UMA PLATAFORMA INTEGRADA PARA APLICAÇÕES DE REALIDADE VIRTUAL DISTRIBUÍDA E COLABORATIVA

Pedro Ferreira[1], João Orvalho[2] e Fernando Boavida[3]

Laboratório de Comunicações e Serviços Telemáticos
CISUC – Centro de Informática e Sistemas da Universidade de Coimbra
Polo II, 3030 COIMBRA – PORTUGAL
Tel.: +351-239-790000, Fax: +351-239-701266
E-mail: {*pmferr* , *orvalho, boavida@dei.uc.pt*}

Sumário. Neste artigo descrevemos uma plataforma integrada para suporte a aplicações de realidade virtual distribuída e colaborativa ou DCVE - Distributed Collaborative Virtual Environments. Descrevemos os requisitos deste tipo de aplicações, a arquitectura da plataforma desenvolvida e seus pormenores, culminando por fim com a descrição e análise dos testes efectuados até ao momento.

1. Introdução

O desenvolvimento de aplicações de realidade virtual distribuída e colaborativa – DCVE – Distributed Collaborative Virtual Environments – possui vários aspectos e obedece a múltiplos requisitos.

Os aspectos mais importantes são dois. O primeiro é o problema da visualização da informação tridimensional, sua representação interna no computador e transformação visual de forma a que o utilizador perceba o que está a ver. Para resolver este problema, existem várias APIs já provadas e testadas, como sejam a OpenGL – Open Graphics Library [25] – e a Java 3D [24], para além da DirectX/Direct3D [26].

---

[1] Departamento de Engenharia Informática da Escola Superior de Tecnologia, Instituto Politécnico de Tomar
[2] Escola Superior de Educação, Instituto Politécnico de Coimbra
[3] Departamento de Engenharia Informática da Faculdade de Ciências e Tecnologia, Universidade de Coimbra

Mas o desenvolvimento deste tipo de aplicações envolve também a transmissão do estado global do ambiente virtual, que pode ser decomposto, como sugere Georganas [9], nos estados individuais de cada objecto representado no mundo que a aplicação manipula, através da rede.

Para este último problema, embora existam várias soluções específicas para cada aplicação desenvolvida e extensa investigação na área, ainda não existe uma plataforma de middleware ou uma API específica para resolver este tipo de questões.

Tendo isto em consideração iniciámos a concepção, especificação, desenvolvimento e programação de uma plataforma integrada para suporte a aplicações de CSCW – Computer Supported Collaborative Work – síncronas e, de forma mais especial, de DCVE – Distributed Collaborative Virtual Environments.

A plataforma que estamos a desenvolver permitirá a eficiente programação de vários tipos de aplicações integradas no seu sistema de conferência, utilizando as várias APIs, específicas para cada tipo de aplicação, que serão desenvolvidas.

Neste artigo descrevemos a arquitectura geral desta plataforma de suporte a aplicações de CSCW e DCVE, descrevendo de forma sucinta os seus principais blocos constituintes. Descrevemos em especial pormenor a API STF – Status Transmission Framework [1] – específica para o desenvolvimento de aplicações DCVE.

2. Requisitos das aplicações DCVE

As aplicações de DCVE – Distributed Collaborative Virtual Environments – possuem uma série de requisitos específicos [7], para além dos restantes requisitos que são comuns a qualquer aplicação de trabalho colaborativo.

Entre estes requisitos está a escalabilidade da plataforma de middleware que suporta a aplicação, a necessidade de uma boa interacção com o utilizador e a garantia de que a consistência do

ambiente virtual distribuído se mantém[10][11]. Muitas aproximações e algoritmos existem para tentar resolver estes problemas [8].

Para manter a consistência do ambiente virtual distribuído é necessário manter em valores aceitáveis factores como a latência ou o jitter [16], que influenciam decisivamente a interacção de um humano com o ambiente virtual representado[18]. Os valores de latência máximos normalmente aceites de forma a não perturbar a interacção são 100 ou 200 milisegundos, conforme os autores, para a latência entre transmissor e receptor.

Outros factores podem influir no correcto funcionamento destas aplicações, nomeadamente a enorme quantidade de tipos de média a suportar – heterogeneidade dos dados, a alta frequência dos eventos, o elevado número de participantes, atrasos de sincronização e a frequência de actualização do écran de visualização[12].

É comum e desejável a utilização de plataformas de comunicação confiável, que garantam a entrega das mensagens quando necessário. Muitas aplicações necessitam também de outros mecanismos, incluindo a criação de diferentes tipos de ADU – Application Data Units -, um mecanismo simples mas confiável de detecção de mensagens perdidas e fora de ordem, um meio de distinguir diferentes tipos de dados, um meio de identificar participantes e por fim um mecanismo de timestamping.

Todos estes requisitos adicionam grande complexidade ao nível da aplicação. Foi para aliviar essa complexidade e movê-la para o middleware que a plataforma integrada CONCHA 2.0, ARMS [6] e STF [1], foi desenvolvida.

3. A plataforma integrada CONCHA – ARMS - STF

A plataforma integrada de middleware desenvolvida, representada no diagrama da Figura 1, está construída com base em diversos módulos interdependentes.

O primeiro destes módulos é o CONCHA – CONference system based on java and corba event CHAnnels – versão 2.0. Trata-se de um sistema de conferência que funciona e foi desenvolvido em Java utilizando o JSDT – Java Shared Data Toolkit versão 2.0 [28]. Este sistema inclui também suporte para a utilização de canais de comunicação ARMS[6] para as aplicações em funcionamento no sistema de conferência.

O segundo módulo é o ARMS, integrado no sistema CONCHA, que é responsável pela entrega de eventos via um protocolo de IP multicast confiável.

A funcionar por cima de canais ARMS ou de canais JSDT, temos a API STF – Status Transmission Framework [1], que constitui o terceiro módulo da plataforma. Esta API foi desenvolvida tendo em conta os requisitos específicos dos DCVEs e pretende fazer a interface entre este tipo de aplicações e o resto do sistema de conferência.

Temos pois que as aplicações suportadas por esta plataforma podem fazer uso da API básica do CONCHA, caso sejam uma aplicação CSCW – Computer Supported Collaborative Work - comum, ou da mais específica API STF, caso se trate de aplicações DCVE.

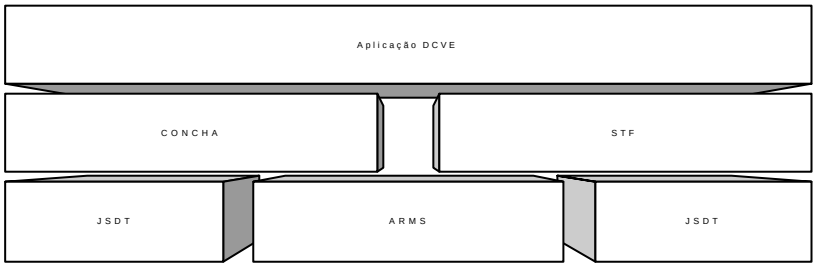

Figura 1 - Diagrama de blocos da plataforma

Passamos agora a descrever cada um destes blocos mais em pormenor.

3.1. ARMS – Augmented Reliable Multicast Corba Event Service

O ARMS – Augmented relyable Multicast Corba Event Service [6] – é uma extensão ao serviço de eventos CORBA padrão de modo a incluir suporte a distribuição multicast confiável de eventos, via protocolo lrmp[22]. Esta extensão é completamente compatível com o serviço de eventos CORBA padrão e pode mesmo ser utilizada de forma mista, com alguns emissores e receptores a utilizar multicast e outros a utilizar o serviço de eventos padrão.

O ARMS fornece uma arquitectura de comunicação com mecanismos de suporte a QoS (Qualidade de Serviço). A API possui mecanismos para garantir a confiabilidade em ambientes multicast, controlo de congestão e controlo de jitter.

O mecanismo de gestão de QoS é suportado por funções de adaptação e monitorização, segundo uma arquitectura orientada a objectos.

A API ARMS possui também objectos específicos para monitorização de perdas e jitter.

3.2. STF – Status Transmission Framework

A STF – Status Transmission Framework [1] – é uma API para manuseamento de estados em ambientes de realidade virtual distribuída. Esta API é capaz de funcionar por cima de qualquer tipo de canal suportado pelo sistema de conferência CONCHA, o que actualmente corresponde a canais JSDT[28] – sobre http, lrmp ou sockets – e a canais ARMS [6] padrão, multicast ou mistos.

A STF permite que a aplicação transmita e receba estados seja qual for a sua precisa composição. A API fornece métodos para transmitir e receber estados, representados por uma interface Java específica para o efeito. Para a STF, um estado pode ser categorizado segundo três critérios.

Segundo a sua importância, um estado pode ser essencial ou redundante. Enquanto um estado essencial chega sempre ao destino, um estado redundante pode não chegar ao destino. Caso o transmissor

esteja a fornecer estados para transmissão pela STF mais rapidamente do que esta os consegue processar e transmitir, os estados mais velhos são efectivamente descartados e substituídos pelos mais novos.

Segundo a sua volatilidade, um estado pode ou não ser volátil. Se um estado for volátil, pode nunca chegar ao destino, podendo inclusivamente perder-se na rede, isto é, na camada ARMS.

Com respeito ao processo de junção tardia, mecanismo para permitir que qualquer cliente se junte a uma sessão de conferência em curso e automaticamente reconstrua o estado actual do ambiente de trabalho partilhado [2][3][4][5], um estado pode ser considerado como independente ou cumulativo. Se um estado for independente, só o ultimo estado transmitido necessita de ser arquivado no nó local para posterior transmissão em processos de junção tardia. Um estado cumulativo é um estado transmitido que na realidade representa uma mudança incremental no estado de um objecto. Assim, é necessário arquivar no nó local todos os estados transmitidos até agora de forma a poder reconstruir futuramente o estado actual do objecto.

A STF organiza as mensagens de estado em chamadas streams de interacção, representadas na Figura 2, que correspondem aos objectos existentes no ambiente virtual. Cada stream de interacção consiste nas mensagens de estado transmitidas para esse objecto – identificado por uma chave – ordenadas por timestamp. Cada mensagem de estado – STFStateMessage – inclui a chave, o estado propriamente dito (dados e categorização), o timestamp e uma prioridade variável.

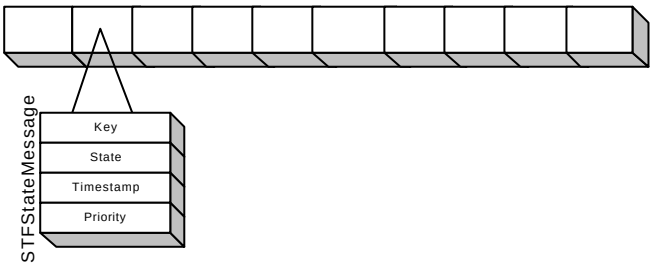

Figura 2 - Streams de interacção STF

A STF possui também mecanismos para controle de lag, que pode ser de recepção e transmissão. Este lag destina-se a evitar a ocorrência demasiado frequente do fenómeno de timewarp – uma mensagem que chega á aplicação receptora fora de ordem e que causa um reajuste forçado do ambiente virtual representado no nó receptor ou mesmo a completa disrupção do serviço[10]. Quando o timewarp não pode ser evitado, a STF fornece também meios para notificar a aplicação de forma simples e eficaz quando tal acontece, simplificando ao máximo o tratamento destas ocorrências.

A sincronização temporal dos participantes no ambiente virtual comum é bastante importante para o bom funcionamento da STF. Por isso a STF possui um mecanismo interno de sincronização, baseado em um algoritmo distribuído. Este mecanismo pode ser selectivamente activado, uma vez que não é incomum a possibilidade de sincronizar externamente os computadores através do protocolo NTP[13].

Por último a STF suporta também processos de junção tardia[2][3][4][5], fornecendo métodos para controlo do arquivo automático de estados transmitidos, de categorização dos estados a transmitir e de controle do próprio processo de junção tardia, baseado em um algoritmo completamente distribuído que dispensa completamente a existência de um servidor central de junção tardia. A arquitectura desenvolvida liberta assim a aplicação da responsabilidade de suportar este tipo de funcionalidade, movendo-a para o middleware.

A arquitectura suporta também checkpointing, ou seja, apagamento de anteriores mensagens arquivadas em determinado ponto no tempo. Este checkpointing pode ser local ou distribuído simultaneamente por todos os nós do sistema. Alternativamente, podemos ter a STF a gerar automaticamente checkpoints em períodos de tempo configuráveis.

3.3. CONCHA – CONference system based on java and corba event CHAnnels

O CONCHA – CONference system based on java and corba event CHAnnels[30] – é um sistema de controle de conferências

implementado sobre JSDT – Java Shared Data Toolkit[28] – que permite utilização de canais lrmp – lightweight relyable multicast - , http e sockets.

O CONCHA evoluiu agora para a sua versão 2.0. Foi incluída agora funcionalidade sobre canais de eventos ARMS – padrão e multicast – e também a API STF – Status Transmission Framework - a funcionar sobre qualquer tipo de canal suportado pelo sistema. Foi também incluído suporte a bases de dados via JDBC – Java Database Connectivity [31]– para as bases de dados de controle do sistema.

O CONCHA 2.0 suporta qualquer tipo de aplicação de colaboração distribuída e possui agora, com a utilização da API STF, suporte específico para ambientes de realidade virtual distribuída.

O sistema baseia-se na utilização do JSDT – Java Shared Data Toolkit[28] – baseado nas normas ITU T.124 e os seus mecanismos de controle são baseados na norma ITU T.120.

A interface gráfica do sistema está representada na Figura 3, onde se podem ver os vários canais e aplicações existentes em dado instante. Podem também ser vistas duas aplicações, uma de conversação – chat – e outra de desenho livre colaborativo – whiteboard, implementados sobre a API aplicacional básica CONCHA.

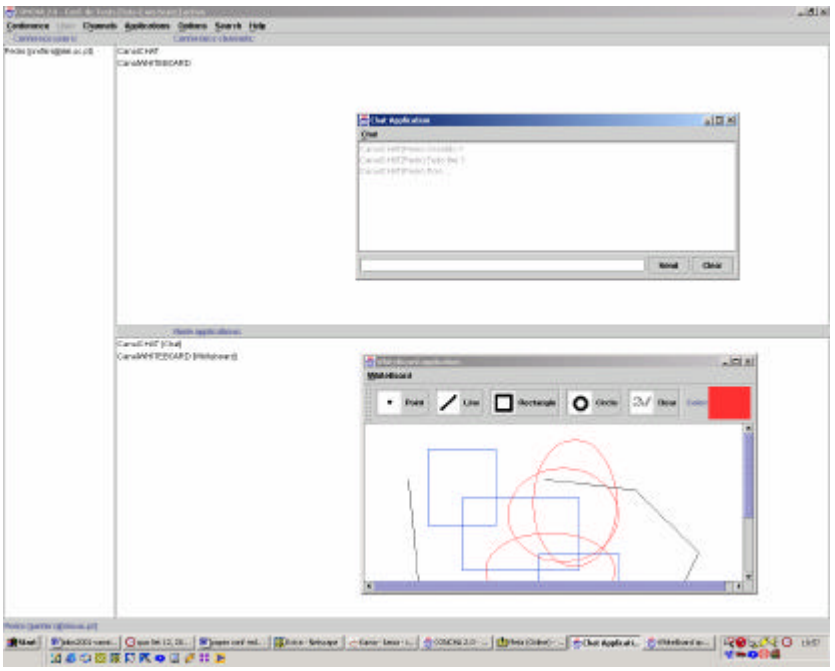

Figura 3 - O sistema CONCHA 2.0 em funcionamento

O sistema CONCHA baseia-se numa arquitectura de controle hierárquica, em que é possível e desejável a existência de nós proxy, que descentralizam as operações de controle e diminuem as necessidades de tráfego através da árvore de canais de controle. A comunicação entre as aplicações propriamente ditas é feita através de canais multicast ARMS ou canais JSDT a funcionar como canais multicast. Todo o sistema é implementado em Java, incluindo os subsistemas ARMS e STF.

4. Testando o sistema

4.1. O que testar

Testar o sistema equivale na prática a testar como uma aplicação desenvolvida utilizando a API de mais alto nível – a STF – se

comportaria sob várias condições. No entanto, são tantas as combinações possíveis se considerarmos o variável número de streams, de categorias de estados, de tamanhos de estados e de chaves, das várias opções possíveis a nível de particionamento do mundo virtual, dos protocolos de junção tardia e de sincronização temporal, que é praticamente impossível testar exaustivamente a API em todos os aspectos.

Assim, foi decidido que os primeiros testes a ser efectuados incidiriam na questão de transmissão e recepção de estados para casos que podem ser considerados representativos de situações comuns em ambientes de realidade virtual distribuída.

O objectivo destes testes é avaliar o comportamento da STF sob várias condições em termos de ritmo de transmissão de mensagens, categorização dos estados e tamanhos desses estados. Foram avaliados os efeitos destas várias variáveis na latência obtida assim como o comportamento manifestado pela STF no aproveitamento da capacidade de transmissão fornecida pela camada ARMS.

Os testes efectuados foram dois. Em ambos estes testes foram manuseadas três streams diferentes, representativas de casos comuns:

- Stream 1: Estados redundantes, voláteis e independentes
- Stream 2: Estados redundantes, não voláteis e independentes
- Stream 3: Estados essenciais, não voláteis e independentes

No primeiro teste variou-se o ritmo de transmissão de mensagens e fixou-se o tamanho do estado e da chave em todas as streams. O tamanho do estado usado foi de 22 bytes e o tamanho da chave de 8 bytes, totalizando 28 bytes. Estes tamanhos são suficientes para transmitir informação sobre posicionamento e orientação tridimensionais e para a identificação de um gigantesco número de objectos no mundo virtual. O ritmo de transmissão de mensagens variou desde 5 mensagens por segundo até 400 mensagens por segundo em cada stream, totalizando 1200 mensagens por segundo no total das três streams. O intervalo de variação era de 5 mensagens por segundo em cada stream.

No segundo teste fixou-se o ritmo de transmissão e variou-se o tamanho da chave, desde 22 bytes para cima, aumentando sempre em passos de 100 bytes. Em ambos os testes foi convencionado que estes parariam caso se atingissem valores de latência na ordem de 1 segundo, o que seria inaceitável. Em ambas as sequências de teste, cada subteste individual executou-se durante 20 segundos. Todas as mensagens em todas as streams tinham a máxima prioridade.

4.2 A plataforma de testes

Os testes foram efectuados em uma rede local Ethernet de 100 Mbit full duplex em que três computadores com a plataforma instalada foram interconectados através de um switch. Todos os computadores possuíam CPU Pentium III 500 a 733Mhz com 128Mb de RAM e com o sistema operativo Windows 2000 Professional instalado. Todos os computadores foram sincronizados via NTP com um servidor externo de tempo [29], tendo sido pois desligada a sincronização temporal interna da STF.

4.3 A aplicação de testes

Tornou-se evidente a necessidade de desenvolver uma aplicação que permitisse realizar o maior número possível de testes ao sistema controlando o maior número possível de variáveis. Foi pois concebida e programada mais uma aplicação integrada no sistema desenvolvido.

A aplicação desenvolvida, a que foi dado o nome representativo de STF Intensive Testing Application – Aplicação de Teste Intensivo da STF, visa sobretudo testar a API STF, uma vez que esta é a principal API que as aplicações DCVE que futuramente serão desenvolvidas na plataforma middleware integrada irão utilizar.

A aplicação de testes possui uma interface intuitiva baseada, assim como todo o sistema de conferência CONCHA, na API de interface gráfica Java denominada Swing. Esta interface permite à pessoa que pretende executar um ou vários testes à plataforma modificar um número considerável de variáveis.

Quanto ao mecanismo de sincronização temporal, podemos controlar se a sincronização temporal interna se encontra activa e, caso esteja, o tempo em milisegundos entre mensagens de sincronização temporal, o tempo em milisegundos máximo que qualquer nó da conferência espera por uma mensagem de sincronização temporal, e o tempo máximo que o nó que inicia o processo de votação para um novo servidor principal de tempo espera por votos positivos ou negativos dos restantes nós da conferência.

Quanto á transmissão de estados, podemos controlar se temos ou não o lag de transmissão activado e, caso este esteja activado, qual o seu tempo em milisegundos.

No que diz respeito á recepção de estados, podemos controlar se o lag de recepção está activado ou não e, se este estiver activado, qual o seu tempo em milisegundos. Podemos também ligar e desligar a detecção de timewarps pela STF.

Por sua vez, quanto á junção tardia, podemos ligar e desligar o checkpointing distribuído, por oposição ao checkpointing local. Podemos também ligar e desligar o checkpointing automático periódico por parte da STF, assim como controlar o intervalo de tempo em milisegundos entre esses checkpoints periódicos. Podemos também controlar o intervalo de tempo, em milisegundos, em que o processo de junção tardia fica á espera de resposta por parte dos servidores de junção tardia – todos os restantes nós da conferência.

A Figura 4 mostra o quadro que diz respeito ás variáveis que controlam cada uma das streams que vão ser transmitidas pelo nó em que a aplicação está a correr. Neste quadro, o utilizador da aplicação de testes pode criar e apagar streams e controlar vários parâmetros para cada uma delas.

Entre os parâmetros que podem ser controlados incluem-se as características de categorização dos estados a transmitir, nomeadamente se estes são ou não voláteis, se são redundantes ou essenciais e se são independentes ou cumulativos no que diz respeito ao processo de junção tardia.

Podemos também controlar, para cada stream, o tamanho do estado propriamente dito – dos seus dados – e da chave identificativa da stream. A prioridade das mensagens de transmissão de estados também pode ser alterada para um dos níveis predefinidos de prioridade. Por fim, podemos controlar o ritmo de transmissão de mensagens de estado para cada uma das streams, expresso em mensagens por segundo.

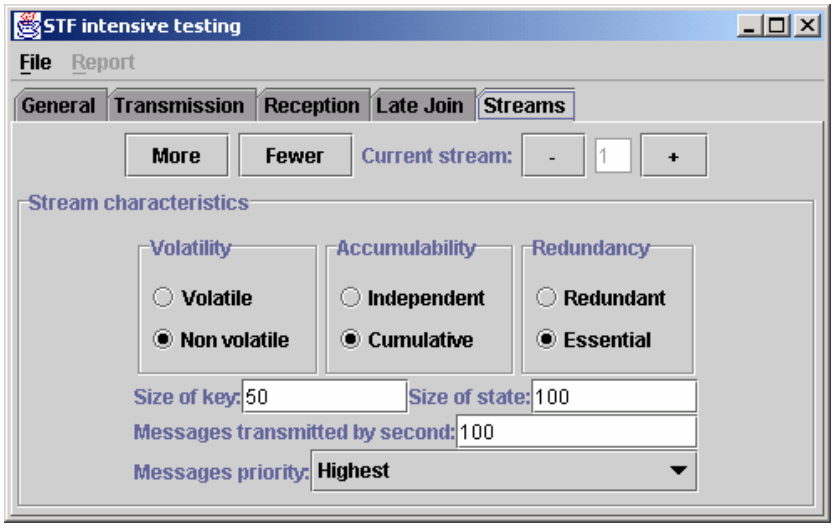

Figura 4 - Aplicação de teste: Quadro de streams

O menu de controle da aplicação de testes possui opções para iniciar e parar o gestor de canal STF, para executar a transmissão de testes durante um período de tempo expresso em milisegundos e também para executar testes automáticos.

A opção de executar testes automáticos foi criada especificamente para a sequência de testes de transmissão e recepção de estados previamente descrita, uma vez que seria praticamente impossível executar os testes necessários um a um, em tempo útil. Assim, se o utilizador da aplicação de testes seleccionar esta opção do menu de controle, a aplicação de testes entra em modo automático, executando a sequência de testes predeterminada sem qualquer intervenção humana.

É claro que a aplicação de testes, para além do controle e execução dos testes, necessita também da capacidade de criar relatórios sobre o resultado desses testes.

Caso o utilizador da aplicação seleccione a opção de teste automático, a aplicação exibe caixas de diálogo que permitem ao utilizador especificar um ficheiro onde toda a informação resultante dos testes é escrita, em formato CSV para fácil processamento por parte de aplicações de folha de cálculo.

Para testes individuais, não incluídos na sequência de testes automáticos, a aplicação de testes fornece um menu com várias opções de relatório e visualização. Aqui podemos escolher entre visualizar no écran os dados gerais, acumulados de todas as streams existentes, ou os dados individuais de cada uma das streams individuais. Podemos também escrever num ficheiro texto, em formato CSV, todos os dados que foram recolhidos para o teste efectuado.

4.4 Resultado dos testes e sua análise

Depois de termos utilizado a aplicação de testes descrita para efectuar os testes que predeterminámos, procedemos à análise dos resultados e á sua correlação. Para melhor entender os resultados que aqui apresentamos, a Figura 5 mostra os pontos de referência utilizados no processo de teste.

Temos pois quatro principais pontos que nos interessam no percurso de uma mensagem de estado desde que esta é transmitida pela aplicação no nó transmissor até que chega à aplicação no nó receptor.

O ponto A representa o momento em que a aplicação transmissora entrega a mensagem de estado á STF para processamento. Na Figura 8 estas mensagens são representadas como mensagens processadas. O ponto B representa o momento em que a camada STF no nó transmissor entrega uma mensagem para a camada ARMS manusear para transmissão. Na Figura 8 estas mensagens são representadas como mensagens transmitidas.

Depois da camada ARMS transmitir as mensagem de estado sobre a rede IP, estas chegam aos vários nós receptores. O ponto C representa o momento em que a camada ARMS em um dos nós receptores entrega uma mensagem de estado à camada STF para processamento. Verificou-se nos testes que estas mensagens são efectivamente as mesmas que as mensagens transmitidas, o que se compreende por os testes serem efectuados utilizando a tecnologia de multicast confiável ARMS sem que este se encontre em condições de stress, pelo que não existem mensagens perdidas na rede IP. Assim, na Figura 8 estas mensagens não são representadas. O ponto D representa o ponto em que a camada STF em um dos nós receptores entrega à aplicação uma mensagem de estado já processada. Na Figura 8 estas mensagens são representadas como mensagens recebidas.

Nos gráficos de latência, esta é definida como o intervalo de tempo, em milisegundos, entre a passagem da mensagem de estado pelos pontos A e D.

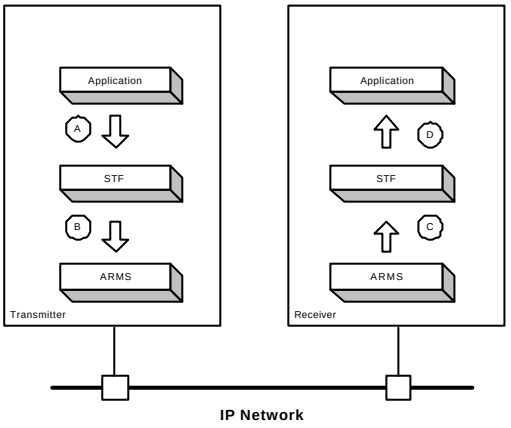

Figura 5 - Resultados dos testes: Pontos de referência

A Figura 6 mostra a latência como função do ritmo de transmissão de mensagens enquanto que a Figura 7 mostra a latência como função do tamanho do estado. A Figura 8 pretende ilustrar a relação existente entre o ritmo de mensagens processadas pela STF e o número de

mensagens efectivamente transmitidas para a rede IP e recebidas pela aplicação da STF no nó receptor.

Estas três figuras ilustram que a latência introduzida por todo o sistema, STF, CONCHA e ARMS, é perfeitamente aceitável encontrando-se sempre abaixo dos 30 milisegundos e normalmente mesmo abaixo dos 20 milisegundos. Ilustram também que a STF tem sucesso no seu objectivo de optimizar ao máximo a capacidade disponível na camada ARMS baseando-se na categorização dos estados de acordo com a sua importância. Note-se que, nos gráficos, os picos de latência existentes, sem lógica aparente, são muito provavelmente gerados pela existência de outros processos a correr nas máquinas de teste e também a variações nas condições da rede utilizada.

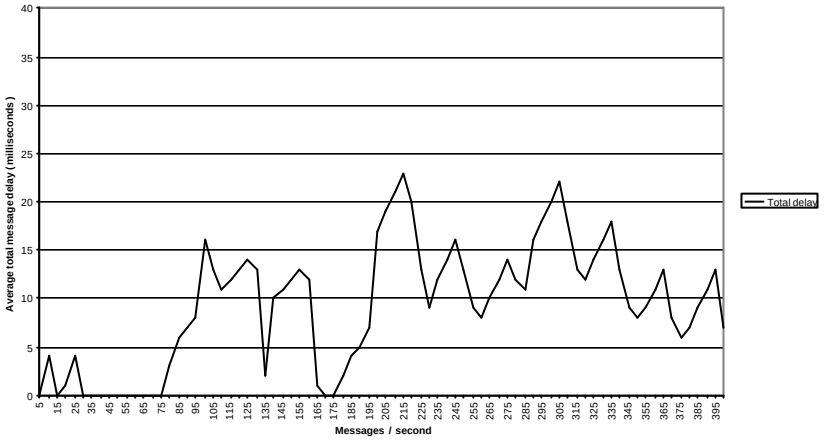

Figura 6 - Latência como função do ritmo de transmissão

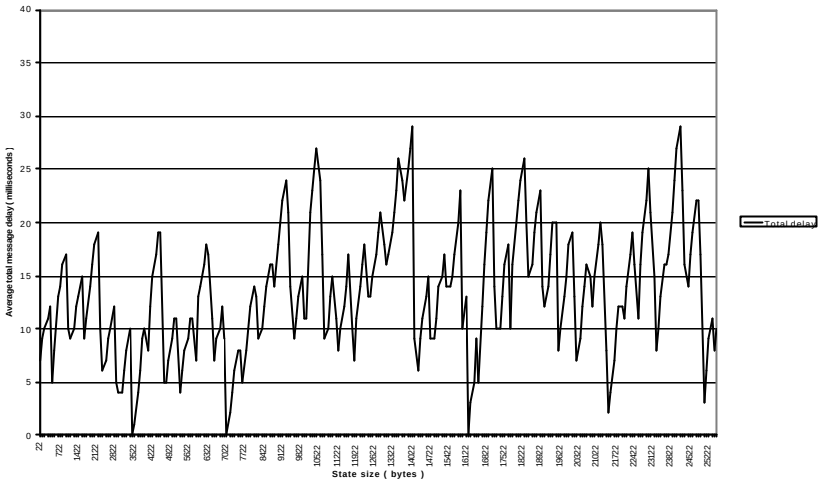

Figura 7 - Latência como função do tamanho do estado

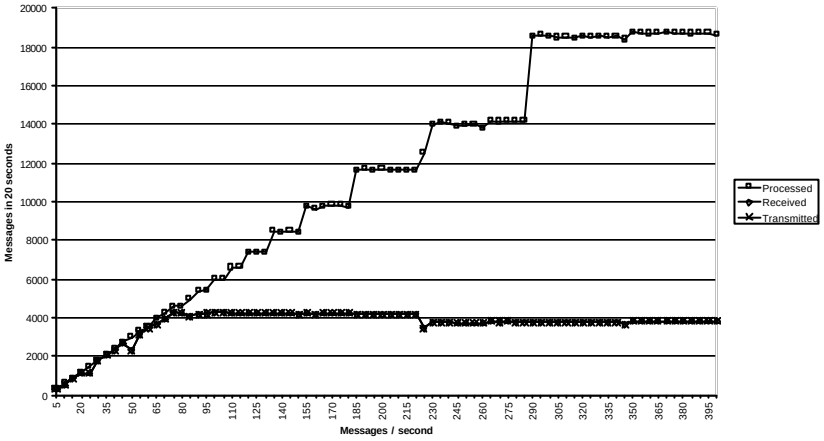

Figura 8 - Mensagens como função do ritmo de transmissão

5. Conclusões e trabalho futuro

As plataformas de middleware que pretendem suportar ambientes de realidade virtual distribuída colaborativos devem suportar uma série de mecanismos para processamento, transmissão e recepção de estados assim como para visualização tridimensional. Enquanto que no caso da representação tridimensional já existem bastantes soluções testadas, para processamento, transmissão e recepção de estados isso não acontece.

Neste artigo descrevemos um trabalho em curso para o desenvolvimento de uma plataforma de middleware para estes efeitos. Esta plataforma de middleware deve seguir religiosamente os requisitos normalmente aceites em termos de fiabilidade, delay e jitter para este tipo de aplicações, assim como fornecer uma série de mecanismos para transmissão, recepção, processamento e categorização dos estados transmitidos.

Depois de termos descrito toda a arquitectura da plataforma de middleware desenvolvida, desde a camada ARMS para multicast confiável via CORBA, a plataforma CONCHA para controle de conferências e culminando na API STF para manuseamento de estados em ambientes de realidade virtual distribuída, apresentámos os testes efectuados e seus resultados, que evidenciam o correcto funcionamento da plataforma e a sua adequação aos objectivos para os quais foi construída.

O trabalho futuro nesta plataforma incluirá concerteza novos testes, nomeadamente de escalabilidade, do processo de junção tardia e do efeito que os lags possuem sobre o delay e jitter. Novas optimizações serão possíveis através de uma maior integração entre a camada STF e a camada ARMS incluindo a utilização das novas capacidades de gerenciamento de qualidade de serviço que estão agora a ser desenvolvidas na plataforma ARMS e que incluem múltiplos níveis de fiabilidade. Por fim, poderemos incluir na plataforma suporte para outros tipos de conteúdos multimédia como sejam som, vídeo e vrml streaming.

Agradecimentos

Este trabalho foi financiado em parte pela Fundação Portuguesa para a Ciência e Tecnologia, FCT.

Referencias

[1] João Orvalho, Pedro Ferreira and Fernando Boavida, "State Transmission Mechanisms for a Collaborative Virtual Environment Middleware Platform", Springer-Verlag, Berlin Heidelberg New York, 2001, pp. 138-153, ISBN 3-540-42530-6 (Proceedings of the 8th International Workshop on Interactive Distributed Multimedia Systems – IDMS 2001, Lancaster, UK, September 2001)

[2] Jurgen Vogel, Martin Mauve, Werner Geyer, Vulker Hilt, Cristoph Kuhmunch, "A Generic Late-Join Service for Distributed Interactive Media"

[3] Lassaâd GANNOUN, Jacques LABETOULLE, "Shared Window System for Large Groups Based on Multicast", http://www.isoc.org/inet98/proceedings/1j/1j_2.htm

[4] Goopel Chung, Kevin Jeffay, Hussein Abdel-Wahab, "Accommodating Latecomers in Shared Window Systems", In IEEE Computer, Vol.26, No. 1 (January 1993) pp. 72-74

[5] Goopel Chung, Prasun Dewan, Sadagopan Rajaram, "Generic and Composable Latecomer Accomodation Service for Centralized Shared Systems"

[6] João Orvalho, Fernando Boavida, "Augmented Reliable Multicast CORBA Event Service (ARMS): a QoS-Adaptive Middleware", in *Lecture Notes in Computer Science, Vol. 1905: Hans Scholten, Marten J. van Sinderen (editors), Interactive Distributed Multimedia Systems and Telecommunication Services*, Springer-Verlag, Berlin Heidelberg, 2000, pp. 144-157. (Proceedings of IDMS 2000 – 7th International Workshop on Interactive Distributed Multimedia Systems and Telecommunication Services, CTIT / University of Twente, Enschede, The Netherlands, October 17-20, 2000).

[7] S. Singhal, M. Zyda. “Networked Virtual Environments Design and Implementation”, ACM press, New York, 1999.

[8] Martin Mauve. “How to Keep a Dead Man from Shooting”, in *Lecture Notes in Computer Science, Vol. 1905: Hans Scholten, Marten J. van Sinderen (editors), Interactive Distributed Multimedia Systems and Telecommunication Services*, Springer-Verlag, Berlin Heidelberg, 2000, pp. 144-157. (Proceedings of IDMS 2000 – 7th International Workshop on Interactive Distributed Multimedia Systems and Telecommunication Services, CTIT / University of Twente, Enschede, The Netherlands, October 17-20, 2000).

[9] Shervin Shirmohammadi and Nicolas D. Georganas. “An End-to-End Communication Architecture for Collaborative Virtual Environments”, Computer Networks Journal, Vol.35, No.2-3, Febr. 2001, pp.351-367.

[10] Martin Mauve, “Consistency in Continuous Distributed Interactive media”, Technical Report TR-9-99, Reihe Informatik, Department for Mathematics and Computer Science, University of Mannheim, November 1999.

[11] Martin Mauve, “Distributed Interactive Media”, Ph.D. Thesis, University of Mannheim, Germany, September 2000.

[12] L. Gautier and C. Diot, “Design and evaluation of MiMaze, a Multiplayer Game on the Internet”, IEEE Multimedia System Conference, Austin, June 28 - July 1, 1998.

[13] David L. Millis, “Network Time Protocol (version 3) specification, implementation”, Request For Comments 1305, IETF, March 1992.

[14] Schulzrinne, Casner, Frederic, Jacobson, “RTP: A transport Protocol for Real-Time Applications”, revision of RFC 1889, Internet-Draft (draft-ietf-avt-rtp-new-04.ps), June 25 1999.

[15] S. E. Deering, “Multicast Routing in Datagram Internetwork”, Ph.D. dissertation, Standford University, December 1991.

[16] Dimitrios Makrakis, Abdelhakim Hafid, Farid Nait-Abdesselem, Anastasios Kasiolas, Lijia Qin, “Quality of Service Management in Distributed Interactive Virtual Environment”, Progress Report of DIVE project. http://www.mcrlab.uottawa.ca/research/QoS_DIVE_Report.html

[17] M. M. Wloka, “Lag in Multiprocessor VR”, Presence:

Teleoperators and Virtual Environments (MIT Press), Vol 4, Nº 1, Spring 1995.

[18] K. S. Park And Robert V. Kenyon, "Effects of Network Characteristics on Human Performance Collaboration Virtual Environment", IEEE International Conference on Virtual Reality (VR '99), Houston, Texas, March 1999.

[19] Colin Perkins and Jon Crowcroft. "Notes on the use of RTP for shared workspace applications", ACM Computer Communication Review, Volume 30, Number 2, April 2000.

[20] J. Crowcroft, L. Vicisano, Z. Wang, A. Ghosh, M. Fuchs, C. Diot, and T. Turletti, "RMFP: A reliable multicast framing protocol", March 1998. Work in progress (Internet draft).

[21] B. DeCleene, S. Bhattacharaya, T. Friedman, M. Keaton, J. Kurose, D. Rubenstein, and D. Towsley, "Reliable multicast framework (RMF): A white paper", March 1997.

[22] S. Floyd, V. Jacobson, S. McCanne, C. -G. Liu, and L. Zhang, "A reliable multicast framework for light-weight sessions and applications level framing", *IEEE/ACM Transactions on Networking*, December 1997.

[23] M. Handley and J. Crowcroft, "Network text editor (NTE): A scalable shared text editor for the Mbone", In *Proceedings ACM SIGCOMM'97*, Cannes, France, September 1997.

[24] JavaSoft, www.javasoft.com

[25] OpenGL, www.opengl.org

[26] Microsoft Developer Library (MSDN), http://msdn.microsoft.com

[27] João Orvalho, Luís Figueiredo, Tiago Andrade, Fernando Boavida, "A platform for the study of reliable multicasting extensions to CORBA Event Service", in *Lecture Notes in Computer Science, Vol. 1718: Michel Diaz, Philippe Owezarski and Patrick Sénac (editors), Interactive Distributed Multimedia Systems and Telecommunication Services*, Springer-Verlag, Berlin Heidelberg, 1999, pp. 107-120. (Proceedings of the 6th International Workshop on Interactive Distributed Multimedia Systems and Telecommunication Services, IDMS'99, IEEE, LAAS-CNRS, ENSICA, Toulouse, France, October 12-15, 1999).

[28] Java Shared Data Toolkit (JSDT), SUN Microsystems, JavaSoft Division, http://java.sun.com/people/richb/jsdt.

[29] The NTPTime client for the network time protocol, http://home.att.net/~Tom.Horsley/ntptime.html.

[30] João Orvalho, Tiago Andrade, Luís Figueiredo, Fernando Boavida, “CONCHA – CONference system based on java and corba event service CHAnnels”, Proceedings of SPIES's symposium on Voice, Video, and Data Communications conference on Quality of Service Issues Related to Internet II, Boston, MA, USA, September 19-22, 1999.

[31] JDBC Data Access API, http://java.sun.com/products/jdbc/index.html, JavaSoft