UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO

(Bacharelado)

PROTÓTIPO DE UM AMBIENTE VIRTUAL DISTRIBUÍDO UTILIZANDO A PLATAFORMA DE DESENVOLVIMENTO DIVE

TRABALHO DE CONCLUSÃO DE CURSO SUBMETIDO À UNIVERSIDADE REGIONAL DE BLUMENAU PARA A OBTENÇÃO DOS CRÉDITOS NA DISCIPLINA COM NOME EQUIVALENTE NO CURSO DE CIÊNCIAS DA COMPUTAÇÃO — BACHARELADO

JACYR JEFFERSON POPENDA

BLUMENAU, JUNHO/2002

2002/1- 41

PROTÓTIPO DE UM AMBIENTE VIRTUAL DISTRIBUÍDO UTILIZANDO A PLATAFORMA DE DESENVOLVIMENTO DIVE

JACYR JEFFERSON POPENDA

ESTE TRABALHO DE CONCLUSÃO DE CURSO, FOI JULGADO ADEQUADO PARA OBTENÇÃO DOS CRÉDITOS NA DISCIPLINA DE TRABALHO DE CONCLUSÃO DE CURSO OBRIGATÓRIA PARA OBTENÇÃO DO TÍTULO DE:

BACHAREL EM CIÊNCIAS DA COMPUTAÇÃO

_____

Prof. Dalton Solano dos Reis — Orientador na FURB

_____

Prof. José Roque Voltolini da Silva — Coordenador do TCC

BANCA EXAMINADORA

_____

Prof. Dalton Solano dos Reis

_____

Prof. Paulo Rodacki Gomes

_____

Prof. Sérgio Stringari

DEDICATÓRIA

“Escale a montanha dos seus ideais e conquiste-a.”

Dedico este trabalho aos meus pais, minhas irmãs e meu cunhado que sempre me apoiaram e me deram forças para terminar essa escalada, principalmente nos momentos em que eu olhava para baixo e sentia vontade de desistir.

AGRADECIMENTOS

Em especial ao meu orientador Dalton Solano dos Reis, que soube me apoiar nas horas mais difíceis, me indicando o caminho a ser seguido para atingir meu objetivo.

E também a meus colegas de faculdade e amigos, pelo companheirismo e bons momentos vividos.

Por fim, agradeço a todos aqueles que contribuíram para a conclusão desse trabalho.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE QUADROS

RESUMO

Este trabalho apresenta o desenvolvimento de um protótipo de ambiente virtual distribuído, utilizando-se para isto a plataforma de desenvolvimento DIVE (*Distributed Interactive Virtual Enviroment*). Inicialmente, aborda alguns conceitos e características relevantes à criação deste tipo de ambiente. Á seguir apresenta a plataforma DIVE, ressaltando suas principais características bem como os métodos existentes na mesma e que possibilitam a implementação de ambientes virtuais distribuídos. Por fim, apresenta o processo de especificação e implementação do protótipo e também seu funcionamento.

ABSTRACT

This work presents the development of a distributed virtual environment archetype, using for this the platform of development DIVE (Distributed Interactive Virtual Enviroment). Initially, it approaches some excellent concepts and characteristics to the creation of this type of environment. To follow it presents platform DIVE, standing out its main characteristics as well as the existing methods in the same one and that they make possible the distributed virtual environment implementation. Finally, it also presents the process of specification and implementation of the archetype and its functioning.

1 INTRODUÇÃO

1.1 CONTEXTUALIZAÇÃO / JUSTIFICATIVA

Os avanços alcançados na indústria de computadores nos últimos anos têm consolidado os sistemas multimídia como forma de interface homem-máquina amplamente disponíveis nos dias atuais. A busca por uma interação mais simples, direta e natural entre os usuários e os seus computadores vem indicando a tecnologia de realidade virtual como a forma mais adequada de chegar a estes objetivos.

Teixeira (1999) ressalta que alguns autores diferenciam o termo realidade virtual de ambiente virtual. O termo realidade virtual, como um termo mais abrangente, significa a utilização de tecnologias onde através de suas interfaces, pode-se dominar os sentidos de seus usuários humanos de forma com que eles interajam intuitivamente com um ambiente gerado por computador. Já o ambiente virtual, pode ser conceituado como uma representação tridimensional gerada por computador que apenas sugere um espaço real ou imaginário, não tendo o realismo fotográfico e o senso total de imersão como objetivos principais.

Com o crescimento vertiginoso da internet nos últimos anos, o interesse pela utilização de realidade virtual para criação de ambientes virtuais também aumentou, devido ao grande potencial de aplicações que podem utilizar suas técnicas, abrangendo áreas que vão desde o entretenimento até aplicações de medicina ou simulação militar. Estas aplicações podem ser classificadas de acordo com o tipo de ambiente, podendo ser um ambiente puramente virtual ou possuindo algum tipo de interação com o mundo real; a participação do usuário, se o usuário é ativo – modificando o ambiente – ou passivo; a distribuição do sistema, se o sistema está concentrado em uma única estação ou se está dividido em estações distintas; e o número de usuários participando simultaneamente do ambiente virtual.

Segundo Kirner (2000) são três as idéias básicas que caracterizam um ambiente de realidade virtual: imersão, interação e envolvimento. A idéia de imersão está relacionada com a sensação que o usuário tem de estar realmente inserido no ambiente virtual, possuindo a ilusão de que os objetos dispostos no cenário estão realmente à sua volta. Interação está ligada à capacidade do ambiente virtual de detectar ações do usuário e reagir conforme estas ações. Por sua vez, a idéia de envolvimento exprime o grau de motivação do usuário em engajar-se na atividade proposta pelo ambiente virtual.

Um ambiente virtual, mostrando uma representação tridimensional de um espaço real ou imaginário ou ainda a combinação de ambos e que suporte o acesso simultâneo por vários usuários permitindo a interação destes entre si e também com o ambiente apresentado, é chamado de ambiente virtual distribuído ou DVE *(Distributed Virtual Evironment)* (Raposo, 2000).

A construção de ambientes virtuais distribuídos envolve uma série de aspectos de extrema importância tais como resposta rápida aos requisitos do sistema, suportar interação em tempo real, fidelidade de inserção do usuário no ambiente virtual em relação a uma referência, controle de atualização/sincronização dos estados dos objetos e do próprio ambiente virtual para todos os usuários, entre outros (Kirner, 2000).

Dentre as diversas plataformas para desenvolvimento de softwares de realidade virtual, pode-se destacar o sistema DIVE (*Distributed Interactive Virtual Enviroment*), desenvolvida pelo SICS (Instituto Sueco de Ciências da Computação) e que tem sido usado como *toolkit* para criação de várias aplicações de realidade virtual multi-usuário (Raposo, 2000).

Esta plataforma se caracteriza pela criação de ambientes com dados compartilhados distribuídos e atualização ponto-a-ponto onde a descrição do mundo virtual é distribuída entre as estações envolvidas, tendo sua base atualizada conforme os atores ou avatares entram ou saem, do mundo ou de um grupo, através de um protocolo de *multicast*, sendo isto essencial para que se possa utilizar melhor os recursos de rede reduzindo assim, indiretamente, a latência (Stenius, 1996).

Segundo Raposo (2000), as questões à respeito das percepções dos usuários também são muito bem tratadas no DIVE pois os usuários podem ser representados de diversas maneiras, utilizando-se desde formas mais simples, que irão apenas informar a presença, localização e orientação de um usuário, até formas mais complexas que mapeiam uma foto estática do usuário na cabeça do Avatar.

Ainda de acordo com Raposo (2000) o funcionamento do DIVE é muito semelhante ao VRML 2.0, que é uma linguagem muito utilizada para descrição de ambientes virtuais na internet, onde os mundos são definidos por uma linguagem baseada em hierarquias de nós, com associação de comportamento aos objetos do mundo utilizando-se scripts Tcl, e tendo sua visualização feita por software específico que pode ser utilizado como *Helper Application* dos *Web browsers* convencionais.

Considerando os assuntos anteriormente citados, procurou-se desenvolver um protótipo de ambiente virtual distribuído e multiusuário, com uma interface de realidade virtual não imersiva, utilizando-se da plataforma DIVE. Dessa forma este trabalho permite que se tenha uma visão mais abrangente da utilização desta ferramenta para implementação de ambientes virtuais distribuídos, bem como de sua real eficácia para solução dos principais problemas existentes.

Além disso, a plataforma DIVE é relativamente nova sendo com isto pouco utilizada para a modelagem e construção de ambientes virtuais, em contrapartida ao VRML e ao JAVA que são utilizados na maioria dos casos. E também, por possuir a facilidade já incorporada na própria plataforma, de se poder implementar tanto a construção do mundo virtual quanto os mecanismos responsáveis pelo controle da comunicação entre os usuários, não havendo necessidade de recorrer a outros protocolos de comunicação de dados.

1.2 OBJETIVOS

O objetivo principal do trabalho é implementar um protótipo de um ambiente virtual distribuído sobre uma rede local, com suporte a multiusuários e com uma interface de realidade virtual não imersiva.

Os objetivos específicos do trabalho são:

a) criar um mundo virtual 3D com objetos simples e com baixo grau de realismo;
b) neste mundo virtual, ter um Avatar que possa interagir com os objetos descritos e outros Avatares.

1.3 ORGANIZAÇÃO DO TRABALHO

O trabalho está organizado conforme descrito abaixo:

O capítulo um trata da contextualização do trabalho, apresentando também os objetivos principais e específicos desejados com a elaboração do mesmo

O capítulo dois aborda de forma mais concisa os aspectos históricos dos ambientes virtuais distribuídos, os principais problemas existentes para implementação dos mesmos e ainda alguns dos principais sistemas virtuais distribuídos existentes.

O capítulo três apresenta a plataforma de desenvolvimento DIVE, ressaltando suas principais características e métodos para criação de ambientes virtuais e implementação dos

mecanismos necessários a manutenção destes ambientes. O capítulo quatro apresenta o desenvolvimento de aplicações no Dive, focalizando principalmente o modelo de implementação em camadas existente na plataforma. O capítulo cinco apresenta o desenvolvimento do protótipo de ambiente virtual multiusuário distribuído bem como sua especificação, implementação e funcionamento .

Finalmente, no capítulo seis são apresentadas as conclusões provenientes da execução desse trabalho, bem como as possíveis extensões que dele podem ser desenvolvidas.

2 AMBIENTES VIRTUAIS DISTRIBUÍDOS

De acordo com Kirner (2000), ambientes virtuais distribuídos, vistos como um meio de comunicação, apresentam um grande potencial de aplicação em diversas áreas. O principal objetivo de um ambiente virtual é motivar o usuário a acreditar que ele está imerso no cenário sintético apresentado. Para alcançar esta ilusão são utilizadas técnicas de realidade virtual distribuída.

2.1 REALIDADE VIRTUAL DISTRIBUÍDA

Um ambiente virtual mostra a representação tridimensional sintética de um espaço real, imaginário ou de uma combinação de ambos. Caso o ambiente permita o acesso de vários usuários simultâneos e também possibilite a interação destes entre si e com o ambiente apresentado, ele é chamado de ambiente virtual distribuído (Raposo, 2000).

Os usuários são representados em um ambiente virtual por uma entidade conhecida como Avatar. O Avatar é a forma tridimensional pela qual o usuário será visualizado pelos outros usuários inseridos no mesmo ambiente virtual compartilhado. Ou seja, as ações tomadas pelo usuário serão executadas no ambiente virtual pelo seu Avatar.

2.2 DESENVOLVIMENTO DE AMBIENTES VIRTUAIS DISTRIBUÍDOS

Segundo Raposo (2000), há pelo menos duas décadas existem pesquisas em DVEs (*Distributed virtual environment*). Porém, poucos tinham acesso a computadores capazes de executar com eficiência o processamento demandado por um sistema de DVE e menos pessoas ainda tinham acesso a conexões de rede de computadores de longa distância com a grande largura de banda passante e a baixa latência exigida pelos DVEs. No entanto, nos últimos anos, a capacidade computacional dos processadores vem aumentando gradativamente enquanto seu custo reduz-se. Os modens caseiros estão alcançando taxas de transmissão que atendem as necessidades de um sistema DVE simples.

Ainda segundo Raposo (2000), as pesquisas com DVEs estão sendo conduzidas basicamente por duas comunidades distintas e que possuem prioridades distintas. Elas são o mundo da internet, apoiada pelos desenvolvedores comerciais e o mundo da Simulação Distribuída Interativa (*Distributed Interactive Simulation* – DIS), suportado pelos desenvolvedores de simulação militar. O objetivo final de ambos é semelhante: um ambiente

virtual de interação com alto grau de complexidade e flexibilidade que suporte simultaneamente um grande número de usuários. Os caminhos escolhidos por ambos, no entanto, divergiram. O grupo da internet prioriza a acessibilidade, enquanto o grupo DIS baseia-se em equipamentos dedicados de alto custo. O mundo da internet defende que um ambiente virtual distribuído deve poder ser executado pelos modelos de computador e acessado pelo tipo de conexão que a maioria dos usuários comuns possui. Surgem problemas como a limitação de banda passante e a latência nas conexões caseiras.

De acordo com Ellis (1994), o início dos trabalhos em DVE na internet ocorreu em meados da década de 70, com o surgimento de um novo tipo de jogo para computador, o *Adventure*, criado por Will Crowter e Donald Woods do centro de pesquisa da Xerox em Palo Alto, Califórnia. Este jogo de computador foi inspirado em um *role playing game* (RPG) chamado *Dungeon and Dragons* e seus participantes utilizam apenas lápis, papel e um conjunto de regras pré-definidas. Toda a descrição do mundo é feita por um dos jogadores, chamado de *Dungeon Máster* (DM). Os demais jogadores interpretam personagens envolvidos em aventuras no mundo virtual descrito pela narrativa do DM. O conjunto de regras serve para resolver conflitos entre os personagens ou dos personagens com os obstáculos colocados pelo narrador da história. O ambiente virtual está na imaginação de cada participante. O *Adventure,* no entanto, era para um único jogador e era executado em um computador de grande porte da *Digital Equipment Corp*. O jogo fornecia a descrição em forma de texto do que estava sendo visto pelo participante e cabia a este formar a imagem do ambiente baseado na descrição apresentada. Com o passar de alguns anos, o *Adventure* tornou-se um gênero de jogos de computador, populares até hoje. A evolução chave deste gênero de jogo, ao menos da perspectiva dos DVEs, foi a possibilidade de múltiplos jogadores interagirem.

Segundo Vilhjálmsson (1997) o primeiro destes jogos multi-usuários foi o *Multiuser Dungeon*, conhecido como MUD, sendo também todo baseado em texto. Devido à popularidade alcançada pelo MUD e os seus sucessores, o termo MUD muitas vezes é utilizado como referência a jogos de computador multi-usuário. Do desenvolvimento posterior do MUD surgiram as salas de conversa (*chat rooms*) da internet. Já outra linha de desenvolvimento, utilizando tecnologia de programação orientada a objetos, permitiu ao

criador do mundo virtual construir um módulo básico, facilmente modificável e expansível. Esta vertente do MUD ficou conhecido como MOO (MUD orientado a objetos).

Certamente, os usuários e desenvolvedores de MUD sempre sonharam com a possibilidade de visualizar e ouvir o mundo virtual originariamente descrito apenas com texto. Com a popularização dos computadores pessoais e a maior facilidade de acesso à rede de computadores, o mundo da internet pôde experimentar a construção de ambiente virtuais com imersão tridimensional e interação de áudio. Alguns esforços foram concentrados em sistemas proprietários, mas também houve um forte movimento defendendo a criação de sistemas abertos. O principal resultado deste movimento foi o surgimento do padrão VRML (*Virtual Reality Modeling Language).*

O mundo do DIS, comenta Raposo (2000), busca simulações altamente realistas para serem utilizadas em treinamento militar. De um sistema DIS é esperada a capacidade de suportar milhares de usuários simultâneos imergindo-os em um ambiente tridimensional satisfatoriamente convincente. Um sistema como este, tipicamente, exige *hardware* caro e dedicado.

Desde a década de 70, o departamento de defesa do governo americano (DoD) possui simuladores imersivos para o treinamento individual dos comandados na operação de diversos veículos militares. A partir da década de 80, o DoD financiou pesquisas para conectar aparelhos de simulação através de redes de alta velocidade. Esta rede ficou conhecida como *Simnet*. Decidiu-se utilizar os simuladores existentes, que possuíam formatos incompatíveis (por exemplo, para a modelagem tridimensional) e fazê-los funcionar em conjunto, ao invés de desenvolver padrões para conteúdo portável. Partindo-se desta decisão, chegou-se ao padrão IEEE DIS, cujo enfoque está em como os simuladores envolvidos podem comunicar informações dinâmicas como a posição de objetos que retratem tropas ou veículos virtuais. De acordo com Teixeira (1999), o padrão DIS serviu como base para toda uma indústria de simulação e sistemas de treinamento militar.

Oliveira (2001) comenta que o DIS sofre ainda hoje pela falta de padronização em conteúdo portável, dificultando a sua extensibilidade, ou seja, caso se deseje acrescentar um novo modelo de tanque de guerra, por exemplo, os fabricantes de cada modelo de simulador envolvido precisam desenvolver o modelo tridimensional do novo objeto em seu software

proprietário. Esforços estão sendo realizados para o desenvolvimento de uma arquitetura de alto nível que permita a reutilização do *software* dos simuladores visando com isto a redução de custo de aquisição e manutenção de sistemas.

2.3 CONSIDERAÇÕES SOBRE AMBIENTE VIRTUAIS DISTRIBUÍDOS

Segundo Kirner (2000), ambientes virtuais distribuídos possuem objetivos de desempenho extremamente ambiciosos, dentre eles o projeto de interfaces naturais em tempo real para ambientes fisicamente corretos. A fidelidade na representação do ambiente virtual não necessita de um realismo fotográfico, porém deve exprimir com a máxima realidade possível os objetos do mundo real aos quais deseja-se retratar.

De acordo com Kirner (2000) o desenvolvimento de ambientes virtuais distribuídos encontra grandes desafios tecnológicos, tais como o gerenciamento da grande quantidade de dados envolvida, a interação entre os vários objetos envolvidos e a escolha da topologia de rede mais adequada.

Os objetos virtuais inseridos no mundo virtual e que possuam qualquer tipo de estado mutável no tempo são chamados de entidades ou, se controlados por um usuário, de avatares.

A tarefa de um sistema DVE é intermediar a troca de informações sobre o estado dinâmico das entidades entre as estações da rede. A mudança de estado de um Avatar, comandada por um participante, deve ser comunicada a todas as outras estações da rede. Através desta atualização de dados é possível manter a consistência na visualização do mundo virtual por todos os participantes. Logo, os pacotes de informações trocados entre as estações envolvidas em um sistema de DVE tipicamente retratam o estado atual de entidades.

Segundo Raposo (2000), os desafios básicos na implementação de um sistema DVE está na decisão de como lidar com a latência alta e variável, e com a largura de banda limitada das comunicações de rede.

Uma abordagem inicial ao compartilhamento de informações sobre o estado de entidades levaria a cada entidade enviar continuamente informações sobre seu estado a todas as outras estações na rede. O problema aparece quando o número de usuários cresce e o mundo virtual torna-se mais complexo, aumentado com isto, o volume de dados trocados

entre as estações envolvidas até o ponto em que a banda passante disponível torna-se o fator limitante. Muitas vezes os participantes recebem pacotes com informações para si que são irrelevantes, gerando processamento desnecessário.

O problema de banda passante limitada para o potencial grande volume de dados gera outro problema: a latência. Segundo Wray (1999), latência pode ser definida como o período de tempo entre a mudança de estado de uma entidade em uma estação da rede e a efetiva alteração de seu estado em uma outra estação. Quando um usuário deseja que seu Avatar desloque-se por um ambiente virtual, existe um atraso para o *software* local detectar este comando. Outro atraso é gerado quando esta instrução é convertida em informação apropriada a transmissão pela rede. Outro montante de tempo é gasto para a efetiva transmissão da informação, principalmente se a rede utilizada for do tipo de longa distância (WAN- *Wide Área network*). Finalmente, um tempo adicional é necessário para na recepção do pacote o destinatário possa converter as informações contidas efetivamente na visualização tridimensional do cenário virtual. Caso o tempo seja excessivamente grande, haverá um longo período de espera entre, por exemplo, um usuário instruir seu Avatar para mover-se e este realmente deslocar-se à vista dos demais participantes do mundo virtual.

Uma alta latência, segundo Wray (1999), é prejudicial à qualidade de um sistema DVE, assim como uma latência altamente variável. Atualizações de estado em usuários diferentes com intervalos variáveis de tempo podem gerar problemas de consistência ao mundo virtual. O ideal seria se a banda passante fosse ilimitada e a latência desprezível, o que não acontece com a internet atual que não é adequada à interação em tempo real. Com isto os pacotes sofrem atrasos na entrega em virtude da latência ser extremamente variável e imprevisível. Em virtude disto, torna-se necessária a implementação de algumas estratégias para que se possa lidar com o problema de banda passante limitada e atrasos variáveis ou muito altos.

2.3.1 COMPRESSÃO DE DADOS

Segundo Makrakis (2001), o uso de compressão de dados diminui a exigência de banda passante no envio de dados. Em compressão de vídeo, por exemplo, usa-se o fato que, em média, os quadros mudam muito pouco em relação aos seus antecessores imediatos, sendo possível somente enviar as diferenças entre os quadros. A mesma idéia pode ser estendida a

sistemas DVE. O envio constante de atualização de posição em um ambiente virtual distribuído, por exemplo, pode gerar um desperdício de banda passante. A idéia é aproveitar-se da similaridade existente entre o estado em que uma entidade está, em relação ao seu estado anterior.

Um objeto que movimenta-se por um ambiente virtual distribuído pode enviar a sua posição atual e um vetor de velocidade, indicando assim o sentido de seu deslocamento e sua velocidade neste deslocamento. Uma nova atualização somente seria necessária caso o objeto altere o sentido ou a magnitude de seu deslocamento, ao invés de enviar diversas posições intermediárias continuamente. O *software* em cada estação pode deduzir a posição instantânea do objeto pela sua posição inicial e seu vetor de velocidade. Este conceito ficou conhecido como *Dead Reckoning* e foi introduzido no início dos anos 80 no sistema *Simnet* das forças armadas americanas. O nome tem origem na navegação marítima e sugere a obtenção da posição atual a partir da posição e velocidade anteriormente conhecidas.

O protocolo DIS (*Distributed Interactive Simulation)* derivado do sistema *Simnet* transmite o estado de seus objetos fornecendo a sua posição, sua velocidade e sua velocidade angular. Cada simulador calcula a sua posição de forma detalhada, através de um algoritmo que leva em conta a sua posição e velocidades anteriores, a entrada do usuário e outros dados relevantes. Outro cálculo mais simples é realizado levando-se em conta somente os dados enviados aos outros simuladores.

Quando os dois resultados divergem por um determinado valor limite, o resultado do cálculo mais detalhado é enviado para os demais simuladores. Mesmo que não haja qualquer divergência, cada entidade envia, em intervalos regulares de tempo, atualizações de seu próprio estado apenas para que o sistema identifique que elas ainda estão em operação. Estas atualizações são conhecidas como *Heartbeats* ou *Keepalives*.

De acordo com Eduardo (2001), o conceito de *Dead Reckoning* auxilia na questão de banda passante e possibilita tornar o sistema menos vulnerável a latência. Como cada mensagem de atualização também leva uma estampa de tempo (*Timestamp*) registrada, mesmo que o pacote atrase é possível calcular a posição atual do objeto em questão. Esta técnica foi originalmente idealizada para aplicações em sistema de simulação militar, onde o número de objetos diferentes envolvidos é relativamente pequeno e seus modelos

tridimensionais estão replicados em todos os simuladores envolvidos. Basta, portanto, que sejam enviadas informações de posição e deslocamento entre os simuladores.

2.3.2 FILTRAGEM DE DADOS

Em um ambiente virtual compartilhado, normalmente, nem todos os usuários interagem entre si simultaneamente. Técnicas de filtragem de dados podem aproveitar-se do fato de que um determinado usuário não necessite naquele instante das informações de um outro participante. Procura-se minimizar o envio de informações irrelevantes.

De acordo com Teixeira (1999), pode-se agrupar as diversas entidades em classes com relação à área de interesse de cada entidade. Estas classes podem ser de três tipos: espacial, temporal ou funcional. As atualizações de uma entidade somente são enviadas para os membros do mesmo grupo em cada classe. Um grupo da classe espacial reúne entidades que compartilhem o mesmo espaço no ambiente virtual. Os membros de um grupo da classe temporal necessitam de taxas de atualização similares. Por exemplo, em um sistema militar de simulação, aviões em combate aéreo necessitam de um controle apenas superficial da posição de veículos terrestres. Portanto, podem receber atualizações de um determinado veículo a uma taxa menor do que outros veículos terrestres estão recebendo. Dessa forma, os aviões envolvidos no combate aéreo estão em uma classe temporal e os veículos terrestres em outra. Caso este mesmo avião passe a realizar um ataque aos veículos terrestres ele seria inserido na mesma classe temporal destes. Finalmente, um grupo da classe funcional contém entidades que possuam funcionalidades semelhantes, como por exemplo, em uma simulação militar uma classe funcional envolveria um grupo tático de controle aéreo.

Segundo Teixeira (1999) a filtragem de dados mais comum envolve a classe espacial. As entidades inseridas em um mesmo espaço virtual freqüentemente não estão na área de interesse de outras entidades. O espaço virtual pode ser dividido em espaços abertos, como cidades, florestas, áreas ao ar livre em geral, ou confinados, como salas, prédios ou áreas fechadas. Pode-se utilizar a filtragem por oclusão para espaços confinados. Caso exista um obstáculo entre duas entidades, estas não necessitam receber informações uma sobre a outra. Por exemplo, se o usuário A estiver movimentando-se no interior de uma sala, o usuário B, que esteja na sala ao lado, não necessita receber os dados relativos à movimentação de A, pois a visão de B limita-se ao interior de sua sala. O mesmo ocorre com A em relação à B. Espaços

abertos podem sofrer filtragens por oclusão ou por distância. A filtragem por oclusão, neste caso, envolveria o interior e exterior de um espaço confinado. Para a filtragem por distância é necessário estabelecer um alcance para a visão das entidades, ou seja, a partir de uma determinada distância em um espaço aberto as entidades estão fora do alcance da vista de outra entidade e não faz-se necessária a atualização de seus dados.

O escopo de interação do participante de um mundo virtual é delimitado pela filtragem de dados aplicada, diminuindo-se o volume de mensagens trocadas, pois apenas os participantes de um mesmo grupo trocam informações.

Buscando minimizar o tráfego na rede, comenta Oliveira (2001) muitos sistemas DVE estão adotando a comunicação *multicast*. Nesta configuração os pacotes não são endereçados a uma única estação, mas a um grupo de estações. Cada grupo *multicast* define a sua área de interesse para cada entidade e equivale a um grupo formado com base nas classes espacial, temporal ou funcional.

2.3.3 MANUTENÇÃO DE CONSISTÊNCIA

Conforme Teixeira (1999) um grande desafio para um sistema DVE é a manutenção da consistência entre as diversas representações do mundo virtual, uma para cada estação participante. O problema surge quando múltiplos usuários tentam interagir simultaneamente com uma mesma entidade.

Baseado neste problema, Ellis (1994) comenta que várias abordagens podem ser propostas para solucioná-lo. Uma abordagem mais simples defende que o sistema pode confiar em regras de convívio social para evitar conflitos. A partir do momento que cada usuário está ciente das ações dos outros usuários, o primeiro pode deixar que os outros ajam para então acessar um recurso em comum. Por exemplo, dois avatares podem querer abrir e atravessar uma porta simultaneamente, o que acarretará uma perda na consistência do cenário virtual. Um dos dois pode deixar o outro atravessar a porta para segui-lo logo após. Haverá a aplicação de uma regra de convívio social de nosso cotidiano pelos usuários, controladores e seus avatares, no ambiente virtual. Esta abordagem funciona satisfatoriamente para ambientes virtuais colaborativos, porém ela é desaconselhável em sistemas em que hajam disputas entre

os participantes. Em sistemas de simulação de combate ou em jogos, seguir regras de convívio social ou educação significa a morte ou derrota do usuário.

Nas aplicações em que existam disputas entre os participantes, fica a cargo do sistema definir a ordem de solicitação de acesso ao recurso. A primeira mensagem a chegar recebe autorização do *software* de controle para acessá-lo. Esta abordagem pode ser injusta quando, levando-se em conta a latência da rede, principalmente em WANs. Uma estação mais próxima ao servidor de recursos seria beneficiada em detrimento de uma localizada mais distante.

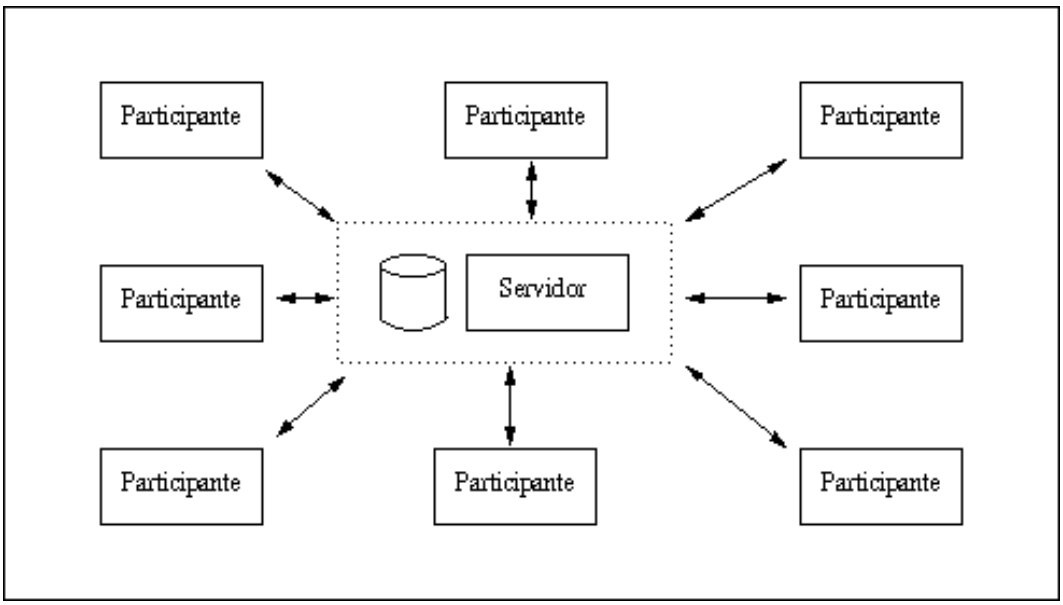
Outra abordagem define que uma entidade ao ser solicitada por um Avatar deve trancar-se (*lock*) aos outros usuários. Somente o usuário, agora proprietário da entidade, pode liberá-la para o uso dos outros. De forma semelhante, uma entidade pode ligar-se (*attachment*) a um usuário, porém a entidade pode desligar-se deste usuário a qualquer momento. Em ambas as abordagens a posição e orientação das entidades trancadas ou ligadas passam a ser relativas ao usuário proprietário ou associado, não adicionando tráfego a rede com sua movimentação.

2.3.4 ARQUITETURA

De acordo com Wray (1999), uma escolha de extrema importância no projeto de sistemas DVE é o protocolo de transporte adotado. O TCP (*Transmission Control Protocol*) oferece uma abordagem orientada a conexão, confiável, mas com o custo de uma alta latência. O UDP (*User Datagram Protocol*) oferece uma abordagem não orientada a conexão, não confiável, mas pode-se utilizar retransmissões periódicas para prover uma maior garantia de que nenhuma das estações perdeu alguma informação crucial. Entre as vantagens e desvantagens de cada abordagem, a maioria dos sistemas DVE existentes utiliza uma combinação de ambos.

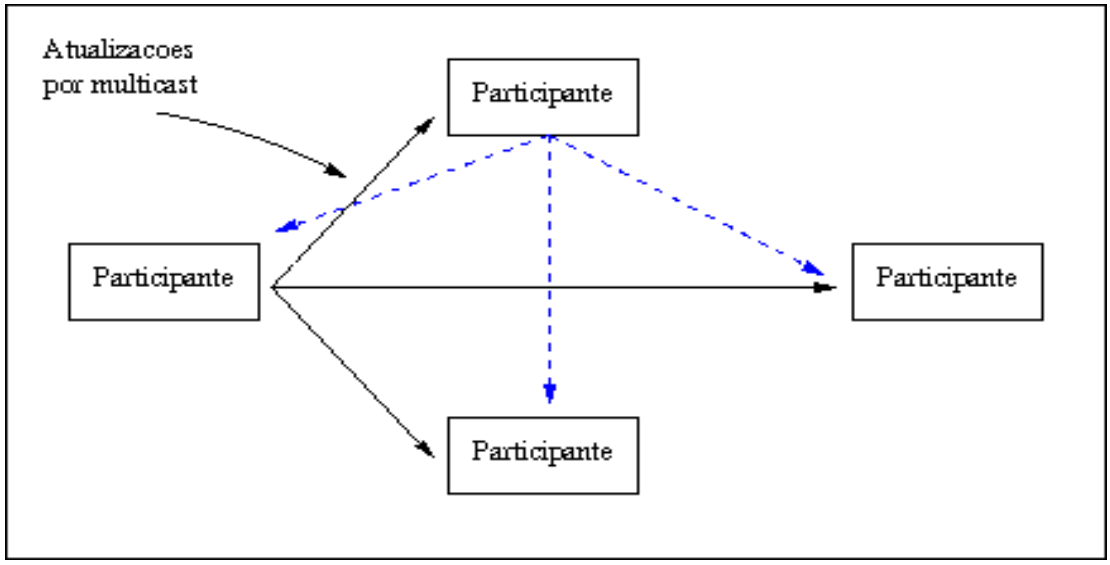
Teixeira (1999) ressalta que a topologia de rede utilizada também é importante. A solução mais simples é um único servidor central, onde todas as estações conectam-se e para onde enviam todas as informações de atualização, como na figura 2.1. O servidor faz a filtragem dos dados possíveis e repassa as informações às estações pertinentes. Esta abordagem funciona bem para ambientes de poucos usuários, mas nitidamente o servidor é um limitante para a escalabilidade do sistema.

Fig. 2.1 Servidor Central



Uma alternativa é o estabelecimento de conexões diretas entre as estações, eliminando-se o servidor central, como mostrado na figura 2.2. No entanto, cada estação deveria enviar **n-1** atualizações, onde **n** é o número de estações envolvidas. Esta configuração acaba tornando-se também não é escalável a milhares de usuários. Para mundos virtuais pequenos pode-se replicar os dados do ambiente em cada estação, deixando que o tráfego na rede limite-se as atualizações. Mundos virtuais maiores e mais complexos podem ser particionados de forma que cada estação possua as informações relacionadas à parte do mundo virtual em que se encontra.

Fig. 2.2 Ambiente com dados compartilhados distribuídos



2.4 SISTEMAS DE AMBIENTE VIRTUAL DISTRIBUÍDO

Aqui serão apresentados resumidamente características básicas de alguns dos principais sistemas DVE existentes ou em desenvolvimento, alguns provenientes do mundo da internet e outros do mundo da simulação militar. Estes sistemas possuem abordagens diferenciadas com relação aos problemas apresentados para o desenvolvimento de DVEs

2.4.1 MASSIVE

Segundo Greenhald (1995), o MASSIVE (*Model, Architecture and System for Spatial Interaction in Virtual Enviroments*) é um projeto desenvolvido na Universidade de Nottingham, Inglaterra. Este sistema permite que usuários comuniquem-se por qualquer combinação de texto, gráfico ou áudio sobre redes LAN ou WAN. O objetivo principal é o desenvolvimento de ambientes virtuais colaborativos escaláveis, que possuam possibilidades de acesso heterogêneas. Em um caso extremo, usuários em sistemas dedicados de realidade virtual poderiam manter algum tipo de interação mínima com outros usuários que estejam acessando de terminais texto.

Ainda segundo Greenhald (1995), o sistema possui controle de consciência da presença de outros objetos (*awareness levels*) baseado nos conceitos de Aura, Focus e Nimbus sendo que, baseando-se nestes parâmetros os atores podem interagir limitadamente com o mundo virtual em que estão inseridos. O parâmetro de Aura de um ator define o seu limite de comunicação com outros atores, pois dois atores somente podem interagir quando suas auras interceptam-se. O parâmetro de Focus determina a quais entidades o ator está dedicando sua atenção, enquanto o Nimbus define o alcance de sua capacidade de ser detectado por outros. Ao manipular estes três parâmetros um usuário pode controlar a capacidade de outros atores detectarem-no e iniciarem uma comunicação. Por outro lado, também permite que um ator limite a sua comunicação a somente aqueles outros atores de seu interesse. Este sistema de controle de percepção também se encontra presente na plataforma DIVE que será apresentada no capítulo 3.

De acordo com Raposo (2000) o sistema MASSIVE também possui um controle de mediação espacial (*Spatial Mediation*) que faz com que a percepção de outros atores, através de diferentes mídias seja influenciada por fatores como a posição e orientação relativas. Por

exemplo, um usuário mantendo o mesmo tom de voz é ouvido com um volume mais alto ao aproximar-se de outro.

2.4.2 NPSNET

O pacote NPSNET foi desenvolvido na *Naval Postgraduate School* para suportar simulações virtuais de treinamento militar em larga escala. Atualmente suporta centenas de usuários, mas objetiva suportar milhares de usuários em uma área de vasta extensão (Npsnet, 2000).

Este sistema baseia-se no padrão DIS para realizar a comunicação entre as entidades envolvidas. O sistema distribui os pacotes DIS utilizando *IP-Multicast*, com o objetivo de minimizar o número de pacotes enviados.

Originalmente, a maioria dos objetos comunicantes consistia de veículos militares. Posteriormente foram desenvolvidos modelos que pudessem representar a figura humana com articulações, que tem sua movimentação efetuada por programas *script* reagindo a comandos simples.

O sistema replica, por ser parte do padrão DIS, a base de dados relativa a todos os objetos inseridos no ambiente e em cada estação, criando assim um fator limitante para uma aplicação com milhares de usuários devido ao alto custo individual das estações necessárias. Atualmente o NPSNET está desenvolvendo o suporte a divisão dos objetos em classes espaciais, temporais e funcionais, com o objetivo de minimizar o volume de informação armazenada em cada estação.

2.4.3 SPLINE

O sistema SPLINE (*Scalable Plataform for Large Interactive Networked Enviroments*) desenvolvido pelo MERL (*Mitsubishi Eletric Research Labs*) possui o objetivo de criar ambientes virtuais multiusuários para trabalho cooperativo (Raposo, 2000). A comunicação entre os participantes pode ser realizada por simulações tridimensionais ou por linguagem falada natural. Ao conteúdo de objetos de um determinado mundo virtual podem ser acrescidos novos objetos temporários ou permanentes desenvolvidos por usuários, ou seja, o desenvolvimento do ambiente virtual não fica apenas a cargo dos implementadores dos

60sistema. É definida uma interface aberta para a interligação destes mundos criados por diferentes usuários.

Segundo Barrus (1996), o mundo virtual no sistema SPLINE, é particionado em zonas conhecidas por locais. Cada local possui um sistema próprio de coordenadas e seus endereços correspondem aos endereços *multicast* usados para o envio de informações sobre o local e seu conteúdo. Também podem ser definidos locais dentro de outros locais. Por exemplo, em uma cidade virtual, um prédio pode ser um local, dividido em vários locais, como salas, elevadores, etc.

Neste capítulo, procurou-se abordar aspectos, de forma geral, referentes à ambientes virtuais distribuídos. Além do aspecto histórico, foram apresentados alguns problemas inerentes ao desenvolvimento destes ambientes, buscando com isso demonstrar também as técnicas existentes para solucionar os mesmos problemas. Por fim foram vistos alguns sistemas de ambientes virtuais distribuídos e também suas principais características.

O próximo capítulo apresenta uma visão mais detalhada da plataforma de desenvolvimento DIVE , a qual será utilizada para o desenvolvimento do ambiente virtual.

3 A PLATAFORMA DIVE

Visando o estudo de aspectos como distribuição, colaboração, interação e participação de múltiplos usuários em ambientes virtuais distribuídos, os pesquisadores do *Swedish Institute of Computer Science* (SICS) desenvolveram em 1991, uma plataforma experimental, inicialmente denominada de *TelePresence* e que mais tarde passou a DIVE (*Distributed Interactive Virtual Environment),* possibilitando assim a criação de um ambiente virtual compartilhado onde diversos usuários puderam encontrar-se e interagirem com outros.

A proposta inicial para o desenvolvimento desta plataforma foi o de permitir o suporte a diversos usuários em um ambiente interativo, apresentando uma interface gráfica simples, mas segundo Stenius (1996), a plataforma tem sido continuamente utilizada na criação de diversos tipos de aplicações, desde as que abrangem os principais fundamentos em nível de rede até aplicações com diversos métodos de interação em ambientes colaborativos com regras sociais pré-definidas. Isto permitiu o desenvolvimento rápido da plataforma, e atualmente ela permite a criação de aplicações usando-se texturas, diversos tipos de animações, utilização de arquivos de vídeo e som e integração com a *World Wide Web*.

Frécon (1998) conceitua o DIVE como uma plataforma que permite o desenvolvimento de ambientes virtuais distribuídos através da utilização de diversas funções existentes na mesma, diferentemente de Táxen (2000), que considera o DIVE não como uma aplicação ou um programa, mas sim como um protocolo para criação, manipulação e visão de mundos virtuais. Na realidade o DIVE é formado por um conjunto de bibliotecas, implementadas utilizando-se basicamente a linguagem C e Tcl, e que através da linkagem destas em um ambiente de programação, permitem a criação de aplicações virtuais distribuídas. Os principais módulos que formam este conjunto de bibliotecas bem como suas principais características são apresentadas abaixo:

- ***Threads*** – Biblioteca responsável pelo interfaceamento entre o ambiente virtual e dispositivos de I/O multiplexados, como luvas e capacetes para interação e visualização entre o usuário e o ambiente.

- ***SID*** – Biblioteca básica para suporte à comunicação através de IP *multicast* e SRM (*Scalable Reliable Multicast).*

- ***DIVE Graphics*** – Esta biblioteca é responsável pela implementação do serviço de renderização da parte gráfica. Essencialmente, a função de renderização é chamada a cada vez que um novo *frame* é desenhado. A função então interage com a base de dados que contém as entidades completas, e renderiza a representação gráfica das mesmas.

- ***DIVE Áudio***- Módulo responsável pela implementação de um sistema de áudio 3D quando houver necessidade de utilização de som em alguma entidade.

- ***DIVE Aux*** – Possui diversos sub-módulos que serão encapsulados no arquivo executável criado após a compilação do aplicativo Dive, e que serão utilizados para chamada de outras bibliotecas externas. Um exemplo é a possibilidade de utilização de scripts em Tcl, no desenvolvimento de algum objeto, e que serão usados para executar algum evento determinado. Ou ainda a utilização de comandos em Tk que possibilitam criar uma interface com o usuário.

- ***DIVE Core*** – A biblioteca DIVE *Core* contém as funcionalidades básicas necessárias para o desenvolvimento de "processos Dive" e é através dela que serão encapsuladas todas as outras bibliotecas necessárias para o desenvolvimento de aplicações Dive. As principais funcionalidades existentes são:

    - Suporte – Responsável pela implementação das estruturas de dados necessárias para criação do ambiente virtual, bem como pela entrada de dados e de configurações.

    - Objetos – Responsável pela criação das entidades que irão compor o ambiente virtual, incluindo as funções de intersecção e de inicialização destas entidades.

    - Distribuição – Utiliza as funcionalidades de comunicação existentes na biblioteca SID para introduzir na rede as entidades que tiverem sido criadas.

Convém ressaltar que a biblioteca Dive Core é o principal módulo utilizado para o desenvolvimento de aplicações na plataforma DIVE. Por isso sua arquitetura será melhor apresentada à partir do capítulo 4.0.

Atualmente, a plataforma encontra-se na versão 3.3 e seu código fonte é distribuído gratuitamente, sob uma licença para uso não comercial e está disponível para diversos sistemas operacionais como IRIX, Solaris, Windows NT/2000 e Linux.

3.1 PRINCIPAIS CONCEITOS DA PLATAFORMA DIVE

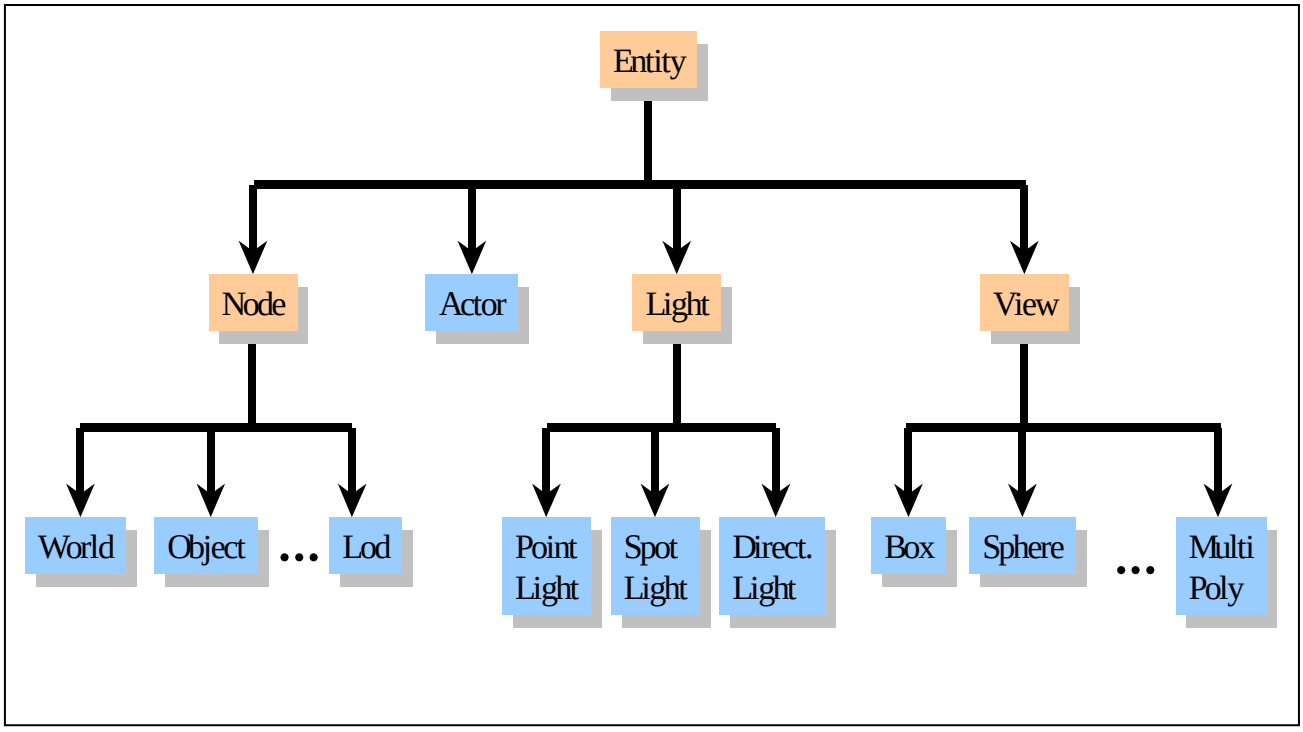
Para possibilitar o desenvolvimento de aplicativos virtuais utilizando o DIVE é necessário compreender os principais conceitos utilizados na plataforma.

3.1.1 ENTIDADES DISTRIBUÍDAS

De acordo com Stenius (1996) um conceito central da arquitetura de programação do DIVE é o compartilhamento da base de dados, onde cada participante conectado a um mundo virtual, possui localmente uma cópia com a descrição deste mundo e as atualizações são continuamente enviadas na rede da qual o usuário está fazendo parte.

Segundo Hagsand (1996) as entidades são os elementos básicos existentes em uma base de dados e que podem ser endereçadas, requisitadas ou distribuídas. Uma entidade é considerada como uma classe hierarquicamente constituída de diversos componentes que irão formar um ambiente conforme mostrado na figura 3.1.

Fig. 3.1 Diagrama hierárquico de componentes de uma entidade



As entidades são estruturadas em um formato de árvore, onde a raiz é o mundo propriamente dito e as folhas são formadas pelos componentes chamados de nós e que contém os objetos que estão inseridos neste mundo, os componentes de luz onde estão as definições de iluminação utilizadas pelo ambiente e pelos objetos, e os componentes de visões que constituem a representação gráfica em 3D que formarão os objetos que constituirão o ambiente virtual. Os componentes descritos como atores são as representações virtuais dos usuários que navegam pelo ambiente virtual e que são ativadas pelos processos requisitados pelas aplicações ao conectarem-se à este determinado ambiente.

Quando uma aplicação Dive conecta-se a um mundo (diga-se Entidade), comenta Frécon (1998), elas operam somente sob a abstração deste mundo e não se comunicam diretamente com alguma outra aplicação também conectada a ele. Em virtude disto à persistência da entidade é garantida somente enquanto uma das aplicações estiver interagindo com o mundo. Quando a última aplicação é finalizada o mundo “morre” e a entidade deixa de existir. Se durante o tempo em que as aplicações estiveram ligadas a esta entidade, tiverem ocorrido mudanças na estrutura da mesma como por exemplo a inclusão ou alteração de algum objeto no mundo virtual, e não houver sido salvo esta entidade atualizada na base de dados dos participantes, as próximas aplicações que conectarem-se à este mundo irão carregá-lo no estado inicial em que ela encontrava-se.

Segundo Frécon (1998), tem se estudado algumas maneiras para que nas próximas versões da plataforma DIVE, exista algum processo que trate o problema da persistência, possibilitando assim um meio de periodicamente salvar o estado da entidade. Muitas questões precisam ser ainda investigadas para que isso ocorra como por exemplo, como delegar responsabilidades para persistência de diferentes partes do mundo e como tratar a posse de um objeto de uma forma geral.

3.1.2 COMPONENTES DE UMA ENTIDADE

A hierarquia que forma uma entidade virtual pode ser composta de diversos componentes, sendo que os principais são apresentados a seguir.

3.1.2.1 OBJETOS DIVE

Segundo Táxen (2000) todas as representações gráficas que constituem um mundo são conhecidas como objetos, inclusive o próprio mundo. A diferença existente entre um objeto qualquer e um objeto mundo é que este contém um conjunto de parâmetros especiais como, por exemplo, a especificação de uma cor de fundo para o ambiente ou ainda o ponto de entrada por onde algum usuário entrará neste mundo, ou seja o ponto inicial em que o Avatar será carregado no mundo.

Os objetos mais comuns quando são criados além de carregar toda sua descrição lógica, podem apresentar também informações referentes à sua capacidade de interação com um Avatar ou mesmo com o ambiente em que ele está inserido e também informações que farão com que este objeto torne-se parte do contexto do ambiente. Ou seja, um objeto contém todas as informações de descrição e orientação geométrica, descrição de materiais, texturas, densidades que formarão este objeto e também podem apresentar comandos responsáveis pelo controle da interação do objeto com algum processo. Um exemplo simples de criação de um objeto é mostrado no quadro 3.1 .

Quadro 3.1 Exemplo de código utilizado para criar um objeto cilíndrico

```
#include "dive.vh"
object {
  name "top"
  material {
    diffuse 0.996 0 0
    ambient 0.996 0 0
  }
  translation
    v    51.6475 0.8  59.8231
  rotation
    v   0.760518   0  -0.649316
    v          0   1     0
    v   0.649316   0   0.760518
  view {
    material_index 0
    texture_index 0
    SPHERE  1  1   1
  }
}
```

O processo de interação de um objeto com algum usuário ou com o ambiente é feita através da utilização de *scripts* feitos em Tcl e que são diretamente associados a este objeto na implementação da estrutura do mesmo, e podem ser disparados quando o objeto é carregado

no ambiente virtual ou através de eventos executados diretamente por um participante do mundo. Um exemplo é o quadro 3.2 onde existe a criação de uma esfera vermelha, que possui em sua implementação um trecho de código em Tcl, que faz com que esta esfera execute uma ação determinada, no caso a esfera pula e se mantém no ar durante um tempo estipulado, assim que o usuário clicar no objeto.

Quadro 3.2 Criação de um objeto cilíndrico que possui interação com o usuário

```
#include "dive.vh"
object {
  name "top"
  prop "Class" "string" "Tcl:Dive" GLOBAL_PROP
  material {
    diffuse 0.996 0 0
    ambient 0.996 0 0
  }
  translation
    v   51.6475 0.8  59.8231
  begin.tcl
      proc on_interaction {type id stype origin src_id x y z} {
      dive_move [dive_self] 0 1.0 0 0
      dive_sleep 200
      dive_move [dive_self] 0 -1.0 0 0
      }
      dive_register INTERACTION_SIGNAL DIVE_IA_SELECT
                                              [dive_self] "" on_interaction
  end.tcl
  view {
    material_index 0
    texture_index 0
    SPHERE 1  1  1
  }
}
```

Além da possibilidade de interação direta de um usuário com um objeto pode-se, utilizando também *scripts* em Tcl, conseguir um tipo de interação indireta feita pelo objeto para com o usuário. Ela é chamada de detecção e implementa uma área circundante ao objeto, também chamada de aura, que possibilita detectar a presença de um usuário, permitindo assim poder-se ativar um determinado evento, seja ele para executar alguma ação ou ainda para gerenciar a possibilidade de colisão ou não com o objeto, sem que para isso ocorra a ação direta do Avatar.

3.1.2.2 VISÕES DIVE

De acordo com Hagsand (1996), visões são representações gráficas 3D passivas que podem ser dinamicamente criadas e modificadas. A declaração de uma visão é utilizada para

designar o modo que a estrutura de um objeto será representado graficamente. Através da declaração da visão define-se como o objeto será apresentado no mundo virtual e isto pode ocorrer de diversas maneiras. Alguns exemplos que podem ser ressaltados são a possibilidade de criar-se objetos desde os mais simples, como uma caixa ou uma esfera, até objetos mais trabalhados e complexos. Também pode-se ressaltar a possibilidade de declarar-se visões que permitirão à um objeto possuir desde uma determinada textura ou cor, ter alterada sua visibilidade, podendo ser mais ou menos visível, até a possibilidade de utilizar-se textos que podem ser apresentados nos objetos que forem criados.

3.1.2.3 MUNDOS DIVE

Para Hagsand (1996) um mundo é a representação de um espaço virtual separado de outros mundos, sendo formado por diversos objetos e podendo ser compartilhado entre diversos usuários distribuídos em pontos geograficamente separados. Frécon (1998) conceitua os mundos Dive como uma base hierárquica de dados, que podem conter desde a representação gráfica de diversos objetos organizados e definições de dados criadas por usuários, até descrições de comportamento autônomos que ditam regras as quais são necessárias para que usuários possam conectar-se a este mundo e interagirem de forma compartilhada.

Como já foi dito anteriormente, um mundo Dive também é um objeto mas o que o diferencia de outro objeto comum é que ele possui uma série de parâmetros setados em sua criação e que afetam, além de todos os objetos existentes no mundo, também todos aqueles usuários que conectarem-se à ele. Um dos parâmetros de extrema importância na criação de um mundo Dive é chamado *start* e ele define em que posição do ambiente que um Avatar irá ser carregado quando acontecer a conexão deste com o mundo. Outros parâmetros que podem ser citados são:

- *Background* – define a cor que será mostrada como “pano de fundo” do mundo virtual.
- *Fog* – define a intensidade de neblina que pode existir no ambiente, sendo que a cor desta neblina sempre será automaticamente igual à cor utilizada no parâmetro *background*.
- *Terrain* – Define qual será a descrição do terreno utilizado no mundo, sendo que o mesmo só é carregado quando algum usuário se conecta.

3.1.2.4 ATORES

Um ator, que pode ser comandado por um usuário ou ainda ser um processo autômato, representa a personificação gráfica do usuário conectado ao mundo virtual e que tem a capacidade de realizar diversas ações neste ambiente. O ator pode modificar objetos e parâmetros e enviar mensagens para outros objetos pertencentes a este mundo, mensagens estas que resultam em mudanças concretas na base de dados.

Segundo Stenius (1996) um ator no DIVE é chamada de *virtual body* ou ainda de *body icon*, e além de representar o usuário também indica a posição virtual em que se encontra o personagem e também o seu ângulo de visão. Na figura 3.2, pode-se ver três usuários participando de uma conferência virtual. Eles são representados no ambiente virtual por complexos atores que são capazes de demonstrar diferentes posturas dependendo do que eles estiverem realizando no momento.

Fig 3.2 Um ambiente DIVE representando uma sala de conferências

De acordo com Hagsand (1996) o DIVE trata muito bem um dos principais problemas que podem ocorrer quando existem diversos atores compartilhando um determinado ambiente virtual. É o problema de concorrência de processos para modificação de um determinado objeto neste mundo, que pode acontecer desde que exista a possibilidade de qualquer ator

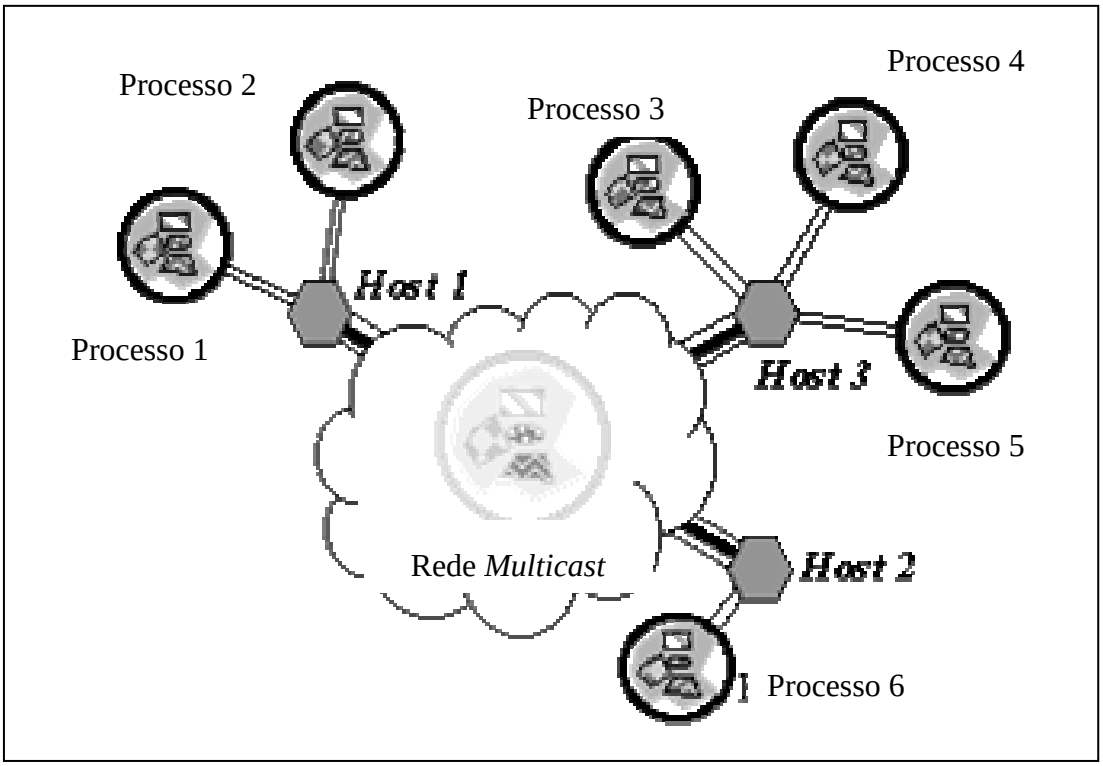
modificar algum objeto presente, e não apenas o criador do mundo. No DIVE, quando algum ator requisita uma modificação nos parâmetros de um objeto, este fica em um estado de bloqueio para qualquer outro ator, e não pode sofrer mais nenhuma alteração enquanto a solicitação anterior não tiver sido atendida e o objeto não tiver sido desbloqueado. Isto é feito através da utilização de um algoritmo de passagem de *token*, que envia um sinal para o objeto, bloqueando-o para qualquer outra solicitação de alteração e que só voltará a ser liberado quando o sinal de *token* retornar para o ator que solicitou a alteração.

Outra funcionalidade existente na plataforma, é a possibilidade de um ator mudar de mundo dinamicamente, utilizando um *gateway*. Um objeto pode possuir características de um portal que serve como porta de comunicação de um mundo para outro. Quando um ator passa por um objeto do tipo portal é disparado um sinal que requisita o novo endereço do mundo determinado no objeto portal fazendo assim com que o ator seja transferido para este novo mundo, carregando consigo todas as características e comportamentos que este possuía no mundo em que se encontrava.

3.2 REPLICAÇÃO DA BASE DE DADOS

A plataforma DIVE utiliza um sistema de replicação ativa de partes da sua base de dados para que exista sempre uma cópia das entidades que são conectadas através de um processo gerado por uma aplicação. De acordo com Frécon (1998) este modelo permite um melhor desempenho em virtude de possibilitar uma baixa latência nas interações entre o ambiente virtual e seus usuários, já que as aplicações acessam a base de dados que contém os mundos, diretamente da memória. Um exemplo pode ser verificado na figura 3.3, onde seis processos interagem um com o outro tendo em comum um mundo virtual simples. Cada processo possui sua própria cópia do mundo em si e também dos objetos mais relevantes contidos nele, mas o mundo todo pode ser visto como residente na rede.

Fig. 3.3 Uma base de dados distribuída entre seis processos interagindo conjuntamente



Este processo de replicação acontece quando alguma entidade ou objeto é adicionado, removido ou modificado, sendo que estes processos são executados primeiramente na cópia local da base de dados e somente depois de executada é distribuída para todos os pontos conectados na rede, isto através de uma mensagem que parte da aplicação que executou o processo de alteração para os outros processos ativos. Por esta razão que a replicação da base de dados é considerada ativa. Este conceito pode levar a uma interpretação da existência de uma central global da base de dados residindo em algum local da rede, mas na realidade isto não acontece pois a base de dados é replicada identicamente para cada processo conectado à ela.

Frécon (1998) comenta que tradicionalmente, os sistemas que utilizam base de dados distribuída, na maioria das vezes obrigam que todos os processos existentes concordem com as informações de alteração da base de dados, antes que estas ocorram.

Isto não acontece com a plataforma DIVE, pois além da utilização de um sistema de replicação parcial da base de dados que minimiza os problemas existentes na utilização deste modelo, também foram utilizadas outras técnicas que permitem uma maior focalização no

desenvolvimento de aplicações altamente interativas. Uma destas técnicas é a utilização de *Dead-reckoning* e de mecanismos de atualização de objetos *run-time*, que permitem que as aplicações desenvolvidas utilizando-se a plataforma DIVE, tolerem que as cópias existentes dos mundos possam ser ligeiramente diferentes, implementando para isso serviços que garantam sua igualdade na maioria das vezes.

Como já foi comentado anteriormente, todas as modificações efetuadas no mundo virtual são sempre aplicadas, primeiramente na base de dados do processo que requisitou a modificação no mundo, para somente depois disto feito ser enviada para os outros processos conectados esta alteração. Um dos problemas ocasionados por este processo de atualização é que existe um tempo de latência entre a atualização local, o empacotamento da mensagem, seu envio para as outras estações conectadas, o posterior recebimento deste pacote e enfim a efetiva atualização das outras bases de dados. Esta latência varia de acordo com a distância em que se encontram os processos conectados à rede uns dos outros. Para sanar este problema, o DIVE possui implementado um sistema que executa sincronizações periódicas utilizando números seqüenciais, e que mantém organizadas todas as entidades que sofreram atualizações. Como resultado todos os pontos conectados podem possuir arquivados o mesmo estado inercial para estas entidades, tornando possível requisitar alguma entidade existente neste arquivo e que não esteja alterada no ambiente virtual.

Outros problemas que podem ocorrer com a utilização de dados distribuídos e com o constante envio de mensagens na rede, é o congestionamento da mesma e a conseqüente sobrecarga de tráfego. Para remediar estes problemas, e para prover a possibilidade de compartilhamento de um ambiente em comum por dezenas de usuário, o DIVE possui um mecanismo que pode dividir um mundo em sub-hieraquias, que podem ser replicados em um grupo menor de aplicações onde existam usuários que possuam algum interesse em comum. Cada sub-hierarquia passa a ser associada com um canal de comunicação *multicast*, chamado de *light-weight group*. Como resultado disto, processos que não estão interessados nestas sub-hierarquias podem simplesmente ignorar este ramo da árvore, reduzindo assim o tráfego na rede. Este assunto é melhor tratado na seção 3.3.3 grupos de interesse.

3.3 ARQUITETURA DE COMUNICAÇÃO DO DIVE

A plataforma DIVE apresenta conceitos distintos referentes à arquitetura de comunicação utilizada, sendo que os principais são apresentados a seguir.

3.3.1 COMUNICAÇÃO PONTO A PONTO

A arquitetura de comunicação utilizada pela plataforma DIVE é oposta ao clássico modelo cliente-servidor e baseia-se em um modelo de distribuição onde os processos são conectados no formato ponto a ponto através da utilização de protocolos *multicast*.

Segundo Hagsand (1996) este modelo foi adotado pois, como a arquitetura cliente-servidor exige que os processos conectados a rede tenham que se comunicar com o servidor sempre que uma entidade é modificada, isto gera implicações negativas no tempo necessário para uma interação real e efetiva e interfere no foco principal da plataforma que é permitir o desenvolvimento de aplicativos altamente interativos.

Quando um processo Dive deseja unir-se a algum mundo ele deve primeiramente contactar um servidor de nomes que forneça o endereço *multicast* do mundo ao qual ele deseja juntar-se. Uma descrição inicial de um mundo pode existir em um sistema remoto acessível através de um endereço URL (*Universal Resource Locator)*, só que somente um membro pode carregar esta descrição inicial ao iniciar-se uma sessão. A partir deste momento, todos os processos subseqüentes irão requisitar o *status* deste mundo a partir de um outro ponto já conectado a esta sessão. Todos os novos processos que desejarem conectar-se, irão requisitar o estado inicial do mundo através do endereço *multicast* deste e receberão uma réplica atualizada com a última versão do mundo diretamente dos pontos conectados e ativos nesta sessão. Assim todos os pontos conectados no mesmo grupo *multicast* interagem fazendo acessos concorrentes à base replicada dos dados e enviando mensagens entre si.

Segundo Frécon (1998) o DIVE possui implementado dois tipos de mensagens que podem ser enviados durante uma sessão e que correspondem a diferentes tipos de dados. Uma delas é a mensagem que contém todas as alterações efetuadas no ambiente virtual, enviadas através da utilização de um protocolo *multicast* confiável que é utilizado para minimizar o número de mensagens duplicadas entre os transmissores e os receptores. Este protocolo *multicast* confiável é necessário para garantir que todos os processos possuam cópias iguais dos mundos.

Outro tipo de mensagem utilizada é aquela responsável pelo envio de dados contínuos, como por exemplo áudio e vídeo, e que são enviados utilizando-se um protocolo *multicast* não confiável. A diferença na utilização de um protocolo confiável, nas mensagens com modificações da base de dados, e não confiável para outros tipos de dados é que, neste caso,

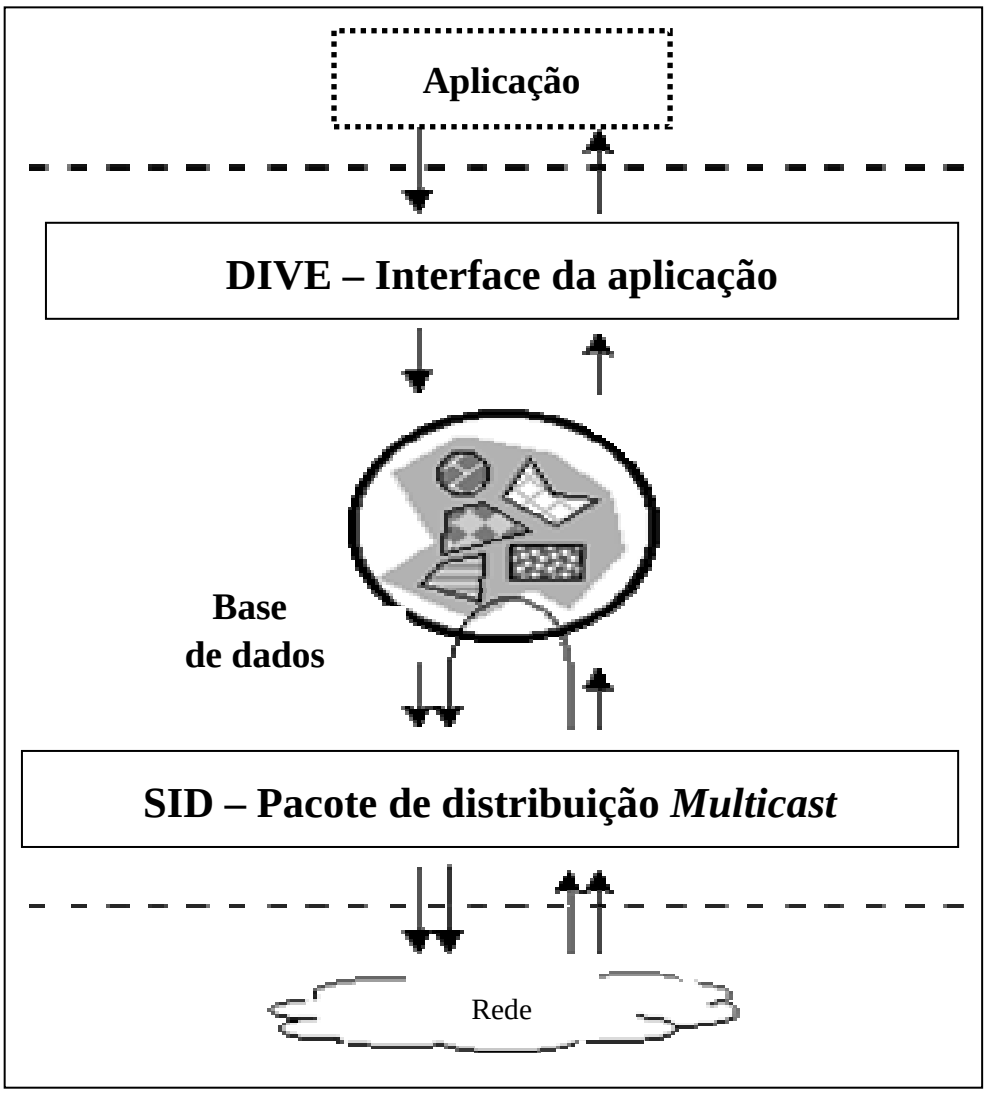
não existe a necessidade de manter-se uma consistência, mas apenas manter a continuidade no envio de arquivos de som ou de imagens utilizando-se para isto uma série de mensagens distintas.

3.3.2 PROTOCOLOS MULTICAST E DE REDE

Para solucionar os problemas de transporte de pacotes contendo as mensagens na rede, os pesquisadores do SICS (*Swedish Institute of Computer Science)* desenvolveram uma biblioteca denominada SID, que foi incorporada a plataforma DIVE, e que possibilita reduzir a quantidade de mensagens enviadas na rede minimizando assim o tráfego nela aumentando sua escalabilidade. O método utiliza bastante o sistema *multicasting*, realiza comunicação baseada em entidades e é confiável pois utiliza um esquema negativo de reconhecimento de requisição e resposta.

Neste sistema os objetos comunicados são equivalentes as entidade Dive. O protocolo enviado pelos pontos não necessita armazenar mensagens até sua chegada ser confirmada por todos os recipientes, como é feito no TCP por exemplo. Ao invés disto, pode haver uma requisição da réplica dos últimos objetos sendo feita diretamente por uma aplicação local utilizando uma chamada. Uma visão simplificada do processo de comunicação utilizada pelo DIVE pode ser verificada na figura 3.4, bem como a necessidade da utilização dos mecanismos de chamada utilizando um sistema *multicast* confiável e escalável, que ativa a comunicação entre o protocolo de rede e a base de dados.

Fig. 3.4 Processo de comunicação utilizado pela plataforma DIVE



Frécon (1998) comenta que na utilização deste processo, se um ponto ativo na rede detecta a perda de uma mensagem de atualização ou de requisição de objeto, ele simplesmente faz novamente uma requisição deste objeto associando a ele um endereço *multicast*. Assim o ponto mais próximo que possuir a última versão deste objeto responde a solicitação, também utilizando o *multicast* para inibir outras réplicas similares. Com isto a rede não é sobrecarregada pelo transporte inútil de réplicas.

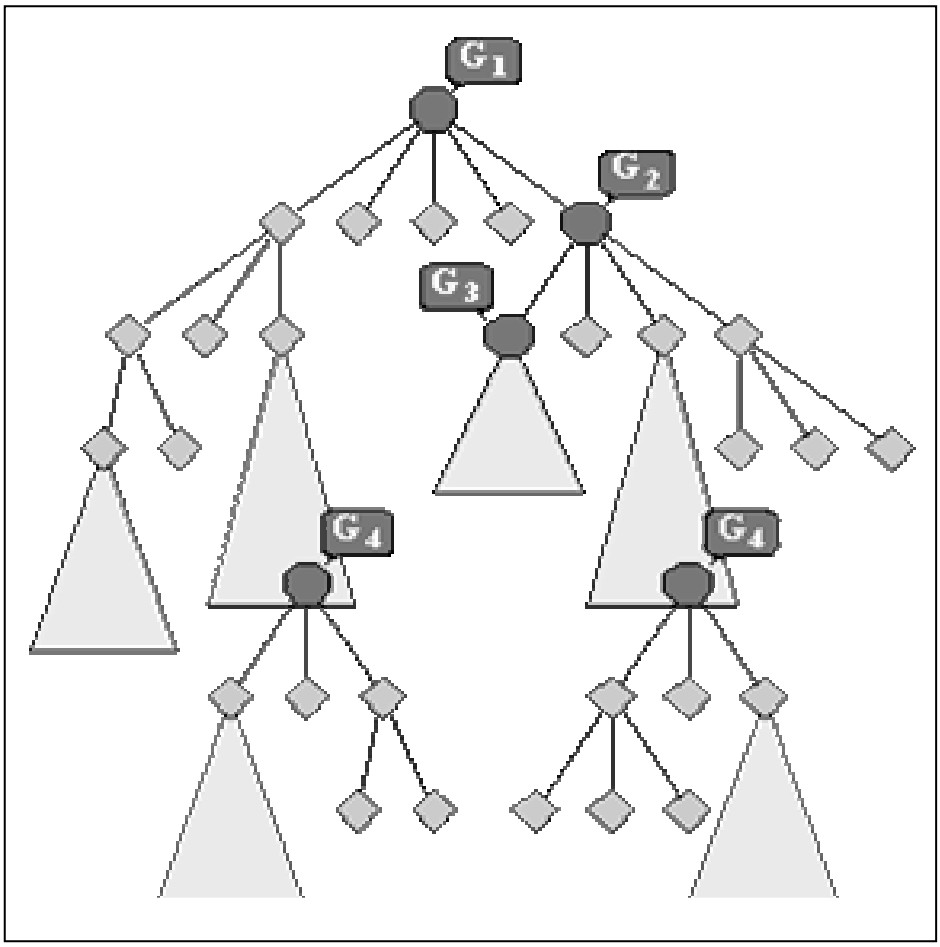
3.3.3 GRUPOS DE INTERESSE (LIGHT-WEIGHT GROUPS)

Como foi mencionado anteriormente, o DIVE utiliza dois diferentes níveis de comunicação *multicast* na hierarquia dos mundos. A entidade mais ao topo da hierarquia está associada com um grupo *multicast* que é utilizado por padrão em todas as comunicações de

dados feitas dentro deste mundo. Mas toda e qualquer entidade na hierarquia pode ter associado a ela um grupo *multicast* diferente. Quando uma mensagem de modificação referente a uma determinada entidade é enviada, o algoritmo utilizado faz com essa mensagem vá até o topo da hierarquia e inicie seu reconhecimento pela primeira entidade existente na árvore. Se durante este processo de ascensão for encontrado um grupo *multicast*, este pode ser utilizado como um ponto de comunicação intermediário. Se nenhum grupo for encontrado neste processo o Dive utiliza então o grupo padrão associado à entidade mais ao topo da hierarquia.

Um exemplo da utilização de grupos *multicast* para particionar as diversas entidades dentro de uma rede pode ser visto na figura 3.5, onde as entidades associadas a um grupo *multicast* são representadas como um octágono com o nome do grupo marcado. Outras entidades são representadas por losangos e outras hierarquias de entidades são simplificadas através da utilização de triângulos. A entidade G1 é o grupo padrão enquanto G2 e G4 são grupos de interesse criados, sendo que G4 é utilizado por dois grupos diferentes.

Fig. 3.5 Processo de utilização de grupos *multicast* nas entidades de uma rede



A utilização deste método é vantajoso em virtude de minimizar o tráfego excessivo de mensagens na rede, pois com a criação de grupos de interesse, todas as mensagens enviadas utilizam o endereço *multicast* destes grupos criados sendo direcionadas somente para os pontos que fizerem parte destes grupos.

Outra vantagem é que os mecanismos necessários para criação destes grupos podem ser interfaceados e controlados utilizando-se scripts Dive/Tcl que possibilitam um alto nível de abstração para criação dos mesmos e que podem ser associados a qualquer entidade Dive.

3.4 ARQUITETURA *RUN –TIME* DO DIVE

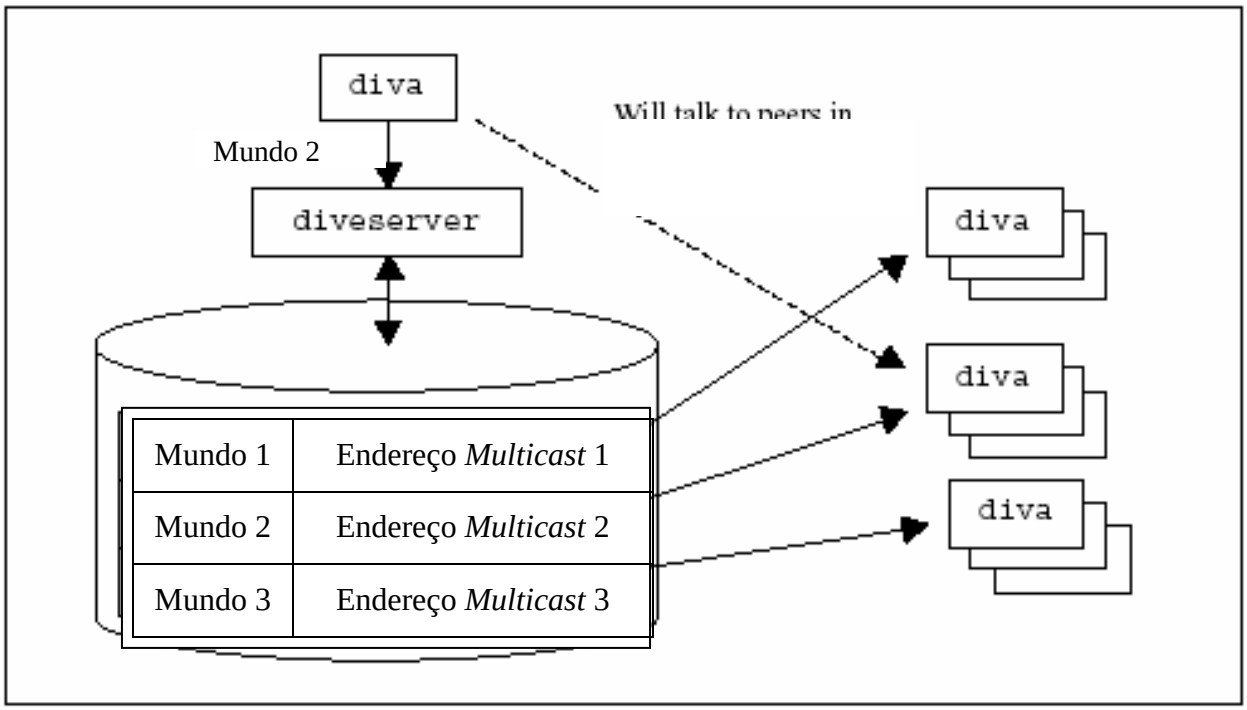
A plataforma DIVE utiliza alguns conceitos de serviços *run-time*, ou seja são aplicações executadas conjuntamente com o ambiente virtual distribuído, auxiliando assim em sua plena execução. Estas aplicações são apresentadas a seguir.

3.4.1 DIVESERVER

De acordo com Frécon (1998) a aplicação chave que permite com que os processos Dive entrem em um determinado mundo é chamado de *Diveserver*. Ele é um servidor de nomes que fornece uma espécie de "*ticket*" de entrada para aquelas aplicações que desejarem conectar-se a um determinado mundo. Este *ticket* é simplesmente composto de um canal *multicast* onde todas as comunicações do mundo serão executadas.

Segundo Táxen (2000) o *Diveserver* é um servidor que contém um lista de nomes de mundos juntamente com seus respectivos grupos de IP's *multicast*. Quando um aplicativo Dive conecta-se ao servidor ele informa qual é o mundo que deseja entrar. Se o mundo desejado estiver na lista do servidor, este então responde com uma mensagem que conterá o endereço do grupo *multicast* deste mundo e que é compartilhado por outros aplicativos Dive. Um exemplo de como este processo acontece é demonstrado na figura 3.6.

Fig. 3.6 Organização do *Diveserver* e as aplicações Dive em modo multi-usuário



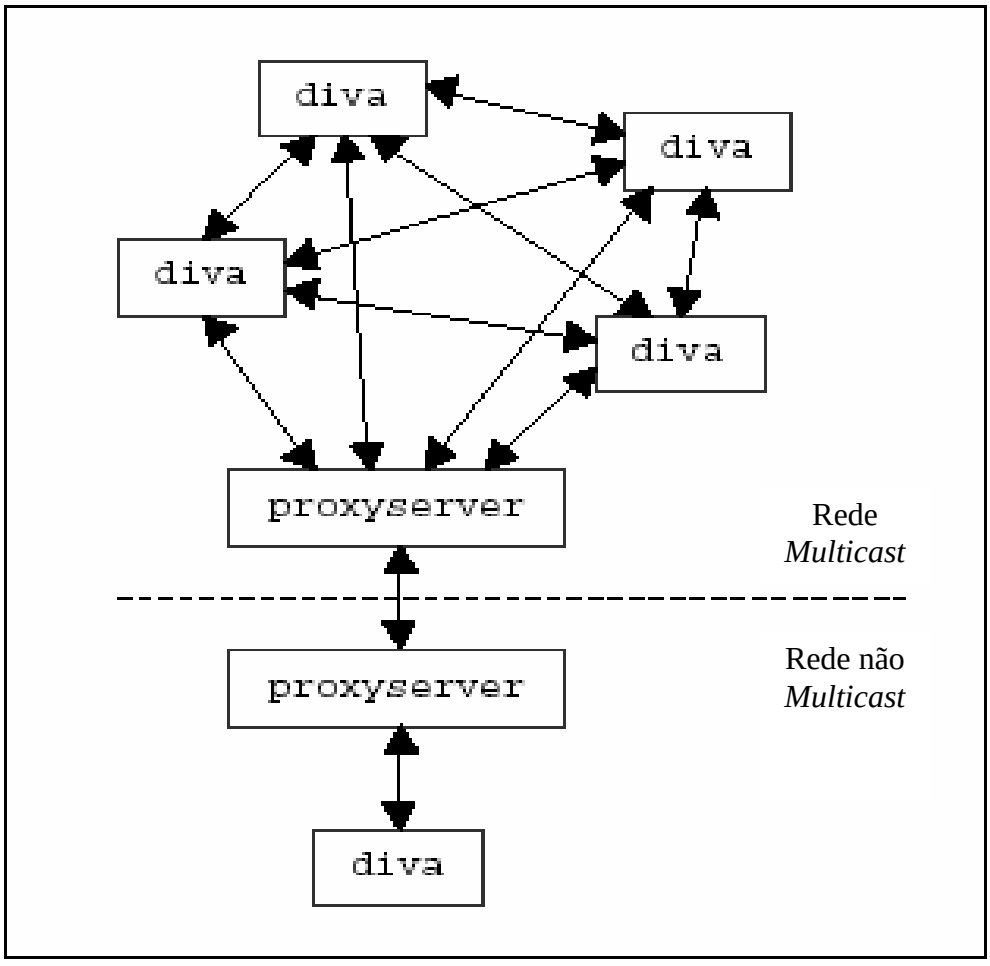
Como o *Diveserver* funciona como um servidor de conexão, o trabalho executado por ele é proporcional ao número de processos que desejam entrar em um determinado mundo ao mesmo tempo e não ao total de usuários já conectados durante a sessão. Isto porque O *Diveserver* é o ponto central necessário com que cada processo Dive deve comunicar-se antes de entrar em um mundo, já que é ele que escala as dezenas de participantes simultâneos que requisitam uma conexão inicial.

Frécon (1998) ressalta que esta comunicação inicial com o *Diveserver* pode ser feita através de um canal *multicast* fixo ou usando a tradicional requisição cliente-servidor ponto a ponto. Uma vez que o processo tenha recebido seu *ticket* ele não precisará mais comunicar-se com o *Diverserver*. Se o processo é o primeiro a se conectar a este *Diverserver*, ele pode ler o estado inicial de algum mundo através de um endereço de URL ou carregando um mundo existente em sua própria base de dados local. Se houver processos já conectados nesta sessão, os novos processos que se conectarem receberão, o mundo completo se não o possuírem em sua base de dados, ou então o *status* em que se encontra este mundo dos processos já conectados.

3.4.2 DIVE PROXYSERVER

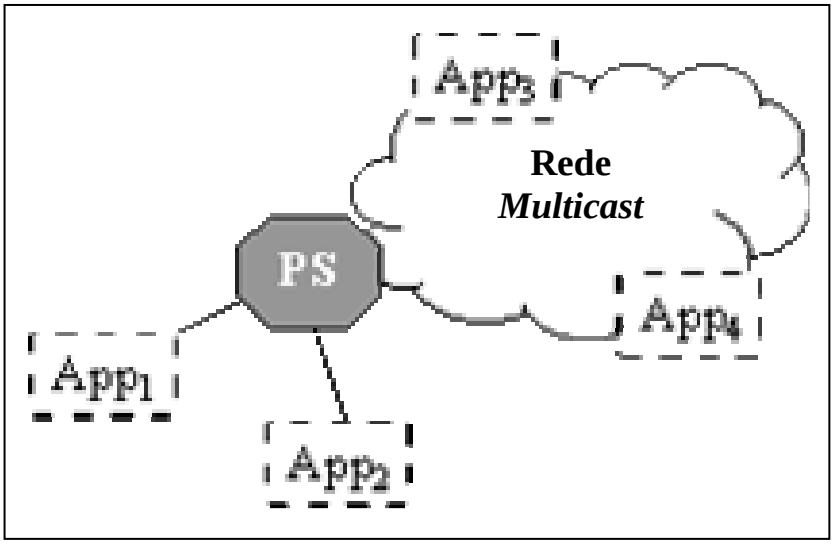
Outra aplicação utilizada no acesso de processos Dive é o *Dive Proxyserver*. Segundo Dive (2001), o *Proxyserver* foi desenvolvido para ser utilizado como uma ponte entre uma rede com capacidade de trabalho *multicast* e outras que não possuam estas capacidades, conforme figura 3.7. Ele é utilizado para criar sessões envolvendo pontos localizados em diferentes locais de uma rede criando assim uma estrutura para comunicação interativa através da formação de um *backbone multicast*.

Fig. 3.7 O *Dive Proxyserver* servindo como uma ponte entre redes sem *multicast*



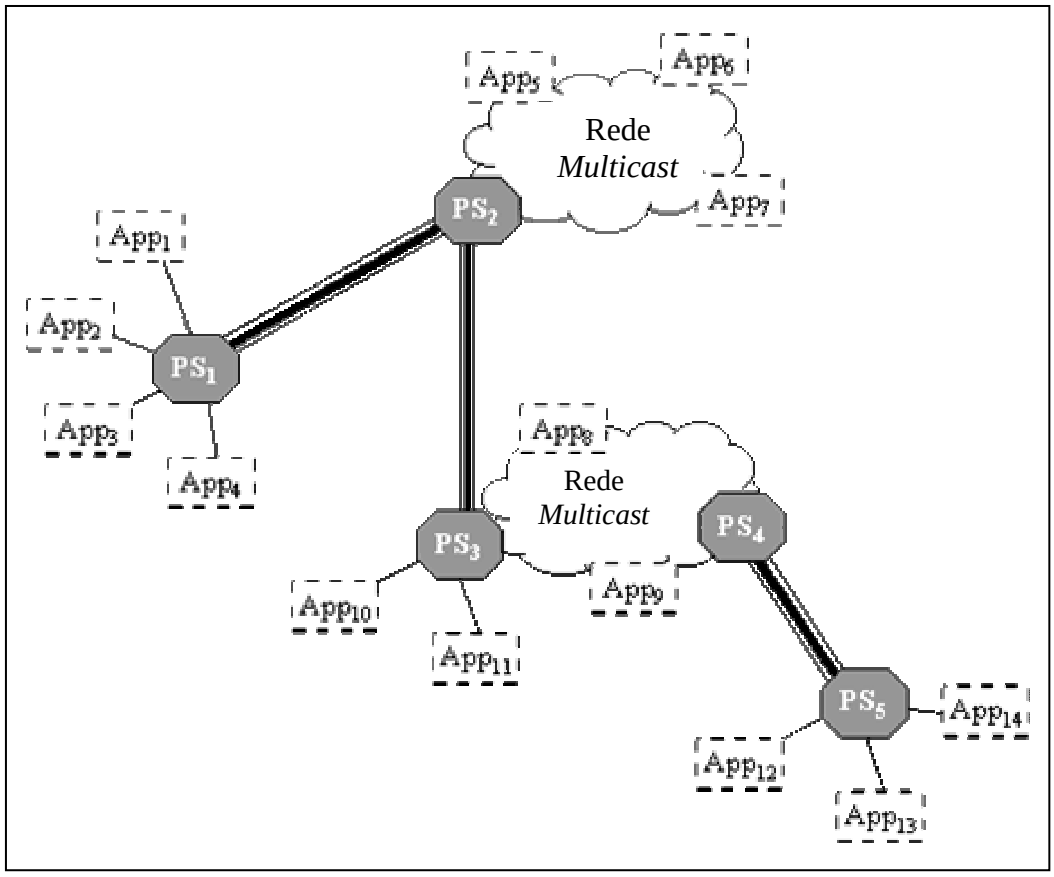
O *Proxyserver* pode atuar como um simples replicador de mensagens para clientes individualmente, replicando todas as mensagens *multicast* do Dive enviadas pelas aplicações, para os outros clientes conectados ou para um determinado grupo *multicast*. Um exemplo de sua utilização é mostrado na figura 3.8.

Fig 3.8 O *Dive Proxyserver* como um simples replicador de mensagens



Outra função que pode ser desempenhada pelo *Proxyserver* é o de servir como um túnel *multicast*, onde diversos *Proxyservers* podem ser unidos entre si permitindo criar uma rede *multicast* conectando diretamente as aplicações Dive. Um exemplo de sua utilização é mostrado na figura 3.9.

Fig. 3.9 O Dive *Proxserver* como um túnel *multicast* a nível de aplicação



Atualmente, a evolução do Dive *Proxyserver* vem sendo direcionada para o desenvolvimento de um sistema mais inteligente que permita executar diversos outros serviços, além da replicação de mensagens, como por exemplo a compressão e pré-formatação de dados nas mensagens antes delas serem enviadas aos clientes. Outra evolução é a possibilidade de uma duplicação automática dos *Proxyservers*, trabalhando com os diversos recursos computacionais e a carga que acontece na rede em virtude da introdução de novos clientes.

3.5 RELAÇÃO ENTRE A PLATAFORMA DIVE E O DIS

Os sistemas que possuem uma maior relação com o DIVE são aqueles que utilizam métodos parecidos para o desenvolvimento de ambientes virtuais distribuídos como por exemplo, controle de consistência, utilização de base de dados particionada e também a não utilização de um servidor central.

Um dos pioneiros na utilização de algumas destas técnicas foi o sistema DIS, e em virtude disto a plataforma DIVE assemelha-se muito à ele. A chave principal para esta semelhança, é que os sistemas escaláveis distribuídos somente podem possuir uma performance em tempo real se a visão dos dados em comum possuírem consistência em todos os pontos que participam de uma simulação. Este ponto é muito forte em ambos os sistemas e a busca da perfeição neste quesito é contínua e árdua. Além deste ponto em comum, pode-se destacar outros pontos que, ao mesmo tempo, aproximam e diferenciam as plataformas DIVE e DIS. São eles respectivamente:

- Como descrito acima, DIVE utiliza o modelo de comunicação ponto a ponto;
- DIVE não necessita de um servidor central (com exceção do *Diveserver*) para possibilitar escalabilidade e tolerância à falhas;
- DIVE utiliza um mecanismo de *dead-reckoning* baseado na velocidade linear e angular de um objeto com isso reduz o número de mensagens geradas quando o objeto se move.

Contudo, o DIVE foi um pouco além do sistema DIS nos seguintes aspectos:

- DIVE é uma plataforma completa para implementação de dispositivos virtuais multi-usuários é não é somente um padrão de comunicação;
- DIVE suporta comunicação utilizando-se áudio e vídeo;

- Os processos que compartilham uma sessão não necessitam obrigatoriamente possuir objetos. Como resultado disto, um processo pode conectar-se, carregar um novo objeto autônomo no mundo e então sair deste sem que com isso o objeto carregado deixe de existir. Isto acontece pois na carga do objeto, suas características são transmitidas para os outros pontos conectados e enquanto estes processos estiverem ativos, o objeto também continuará existindo;
- DIVE permite dividir os mundos em sub-regiões que podem ser associados a grupos de interesses comuns à alguns usuário, facilitando assim o transporte das mensagens de atualização entre os pontos;
- As mensagens de rede utilizadas pelo DIVE não contém objetos inteiros, mas somente os parâmetros que irão definir quais as modificações que serão aplicadas em algum determinado objeto.

No capítulo 3 foram vistos assuntos relacionados à plataforma DIVE. Inicialmente foi apresentada a plataforma de uma maneira geral, destacando seu aspecto histórico e também abordando superficialmente as principais bibliotecas que a compõem. Após foram apresentados os principais conceitos utilizados no DIVE necessários ao desenvolvimento de ambientes virtuais, bem como o modelo de comunicação responsável pelo controle da consistência dos mundos criados. Finalmente são apresentados tópicos referentes à arquitetura de *run-time* utilizada pela plataforma e também as características principais que relacionam a plataforma DIVE e o DIS. O capítulo seguinte apresenta os métodos de desenvolvimento utilizados no Dive para o desenvolvimento de aplicações.

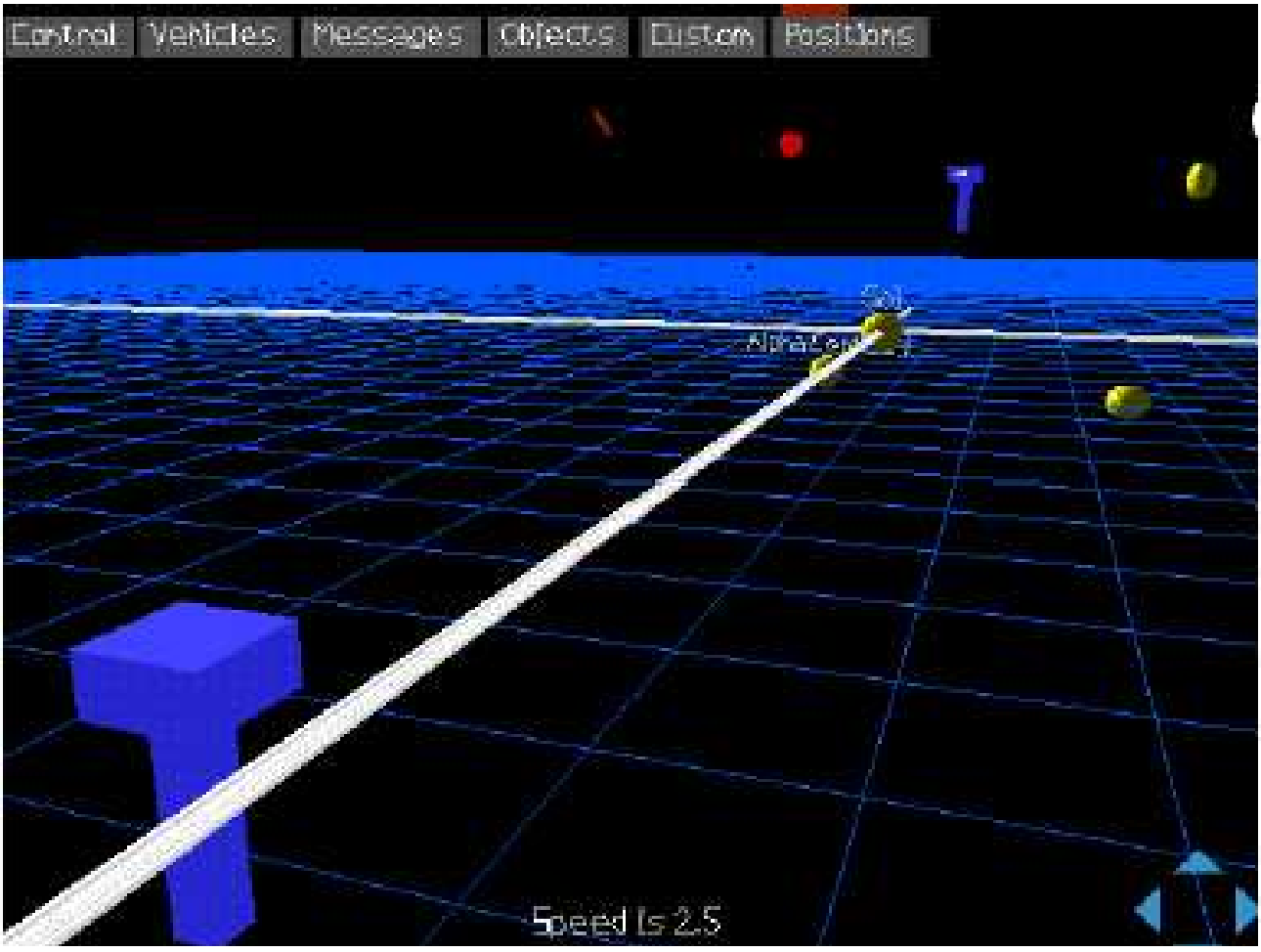
4. DESENVOLVIMENTO DE APLICAÇÕES NO DIVE

Segundo Keukelaar (1996), o modelo básico que melhor descreve a plataforma DIVE é a existência de uma base de dados compartilhada em uma rede, onde existe um conjunto de processos interagindo através de acessos concorrentes a esta base, e também enviando sinais para objetos e outros processos existentes.

Entende-se por processos todos os dispositivos participantes de um mundo virtual, sendo que estes dispositivos podem ser divididos conceitualmente em dois grupos:

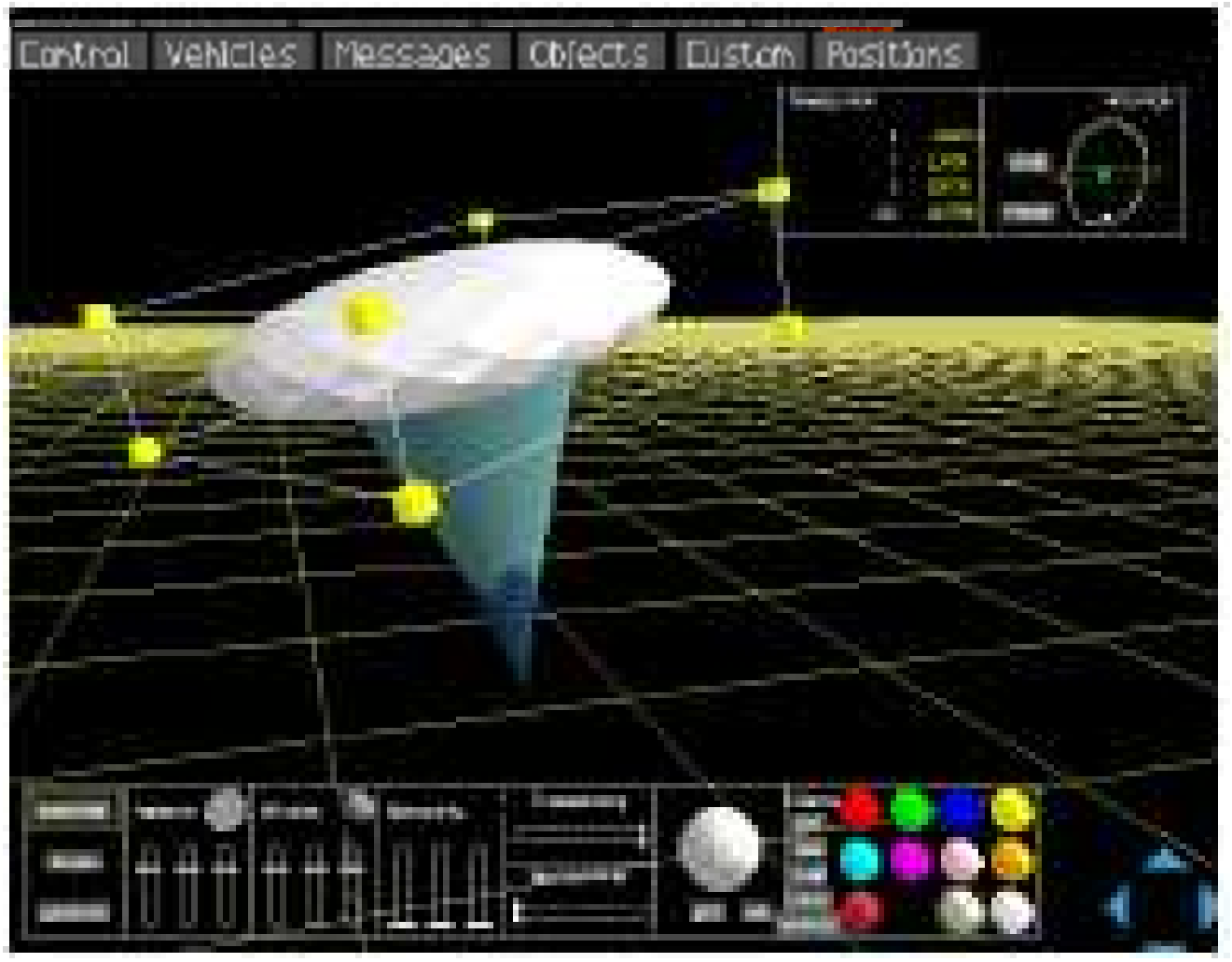
a) Dispositivos de visualização, que apresentam um mundo para usuários humanos permitindo que eles participem ativamente, interagindo com o mundo e com outros usuários presentes, conforme figura 4.1;

Fig. 4.1 Dispositivo de visualização



b) Dispositivos de aplicação, também chamados de AI's (*Application Introduce)* que introduzem aplicações dentro de um mundo virtual, permitindo executar operações em objetos existentes neste mundo, conforme figura 4.2.

Figura 4.2 Dispositivo de aplicação



O desenvolvimento destes dispositivos com a plataforma Dive pode ser feito de três maneiras diferentes. Uma delas é utilizando-se o ambiente de programação C/C++ para criação do sistema de maneira geral, através da linkagem com as bibliotecas do DIVE, sendo necessário no mínimo as bibliotecas *Threads*, SID e a *DIVE Core* para que a aplicação possa funcionar. Estas bibliotecas fornecem funções que são utilizadas para a implementação de todo o sistema, deste a parte de comunicação e distribuição na rede, à parte de interface entre

o sistema e o usuário e além disto também toda a parte de implementação dos ambientes virtuais que serão visualizados nestes dispositivos e compartilhados na rede por outros.

Pode-se também utilizar alguns arquivos arquivos executáveis distribuídos gratuitamente em Dive (2001), que já são compilados e linkados com as principais bibliotecas do DIVE e que fornecem as principais funcionalidades existentes na plataforma, sendo que estes podem ser conectados com aplicações e interfaces criadas utilizando-se as linguagens Tcl/Tk e que permitirão o desenvolvimento do ambiente virtual bem como dos mecanismos necessários para uma interação realista com o mesmo. Este método é bastante eficiente pois permite focar o desenvolvimento de uma aplicação em sua interface de comunicação com o usuário e seus processos, sem que haja a preocupação de desenvolver a parte de comunicação, já que os arquivos arquivos executáveis fornecidos ficarão encarregados disto.

Por último, além de poder-se utilizar os arquivos arquivos executáveis que fornecem as principais funcionalidades do DIVE, pode-se também utilizar um aplicativo, desenvolvido em Tcl/Tk e que também encontra-se disponível em Dive (2001), e que é um visualizador avançado chamado de *Vishnu*, que oferece uma interface bastante prática entre o mundo virtual e o usuário, permitindo desde a conexão com outros mundos virtuais, criação de novos objetos para utilizar no mundo virtual, manipulação do Avatar utilizado para navegação no ambiente, até a utilização de um sistema de *chat* que permite a troca de mensagens entre os usuários conectados em um mesmo mundo. Este método favorece o desenvolvimento, principalmente, do ambiente virtual em si e de todos os componentes que farão parte do mesmo, pois a parte de comunicação e a de interface já são fornecidas pelos arquivos arquivos executáveis e também pelo visualizador.

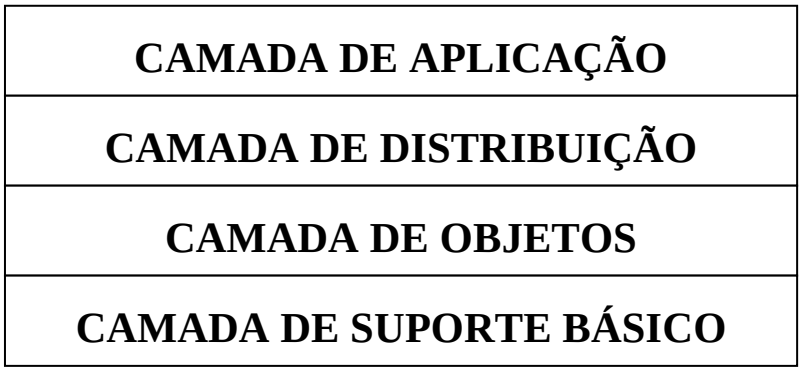
Estes métodos de desenvolvimento são possíveis em virtude da existência de um modelo de implementação, no qual foi baseado a arquitetura do módulo DIVE Core, que é o principal módulo utilizado para a criação de aplicações, e que possibilita uma maior flexibilidade para a implementação de ambientes virtuais distribuídos. Esta arquitetura bem como suas principais características são mostradas a seguir.

4.1 ARQUITETURA DIVE CORE

Conforme foi apresentado no capítulo 3, a plataforma DIVE é formada por um conjunto de bibliotecas que, utilizadas conjuntamente com outras tecnologias, permitem o desenvolvimento de dispositivos virtuais distribuídos. A biblioteca básica necessária para o

desenvolvimento destes dispositivos é a DIVE Core e, é baseando-se em sua arquitetura que os dispositivos podem ser implementados. Esta arquitetura consiste de um núcleo central, e um conjunto de “núcleos externos”. Esta arquitetura pode ser visualizada na figura 4.3.

Fig. 4.3 Arquitetura Dive Core



Cada camada consiste de vários módulos, possuindo funções específicas, sendo que durante o desenvolvimento de uma aplicação Dive, funções de diferentes camadas podem ser chamadas. A arquitetura descreve também que quanto mais alta é a camada, maior é o nível de abstração necessário para sua implementação, pois cada camada pode utilizar determinadas funções, tipos ou variáveis globais que são relacionadas assim com as camadas posteriores a esta. A seguir uma descrição de cada camada existente na arquitetura DIVE Core.

4.1.1 CAMADA DE SUPORTE BÁSICO

Esta camada está relacionada com o interfaceamento do dispositivo Dive para com o sistema operacional utilizado. A camada de suporte interage com o sistema operacional permitindo a comunicação entre ele e o dispositivo Dive.

Um módulo, existente no conjunto de bibliotecas do Dive, e que é bastante utilizado nesta camada é o módulo *Threads* que permite a utilização de diversos dispositivos de entrada e saída, como por exemplo luvas, capacetes para visualização e mais comumente o mouse e o teclado. Esta camada também é responsável pela entrada de parâmetros referentes as variáveis de ambiente existentes no sistema, necessários ao funcionamento correto do dispositivo. Por fim, apresenta também funções definidas como *callbacks*, que são utilizadas para tratar mensagens que podem ser enviadas do dispositivo para o sistema e assim ser visto pelo

usuário. Estas mensagens podem ser referentes a algum erro acontecido no sistema ou ainda algum tratamento de evento relacionado com alguma ação executada pelo usuário.

É na camada de suporte básico que também são utilizadas as funções existentes no módulo SID, que possibilitam implementar todo o protocolo responsável pelo encaminhamento das mensagens ocorridas nos dispositivos virtuais.

As funções existentes na plataforma DIVE e que permitem a implementação desta camada foram todas desenvolvidas utilizando-se a linguagem C, por isso para a criação de algum dispositivo relacionado com a camada de distribuição é necessário, além de um conhecimento básico das funções Dive, também um bom conhecimento na linguagem C.

4.1.2 CAMADA DE OBJETOS

Esta camada está basicamente relacionada com todo o conteúdo da base de dados que integra o dispositivo Dive, incluindo todos os tipos de objetos como mundos, personagens etc. Ela encontra-se abaixo da camada de distribuição, no modelo da arquitetura DIVE Core, mas na realidade é independente tanto da camada de distribuição quanto da camada seguinte que é a de aplicação.

Esta camada é particionada em cinco diferentes módulos responsáveis pela implementação de diversas funcionalidades, sendo elas:

- Objetos *(objects)*;
- Transformação (*transformation);*
- Comportamento *(behaviour);*
- Consciência *(awareness);*
- Personagens *(persons).*

Não existe a necessidade de utilização de todos estes módulos para criação de um objeto. As funções de cada módulo são utilizadas de acordo com as características desejadas para cada objeto.

4.1.2.1 OBJETOS

Os objetos Dive são as estruturas básicas existentes em um sistema pois são eles que irão compor toda a parte gráfica em um ambiente virtual. Os objetos possuem uma determinada

estrutura que contém informações como a posição, o nome e também um identificador para este objeto, e que são utilizados no processo de renderização deste objeto. Os principais campos desta estrutura de composição de um objeto são apresentados abaixo:

- Tipo – É um campo utilizado para determinar uma distinção entre um objeto e outro existente na base de dados. Por exemplo, um mundo e um personagem são ambos objetos mas possuem um tipo diferente que lhes dão características diferentes. Este campo é automaticamente setado quando um objeto é criado e não deve nunca ser alterado;
- Id – É um identificador único para o objeto, também automaticamente setado na criação de cada objeto, e que pode ser utilizado quando existe a necessidade de referenciar-se um determinado objeto em alguma função;
- Material – É o campo onde define-se qual será o material utilizado na visualização deste objeto. Permite também a utilização de texturas para compor este objeto;
- Comportamento – Cada objeto pode ter associado a ele um determinado comportamento que pode ser disparado dinamicamente, e que pode alterar o estado inicial deste objeto fazendo com que ele execute uma determinada ação;
- Rotação – É um campo onde se determina uma matriz 3x3 que nada mais é do que a matriz de rotação local de um objeto. Esta matriz é utilizada para computar a rotação global de um objeto e assim definir qual é a posição de um objeto em um espaço virtual;
- Translação – Campo onde são determinados três pontos que indicarão o ponto de translação de origem de um objeto;
- Visão – Cada objeto pode ter associado um conjunto de visões, que são utilizadas para descrever a aparência de um objeto no processo de renderização deste. Existem ainda classes de visões já definidas como por exemplo*: box, boxvector, polygon, polygon set, grid, ellipse, cylinder, text* e *sphere* que são a implementação de objetos simples e que podem estar contidos na criação de outros objetos, permitindo assim o desenvolvimento de objetos mais complexos;
- Sub-objetos – Objetos podem ser compostos em uma hierarquia onde cada objeto pode possuir um ou mais objetos relacionados a si. Este campo é utilizado para determinar a relação existente entre objetos quando utilizados na composição de um objeto mais complexo;

- *Flags* – Cada objeto possuem um número de *flags* que determinam algum estado para este objeto, sendo que cada *flag* nada mais é do que um valor booleano que irá determinar ou não a utilização daquele estado. Os principais *flags* são:
  - *Visibility* – determina se um objeto será visível ou não;
  - *Nograsp* – determina se um objeto poderá ser manipulado ou não em um ambiente virtual;
  - *Wireframe* – determina se o objeto será mostrado em formato gradeado ou com um formato sólido;
  - *Gateway* – determina se um objeto pode ser utilizado como um portal ou não;
  - *Locallight* – determina se um objeto pode ser considerado como uma fonte de luz. Se este campo for setado, a definição do material para este objeto será interpretado como a fonte de luz e a posição do objeto será o ponto de origem desta;
  - *Collision* – determina se deverá ser executado a checagem de um sinal de colisão durante o movimento de algum objeto.

O módulo de objetos, na camada de objetos, é onde são implementados todas as funcionalidades referentes à criação, inicialização e remoção de objetos da base de dados. Isto pode ser feito de duas maneiras:

- Uma delas é utilizado-se funções implementadas na linguagem C desenvolvidas especificamente para camada de objetos, que fornecem todos os recursos necessários para, não somente criar, mas também inicializar e setar todos os atributos e propriedades existentes na estrutura do objeto. Segundo Andersson (1994) este processo pode ser bastante tedioso pois se utiliza uma grande quantidade de linhas de código para implementar um determinado objeto. Para maiores explicações de como utilizar as funções existentes na linguagem C, consulte Andersson (1994).
- Outra possibilidade é utilizar a descrição de arquivos DIVE que pode ser encontrado em Avatare (1999), que permite criar os objetos de maneira mais abstrata, e sem a necessidade de uma ambiente de programação específico. Este arquivo pode ser escrito utilizando-se apenas um editor de textos, sendo então salvo utilizando-se um tipo *.vr* que pode lido e então carregado diretamente para uma aplicação. Para melhores informações sobre como utilizar a descrição de arquivos Dive consulte Avatare (1999).

4.1.2.2 TRANSFORMAÇÃO

Objetos DIVE podem ser transformados de várias maneiras, incluindo alterações na translação e diversos tipos de rotações, sendo que todos são realizados no sistema de coordenadas local do objeto. Geralmente estas transformações estão relacionadas com algum evento ou comportamento associado a um objeto.

4.1.2.3 COMPORTAMENTO

Um objeto Dive pode reagir a determinados eventos que acontecem em um ambiente virtual, disparados por um usuário ou ainda para executar um procedimento do próprio sistema. Um exemplo é um objeto mudar de cor ou ainda gerar um som quando um usuário se aproxima do objeto ou quando o toca. Isto é considerado como um comportamento que o objeto possui e que é executado mediante um determinado evento.

O comportamento é especificado utilizando-se um estado inicial e um conjunto de "arcos" entre os estados seguintes. Cada arco especifica quais ações serão executadas quando algum sinal de transição ocorre. A declaração de comportamento é limitado aos tipos de ação que executam transformações de rotação, translação, alteração de material, visibilidade, execução de algum arquivo de som, alteração na solidez do objeto e ainda o envio de determinados sinais de comportamento para outros objetos.

Os principais sinais que podem afetar o comportamento implementado em um objeto são:

- Sinais de colisão – são gerados quando objetos no mundo virtual interagem entre si, sendo que para que isto ocorra os campos de *collision* e *wireframe*, nas determinação das características de um objeto, devem estar setados como *true.* Estes sinais são distribuídos para todos os processos conectados ao mundo;
- Sinais de interação – são gerados quando um usuário interage com o mundo virtual de diversas maneiras, como por exemplo selecionando um objeto. Este sinal também é distribuído para todos os processos conectados ao mundo;
- Sinais de comportamento – são sinais gerados por processos ou objetos com a intenção de disparar outros comportamentos existentes em outros objetos;
- Sinais de tempo – são sinais gerados em intervalos específicos de tempo e são geralmente utilizados para direcionar certos comportamentos em objetos, possibilitando torná-los objetos com movimentos autônomos.

Existem duas maneiras possíveis de associar um comportamento a um determinado objeto, sendo possível a utilização de funções específicas à esta camada, desenvolvidas também na linguagem C ou ainda, mais facilmente, pode-se utilizar a descrição de formato de arquivo que permitem a construção de comportamentos, utlizando-se *scripts* em Tcl.

4.1.2.4 CONSCIÊNCIA

O módulo de consciência implementa uma variação simples de um modelo de interação espacial. Este modelo é baseado na noção de consciência entre objetos em um espaço compartilhado. A consciência de um objeto pode ser utilizada para determinar quais informações este objeto receberá através de uma entrada visual ou utilizando uma "aura". Os principais campos que determinam o cálculo de consciência existente em um objeto são:

- *Aura* – a aura é um sub-espaço onde potencialmente um objeto pode interagir com outros, simplesmente pela detecção da presença destes neste sub-espaço;
- *Focus* – o campo de foco determina onde o objeto direcionará sua atenção, permitindo assim ser visto como um seletor de informação;
- *Nimbus* – o campo de *Nimbus* determina onde o objeto disponibilizará as informações para outros objetos, podendo ser visto como um projetor de informações.

No Dive, aura, focus e nimbus são implementados como objetos comuns que podem ser atachados em outros objetos, tornando-os sub-objetos destes. A aura é um sub-objeto do objeto primário e o focus e o nimbus são sub-objetos da aura. Uma maior definição da utilização destes conceitos pode ser visto na seção 2.4.1 .

4.1.2.5 PERSONAGEM

Segundo Andersson (1994) tradicionalmente, programação distribuída é vista como um conjunto de processos comunicando-se entre si através da passagem de mensagens. O conceito de personagem é utilizado com a finalidade de permitir um aumento do nível de abstração na programação de aplicações distribuídas. Como tal, um personagem é um conjunto de processos que compartilha alguns estados associados a um usuário.

Na estrutura de dados de um personagem, existem algumas propriedades relacionadas com aspectos de um usuário humano residente no sistema, sendo que estas propriedades

incluem principalmente: Um único identificador, um nome de usuário, um nome de pessoa único e também descrições do usuário.

Além destas temos uma descrição virtual do corpo físico do usuário que serve como um ícone que representa graficamente o usuário e que é chamado de *body icon.*

Um personagem é inicializado e distribuído como um objeto para todos os processos participantes do mundo virtual e cada personagem passa a fazer parte da base de dados replicada sendo possível acessar personagens similarmente como um objeto.

4.1.3 CAMADA DE DISTRIBUIÇÃO

Os objetos Dive que os processos existentes em um mundo virtual podem criar e manipular, ficam armazenados em uma base de dados mantida na memória de um sistema. Desde que exista um número maior de processos rodando ao mesmo tempo, geralmente em máquinas diferentes, cada processo terá uma cópia separada da base de dados sendo que cada mudança efetuada por um processo nesta base, deve ser informada aos outros participantes. Por isso existe a camada de distribuição, que é responsável pela replicação da base de dados local e pela imediata distribuição destas replicações aos outros processos, fazendo com que esta base de dados passe a ser compartilhada.

No Dive, funções que criam e manipulam objetos possuem duas versões. Uma que somente executa as operações localmente e outra que distribui a operação para os outros processos.

Muitas das funções utilizadas na camada de objetos são similares as funções existentes na camada de distribuição, mas estas são encarregadas pela distribuição dos objetos. Para diferenciar a funções da camada de distribuição, todas usam o prefixo *distrib* que identifica a qual camada pertence esta função. Além disso, no processo de criação de um mundo virtual, sempre se utiliza as funções de distribuição após a criação ou alteração de um objeto, bem como de sua posterior inicialização.

A camada de distribuição gerencia também um número de variáveis globais que indicam o estado em que se encontra a distribuição de um objeto. Elas incluem o mundo corrente, onde os processos estão conectados e o estado local de cada processo. As características destas variáveis globais são apresentadas abaixo:

- *Worlds* – indica o estado global do grupo de processos conectados, sendo que este estado é independente das bases locais de cada processo. As informações que fazem parte deste estado global são: os limites geométricos que formam o mundo, a cor de fundo utilizada, a definição das características de luz utilizada e o número de membros conectados;
- *Process – o*s processos conectados ao mundo virtual, possuem alguns parâmetros que são significantes ao processo de distribuição de estados. Elas incluem o nome do programa e o personagem atualmente conectado com o processo.

O conjunto de funções utilizadas na camada de distribuição e sua correta utilização, bem como de todas as funções existentes nas outras camadas, podem ser vistas em Andersson (1994).

4.1.4 CAMADA DE APLICAÇÃO

A camada de aplicação é constituída por um conjunto de módulos que dão suporte para o desenvolvimento de novas aplicações utilizando-se a plataforma DIVE. A intenção deste conjunto de módulos é permitir a implementação de novas idéias fácil e intuitivamente, sem que haja a necessidade de aprofundar-se muito no nível mais baixo do desenvolvimento de interface para sistemas.

Alguns módulos existentes nesta camada são:

- Módulo de Utilitários – Permite implementar uma série de ferramentas que podem ser utilizadas pelas aplicações e que permitem criar e gerenciar no mais alto nível diversas ações no Dive;
- Módulo de Veículo – Este módulo traduz todas as ações de um usuário, como por exemplo o movimento do mouse ou o clique de um botão, em ações que ocorrem dentro do dispositivo virtual. Desde que haja uma interface bem definida entre este módulo e as camadas anteriores, um usuário pode trabalhar em uma aplicação utilizando diversos dispositivos físicos de entrada e saída, sem a necessidade de algum tipo de configuração pré-definida. Alguns destes dispositivos que podem ser relacionados são: mouse, HMD *(Head mounted display)* e *dataglove*;
- Módulo de Renderização – Este é o módulo responsável pela renderização gráfica de um objeto existente na base de dados do dispositivo virtual. As funções existentes neste

módulo trabalham diretamente com a camada de objetos, interagindo com a base de objetos local existente.

Os principais módulos que formam o conjunto de bibliotecas da plataforma Dive, mais utilizados na camada de aplicação são o módulo *Dive Graphics*, o *Dive áudio* e também o *Dive aux*.

No capítulo 4 foram apresentados os métodos para desenvolvimento de aplicações utilizando-se a plataforma Dive, sendo ressaltado principalmente o modelo de implementação em camadas utilizado na arquitetura do módulo Dive Core.

Em virtude do tempo disponível para confecção deste trabalho, optou-se por utilizar as funções existentes na camada de objetos como forma de implementação da base de dados necessária à criação do ambiente virtual. Conjuntamente serão utilizados os arquivos executáveis disponíveis em Dive (2001) responsáveis pela parte de comunicação bem como o visualizador *Vishnu* responsável pela visualização e distribuição da base de dados implementada. A especificação, implementação e funcionamento destes processos são apresentados a partir do capítulo 5.

5 DESENVOLVIMENTO DO SISTEMA

O desenvolvimento do protótipo foi baseado na aplicação chamada *Vishnu*, encontrada em Dive (2001), onde através de sua utilização foi possível implementar as principais funções aplicadas no protótipo, e também criar um mundo virtual que foi anexado à base de dados do sistema, tornando possível seu acesso por outros usuários.

A seção 5.1 apresenta a especificação e implementação (optou-se por apresentar a especificação e implementação juntas para facilitar o entendimento das principais funções existentes no protótipo). Já a seção 5.2 apresenta o processo de instalação, o funcionamento e a interface do protótipo.

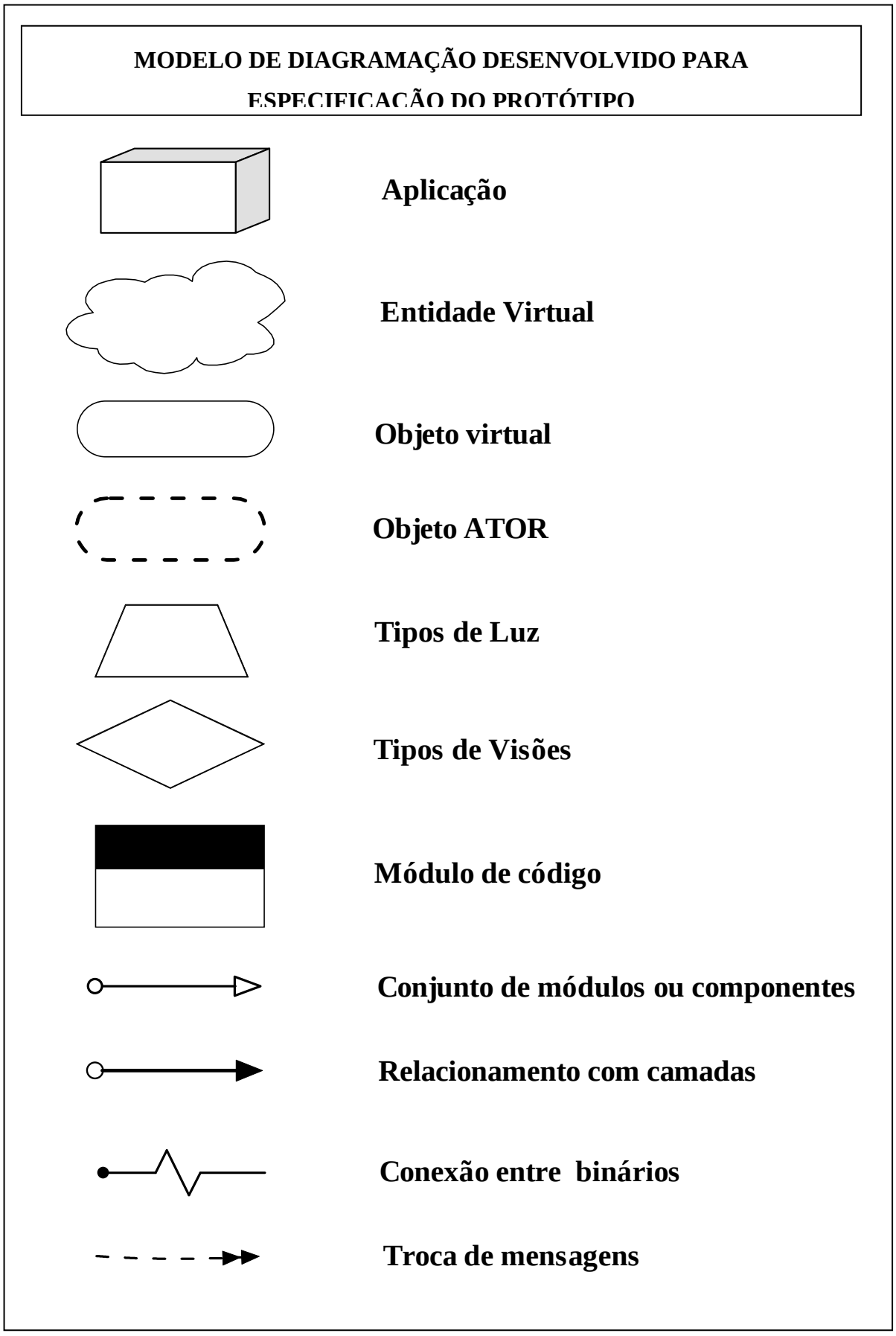
5.1 ESPECIFICAÇÃO E IMPLEMENTAÇÃO

Em virtude da utilização da linguagem Tcl/Tk para implementação do visualizador Vishnu encontrado em Dive (2001), continuou-se utilizando a mesma linguagem, mais especificamente Tcl 8.0 e Tk 8.0, para implementação do protótipo deste trabalho. Também foi utilizado o ambiente ASED Tcl/TK-IDE 2.0.7.3, como um editor para manipulação do código em Tcl/Tk responsável pela implementação das principais funções existentes no Vishnu.

Já para criação do mundo virtual que é utilizado no protótipo de ambiente virtual distribuído, utilizou-se o modelo de descrição de arquivos DIVE, encontrado em Avatare (1999), para implementação da principal entidade bem como dos objetos que estão contidos na mesma, os quais foram anexados a base de dados distribuída do protótipo. Para descrição desta entidade e de seus objetos foi utilizado, além do Microsoft *WordPad 5.0*, o próprio editor de objetos existente na aplicação DIVE.

Para especificar o processo de criação e funcionamento do protótipo e a relação existente entre cada parte deste processo com o modelo de especificação da arquitetura Dive Core apresentado na seção 4.1, foi utilizado uma série de diagramas criados utilizando-se a ferramenta *SmartDraw 6.0*. Já que não houve a possibilidade de utilizar um modelo de diagramação que pudesse representar a contento a especificação deste protótipo, foi criado o modelo de diagramas apresentado na figura 5.1, que visa mostrar de forma clara a lógica utilizada no desenvolvimento do protótipo.

Figura 5.1 Modelo de Diagramação



Em seguida tem-se uma breve explicação de cada componente existente no modelo de diagramação anterior:

- **Aplicação:** como o próprio nome já diz, este diagrama representa as aplicações e arquivos executáveis que são necessárias ao funcionamento do protótipo como um todo. Os arquivos executáveis são implementados na linguagem C e o visualizador *Vishnu* é uma aplicação implementada na linguagem Tcl/Tk;

- **Entidade Virtual:** representa o nó raiz de uma hierarquia de componentes. É na realidade um arquivo único que pode conter diversos tipos de componentes como descrições de mundos, de objetos, de tipos de luz utilizadas, de visões, etc.;

- **Objeto Virtual:** representa a descrição de um objeto. Estes objetos podem ser de diversos tipos como mundos, objetos compostos de sub-objetos, etc. A definição de objetos no ambiente virtual Dive é bastante generalizada pois praticamente todas as descrições podem ser consideradas como objetos. O que muda são as características utilizadas por eles de acordo com o tipo de objeto desejado. Maiores detalhes sobre objetos são encontrados na seção 3.1.1;

- **Objeto ATOR:** representa a utilização da descrição de um objeto ator. A diferença deste tipo de objeto para outros é que, na maioria das vezes, estes são utilizados como Avatares e representam um processo do usuário que é inserido no ambiente virtual. Apesar deste estar presente na hierarquia de uma entidade não faz parte dela, ou seja, sua descrição não estará contida na entidade;

- **Tipos de luz:** representa a descrição de um tipo de luz presente na entidade. Os principais tipos são *Point Light*, *Spot Light* e *Directional Light*;

- **Tipos de visões:** representa a descrição da utilização de uma classe de objeto existente e que é geralmente utilizada para criação de objetos mais complexos. Os principais tipos de visões existentes são *line*, *box*, *sphere*, *ellipse*, *cylinder*, *n_m_poly*, etc;

- **Módulo de código:** representa a união dos principais *scripts* em Tcl que formam uma determinada aplicação;

- **Conjunto de módulos ou componentes:** representa a utilização de um determinado módulo de código ou conjunto de componentes que formam, respectivamente, uma aplicação ou entidade;

- **Relacionamento com camadas:** representa o relacionamento de uma determinada camada da arquitetura Dive Core com uma aplicação ou entidade;

- **Conexão entre arquivos executáveis:** representa a existência de uma conexão entre os arquivos executáveis;

- **Troca de mensagens:** representa a existência de troca de mensagem entre as aplicações.

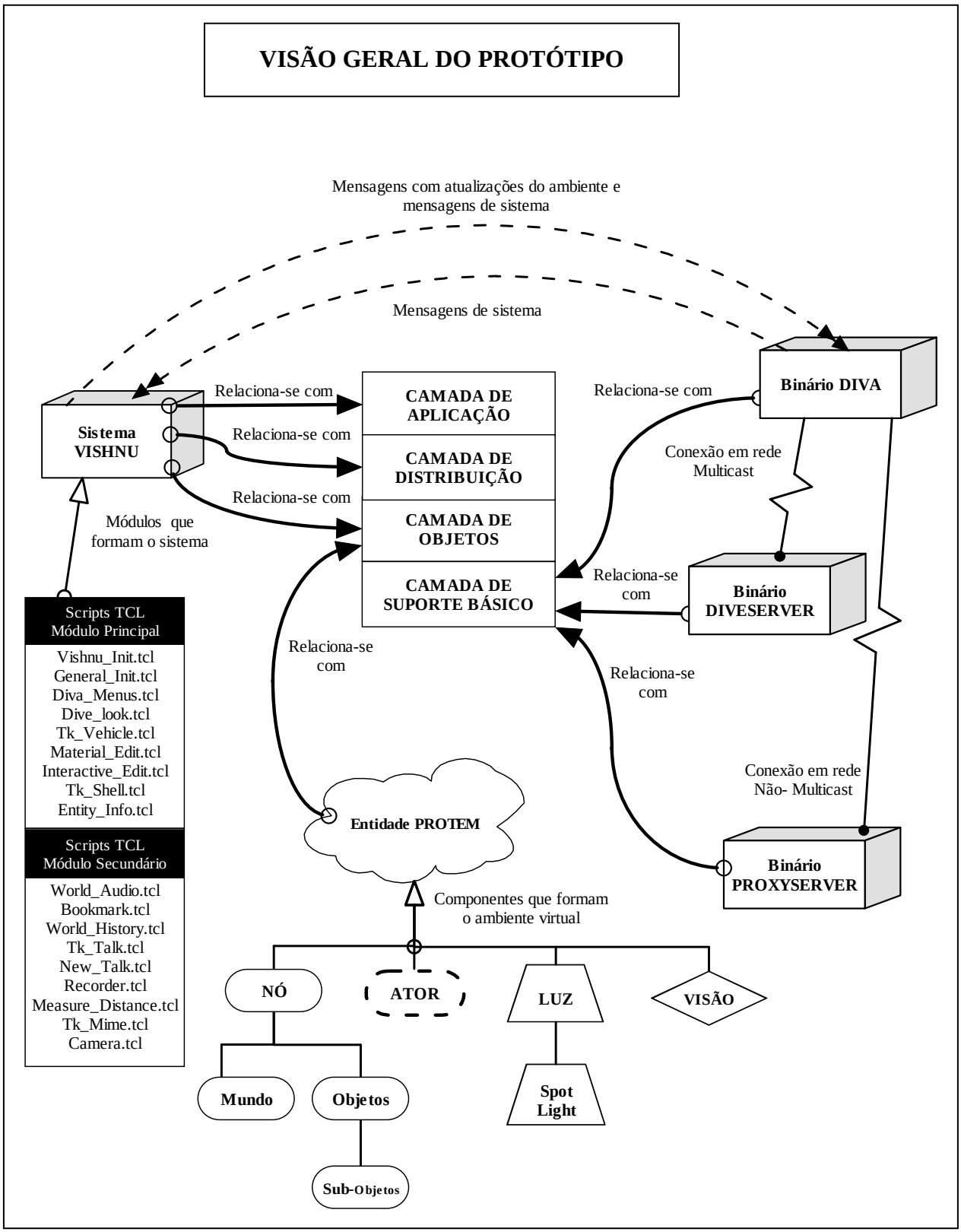
Como ponto de partida para especificação do protótipo será apresentado o diagrama que proporciona uma visão geral do protótipo permitindo assim uma melhor compreensão do funcionamento do mesmo. Na seqüência serão apresentados os diagramas que especificam cada parte dos processos vistos no diagrama principal e, em alguns casos, serão apresentados trechos de código para facilitar o entendimento e exemplificar como estes diversos processos foram implementados.

5.1.1 VISÃO GERAL DO PROTÓTIPO

A figura 5.2 apresenta a estrutura de forma geral dos processos existentes na implementação e funcionamento do protótipo Dive, procurando demonstrar a relação existente entre cada um com as camadas do modelo de implementação da arquitetura Dive Core.

Nela pode-se verificar a existência de três processos distintos, sendo eles, a utilização de arquivos executáveis responsáveis por funções de suporte básico no ambiente operacional e de rede, a implementação de uma aplicação através de *scripts* em Tcl os quais desempenham diversas funções, podendo-se destacar as funções de criação da interface com o usuário, funções de renderização e edição de objetos virtuais, funções que permitem o controle do ambiente virtual utilizando diversos tipos de dispositivos como por exemplo mouse, luvas, capacetes de visualização, etc. E por fim o processo de criação de entidades virtuais compostas de diversos componentes, através da utilização de uma linguagem de descrição de objetos existente na plataforma DIVE.

Figura 5.2 Visão geral do protótipo



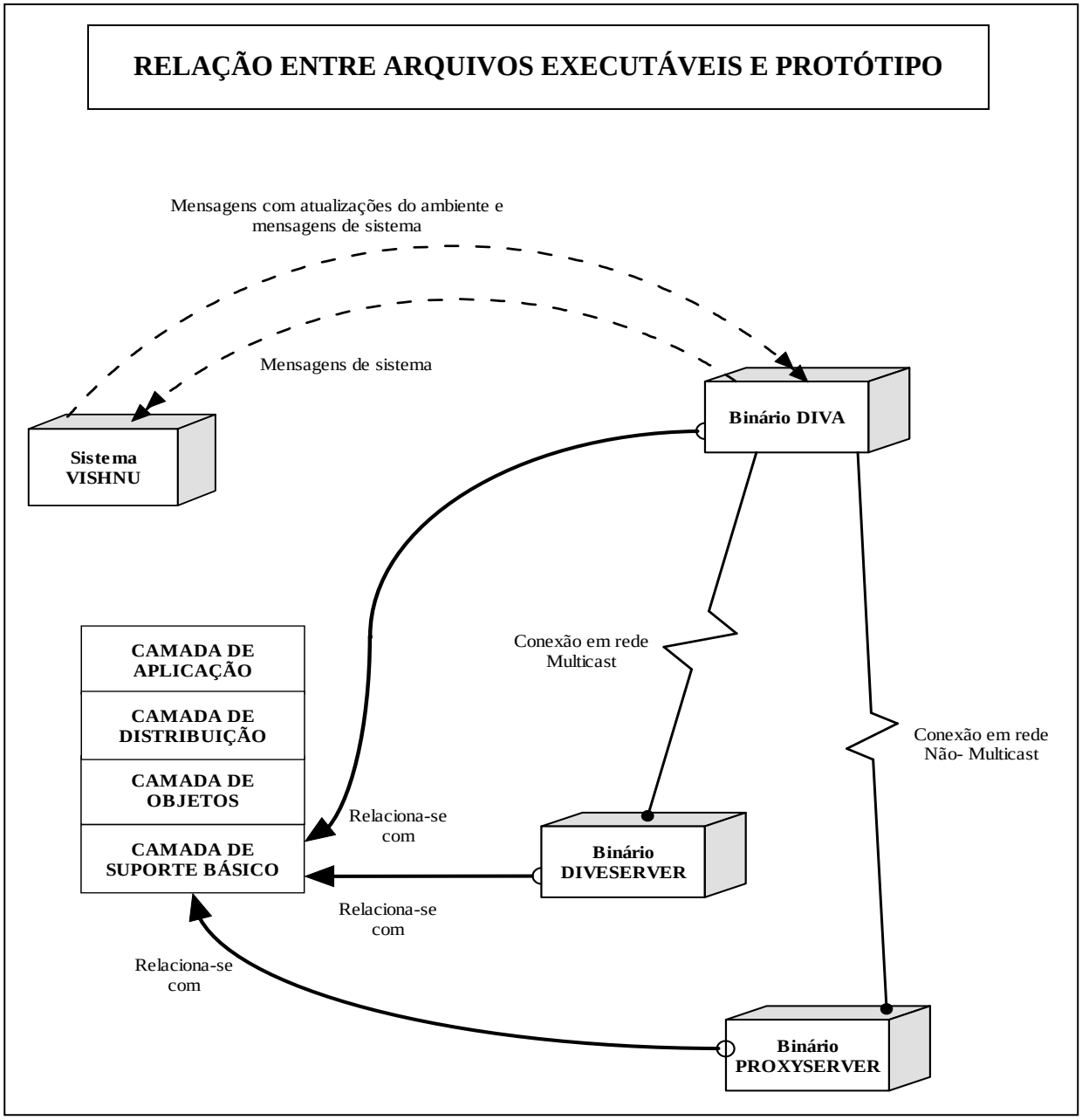
Estes processos são apresentados de forma separada nas seções a seguir.

5.1.2 RELAÇÃO ENTRE OS ARQUIVOS EXECUTÁVEIS E O PROTÓTIPO

Como foi explicado anteriormente no capítulo 4, a plataforma DIVE permite uma maior flexibilidade no desenvolvimento de aplicações, podendo-se desenvolver um aplicativo único que incorpora funções de todas as camadas da arquitetura Dive Core sendo este implementado usando a linguagem C, ou ainda permite o desenvolvimento do aplicativo em partes utilizando-se conjuntamente as linguagens C e Tcl/Tk.

A figura 5.3 apresenta o processo de utilização de arquivos executáveis que fornecem, basicamente, serviços de suporte de rede e de sistema operacional.

Figura 5.3 Relação entre arquivos executáveis e protótipo



Os arquivos executáveis *Diva*, *Diveserver* e *Proxyserver* são arquivos devidamente compilados e linkados com as bibliotecas da plataforma *Dive* que fornecem as principais funções existentes na camada de suporte básico, tendo sido implementados utilizando a linguagem C.

Tanto o *Diveserver* quanto o *Proxyserver* são mais detalhadamente explicados na seção 3.4, mas basicamente funcionam como servidores de aplicação, pois armazenam uma lista com todos os endereços *multicast* dos usuários conectados. Com a conexão do arquivo executável *Diva*, este recebe os endereços *multicast* existentes nesta lista permitindo assim o compartilhamento de um ambiente virtual com outros usuários. Um trecho do código implementado do arquivo executável *Diveserver* pode ser visto no quadro 5.1 e apresenta a estrutura criada para armazenar a lista de usuários que conectaram-se ao *Diveserver*.

Quadro 5.1 Estrutura da lista de usuários conectados

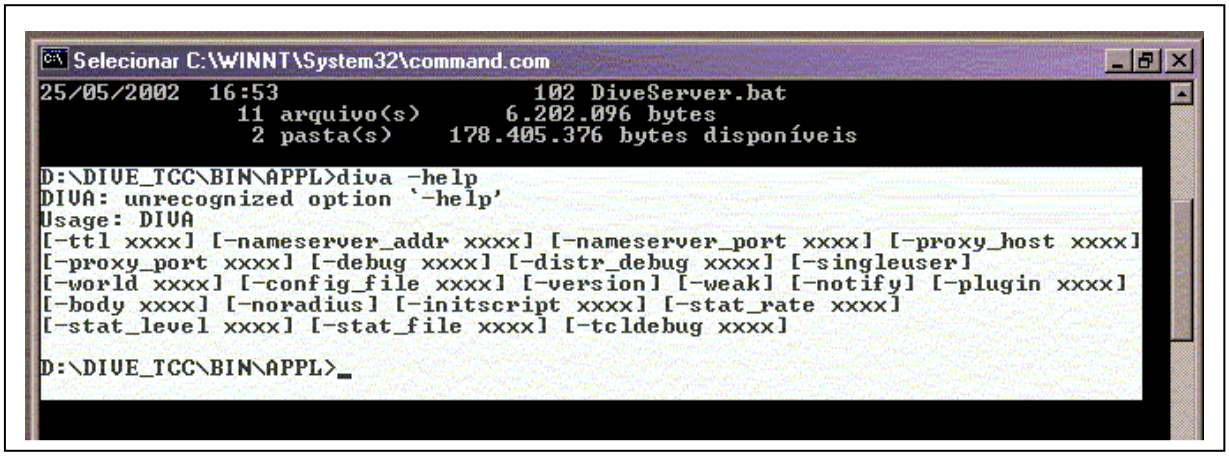
```
#define DIVESERVER_FILENAME ".diveserver.data"

struct ns_entry{
  struct ns_entry *next;
  char *domain; /* Domain/application name */
  char *name; /* Name of group */
  char *addr; /* Address in "..." format */
  unsigned short port; /* Port number*/
  unsigned ttl;        /* Multicast time-to-live */
  unsigned flags;
  unsigned int incarnation;  /* Counter for order of joins. */
};
static struct ns_entry *ns_list = NULL; /* List of nameserver entries */
static sid_addr ns_addr; /* Address that nameserver communicate on */
```

As funções básicas utilizadas para implementação dos arquivos executáveis *Diveserver* e *Proxyserver* são procedentes da biblioteca SID, que é um modelo de protocolo de comunicação criado também pelos pesquisadores do *Swedish Institute of Computer Science* (SICS) e que fazem parte da camada de suporte.

Um aplicativo fundamental para o processo de funcionamento do protótipo como um todo é o arquivo executável *Diva*. Ele executa diversos serviços podendo-se destacar a inicialização do *script* em Tcl responsável por todo processo de carregamento e ativação da interface gráfica do visuzalizador *Vishnu*, o processo de inserção do objeto Ator no ambiente

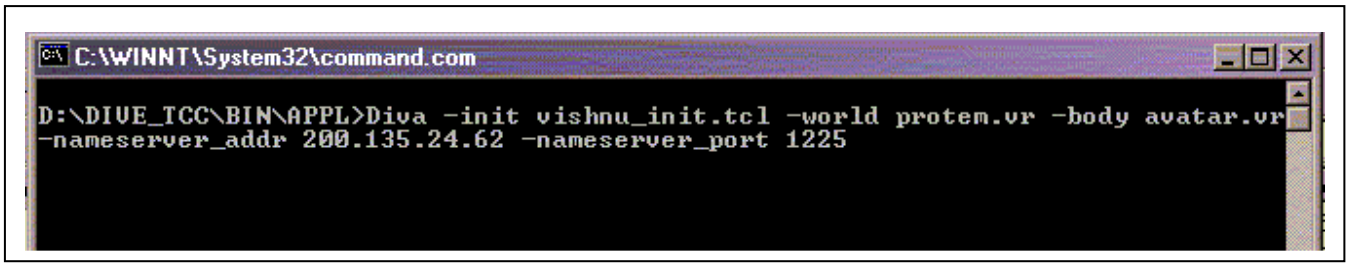
virtual, o processo de conexão com os servidores de aplicação *Diveserver* ou *Proxyserver*, a configuração de diversos parâmetros utilizados pelo visualizador e também o controle de mensagens provenientes do sistema operacional indicando possíveis erros ocorridos. Os principais parâmetros que podem ser utilizados com o arquivo executável Diva pode ser visto na figura 5.4

Figura 5.4 Principais parâmetros existentes no arquivo executável Diva



Estes parâmetros podem ser passados em tempo de execução, utilizando-se um arquivo *.bat* com os comandos necessários, ou ainda podem ser configurados diretamente no visualizador *Vishnu* que possui uma função específica para isto, arquivando os valores setados em um arquivo com extensão *.dive_configure*. Um exemplo da utilização de passagem de parâmetros para serem executados no visualizador Vishnu pode ser visto na figura 5.5

Figura 5.5 Exemplo de passagem de parâmetros em tempo de execução



O exemplo acima irá executar o arquivo executável Diva passando inicialmente qual é o *script* em Tcl responsável pela inicialização da interface gráfica, que neste caso é o visualizador *Vishnu*, passando também que a entidade que deve ser carregada (diga-se mundo virtual) é a entidade Protem, que a representação gráfica do usuário (diga-se Ator) que será utilizado é o objeto chamado Avatar e por fim já solicita conexão com o servidor de aplicação

Diveserver passando o endereço IP aonde o servidor está sendo executado e também a porta de entrada. Convém ressaltar que não existe a necessidade de executar isto manualmente, pois como será visto no capítulo seguinte, pode-se setar todos estes valores diretamente na aplicação os quais serão armazenados em um arquivo de configurações próprio da aplicação.

Nos quadros 5.2 e 5.3 temos trechos de código do aplicativo *Diva* mostrando, respectivamente, as funções de execução do *script* Tcl inicial que carrega o visualizador *Vishnu* e a função principal para inserção do objeto Ator no mundo virtual.

Quadro 5.2 Função que executa *script* Tcl inicial

```
Local variables
=========================================================================*/
static char *person_filename;
static char *initscript = NULL;
/*=========================================================================
/* Find tcl script init file from string and return it, NULL if the
   script could not be found from the DIVEPATH. */
static char* script_from_command(char *s0)
{
  char *s;
  char *s1;

  if (s = strrchr(s0, '/'))   /* Last '/' found */
    s++;
  else      /* No '/' found */
    s = s0;
  /* s is filename. */
  if (s1 = expand_filename("DIVEPATH", s, "_init.tcl"))
    return s1;
  return NULL;
}
```

Quadro 5.3 Função principal para inserir objeto Ator

```
/* Migrate the actor implemented by this process from the private init
   world to the starting world given on the command line and passed as
   an argument to this thread */
static void *start_actor(void *pid, void *name)
{
  actor *a = find_actor(pid);

  /* Migrate actor only if it exists */
  if (a == NULL)
    return NULL;
  distr_actor_world_change((objid_t*)pid, (char*)name, NULL, TRUE, NULL,
                       NULL);
  return NULL;
}
```

Uma outra função existente no arquivo executável Diva é o envio de uma mensagem para o visualizador *Vishnu* de tempos em tempos, e que funciona como um alarme que dispara a renderização do ambiente virtual visto pelo Ator inserido no processo. O trecho de código que executa esta função pode ser visto no quadro 5.4

Quadro 5.4 Função que gera sinal de alarme para renderização do ambiente virtual

```
void *start_diva(void *arg)
{ actor *p;
  struct timeval T;
  int alarm_freq;
  objid_t *topid;
  int i;
  register_config_cb(alarm_update, NULL, "alarm_freq", NULL);
  config_get_int(configure_table, "alarm_freq", &alarm_freq);
  if (alarm_freq == 0)
    alarm_freq = 10;
  if (!should_render) {
    alarm_init(1000/alarm_freq);
    return NULL;
  }
  T.tv_sec = 1;
  T.tv_usec = 0;
  p = find_actor((objid_t*)arg);
  if (!p)
    dive_error("start_vishnu: no person");
  if ((config_get_int(configure_table, "person_collision", &i)>-1) &&
      (i > 0)) {
      if (topid = find_assoc_id(p, "top")) {
        collision_register(p, topid);
      }
  }
  if ((config_get_int(configure_table, "person_gravity", &i)>-1) &&
      (i > 0)) {
      point_t dir = {0, -1, 0};
      collision_gravity(p, &dir);
  }
  if (alarm_freq > 0)
    alarm_init(1000/alarm_freq);
  {
      void * attr;
      attr = threads_attr_create();
      if (!attr) {
      dive_warning("start_diva: error when creating render thread attr");
      }
      else {
      threads_attr_init(attr);
      threads_attr_setstacksize (attr, 64*1024);
      threads_create(NULL, attr, render_loop, (void*)&p->id);
      threads_attr_free(attr);
      }
  }
#ifdef EVENT_HANDLER
  threads_fork(eventhandler, &p->id);
#else
  /* Initialize interaction context for Keyboard and Mouse */
  inctx_XKM_init(p);
#endif
  return NULL;}
```

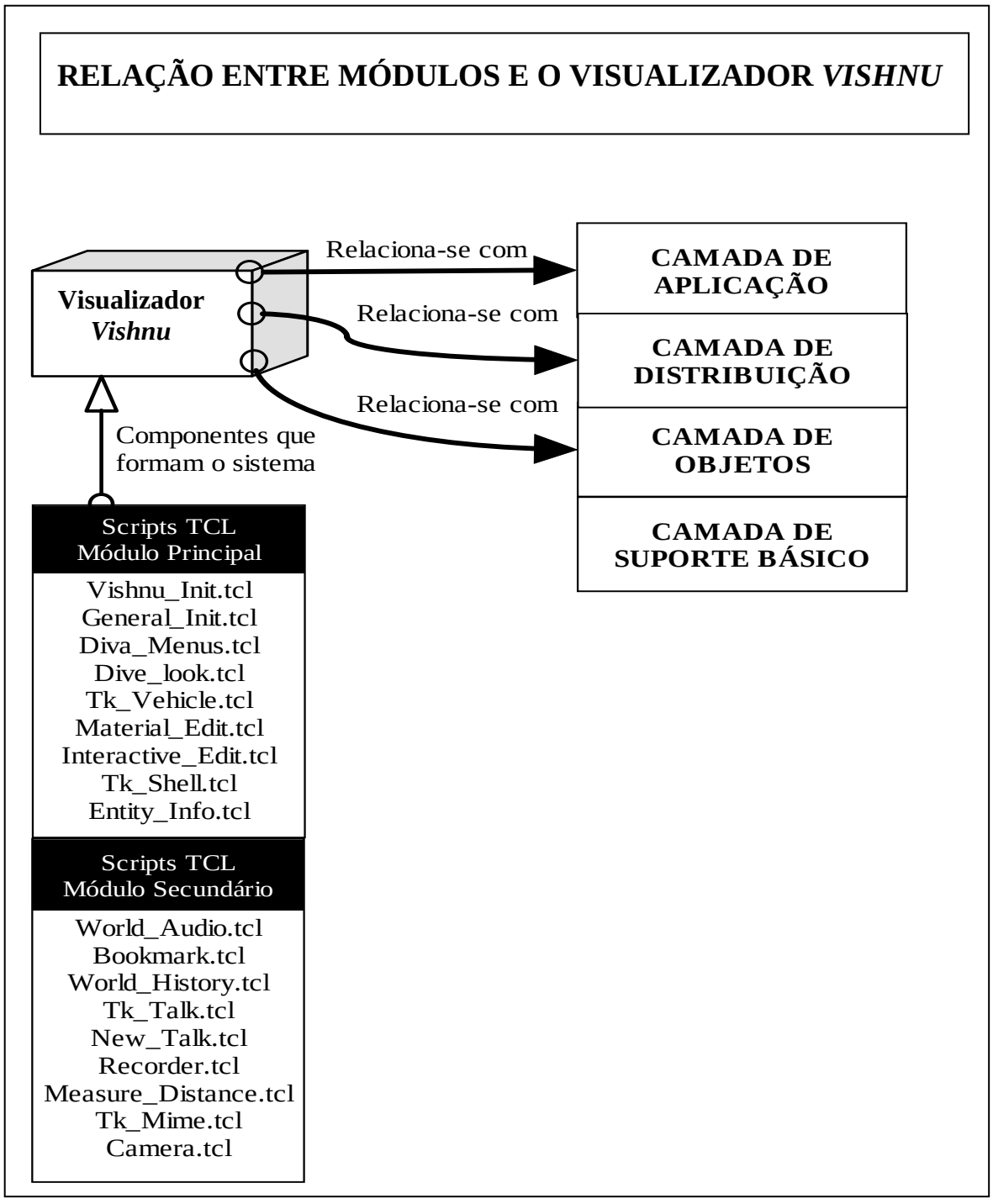
O arquivo executável Diva possui relação com praticamente quase todas as funções da camada de suporte básico, sendo que os módulos mais utilizados são o *Threads* que fornece suporte a parte de hardware e periféricos e o *Callbacks* que fornece suporte a parte de ambiente operacional.

A utilização destes arquivos executáveis possibilitam uma facilidade a mais já que não houve a necessidade de preocupar-se com a parte de comunicação e gerenciamento do serviço de rede. E também, como os códigos fonte das aplicações também são fornecidos, permitem um maior entendimento do seu funcionamento e das principais funções existentes na plataforma Dive e que possibilitaram o desenvolvimento destes arquivos executáveis.

5.1.3 RELAÇÃO ENTRE OS MÓDULOS E O VISUALIZADOR *VISHNU*

Outro processo distinto no desenvolvimento do protótipo é a implementação de *scripts*, descritos utilizando-se a linguagem Tcl/Tk, que relacionados com as camadas de aplicação, distribuição e de objetos, permitem a utilização de diversas funções que possibilitam o desenvolvimento de um visualizador bastante completo. O diagrama que apresenta os principais *scripts* utilizados bem como sua relação com as camadas pode ser visualizado na figura 5.6

Figura 5.6 Relação entre módulos de códigos e o visualizador *Vishnu*



A facilidade na utilização de ambientes gráficos por parte dos usuários, favorece cada vez mais o aparecimento de linguagens que permitam o desenvolvimento destas aplicações. Esta pode ser considerada uma das vantagens da plataforma DIVE que além de possibilitar o desenvolvimento de aplicações utilizando a linguagem C, também possui funções específicas em Tcl/Tk que permitem o desenvolvimento de diversos tipos de aplicações visuais.

Alguns dos principais *scripts* utilizados na implementação do protótipo bem como os serviços por eles oferecidos e sua relação com as camadas da arquitetura Dive Core são mostradas a seguir:

- ***Vishnu_init*** – é o *script* inicial que é linkado com o arquivo executável Diva e possibilita a inicialização dos demais *scripts* que compõem o visualizador Vishnu;

- ***General_init*** – é responsável principalmente pela criação do menu principal existente no visualizador, pela inicialização da janela na qual será renderizado o ambiente virtual e também possui uma função que obtém a cada segundo a posição do Avatar após ele ter sido inserido no visualizador, sendo que o trecho do código que executa esta função pode ser visto no quadro 5.5. As funções utilizadas neste *script* são relacionadas com a camada de aplicação, mais diretamente com a implementação de menus e interfaces;

Quadro 5.5 Função que obtém a posição do Avatar

```
# Compute avatar position once a second
proc newpos {} {
   global position
   dive_entity_info [dive_find_assoc [dive_self] "top"] info
   set x [lindex $info(T0) 0]
   set z [lindex $info(T0) 2]
   set position "($x, $z)"
}
```

- ***Diva_menus*** – além de criar os principais sub-menus do visualizador também executa as principais funções relacionadas com entidades e objetos existentes na base de dados distribuída do protótipo. Possui funções para leitura das entidades e objetos existentes carregando-as no ambiente virtual, funções que criam um editor de objetos em tempo de execução, funções para carregar objetos descritos em Vrml permitindo salvá-los como objetos Dive e também funções relacionadas com o arquivo de configuração lido pelo visualizador. Possui relacionamento direto com as camadas de aplicação e principalmente com a camada de distribuição e de objetos

- ***Entity_info*** – é também um dos *scripts* mais importantes do visualizador pois as funções utilizadas nele desempenham principalmente o controle das informações das entidades existentes na base de dados distribuída e pelo envio das mensagens ao arquivo

executável Dive com as atualizações existentes na entidade virtual, que serão repassados as outras aplicações conectadas. Um exemplo pode ser visto no quadro 5.6 que apresenta um trecho de código no qual existe uma função que envia uma mensagem para avisar que a aplicação ainda está “viva” e outra função que checa se a mensagem enviada foi recebida. As funções utilizadas neste *script* estão relacionadas com a camada de distribuição da arquitetura Dive Core;

Quadro 5.6 Funções de envio e recebimento de mensagens de atualização

```
proc send_alive_ping_server_pings {} {
   global Iam_alive_ping_server Simple_actors Alive_ping_server_life
   incr Alive_ping_server_life -1
   if {$Iam_alive_ping_server} {
         for {set i 0} {$i < $Simple_actors(number)} {incr i} {
            dive_entity_message $Simple_actors($i) "alive_ping" 1 "" [dive_self]
         }
   }
   if {!$Alive_ping_server_life} {
         set Iam_alive_ping_server 1
   }
}

proc receive_alive_ping_server_pings {event id origin name type msg_list} {
   global Iam_alive_ping_server Simple_actors Alive_ping_server_life

   if {$name == "alive_ping" && $origin != [dive_self]} {
         set Alive_ping_server_life 10
         set Iam_alive_ping_server 0
   } elseif {$name == "alive_ping" && $origin == [dive_self]} {
         set Alive_ping_server_life 10
         set Iam_alive_ping_server 1
   }
}
```

Existem cerca de 160 funções em Tcl na plataforma DIVE possíveis de serem utilizadas e que implementam os mais variados serviços, dependendo do tipo de aplicação desejada. Para maiores informações, consulte Dive (2001).

As principais implementações efetuadas no protótipo dizem respeito à parte de interface, que foi estudada e alterada para possibilitar uma melhor compreensão e posterior utilização por possíveis usuários. Em grande parte o estudo e implementação foi referente à utilização

do modelo de descrição de arquivos Dive, que pode ser encontrado em Avatare (1999), que possibilitou o desenvolvimento do ambiente virtual anexado à base de dados da aplicação e que pode ser compartilhado por outros usuários. Este processo pode ser visto na próxima seção.

5.1.4 RELAÇÃO ENTRE ENTIDADE E O PROTÓTIPO

Por fim, o último processo existente no diagrama apresentado na seção 5.1.2, é a criação de entidades virtuais que compõem a base de dados distribuída do protótipo.

Andersson (1994), apresenta um modelo de desenvolvimento destas entidades virtuais totalmente implementados em linguagem C através da utilização das principais funções existentes na camada de objetos da arquitetura Dive Core.

A vantagem de se utilizar este modelo é que como são implementados em C, podem ser compilados diretamente na aplicação, não dependendo desta para serem executadas ou ainda distribuídas. A desvantagem é que, além de haver a necessidade de uma grande quantidade de linhas de código para implementar uma entidade, possue também um nível de abstração baixíssimo tornando seu desenvolvimento extremamente maçante e complicado.

Por isso, o modelo escolhido para a implementação destas entidades foi o proposto por Avatare (1999), que apesar de possuir a dependência de aplicação para serem visualizados e distribuídos, possue métodos razoavelmente fáceis de serem utilizados e também um nível de abstração maior que facilitam a implementação.

Este modelo trabalha com uma série de tipos de arquivos com formatos diferentes, os quais contém a descrição das diversas entidades que podem ser acessadas por uma aplicação.

Os principais tipos de arquivos são:

- Arquivos com extensão *.vr* – é o tipo de arquivo mais usado para armazenar as definições das principais entidades gráficas existente na base de dados, que são os mundos, os diversos tipos de objetos e os atores;

- Arquivos com extensão *.vh* – são os arquivos de *header*, e são utilizados para armazenar definições de macros que podem ser incluídas em um grande número de mundos e

arquivos de objetos permitindo a criação de objetos em um mundo virtual, apenas com a passagem de alguns parâmetros;

- Arquivos com extensão *.data* – são arquivos especiais que podem conter diversos componentes, os quais podem ser utilizados na descrição de um objetos. Por exemplo, podem conter nomes de texturas, de materiais ou de fontes permitindo sua utilização mais facilmente;

- Arquivos com extensão *.Tcl* – arquivos com *scripts* contendo comandos em Dive/Tcl utilizados especificamente para a descrição de comportamentos, podendo carregados na descrição de um objeto.

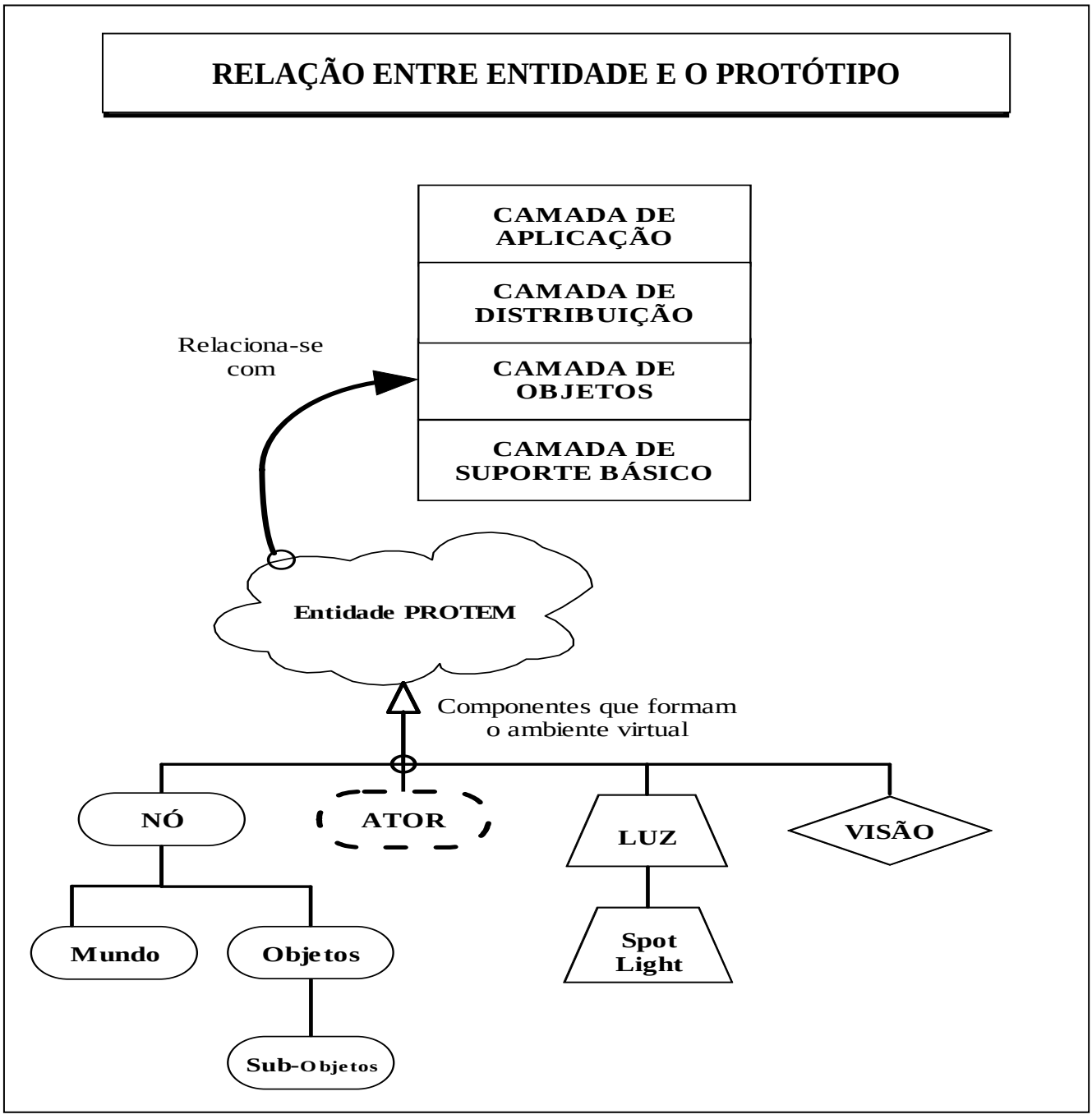
Todos os arquivos Dive são pré-processados por um pré-processador C sendo que, com isso, praticamente todas as diretivas usadas em C podem ser utilizadas na descrição de arquivos Dive.

Estes arquivos podem ser criados através da utilização de um simples editor de texto, salvando os tipos de arquivos de acordo com a função desejada para o mesmo. Ou ainda carregando arquivos existentes diretamente na aplicação e abrindo o editor de objetos, o qual além de permitir salvar o arquivo editado na extensão padrão do Dive (.vr), possibilita salvar também o arquivo com diversos outros tipos de extensão como por exemplo o Vrml 1.0.

Para criar a entidade virtual que será carregada no ambiente virtual, escolheu-se a descrição do ambiente Protem, que é o laboratório dos estudantes do curso de ciências da computação da Furb, em Blumenau.

O diagrama para especificar o modelo hierárquico básico que permitiu a implementação da entidade desejada, bem como a relação das funções utilizadas com a camada de objetos pode ser visualizado na figura 5.7.

Figura 5.7 Modelo básico da relação entre Entidade e o protótipo.



Conforme já apresentado na seção 4.1.2, a camada de objetos é particionada em cinco módulos onde cada um deles possui as suas próprias funções e métodos permitindo assim compor uma entidade virtual. A entidade Protem, que foi a descrição escolhida para ser representada, possui funções e métodos de três dos cinco módulos existentes na camada. São eles os módulos de objeto, de transformação e de comportamento. Na realidade eles se

complementam, pois no Dive praticamente tudo pode ser considerado como objeto. Um exemplo de declaração de um objeto utilizado na entidade implementada, foi o objeto chamado de “Porta_da_frente”. O código referente à este objeto pode ser visualizado no quadro 5.7.

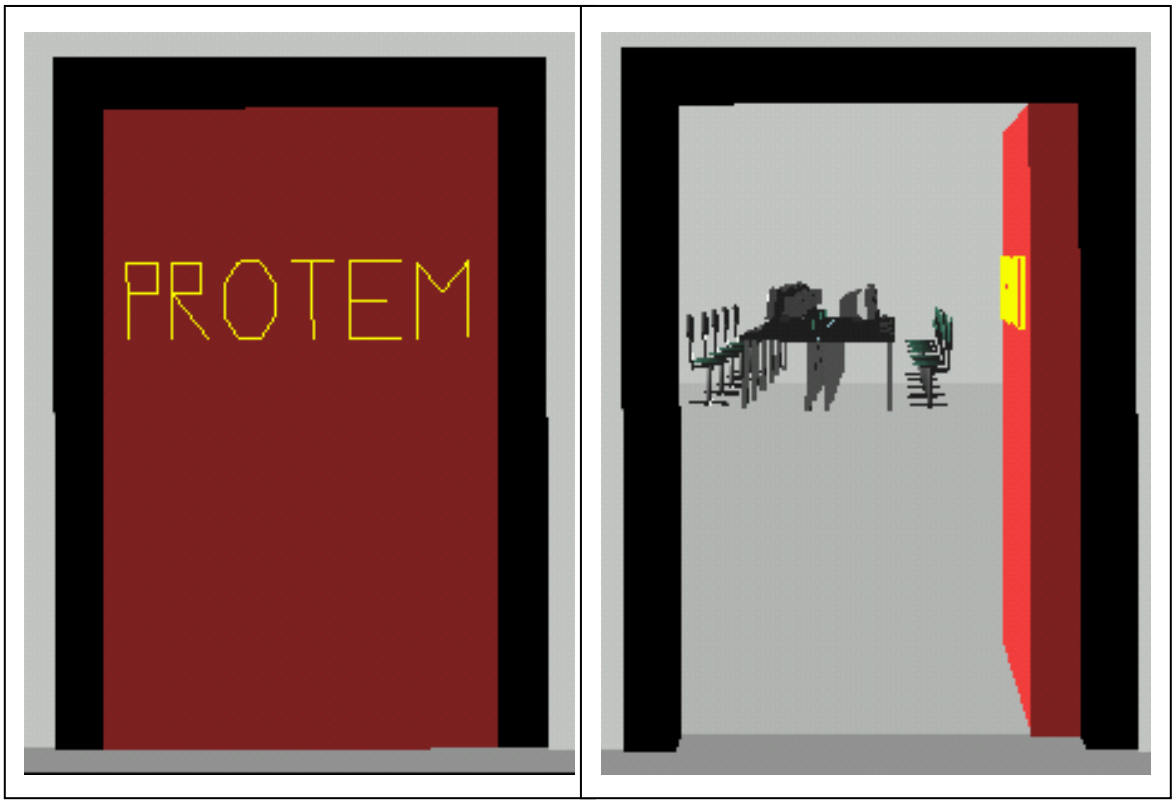
Quadro 5.7 Declaração do objeto “ Porta_da_Frente”

```
#include "dive.vh"
object {
  name "Porta_da_Frente"
  material {
    diffuse 0.996 0.996 0.996
    ambient 0.996 0.996 0.996
  }
  translation
    v         13          0         -1
  rotation
    v   0.999995 -0.00318521          0
    v  0.00318521   0.999995          0
    v          0          0          1
  object {
    material {
      diffuse 0.645 0.164 0.164
      ambient 0.645 0.164 0.164
    }
    translation
      v       -0.8          0        0.2
    view {
      material_index 0
      texture_index 0
      RBOX
        v          0          0       -0.2
        v        1.6        2.5          0
    }
    object {
       material "YELLOW"
       name "Aviso_Porta1"
       translation
           v       1.5          1.6      -0.207
       fixedxyz v 0  3.14  0
       view{
         TEXT 0.3
         "PROTEM"
         "default"
       }
    }
  begin.tcl
      proc on_interaction {type id stype origin src_id x y z} {
        dive_fixedXYZ [dive_self]  0 -1.57 0 LOCAL_C
        dive_sleep 10000
        dive_fixedXYZ [dive_self]  0 1.57 0 LOCAL_C
        }
      dive_register INTERACTION_SIGNAL DIVE_IA_SELECT [dive_self] ""
on_interaction
    end.tcl
  }
```

A declaração do objeto acima possui inicialmente a descrição do objeto principal, o qual foi dado o nome de “Porta_da_Frente”, juntamente com a declaração do material usado e também a localização do objeto no mundo virtual. Em seguida é declarada uma visão que utiliza uma classe de objeto chamada de RBOX para renderizar um sub-objeto com características específicas. Após temos uma declaração de outro sub-objeto chamado de “Aviso_Porta1”, seguido de declarações de tipo de material e de localização deste sub-objeto, ainda uma outra visão que utiliza uma classe chamada TEXT e que insere um texto neste sub-objeto. Finalmente temos a declaração de um comportamento específico deste objeto e que após um clique com o mouse no mesmo gerará uma transformação na posição inicial do objeto para uma outra especificada pela função de transformação.

A visão do objeto renderizado no ambiente virtual é apresentada na figura 5.8, que mostra dois momentos, um antes da execução do comportamento e da transformação descritas, e outro momento após a execução dos mesmos.

Fig. 5.8 Renderização do objeto “Porta_da_Frente”



Este objeto não foi descrito diretamente no arquivo que armazena a entidade Protem, principalmente por que aumentaria o tamanho do arquivo em virtude da quantidade de objetos existentes no mesmo, tornando também a distribuição da entidade para os usuários que não a possuísse em sua base de dados mais lenta.

Ao invés disto optou-se pela criação do arquivo com a descrição do objeto separadamente, bastando apenas utilizar o comando "*inline*" para carregá-lo na entidade principal.

O trecho de código da entidade Protem na qual foi descrita a chamada para o objeto acima, bem como para os outros objetos possuidores das mesmas características pode ser visto no quadro 5.8.

Quadro 5.8 Chamada de arquivo de objeto externo

```
object {
name "PORTA_PROTEM"
inline "PFrente1.vr"
}
object {
name "PORTA_DEPTO"
inline "PFrente2.vr"
}
object {
name "PORTA_VANDEIR"
inline "PVandeir.vr"
}
object {
name "PORTA_BANHEIRO"
inline "PBanheiro.vr"
}
object {
name "ESCADA_DEPTO"
inline "Escada2.vr"
}
```

A descrição do objeto "Porta_da Frente" utilizado na entidade Protem, foi feito utilizando-se um editor de texto simples para implementar as características do mesmo, e em virtude de ser um objeto simples não houve muita dificuldade em descrevê-lo.

Diferente foi a criação de um outro objeto utilizado no desenvolvimento da entidade principal, que foi o objeto "Computador".

Em virtude da quantidade de vértices necessários ao seu desenvolvimento, optou-se pela utilização de um objeto com as características desejadas, mas que era feito em Vrml 1.0

Utilizou-se então o próprio visualizador para carregar este objeto em Vrml no mundo virtual, que é lido em seu formato original, mas convertido e carregado no ambiente com as características básicas existentes no modelo de descrição de arquivos do Dive. Nas figuras 5.9 e 5.10, respectivamente, pode-se ver o arquivo em Vrml já carregado no ambiente *Vishnu*, e o editor de objetos da ferramenta com a descrição do objeto computador já formatada para o padrão Dive. Já na figura 5.11 pode-se ver o objeto já alterado de acordo com o modelo de objeto desejado.

Fig. 5.9 Objeto em Vrml 1.0 carregado no ambiente

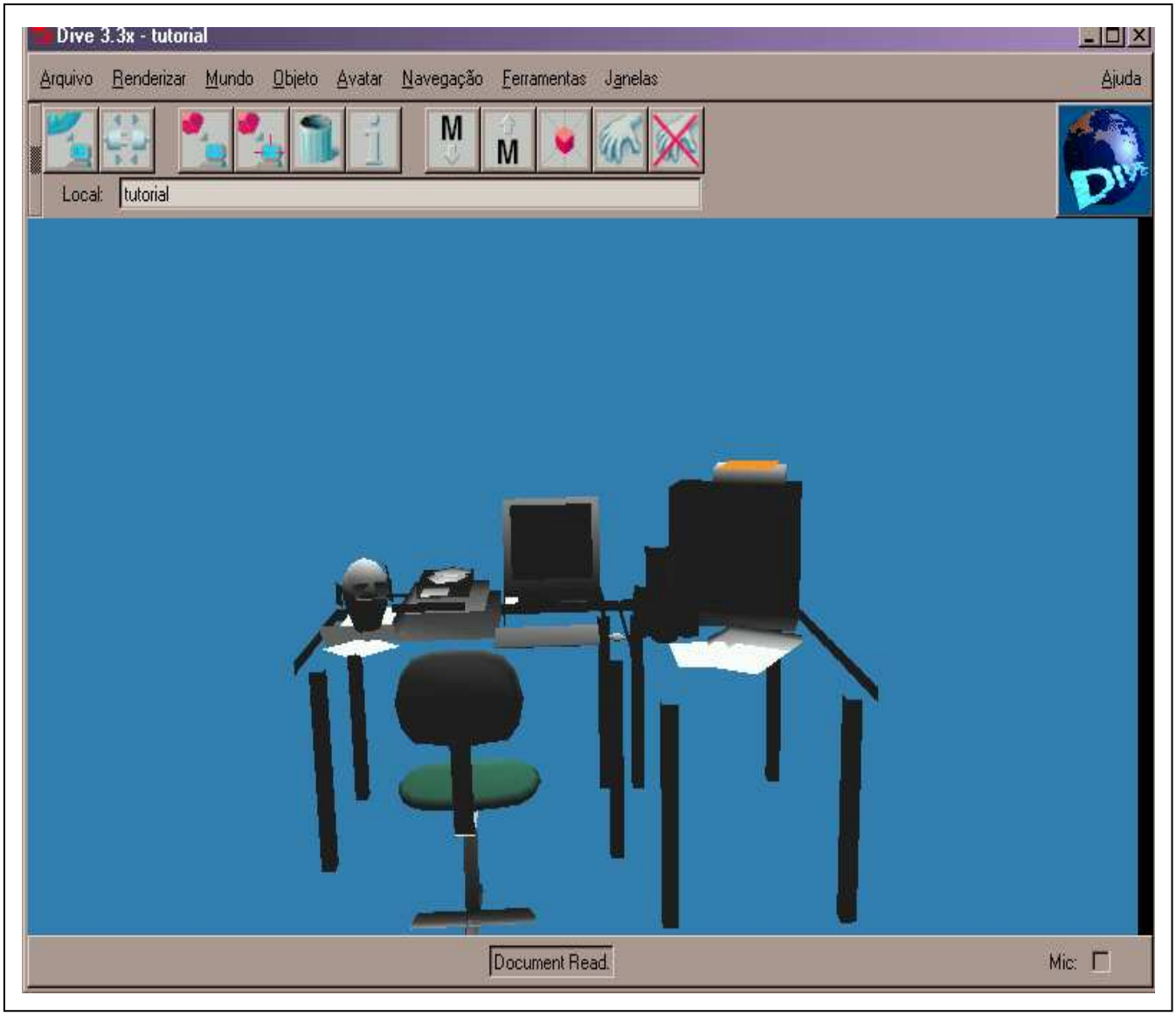


Fig. 5.10 Editor de objetos do visualizador *Vishnu*.

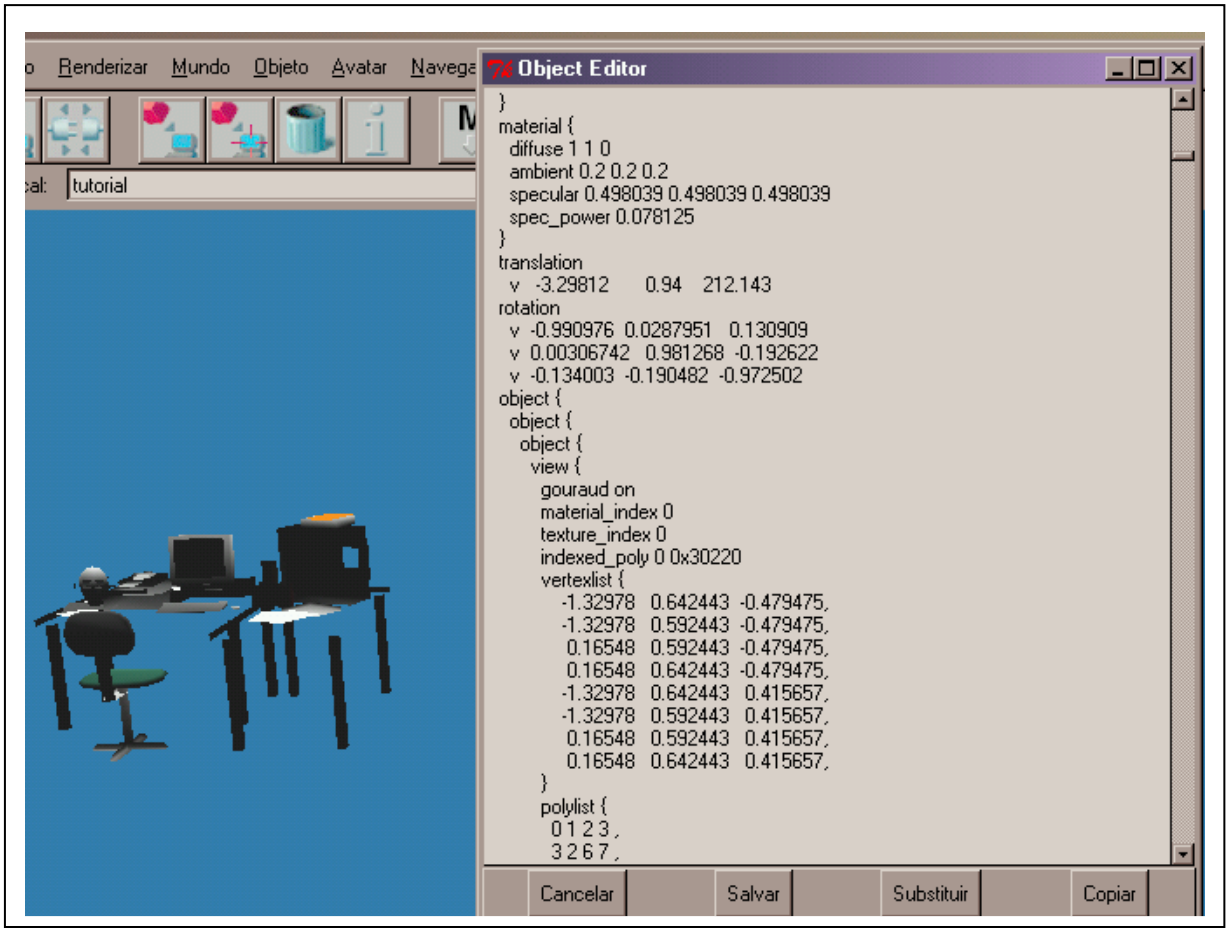
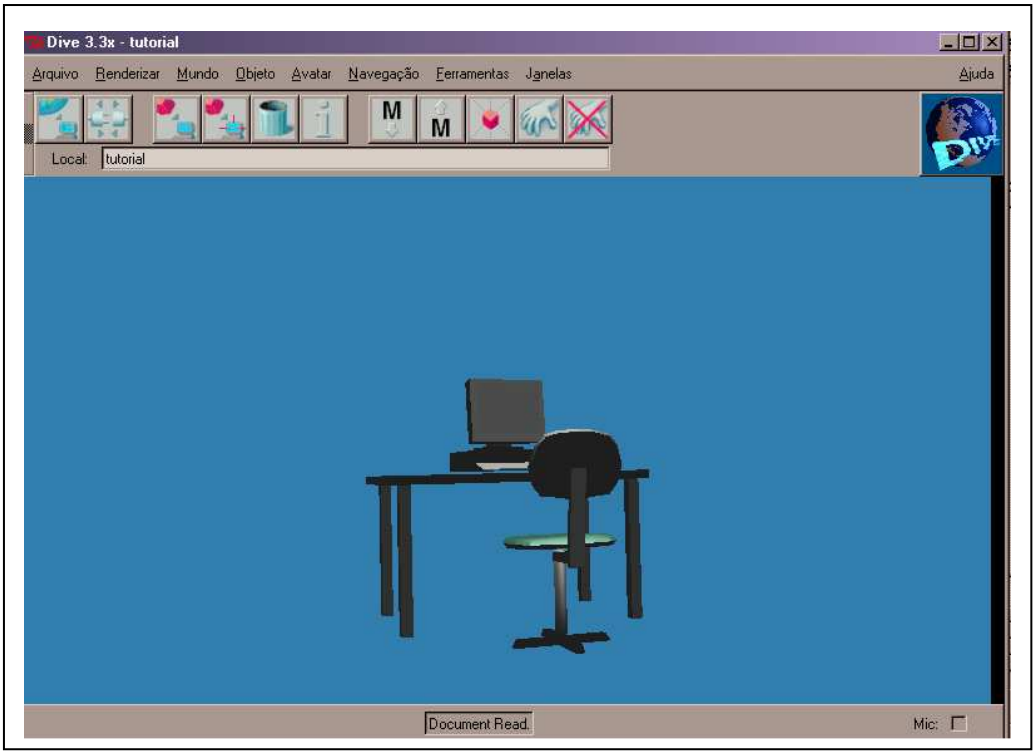


Fig. 5.11 Objeto já alterado para o modelo desejado.



A vantagem em utilizar este processo é que conforme se vai alterando o código de descrição do objeto, pode-se visualizar como que o objeto está ficando. Após as alterações desejadas basta salvar o novo objeto na base de dados com seu respectivo nome.

Com o arquivo contendo a descrição do objeto pronto, bastou somente utilizar o comando "*inline*" para inserir o novo objeto na entidade Protem, especificando a posição do objeto no mundo definido pela sua translação e rotação. O trecho de código que contém a declaração do objeto "computador" na entidade Protem pode ser visto no quadro 5.9.

Quadro 5.9 Declaração de chamada do objeto "computador".

```
object{
  translation
    v    1.15327   0.432701    18.0917
  rotation
    v -0.0232361 -0.00882472  -0.999691
    v -0.0386656   0.999221 -0.00792185
    v   0.998982  0.0384696 -0.0235593
  inline "computador.vr"
}
```

Outra facilidade utilizada foi a utilização de macros (arquivos do tipo *.vh*) e que contém processos específicos para criação de objetos, apenas com a passagem de certos parâmetros.

Um exemplo é a utilização da macro chamada *Walls*.vh que permite criar os mais variados tipos e tamanhos de paredes, contendo também portas e janelas das mais diversificadas. No quadro 5.10 pode-se ver um trecho da macro *Walls*.vh no qual pode-se criar uma parede com duas janelas, passando-se parâmetros como tamanho, altura, largura, material utilizado para a parede, em que altura da parede a renderização da janela começa e onde ela termina, a largura do batente da janela e finalmente o material utilizado para a janela.

Quadro 5.10 Trecho de código da macro *Walls*.vh

```
  +----------+
  |          |
  |          |
  +----------+ --WIN_STOP
  | +--++--+ |
  | |  ||  | |
  | |  ||  | |
  | +--++--+ |
  +----------+ --WIN_START
  |          |
  |          |
  |          |
  +----------+
     LENGTH          */
#define wall_with_two_windows(LENGTH, HEIGHT, WIDTH, MAT, WIN_START,
WIN_STOP, WIN_FRAME, WIN_MAT) \
    material MAT                    \
    view 0 {                              \
      BOXVECTOR 2                         \
      RBOX                          \
     v (-0.5*LENGTH) 0.0 0.0                \
     v (0.5*LENGTH) (WIN_START) WIDTH \
      RBOX                          \
     v (-0.5*LENGTH) (WIN_STOP) 0.0         \
     v (0.5*LENGTH) (HEIGHT) WIDTH          \
    }                               \
  subs object {                           \
    material WIN_MAT                      \
    view 0 {                              \
      BOXVECTOR 5                         \
      RBOX                          \
     v (-0.5*LENGTH) (WIN_START) (0.5*WIDTH)     \
     v (-(0.5*LENGTH)+WIN_FRAME) (WIN_STOP) WIDTH \
      RBOX                          \
     v (0.5*LENGTH) (WIN_START) (0.5*WIDTH)\
     v ((0.5*LENGTH)-WIN_FRAME) (WIN_STOP) WIDTH \
      RBOX                          \
     v ((-0.5*LENGTH)+WIN_FRAME) (WIN_START) (0.5*WIDTH) \
     v ((0.5*LENGTH)-WIN_FRAME) (WIN_START+WIN_FRAME) WIDTH \
      RBOX                          \
     v ((-0.5*LENGTH)+WIN_FRAME) (WIN_STOP-WIN_FRAME) (0.5*WIDTH) \
     v ((0.5*LENGTH)-WIN_FRAME) (WIN_STOP) WIDTH \
      RBOX                          \
     v (-WIN_FRAME) (WIN_START+WIN_FRAME) (0.5*WIDTH) \
     v WIN_FRAME (WIN_STOP-WIN_FRAME) WIDTH\
    }                               \
  }
```

Já no quadro 5.11 pode-se ver a chamada da macro *Walls.vh*, com a passagem dos parâmetros necessários para a criação do objeto desejado, destacando também o comando para inclusão de um arquivo igual a linguagem C.

Quadro 5.11 Chamada da macro *Walls.vh* na entidade Protem

```
#include "dive.vh"
#include "walls.vh"
#define COR_PAREDE "WHITE"
#define COR_JANELA "BROWN"
object{
  name "PAREDE_ESQUERDA"
  object{
    translation
    v    20   0  4
    fixedxyz v 0 1.57 0
    wall_with_two_windows(2,3,0.2, COR_PAREDE, 1.4, 2.4, 0.1,
COR_JANELA)
 }
object {
  translation
    v    20   3  4
  fixedxyz v 0 1.57 0
  wall_with_two_windows(2,3,0.2, COR_PAREDE, 1.4, 2.4, 0.1,
COR_JANELA)
 }
}
```

Para finalizar, um componente que diferencia um objeto qualquer de uma entidade é a declaração de um mundo em sua descrição gráfica. A declaração do mundo dentro da entidade Protem pode ser visto no quadro 5.12.

Quadro 5.12 Declaração de mundo na entidade Protem

```
world "LAB_PROTEM" {
  start      v       12.8    0.1    -13
  position 10 4 -2
  background  0.5 0.5 0.5
  color 1.000000 1.000000 1.000000
  ambient 0.600000 0.600000 0.600000
}
```

A funções existentes na camada de objetos, permitem o desenvolvimento de entidades razoavelmente complexas, com um grau de realismo eficiente e que, juntamente com o auxílio

das ferramentas disponíveis no visualizador Vishnu, torna bastante facilitado e agradável o processo de desenvolvimento destes ambientes.

Na seção seguinte, será mostrado o processo de instalação do visualizador Vishnu, dos arquivos executáveis responsáveis pela comunicação e também da base de dados necessária. Também será apresentado as principais funcionalidades existentes para navegação e compartilhamento do ambiente virtual com os outros usuários.

5.2 INSTALAÇÃO, FUNCIONAMENTO E INTERFACE

A seguir será apresentado os processos de instalação, funcionamento e a interface do protótipo.

5.2.1 INSTALAÇÃO

O processo de instalação é realizado com a utilização do arquivo de *setup* chamado de "Setup_DIVE", o qual quando executado, cria uma pasta de nome DIVE na raiz principal da máquina onde quer se executar o protótipo, copiando para ela além dos arquivos executáveis necessários à parte de comunicação, os *scripts* responsáveis pela criação da interface gráfica, os arquivos que formam a base de dados do sistema bem como outros arquivos necessários ao funcionamento da ferramenta como um todo.

Além destes, existem três arquivos com extensão *bat* chamados de *Bvishnu*, *Bdiveserver* e *Bproxyserver*, que são responsáveis por setar *patchs* que vinculam duas variáveis de sistema, linkando-as com a pasta em que estão os principais arquivos. Além de realizar este serviço, também ativam os arquivos executáveis para comunicação e o aplicativo gráfico.

Portanto qualquer alteração no atalho inicialmente setado na instalação do sistema, deve também ser alterado nos arquivos *.bat* sob risco de não funcionamento do sistema.

O processo de instalação também cria um atalho na área de trabalho para a respectiva pasta que contém os arquivos *.bat* que inicializam o sistema. Também possui um executável que permite desinstalar todo o sistema, excluindo totalmente a pasta anteriormente criada com todo o seu conteúdo com toda segurança.

5.2.2 FUNCIONAMENTO

O processo de funcionamento do protótipo também é bastante simplificado, dependendo do contexto no qual o mesmo será utilizado. Existem três formas possíveis de utilização:

- Num ambiente de rede sem a utilização de um servidor central - para utilizar o sistema em modo multi-usuário, basta inicializar o arquivo chamado Bdiverserver.bat em somente uma das máquinas conectadas à rede para ativar a parte de comunicação, e então inicializar o arquivo Bvishnu.bat para ativar a interface gráfica em todas as outras máquinas que desejarem se conectar, sendo que mesmo a máquina onde está ativado o Bdiveserver.bat pode ter a interface gráfica inicializada sem problema nenhum. O processo de conexão da interface gráfica com a parte de comunicação é automática, não havendo a necessidade de passar nenhum endereço IP;

- Num ambiente de rede com a utilização de um servidor central - aqui, o arquivo Bdiveserver.bat deve ser executado somente no servidor e todas as máquinas clientes devem inicializar a interface gráfica. Na inicialização desta será aberta uma janela que pergunta se a conexão desejada é de usuário simples, de usuário através de um *Name Server* ou um *Proxy Server*, conforme figura 5.12. Neste caso utiliza-se o *Name Server* que abrirá uma janela onde deverá ser colocado o endereço IP e a porta de acesso da máquina servidor onde estám sendo executado o arquivo Bdiveserver.bat, conforme figura 5.13.

Fig. 5.12 Janela inicial do visualizador *Vishnu*

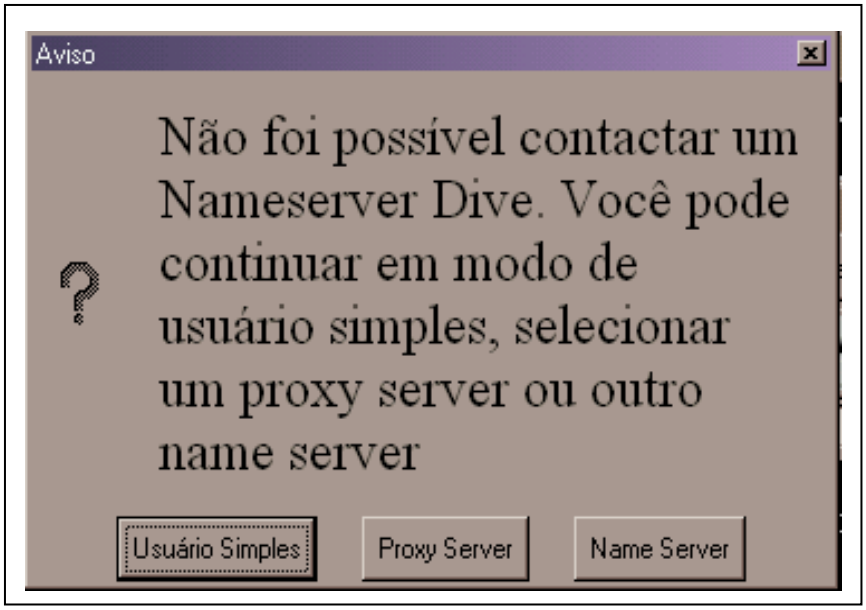
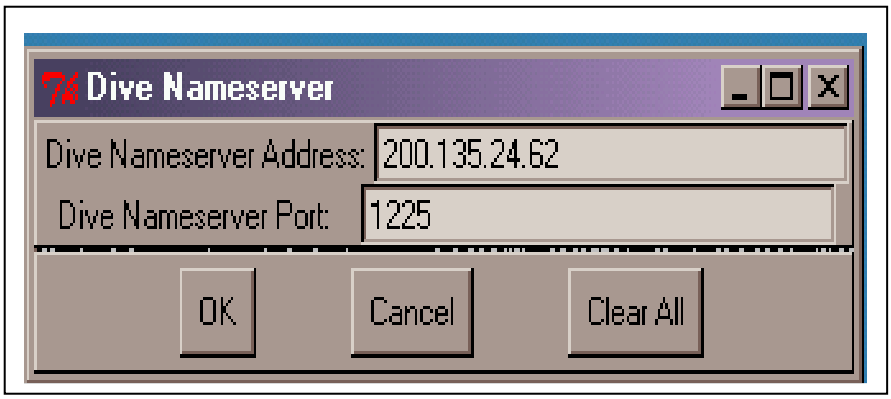
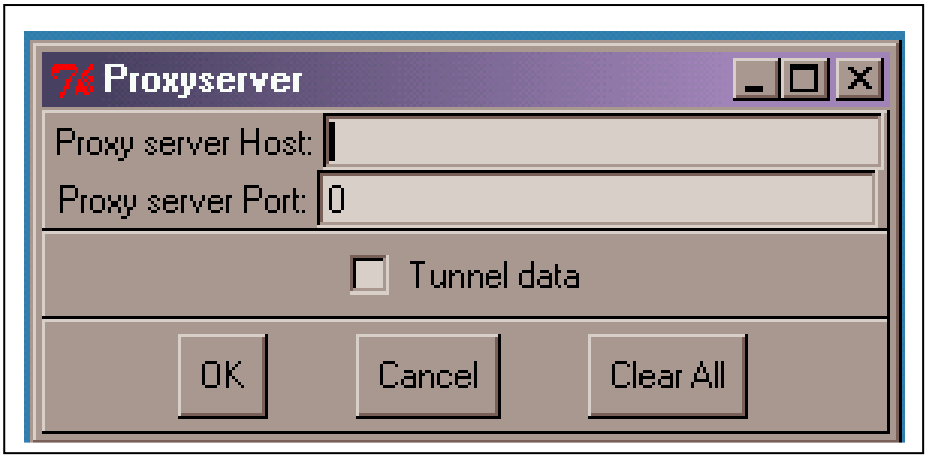


Fig. 5.13 Janela para endereço Name Server



- Acessando um servidor central via conexão discada - aqui, o arquivo Bdiveserver.bat e Bproxyserver.bat devem ser executados no servidor e todas as máquinas clientes devem inicializar a interface gráfica. Na inicialização desta será aberta uma janela que pergunta se a conexão desejada é de usuário simples, de usuário através de um *Name Server* ou um *Proxy Server*, conforme figura 5.12. Neste caso utiliza-se o *Proxy Server* que abrirá uma janela onde deverá ser colocado o endereço IP e a porta de acesso da máquina servidor onde está sendo executado o arquivo Bproxyserver.bat, conforme figura 5.14. Neste caso, o servidor deve ter os dois arquivos arquivos executáveis funcionando pois o *Name Server* será conectado através do *Proxy Server*.

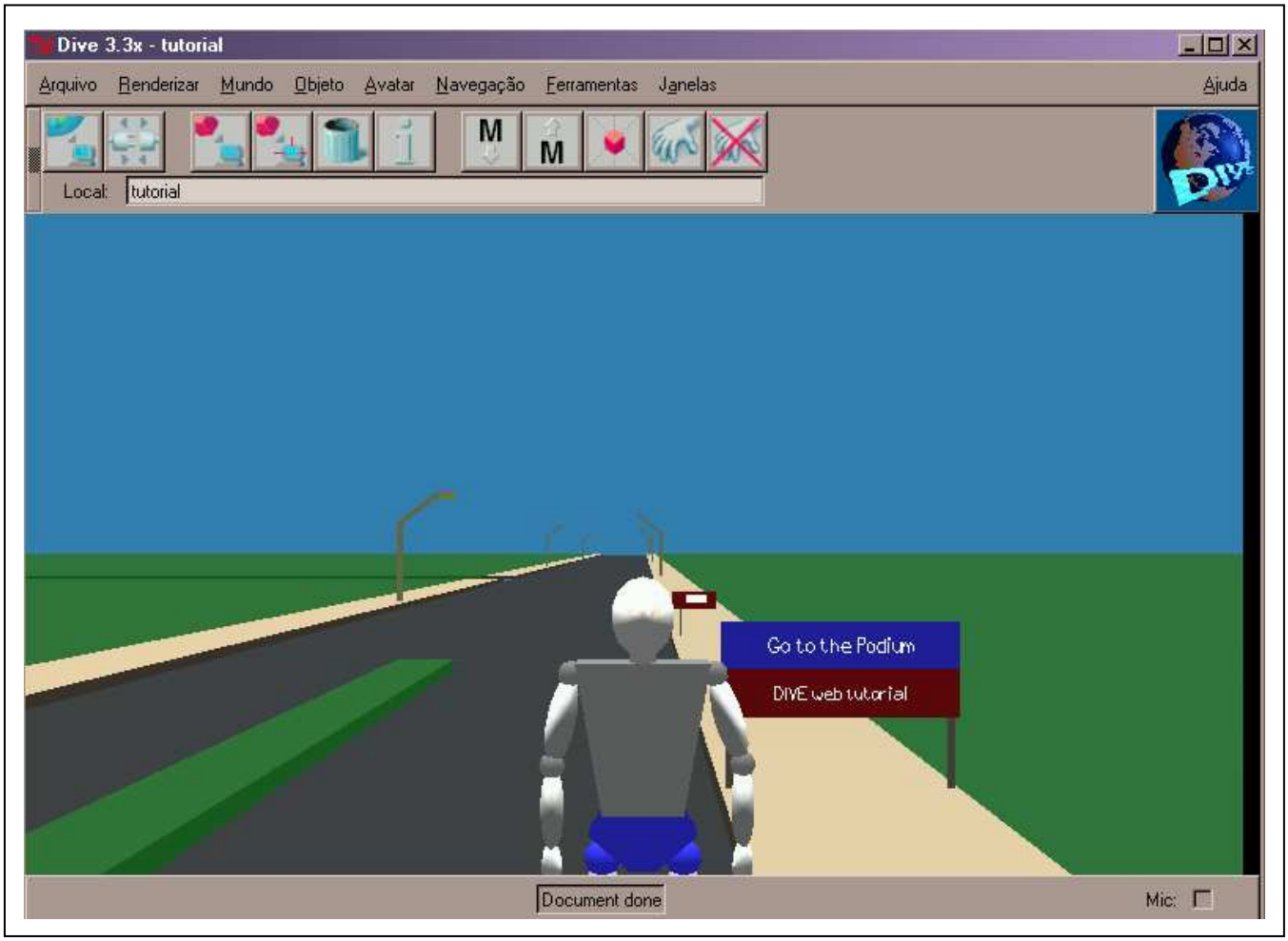
Fig. 5.14 Janela para endereço Proxy Server



5.2.3 INTERFACE

A janela principal do *Vishnu* consiste de três partes: um cabeçalho, um visualizador e uma barra de *status*. O cabeçalho consiste de um conjunto de menus com sub-menus e uma barra de botões com atalhos para algumas operações usuais. O visualizador exibe em 3D o que será visto pelo usuário. A barra de *status* localizada na base da janela mostra informações, tal como o nome do mundo corrente. Uma visão geral do visualizador pode ser visto na figura 5.15.

Fig. 5.15 Visão geral do visualizador



Os menus mais utilizados para navegação no ambiente são os menus a seguir.

5.2.3.1 MENU ARQUIVO

Os comandos mais utilizados no menu de arquivos são:

- **Configurar:** Este é a janela onde contém campos que podem ser alterados em tempo de execução e para alteração do arquivo de configuração *.dive_configure* lido pelo visualizador e que contém informações de inicialização que podem ser alterados. Os itens mais utilizados são:
  - ***Proxy Server*** – Define um novo endereço de acesso para um servidor *Proxyserver;*
  - *Name Server* - Define um novo endereço de acesso para um servidor *Nameserver;*
  - Configurar tabela - Todos valores de *default* no arquivo de configuração são mostrados em uma janela. Os valores de *default* podem ser mudados e salvos;
- **Sair**: Sair da aplicação. Esta opção sempre deve ser utilizada e nunca o botão de fechar na janela da aplicação.

5.2.3.2 MENU MUNDO

O menu do Mundo contém operações que são executadas no ambiente em que o ator está, sendo que os comando mais utilizados são:

- **Carregar:** Conectar a um novo mundo através de uma URL ou nome de arquivo local. O ator deixa o mundo corrente e é transferido para o outro;
- **Salvar:** Salvar mundo completo para um arquivo local.

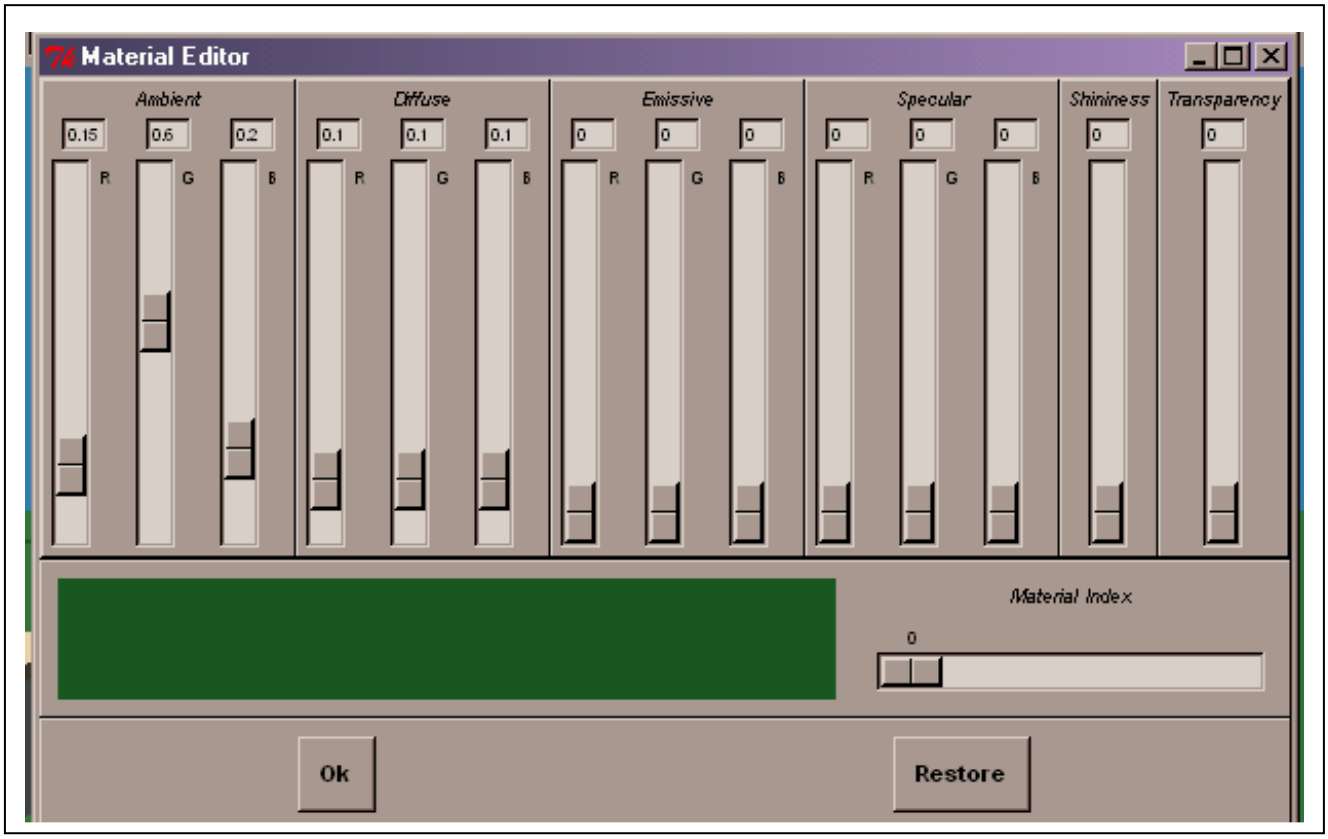
5.2.3.3 MENU OBJETOS

No menu de objetos, diversas operações podem ser desempenhadas. Sendo que as principais são:

- **Carregar em frente do Avatar:** Carregue um objeto através de uma URL ou um arquivo local. O objeto carregado é inserido dentro o mundo corrente em frente do ator;
- **Carregar no local padrão do objeto:** Carregue um objeto através de uma URL ou arquiva local. O objeto carregado é inserido dentro do mundo corrente na posição definida no arquivo (ou URL);
- **Salvar:** Salve o objeto marcado para um arquivo local;

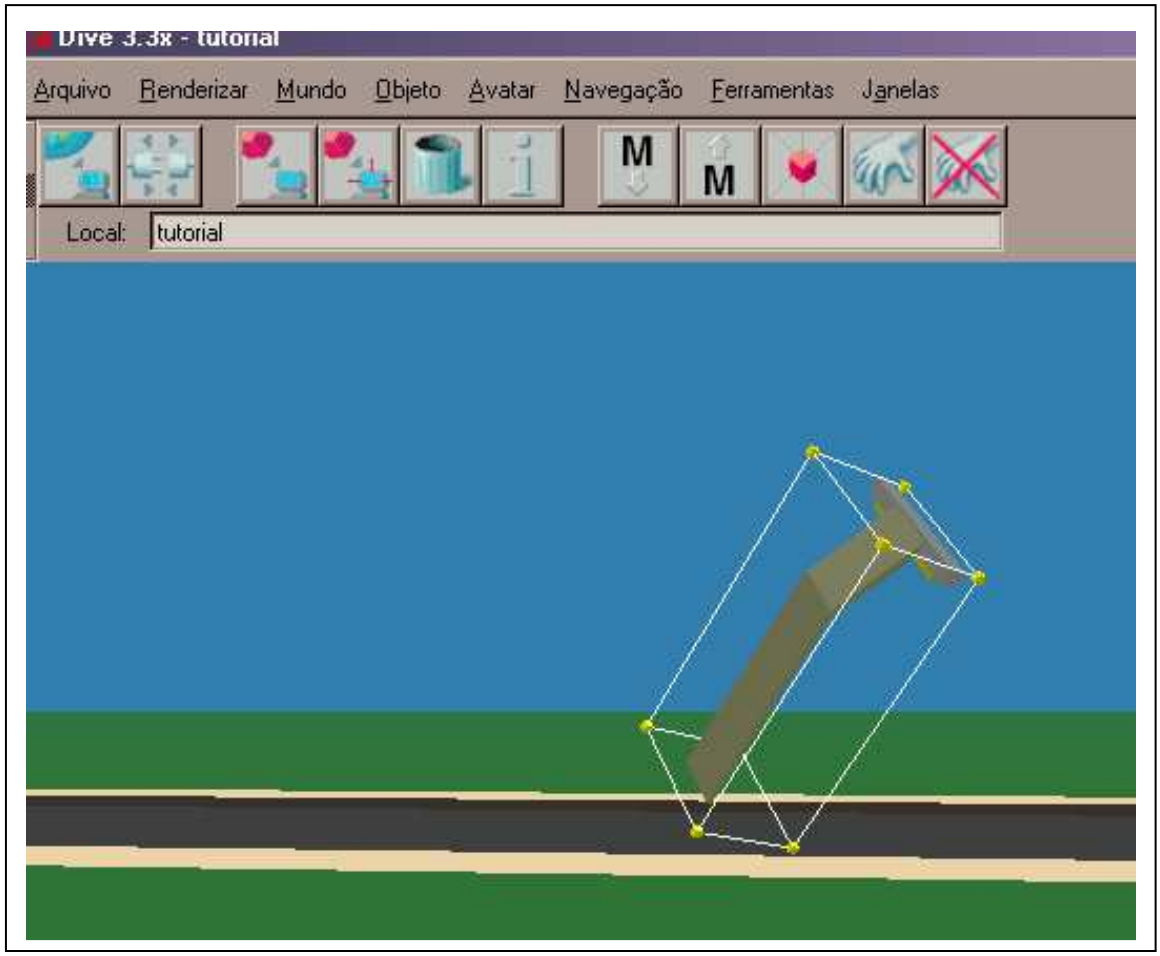
- **Editar/Código:** Exibir o código do objeto selecionado. O código pode ser modificado e salvo ou substituído pelo objeto original;
- **Editar/Material**: Carregar uma aplicação que altera o padrão RGB do material existente em um objeto. A figura 5.16 apresenta este editor;

Fig. 5.16 Editor de materiais em objetos



- **Editar/Caixa de edição**: Cria uma caixa de edição ao redor de um objeto marcado.Permite alterar tamanho, altura e largura de um objeto diretamente no ambiente. A figura 5.17 apresenta este editor;

Fig. 5.17 Editor de objetos diretamente no ambiente



- **Dive/Tcl Console:** Inicia um console onde pode-se inserir *scripts* de comportamento a um objeto em tempo de execução;
- **Deletar:** Excluir objeto selecionado;
- **Adicionar sub-objeto:** Usado para compor um único objeto utilizando-se para isso outros objetos.

5.2.3.4 MENU AVATAR

Possui funções relacionadas com o Ator existente na aplicação. As funções mais utilizadas são:

- **Carregar:** Carregue um novo Avatar através de uma URL ou arquivo local, sendo que o Avatar antigo é deixado para trás;

- **Salvar:** Salve o Avatar atual para arquivo local;
- **Dive/Tcl Console:** Inicia um console onde pode-se inserir *scripts* de comportamento ao Avatar em tempo de execução;
- **Olho Direito:** Muda o ponto de visão para o olho direito do Avatar;
- **Olho Esquerdo:** Muda o ponto de visão para o olho esquerdo do Avatar;
- **Visão Topo:** Muda o ponto de visão para o topo da cabeça do Avatar;
- **Visão Lateral:** Muda o ponto de visão para a lateral do Avatar;
- **Visão Traseira:** É o ponto de visão mais utilizado. Muda o ponto de visão para trás do Avatar;
- **Mudar Avatar:** Mude a consciência do Avatar para algum outro objeto selecionado, sendo que o ponto de visão passa automaticamente para o novo Avatar;
- **Desfaça mudança de Avatar:** Desfazer a mudança de consciência do objeto selecionado novamente para o Avatar.

5.2.3.5 MENU NAVEGAÇÃO

Para navegar no ambiente virtual o menu de navegação possui funções que criam ferramentas chamadas de veículos e que, ou implementa controles próprios para navegar ou utiliza dispositivos específicos que transmitem os movimentos do usuário para o ambiente virtual. Os principais veículos são:

- **Teclado:** Habilita a utilização do teclado para navegação, sendo que certas teclas executam funções de navegação diferentes. As teclas e a operação realizada por elas são:
  - **Seta para Esquerda** -Virar para esquerda;
  - **Seta para direita** -Virar para direita;
  - **Seta para cima** - Mover para frente;
  - **Seta para baixo** - Mover para trás;
  - **Tecla – ou +** - Adiciona mais ou menos zoom ao ponto de vista do Avatar;
  - **Tecla *Page up* ou *Page down*** - Aumenta ou diminui a velocidade do Avatar;
  - **Teclas de F1 até F6** - Gira a câmera e muda a visão de olho.

- **Cabeça:** Uma ferramenta em Tk que habilita o movimento independente da cabeça do Avatar;
- **Translação**: Uma ferramenta em Tk para movimento de translação simples;
- **Ir para o ponto inicial:** Move o Avatar para posição de entrada no mundo, declarada na descrição do mundo;
- **Aumentar velocidade:** Aumenta velocidade de movimentação do Avatar;
- **Diminuir velocidade:** Diminui a velocidade de movimentação do Avatar.

5.2.3.6 MENU DE FERRAMENTAS

Os principais comandos utilizados no menu de ferramentas são:

- **Pontos De Vista:** Exiba uma lista de todos pontos de vista (câmeras) no mundo corrente. Selecione "*Go To*" ou clique duplo em um ponto de vista para ir ao ponto de vista;
- **Mundos:** Apresenta (automaticamente) uma lista atualizada de todo os mundos que foram visitados desde o começo de uma sessão. Para retornar a algum dos mundos listados basta dar um duplo clique no nome do mundo ou clicar uma vez no nome e pressionar o botão "*Go To*" para ser teleportado;
- **Atores:** Exibe uma lista de todos os atores que passaram por um mundo. Um ator pode ser selecionado na lista, e operações que incluem o envio de mensagens de texto, ao ator selecionado são disponíveis;
- **Enviar mensagens para todos:** Envie mensagens para todos os outros participantes de um mundo.

5.2.3.7 INTERAÇÃO COM O MOUSE

A interação entre o usuário e os objetos existentes em um ambiente virtual, são basicamente realizadas com a utilização do mouse sendo que as quatro principais operações de interação disponíveis são ativar, selecionar, mover e rotacionar um objeto.

Toda operação realizado sob um objeto só pode ser realizada estando este objeto marcado. Isto também funciona como um sistema de proteção para manter a fidelidade no

ambiente, já que um objeto selecionado para manuseio só estará liberado para outro usuário a partir do momento em que for liberado pelo primeiro.

Para selecionar um objeto, clique sobre ele utilizando o botão esquerdo do *mouse* e aperte, ao mesmo tempo, a tecla *Shift* do teclado. O objeto piscará uma vez indicando que está selecionado.

Para mover um objeto basta clicar nele com o botão do meio do mouse e mantê-lo pressionado enquanto movimenta o mouse, fazendo com que o objeto se mova também.

Para rotacionar um objeto basta clicar nele com o botão direito do mouse e mantê-lo pressionado enquanto realiza a rotação desejada.

A operação de ativação de um objeto acontece à partir do momento em que se seleciona este objeto. Agora para desativar (ou retirar a seleção do objeto) basta clicar normalmente com o botão esquerdo do mouse e o objeto passa a ficar liberado.

Neste capítulo foi apresentado a especificação e detalhes da implementação do protótipo de ambiente virtual distribuído. Além da apresentação do modelo de especificação utilizado, foram apresentados os principais processos necessários para a implementação do sistema, buscando relacionar as funções utilizadas em cada processo de desenvolvimento com o modelo de implementação da arquitetura Dive Core.

Também foram apresentadas características relacionadas com a instalação, funcionamento e a interface do protótipo.

O capítulo seguinte apresenta as conclusões alcançadas ao longo do desenvolvimento desse trabalho, bem como as possíveis extensões que podem enriquecer ainda mais o conteúdo e protótipo resultantes desse trabalho.

6 CONCLUSÕES

No decorrer deste trabalho, pôde-se observar aspectos importantes relacionados à construção de ambiente virtuais distribuídos e, principalmente, os mecanismos existentes na plataforma DIVE que possibilitam a implementação deste tipo de sistema.

Entre algumas vantagens observadas com a utilização desta plataforma para o desenvolvimento do trabalho, pode-se destacar o fornecimento das bibliotecas que compõem a plataforma e também os códigos fonte dos principais arquivos de sistema, de forma gratuita e aberta permitindo assim obter-se um maior entendimento das soluções utilizadas, por não haver dependência de tecnologia proprietária.

Outra vantagem é a flexibilidade existente com relação ao tipo de linguagem utilizada no desenvolvimento e também no modelo de implementação apresentado, que permite direcionar os esforços para a obtenção de um objetivo específico no que diz respeito aos processos existentes para construção do ambiente virtual distribuído.

Relacionando-se aos objetivos previamente estipulados, pode-se afirmar que foram atingidos, já que inicialmente a intenção era implementar um aplicativo composto com objetos simples e ter-se uma visão em primeira pessoa utilizando o DIVE. Num segundo momento pensou-se em fazer o aplicativo chegando a utilizar a nível de aplicação da linguagem C, mas verificou-se que pela quantidade de linhas de código gastaria-se um tempo considerável. Optou-se por não inicialmente implementar a aplicação como um “todo”, pois tinha como objetivo inicial ter-se um protótipo que validasse o seu funcionamento dentro do cronograma.

Com referência as contribuições do trabalho para o grupo de usuários, possui um caráter positivo, pois apresenta um estudo das características principais da plataforma DIVE, plataforma esta que é relativamente recente e que por isso possui poucos trabalhos relacionados a sua utilização para a construção de ambientes virtuais, e que na maioria das vezes são implementados utilizando-se as linguagens Java e Vrml.

Outro ponto positivo foi relacionado com o hardware necessário para a utilização do protótipo desenvolvido com a plataforma DIVE. O protótipo foi validado utilizando-se dois computadores com processador *Pentium* 450mhz, com 128mb de memória, um deles com sistema operacional Windows 2000 e o outro com Windows NT 4.0, e apresentou uma performance satisfatória, não comprometendo o realismo na interação com o ambiente virtual.

6.1 EXTENSÕES

As possíveis extensões que podem ser feitas a partir deste trabalho estão enumeradas a seguir:

- Desenvolver um estudo mais aprofundado das funções existentes na plataforma e implementadas na linguagem C, bem como das bibliotecas existentes, desenvolvendo um aplicativo completo à partir dos mesmos;

- Desenvolver outros tipos de aplicação, não somente visualizadores de ambiente virtuais, mas também aplicativos que permitam criar e interagir com objetos mais complexos e que possibilitem desenvolver atores que possuam algum nível de “inteligência”;

- Pesquisar alguma política para utilização de mundos distribuídos, evitando a replicação sob demanda dos mesmos;

- Utilizar o modelo de descrição de arquivos Dive e criar mundos virtuais mais complexos, utilizando texturas, sons, vídeos e níveis de interação maior entre os usuários que compartilham o ambiente e também entre os objetos que o compõem.

REFERÊNCIA BIBLIOGRÁFICA

ANDERSON, Magnus ;. **DIVE – *The Distributed Interactive Virtual Environment* .** *Technical reference manual.* Suécia, 1994. Disponível em:< ftp://ftp.yars.free.net / pub/doc/ vr/dive/diveuser.ps>.Acesso em: 21 Abr. 2002

AVATARE, Anneli ; FRÉCON , Emmanuel . **DIVE – *The Distributed Interactive Virtual Environment* .** *DIVE Files Description for DIVE version 3.3.* Suécia, [1999]. Disponível em:< http://citeseer.nj.nec.com/avatare97dive.html>.Acesso em: 08 out. 2001

BARRUS, J.W. *"Locales: Supporting large multiuser virtual environments"*. **IEEE *Computer Graphics and Applications* ,** vol 16, no.6, pp 50-57, Novembro 1996.

DIVE. ***The DIVE home page,*** Suécia, [2001]. Disponível em: <http://www.sics.se/dive/>. Acesso em: 08 out. 2001.

EDUARDO, Vandeir. **Protótipo de um ambiente virtual distribuído multiusuário,** 2001.108 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau/ Blumenau.

ELLIS, S.R. *What are virtual environment?* **IEEE *Computer Graphics and Applications* ,** vol 14, no.1, pp 17-22, Janeiro 1994.

FRÉCON, Emmanuel. **DIVE: A scaleable network architecture for distributed virtual environments,** Suécia, 1998. Disponível em:<http://www.sics.se/~emmanuel /publications/ dsej/dsej.html>. Acesso em: 08 Maio. 2002

GREENHALD, C. ***MASSIVE: A Distributed Virtual Reality System Incorporating Spatial Trading.*** Inglaterra, [1995]. Disponível em: ftp://ftp.crg.cs.nott .ac.uk/pub/ papers / DSC95.ps.gz > Acesso em : 03 Jan 2002

HAGSAND, Olof. *Interactive multiuser Ves in the Dive system* **IEEE *Computer Graphics and Applications* ,** vol 03, no.1, pp 30-39, Janeiro 1996.

KEUKELAAR, Johannes H. ***External event handling in Dive, a virtual system,*** Estocolmo, 1996. Disponível em:_< http://www.nada.kth.se/~johannes/work/report.html>. Acesso em: 06 Maio. 2002.

KIRNER, Cláudio. **Sistemas de realidade virtual,** São Carlos, [2000?]. Disponível em: <http://www.dc.ufscar.br/~grv/tutrv/tutrv.htm>. Acesso em: 26 out. 2001.

MAKRAKIS, Dimitrios; HAFID, Abdelhakim. ***Quality of service management in distributed interactive virtual environment,*** Uottawa, [2001?]. Disponível em: <http:// www. mcrlab. uottawa.ca/research/QoS_DIVE_Report.html>. Acesso em: 01 nov. 2001.

NPSNET, Naval Postgraduate School. **NPSNET *Research* Group.** Monterey: [s.n],[2000?]. Disponível em: http://www.npsnet.org/NPSNET-V/ > Acesso em : 22 Jan 2002

OLIVEIRA, Manuel Fradinho; PEREIRA , João Madeiras. ***Virtual worlds* – uma arquitetura para ambientes virtuais,** Lisboa, [2001?]. Disponível em: <http:// virtual. inesc.pt/virtual/8epcg/actas/c16/>. Acesso em: 01 nov. 2001.

RAPOSO, Alberto Barbosa. **Interação na WEB –** tendências. Campinas, [2000?]. Disponível em: http://www.dca.fee.unicamp.br/courses /IA368F/1s1998 /Monografias/alberto/. Acesso em: 01 nov. 2001.

STENIUS, Marten. ***Collaborative object modelling in virtual environments***, Estocolmo, [1996]. Disponível em: <http://www.d.kth.se/~d90-mst/masters/>. Acesso em: 06 out. 2001

TÁXEN, Gustav. ***A practician´s guide to DIVE,*** Estocolmo, [2000]. Disponível em: <http://citeseer.nj.nec.com/327745.html>. Acesso em: 23 out. 2000.

TEIXEIRA, Renata Cruz .**Uma plataforma escalável para sistemas distribuídos de realidade virtual,** Rio de Janeiro, [1999]. Disponível em: <http://www.cse.ucsd.edu/~teixeira /Teix99.ps.gz >. Acesso em: 29 out. 2000.

WRAY, Mike ; HAWKES, Rychard. ***Distributed virtual environments and VRML:*** *an event-based architecture*, Bristol, [1999?]. Disponível em: <http:// keryxsoft. hpl. hp. com/ documents/dve/vrml.htm>. Acesso em: 29 out. 2000.

VILHJÁLMSSON, Hannes Hogni . ***Autonomous communicative behaviours in avatars,*** Estocolmo, [1997]. Disponível em: <http://www.media.mit.edu/ ~hannes/msthesis/ avatars_thesis.pdf >. Acesso em: 06 out. 2001