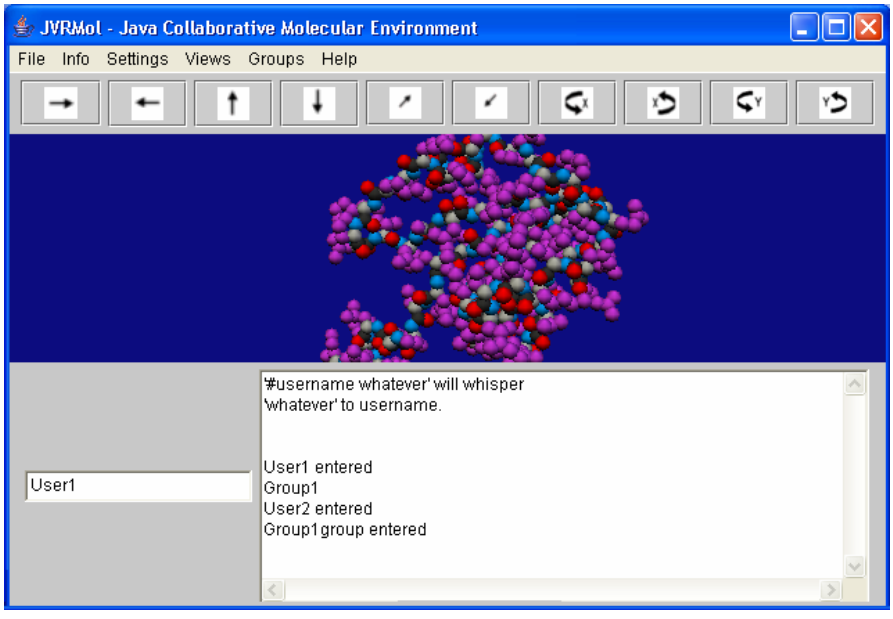

Figura 5.5: Participante 1.

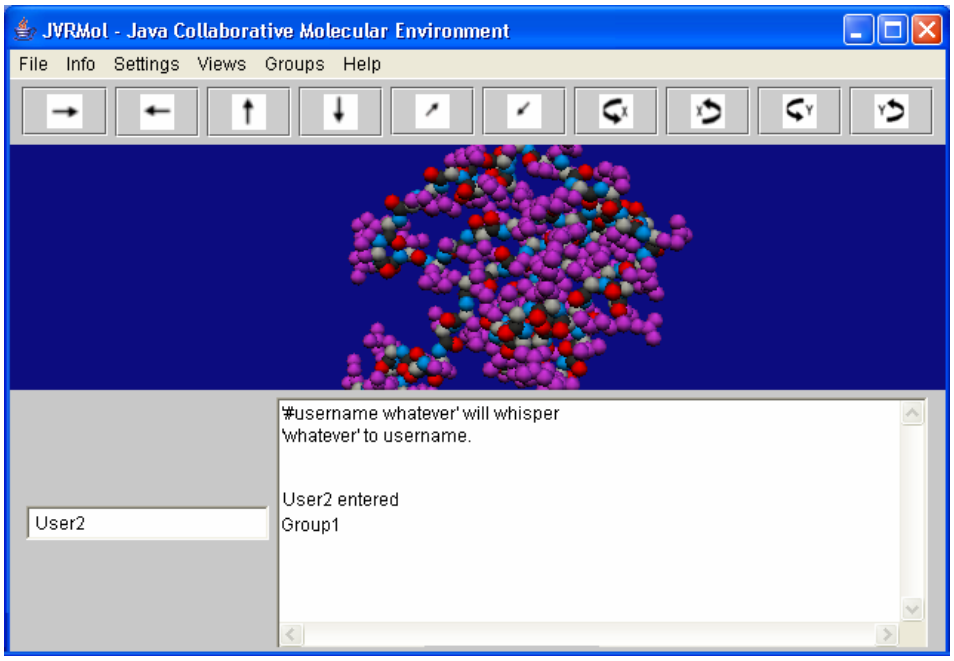

Figura 5.6: Participante 2.

As Figuras 5.7 e 5.8 indicam instâncias dos participantes User3 e User4, pertencentes a Group2.

Verifica-se que o Group2 carrega uma molécula distinta de Group1 e a entrada do participante User4 é comunicada somente ao participante User3.

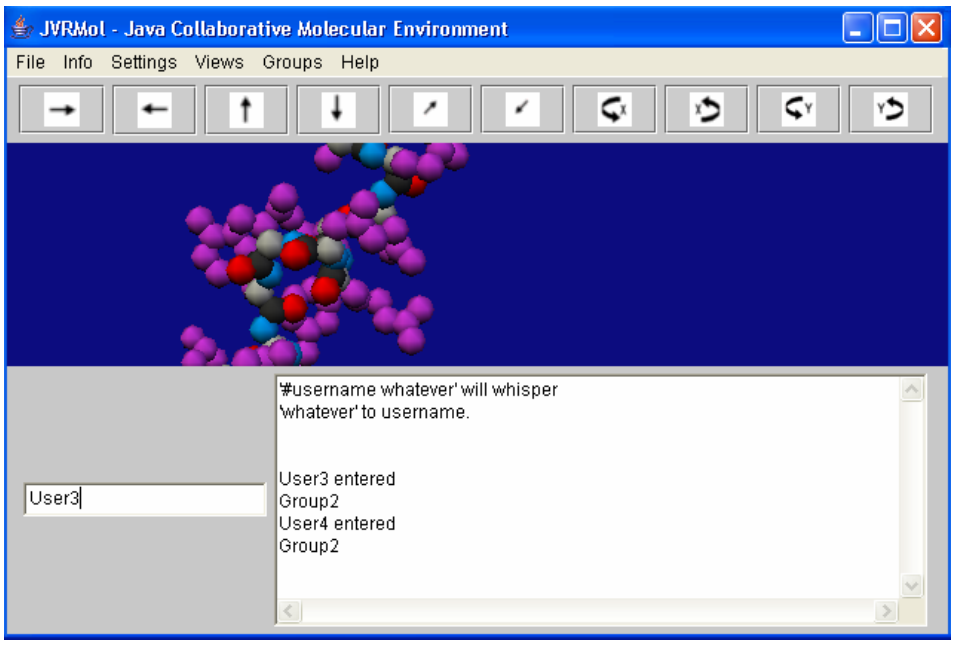

Figura 5.7: Participante 3.

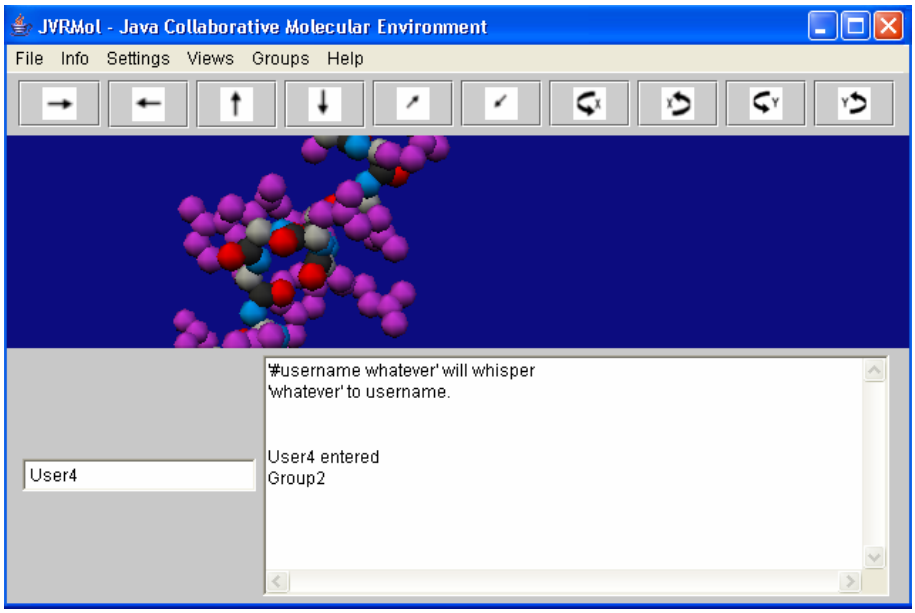

Figura 5.8: Participante 4.

Dentro de cada grupo criado, permiti-se a disseminação de estados entre seus membros, ou seja, garante que a visão da mesma molécula de proteína num dado instante seja uma representação compartilhada pelos outros participantes do grupo.

Visualiza-se que a imagem compartilhada da molécula permanece consistente a todos os participantes do grupo. Cada alteração feita nas coordenadas da molécula é informada aos demais, o que garante esta propriedade.

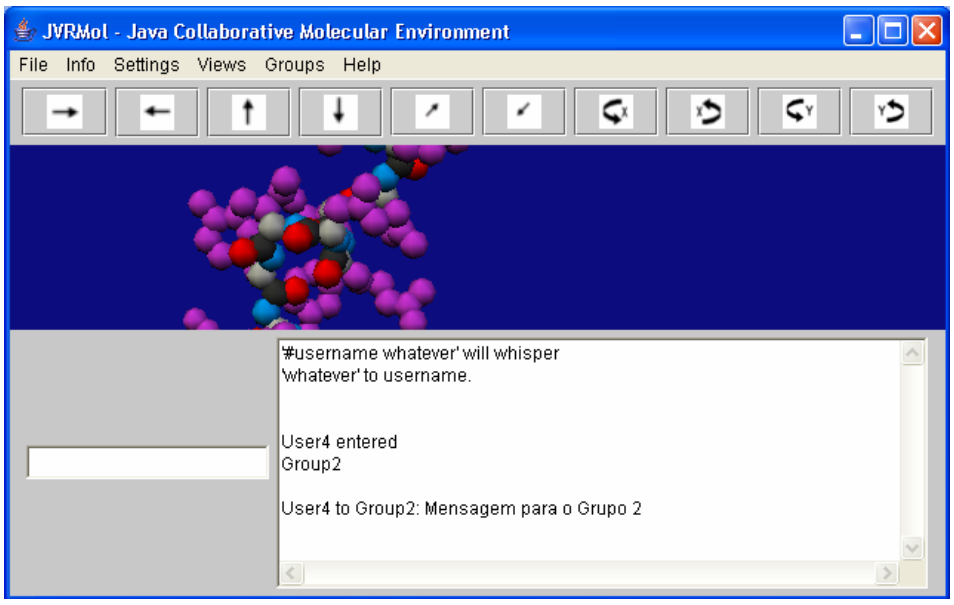

Figura 5.9: Participante 4 envia uma mensagem textual para seu grupo.

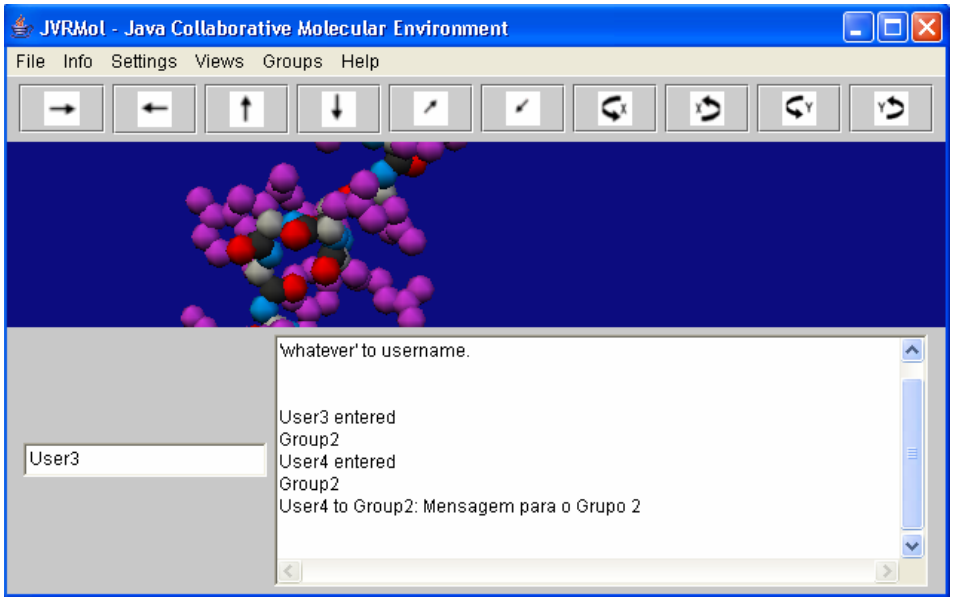

Figura 5.10: Participante 3 recebe uma mensagem textual do grupo a qual pertence.

Do mesmo modo acontecerá quando o participante User4 enviar uma mensagem textual, em que somente os participantes pertencentes ao mesmo grupo a receberá (Figura 5.9 e Figura 5.10).

Isto é feito por meio da operação *chat*, que tem por objetivo difundir uma mensagem de texto enviada por algum dos participantes.

Ressalta-se que vários grupos podem ser criados e um participante pode participar de quantos grupos desejar. Isso adiciona uma funcionalidade maior ao ambiente do JVRMol, pois possibilita o estabelecimento de várias conexões e relações, cada um com diferentes necessidades e demandas.

Um exemplo disso seria dividir a análise de uma determinada molécula de proteína entre os grupos executando em paralelo, e que possuem diferentes opiniões sobre um mesmo problema. Em seguida, os aspectos analisados por cada grupo podem auxiliar na solução do problema por meio do tratamento das informações obtidas por cada grupo.

CONCLUSÃO

A crescente quantidade de informações e o nível de conhecimento exigido dificultam a resolução de problemas e a tomada de decisões por um uma única pessoa. Ao conhecer as vantagens do trabalho em grupo e as dificuldades encontradas na coordenação e no gerenciamento dessas informações, a utilização de ferramentas de apoio à comunicação e ao trabalho colaborativo torna-se fundamental para o aumento da qualidade das informações que disponibilizam.

Percebe-se que o acesso à informação compartilhada e a uma eficiente solução de *groupware* contribui para o aumento da produtividade de equipes de trabalho ou de pesquisa. No desenvolvimento de um projeto, é necessária uma ferramenta que faça a união dos conhecimentos dos participantes para que esses conhecimentos sejam compartilhados contribuindo para a co-realização do projeto.

O trabalho, a produção conjunta e a troca de informações são fatores fundamentais em um projeto compartilhado. A comunicação também tem extrema importância, seja para a transmissão de informações, ou mesmo para a tomada de decisões. Nesse sentido, o compartilhamento das informações se torna essencial para evitar esforços repetitivos.

Neste trabalho, foram implementados mecanismos para criação e gerenciamento de grupos no ambiente do JVRMol. Esses mecanismos incluem a criação e a exclusão de grupos, bem como o gerenciamento dos membros do grupo oferecendo um conjunto de recursos para o trabalho compartilhado em grupo, o que permite que várias equipes trabalhem simultaneamente e participem de projetos em comum, mesmo quando estão geograficamente separadas. Dessa maneira, nota-se que as informações são compartilhadas por todos os participantes de forma facilitada. Quanto à interação, esta se estabelece entre todos eles por meio do compartilhamento contínuo de informações e de idéias trocadas por meio de mensagens. Assim, a construção de conhecimentos é feita em cooperação, em conjunto, o que facilita a troca de informações entre eles e, ainda, uniformiza o conhecimento, bem como facilita a tomada de decisões e chegada a conclusões de maneira mais rápida e eficaz.

Muitas pesquisas são desenvolvidas com o objetivo de fornecer brevemente um cenário adequado para a realização de atividades em grupo em ambientes virtuais de forma eficiente (ASSIS, 2003). Desse modo, este trabalho pretende fornecer uma proposta de biblioteca para a criação e o gerenciamento de grupos em ambientes virtuais compartilhados.

Trabalhos Futuros

Algumas sugestões para trabalhos futuros são:

- Implementar a forma de sincronização no acesso a objetos;
- Mecanismo de marcação de sítios das proteínas;
- Anexar informações aos sítios marcados;
- Salvar contexto do trabalho.

REFERÊNCIAS

ASSIS, Gilda Apararecida. **Ambientes Virtuais Cooperativos e Colaborativos**. Porto Alegre, out. 2003. Disponível em: <http://www.inf.ufrgs.br/ñedel/cmp513/12-cooperavive-vr-p.pdf>. Acesso em: 15 out. 2003.

BATISTA, Nuno et al**. VR-Cool: Uma Plataforma para Trabalho Colaborativo em Ambientes de Realidade Virtual.** Marinha Grande, 2000. Disponível em: < http://virutal. inesc.pt/virtual/9epcg/actas/pdf/artigo7.pdf>. Acesso em: 15 out. 2003.

BENFORD, Steve et al. **Managing Mutual Awareness in Collaborative Virtual Environments**. In: Proceeding Symposium on Virtual Reality Software and Technology. Singapura, ago,1994a.

BENFORD, Steve et al. **Supporting Cooperative Work in Virtual Environments.** The Computer Journal. v.37, n.8. Nottingham. 1994b.

BENFORD, Steve et al. **Collaborative Virtual Environment.** Communication of the ACM, v.44, n.7. julho, 2001.

BRINCK, Tom. Usability First Groupware Section: Introduction to groupware, typical applications, design issues. Disponível em: <http://www.usabilityfirst.com/groupware/cscw.txl>. Acessado em 20/09/2005.

CHANG, Carl K. et al. **Formalization of Computer Supported Cooperative Work Applications**. In: Proceedings The Eighth IEEE Workshop on Future Trends of Distributed Computing Systems, 31 out.-2 nov., 2001. p.185 –191.

CHEVERST, Keith; SMITH, Gareth. **Exploring the notion of information push and pull with respect to the user intention and disruption. 2001**. Disponível em <http://citeseer.ist.psu.edu/536214.html>. Acesso em: 22 nov. 2003.

DAHMANN, Judith S.; FUJIMOTO, Richard M.; WEATHERLY, Richard M. **The Department of Defense High Level Architecture.** In: P*roceedings of the 1997 Winter Simulation Conference ed. S. Andradóttir, K. J. Healy, D. H. Withers, and B. L. Nelson*

DIVE. **The DIVE home page**. Disponível em: <http://www.sics.se/dive/>. Acessado em 10/11/2005.

DIX, A.; FINLAY, J.; ABOWD, G. D.; BEALE, R. **Humam – Computer Interaction**. Prentice Hall. 2004.

FUKS, Hugo; RAPOSO, Alberto B.; GEROSA, Marco A. **Engenharia de Groupware: Desenvolvimento de Aplicações Colaborativas.** In: XXI Jornada de Atualização em Informática. Anais do XXII Congresso da Sociedade Brasileira de Computação. V2, Cap. 3. 2002.

GEROSA, Marco Aurélio et al. **Elementos de Percepção em Ambientes de Aprendizagem como Forma de Facilitar a Comunicação, Coordenação e Cooperação. 2002.** Disponível em: <http://wwwasee.org/international/intertech2002/600.pdf>. Acesso em: 09 nov. 2003.

GREENHALG, Chris. **An Experimental of the Spatial Model.** In: Proceeding Sixth ERCIM workshops. Stockholm, 1-3 jun., 1994.

GREENHALG, Chris. **Supporting Complexity in Distributed Virtual Reality Systems**. Nottingham. 1996. Disponível em: <http://citeseer.nj.nec.com/greenhalgh96supporting.html>. Acesso em: 07 dez. 2003.

GREENHALG, Chris; BENFORD, Steve. **Boundaries, Awareness and Interaction in Collaborative Virtual Environments.** In: Proceedings Sixth IEEE workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, 18-20 jun., 1997. p. 193 – 198.

GRUDIN, Jonathan**. CSCW: history and focus** . Computer**,** maio, 1994.

HOFTE, G. Henri. **Working Apart Together**. Telematica Instituut. 1998.

IQBAL, Rahat et al. A Framework for Interoperability of Heterogeneous Systems. Prooceding of the 14th International Workshop on Database and Expert Systems Applications. 2003.

JAVA set. 2003. Disponível em: <http://java.sun.com>. Acesso em: 15 set. 2003.

JAVA 3D. set. 2003. Disponível em: <http://java.sun.com/products/java-media/3D>. Acesso em: 21 set. 2003.

JAVA RMI. out. 2003. Disponível em: <http:/java.sun.com/products/jdk/rmi/index.htm>. Acesso em: 14 out. 2003.

LOGAN, Brian et al. **Keeping in Touch: Agents Reporting from Collaborative Virtual.** 2002**.** Disponível em: <http://citeseer.ist.psu.edu/507811.html>. Acesso em 22 nov. 2003.

MACEDONIA, Michael R., ZYDA, Michael;. **Taxonomy for networked virtual environments**. IEEE Multimedia, v.4 , n.1, p.48-56, jan,-mar., 1997.

MOECKEL, Alexandre. **CSCW: Conceitos e Aplicações para Cooperação**. Curitiba, 2003.

NIKITINA, Lialia; NIKITIN, Igor,**Implementation of Occlusion Culling.** Disponível em <http://zhurnal.ape.relarn.ru/articles/2003/036e.pdf>. Acesso em 25 out. 2005.

OLIVEIRA, Jauvane C. et al**. Collaborative Virtual Environment for Industrial Training and e-Commerce**. Ottawa, 2000. Disponível em: <www.mcrlab.uottawa.ca/papers/CVE-WS-VRTS2000.pdf>. Acesso em: 15 jan. 2004

PDB. jan. 2004. Disponível em : <http://www.rcsb.org/pdb/>. Acesso em: 28 jan. 2004.

RODELLO, Ildeberto Aparecido. JVRMol **– Um Ambiente Virtual Distribuído para Visualização e Análise de Moléculas de Proteínas**. São Carlos. 2003.

RODELLO, Ildeberto Aparecido et al. JVRMol **– Uma Biblioeteca para Invocação Remota de Métodos em Ambientes Virtuais Distribuídos Desenvolvidos com Java 3D,** Marília. 2005.

SHIRMOHAMMADI, Shervin; GEORGANAS, Nicolas D**. Collaborating 3D Virtual Environments: A Synchronous Architecture**. In: Proceedings IEEE 9th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises. 14-16 jun., 2000. p. 35-45.

SNOWDON, Dave et al. **Collaborative Virtual Environments: Digital Places and Spaces for Interaction.** Londres, 2001.

TANENBAUM, A. S., Distributed Operating Systems. Prentice-Hall, 1995.

TEIXEIRA, Renata Cruz; DUARTE O. C. M. B. A **Plataform for Distributed Virtual Environments.** jan**.** 2004. Disponível em: <http://citeseer.nj.nec.com>. Acesso em: 25 jan. 2004.

TRAMBEREND, H, Avocado: A Distributed Virtual Reality Framework," In: *Proceedings IEEE Virtual Reality '99*, pp. 14–21, IEEE Computer Society, Mar. 13–17 1999.

VALIN, Steven. et al. **Sharing Viewpoints in Collaborative Virtual Environments**. In: *Proceedings of the 34th Annual Hawaii International Conference on System Sciences*. 3-6 de janeiro 2001. p. 257 – 268.

ZOTTO, Ozir Francisco de Andrade. **Ferramentas Groupware psra Intranets**. Bate Byte, n.70, nov, 1997.

ZYDA, Michael; SINGHAL, S. **Networked Virtual Environments: Design an Implementation**. Addison Wesley Pub, Califórnia, 1999.

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo