Serviço *Multicast* na Plataforma JAMP para Suporte a Aplicações Multimídia Cooperativas

Sandro de Paula Mendonça[1]
e-mail: sandrop@dc.ufscar.br

Luis Carlos Trevelin
e-mail: trevelin@power.ufscar.br

Universidade Federal de São Carlos (UFSCar)
Department of Computing
Rod. Washington Luiz, Km 235
São Carlos (SP) – Brazil
13.5565-905 – Postal Box 676
Phone: +55 16 260 8233

RESUMO

Este artigo tem por objetivo apresentar o serviço *multicast* para suporte à aplicações distribuídas utilizando tecnologias de *Frameworks* orientados a objetos e Java/RMI(*Remote Method Invocation*). O serviço *multicast* é uma maneira otimizada de enviar dados de uma fonte para vários destinos fazendo apenas uma requisição ao serviço de transporte. O *multicast* na plataforma JAMP(*Java Architecture for Media Processing*) está sendo disponibilizado para suporte à comunicação em aplicações multimídia cooperativas. A plataforma JAMP foi desenvolvida pelo projeto MultiEng[2] e é usada para o desenvolvimento de aplicações multimídia e cooperativas em ambiente distribuído aberto.

Palavras-Chaves: Sistemas Distribuídos, *Frameworks*, Java/RMI, Plataforma JAMP e *Multicast*.

ABSTRACT

This article has as objective arms to present a multicast service for support to distributed applications using Frameworks and Java/RMI(Remote Method Invocation). The multicast service is a improved way a of sending data from a source to several destinies just making a requisition to the multicast transport service. The multicast in the platform JAMP(Java Architecture for Media Processing) is being made available for supporting communication of the applications cooperative multimedia. The JAMP platform was developed by the project MultiEng[1] and is being used for the development of multimedia and cooperative applications in a open distributed environment.

Word-keys: Distributed systems, Frameworks, Java/RMI, JAMP Platform and Multicast.

1 Introdução

Os sistemas distribuídos vem sendo estudados há algum tempo e possuem paradigmas, arquiteturas e também problemas como ambientes heterogêneos, gerenciamento de configuração, e monitoramento de rede. Hoje é possível notar uma coesão do paradigma de orientação a objetos que se apresenta como sendo uma provável solução para diversos pontos da área de sistemas distribuídos como, fornecer transparência tanto ao

[1] Financiado pela CAPES
[2]Multieng- Projeto patrocinado pela Finep/Recope-Computação de Alto Desempenho – Processo n°3609/96.

programador quanto ao usuário final de forma confiável e eficiente. Isso proporcionou uma melhor forma para integração no processamento das informações distribuídas. A união da orientação objetos com sistemas distribuídos originou a área de Objetos Distribuídos[OLI97].

A área de objetos distribuídos se tornou um ponto de interesse de estudo quando considera-se principalmente, a área em que vem sendo aplicada :Internet e Intranet. Com isto surgiu o desenvolvimento de várias novas aplicações em redes de computadores, baseadas na família de protocolos TCP/IP, e a necessidade de melhorias na comunicação para dar uma resposta efetiva às novas exigências impostas. A fim de prover estas melhorias, a comunidade TCP/IP está trabalhando em novos mecanismos para transferência de informações, de maneira que possa reduzir a ocupação da banda passante o máximo possível. Desta forma, estão sendo desenvolvidos novos conceitos de transferência de dados de um ponto a vários outros chamado de tráfego *multicast*.[DEE89]

Multicast é uma maneira mais otimizada de enviar dados de uma fonte para muitos destinos. Este padrão descreve modos de difundir eficazmente a informação em cima da largura de banda existente.

A comunicação *multicast* utiliza a classe D de endereçamento IP, onde o fluxo de informações gerado por uma determinada origem, atinge a um ou mais *hosts* associados a um grupo de destino, fazendo com que não transitem pacotes idênticos pelo mesmo enlace e replicando-os somente quando necessário.

Alguns dos ganhos e finalidades da comunicação *multicast* são: facilitar o trabalho do *host* que está enviando um pacote para vários destinatários, aliviar a sobrecarga dos *hosts* que não precisam receber pacotes, dar oportunidade a um *host* de se integrar a um ou mais grupos *multicast* e aliviar a sobrecarga dos enlaces que estão transferindo as informações. [DEE89].

A implementação da técnica *multicast* exige funções básicas que devem ser incorporadas às estações (computadores e roteadores) em uma rede TCP/IP. Elas são incorporadas através do *Internet Group Management Protocol* (IGMP) desenvolvido para fazer o gerenciamento dos grupos *multicast* em conjunto com os protocolos de roteamento *Distance Vector Multicast Routing Protocol* (DVMRP), *Multicast Extensions Open Shortest Path First* (MOSPF), *Protocol Independent Multicast* (PIM) e protocolos de transporte.

Vários trabalhos, ao longo dos últimos anos estão em andamento para desenvolver, avaliar e identificar as possíveis melhorias a serem implementadas nos protocolos de roteamento: *multicast*, principalmente após o sucesso das implementações em *Intranets* e na Internet através do *Backbone Multicast* (MBONE) [MEL96].

Um serviço de multicast pode ser definido como um conjunto de procedimentos e interfaces que permitem enviar mensagens a um grupo de participantes em um ambiente de processamento e comunicação. A arquitetura genérica de um serviço multicast pode ser dividida em duas partes: *o gerenciamento de grupo e a construção de uma infra-estrutura de distribuição.* Um grupo é definido como um subconjunto de usuários para qual é possível a transmissão de mensagens, estando associado a um endereço multicast. O gerenciamento de grupo diz respeito a todas as ações relacionadas a composição do grupo, como manipulação de informações sobre seus participantes e o controle sobre entrada e saída de participantes ao grupo. A construção da infra-estrutura de distribuição esta relacionada a forma de coordenação de recursos de processamento e comunicação para que a distribuição das mensagens possa ser efetuada. Em geral, a construção desta infra-estrutura é efetuada de forma a tentar minimizar as replicações de mensagens desnecessárias e aumentar o desempenho dos recursos. Protocolos de roteamento são, em geral, responsáveis por grande parte deste trabalho no que concerne o sistema de comunicação propriamente dito.

O trabalho mostrado neste artigo tem como objetivo apresentar um serviço *multicast* sobre o protocolo TCP/IP, o qual deverá ser disponibilizado como um servidor e como um *framework* para as aplicações clientes da plataforma JAMP, utilizando a tecnologia Java/RMI para comunicação das aplicações multimídia cooperativas.

A plataforma JAMP é um ambiente de programação distribuída que utiliza *frameworks* ("molde") implementados em Java/RMI, em que foi criada devido ao rápido avanço, e à pouca disponibilidade, até então, de ferramentas para o ambiente distribuído. Utilizando as tecnologias citadas, a plataforma JAMP proporciona uma série de vantagens como aumento de desempenho, capacidade de processamento, e tratamentos de erros remotos[FER97].

O serviço *multicast,* a plataforma JAMP e também as aplicações que são desenvolvidas usando *frameworks* da JAMP são multiplataforma, pois utilizam Java/RMI como linguagem de implementação. O RMI integra a tecnologia de sistemas distribuídos diretamente com a linguagem Java. A invocação remota de métodos de Java/RMI permite a comunicação de objetos, através da chamada de métodos em máquinas diferentes, em uma aplicação distribuída.

Os detalhes das tecnologias empregadas neste trabalho são assim apresentadas: Na seção 2 é apresentado uma visão geral sobre Java/RMI. A seção 3 é apresentado Frameworks e a plataforma JAMP. Na seção 4 apresenta um breve estudo sobre *multicasting* e os protocolos utilizados. Na seção 5 é apresentado o serviço de *multicasting* na plataforma JAMP e finalmente, na seção 6, são apresentadas as conclusões deste trabalho.

2 Java/RMI

O Java/RMI [DIS98] foi adotado como linguagem de desenvolvimento no projeto devido a sua integração a tecnologia de sistemas distribuídos diretamente com a linguagem Java. O RMI permite a comunicação de objetos, através da chamada de métodos, em máquinas diferentes em uma aplicação distribuída.

RMI é um conjunto de classes e interfaces em Java que encapsulam vários mecanismos de trocas de dados, que simplificam a execução de chamadas a métodos remotamente localizados em espaço de endereçamento diferentes. Este pode ser visto como uma arquitetura de três camadas: *Stubs* e *Skeletons*, *Remote Reference* e *Transport,* como mostra a Figura 1.

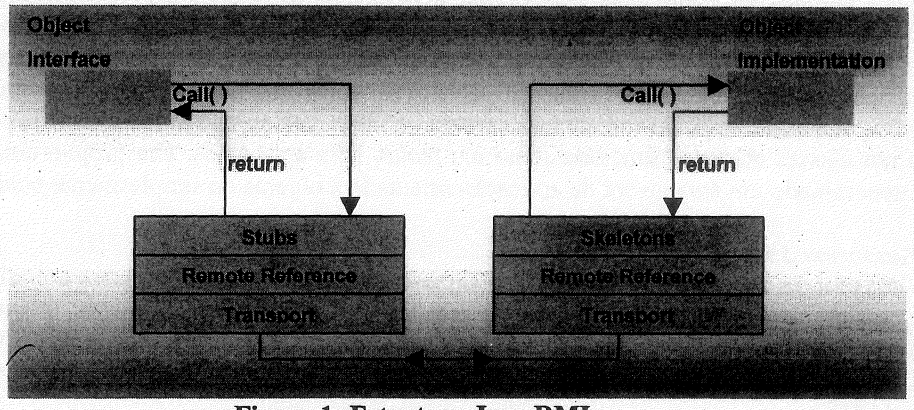

Figura 1- Estrutura Java RMI

A interface do cliente (*Object Interface*) deve conhecer a interface do objeto remoto para poder referenciá-lo, enquanto que o lado servidor (*Object Implementation*) possui a implementação do mesmo.

Quando um cliente chama (*call*) um método de um objeto remoto, os parâmetros do método são convertidos em uma mensagem pelo *Stub* e esta mensagem é enviada à máquina remota (*Remote Machine*) utilizando o protocolo de conexão (*connection protocol*) e uma porta anônima de comunicação (*Transport*). Quando a mensagem chega na máquina remota, o *Skeleton* transforma esta mensagem em argumentos e chama a implementação do método convertendo o resultado em uma mensagem que é redirecionada (*return*) para a máquina cliente (*Client Machine*), cujo *Stub* é responsável pela extração deste valor de retorno.

A camada *Remote Reference* providencia uma interface constante para os *Stubs* e *Skeletons*, escondendo as diferenças entre os diversos tipos de protocolos do servidor, como por exemplo :

- Servidores que podem suportar somente invocação ponto-a-ponto, onde um cliente interage com um único objeto;
- Servidores que podem suportar replicação, onde um número de objetos remotos replicados precisam se manter sincronizados sempre que uma instância é modificada;
- Objetos remotos que podem não estar continuamente ativos, podendo ser carregados de um meio magnético, por exemplo um disco, quando um cliente invocar um de seus métodos.

A camada *Transport* manipula os detalhes da conexão e providencia um canal de comunicação entre as JVMs (*Java Virtual Machines*) que executam o código cliente e o servidor.

Outra tecnologia integrada com Java/RMI no desenvolvimento deste trabalho são os *frameworks* e a plataforma JAMP, apresentados a seguir.

3 *Frameworks* e a Plataforma JAMP

A camada básica da arquitetura da plataforma JAMP é formada por *frameworks*, conforme apresenta a Figura 2. Esta camada é responsável pela invocação remota dos objetos e serialização dos objetos. Utiliza-se *frameworks* orientados a objetos na JAMP para fornecer os serviços e distribuir as aplicações, que herdam além dos serviços oferecidos pela plataforma todos os benefícios da utilização das técnicas de orientação a objetos.

3.1 Frameworks

Sistemas orientados a objetos são construídos por classes que colaboram entre si, através da troca de mensagens, para realizar as tarefas. Através do paradigma da Orientação a Objetos é possível reutilizar um sistema ou parte deste, somente redefinindo o comportamento de algumas subclasses. A partir da utilização de um sistema existente pode-se obter diferentes sistemas, reutilizando tanto o código como o projeto geral deste sistema. Analisando-se o comportamento esperado dos sistemas é possível detectar que existe uma parte comum que pode ser generalizada. Assim pode ser criada uma nova classe, superclasse das anteriores, que contenha toda a parte comum a ambas, enquanto as originais implementam a parte específica do comportamento abstrato.

Classes que representam conceitos genéricos relativos a uma família de objetos relacionados são chamadas de classes abstratas. Cada objeto representa um caso particular da abstração e é representado por uma subclasse concreta, que fornece uma variante específica do comportamento abstrato definido na classe abstrata. Desta maneira, as classes abstratas trabalham como um modelo para as suas subclasses. Assim sendo, um projeto constituído por classes abstratas funciona como um *molde* para aplicações. Um projeto constituído por classes abstratas é denominado um *framework* de aplicação orientado a objetos ou simplesmente *framework* [WIL98].

3.2 A Plataforma JAMP

JAMP oferece mecanismos que embute vários formatos de mídias, serviços e codificações dentro das aplicações, assim como seu processamento e recuperação. A plataforma JAMP possui uma arquitetura distribuída, que é formada por vários servidores, um *Broker* e um conjunto de *frameworks* usados pelas aplicações distribuídas. A Figura 2 mostra aplicações distribuídas e um conjunto de *frameworks* já disponíveis na plataforma JAMP.

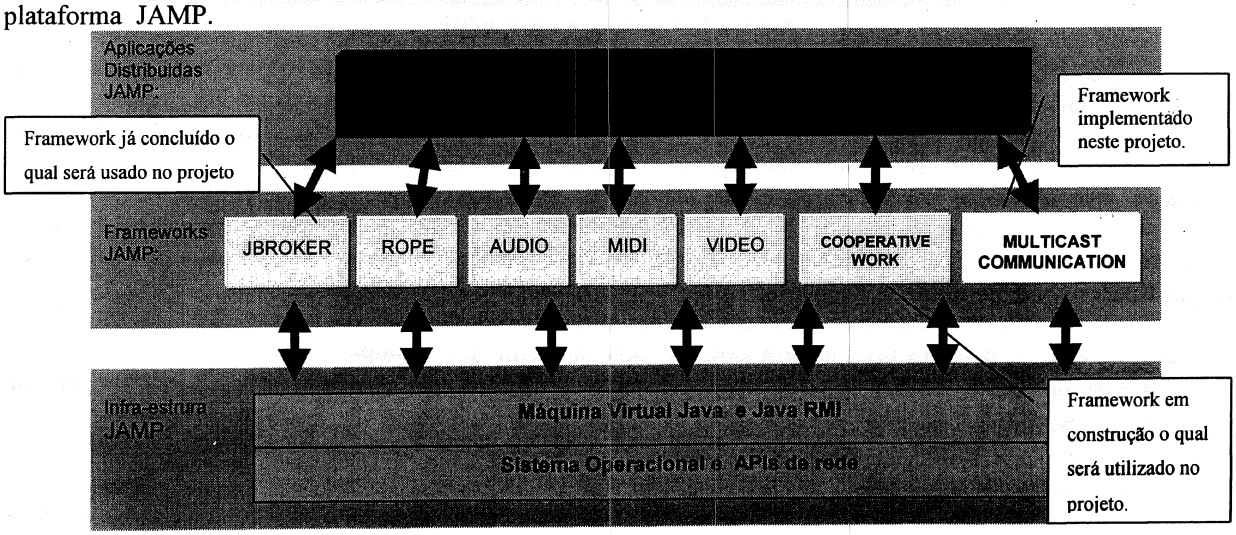

Figura 2 – Plataforma JAMP

Aplicações Distribuídas JAMP: As aplicações distribuídas JAMP podem usar todos os *frameworks* disponíveis na plataforma. Os serviços dos *frameworks* são embutidos nas aplicações através de sua compilação. As aplicações podem ser: *Aplicações Java*, que são executadas pelo interpretador Java; ou *Applets*, que são executadas pelos *Browsers* ou *Applet viewers.*

***Frameworks* JAMP:** Atualmente a camada *frameworks* dispõe dos seguintes *frameworks*: *Rope, Audio, MIDI, Cooperative Work, Multicast Comunication e o Video*. Estes *frameworks* dispõem de APIs [FER98] que são usadas pelas aplicações para incorporar as necessidades de processamento de mídias, trabalho cooperativo e permitir a comunicação distribuída. Nesta camada nota-se a flexibilidade que a plataforma JAMP oferece as aplicações para WEB, pois novos *frameworks* podem serem criados conforme a necessidade, ou seja, a plataforma não está restrita apenas aos *frameworks* citados. Com o surgimento de novos serviços para WEB, novos *frameworks* podem ser criados, estendendo a plataforma com estes novos serviços.

O *JBroker* é um *framework* que oferece importantes funcionalidades para a plataforma. Contém uma base de dados dos servidores disponíveis no momento e permite que os clientes encontrem os servidores distribuídos (objetos remotos) na rede. Um cliente JAMP pode invocar um método de um servidor depois da invocação com sucesso, por referência remota no *JBroker*. O processo de invocação, que inicia com o servidor sendo registrado no *JBroker* para uso direto da aplicação, é chamado de *Trading Process*, conforme apresenta a Figura 3.

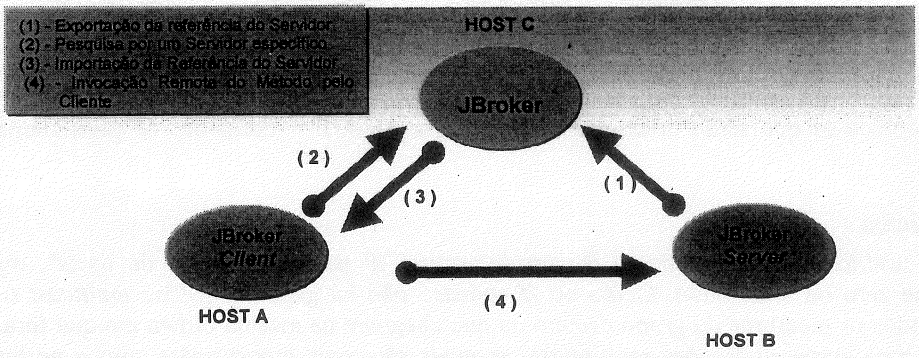

Figura 3 – Trading Process

Infra-estrutura da plataforma JAMP: Implementada em Java, a JAMP permite invocar métodos de objetos remotos (RMI) como se fossem objetos locais. Estes podem estar em diferentes arquiteturas de hardware e diferentes JVM (*Java Virtual Machine*). O TCP/IP é o responsável pela comunicação.

A Arquitetura Distribuída JAMP: é formada pelo *Java Broker*, vários servidores e por um conjunto de *frameworks* usados pela aplicações. A Figura 4 mostra a arquitetura com os servidores JAMP (*Host* B, C e D), o *Java Broker* (*Host* A), e os *frameworks* JAMP embutidos na aplicação multimídia JAMP (*Host* E). As setas *Server Registration* representam o registro dos serviços no *Java Broker* e, as *Client Invocation* mostram os clientes invocando dinamicamente os serviços referenciados pelo *Java Broker*. Após o sucesso da invocação, os clientes podem acessar diretamente por RMI. As setas RMI identificam todas as possíveis conexões RMI.

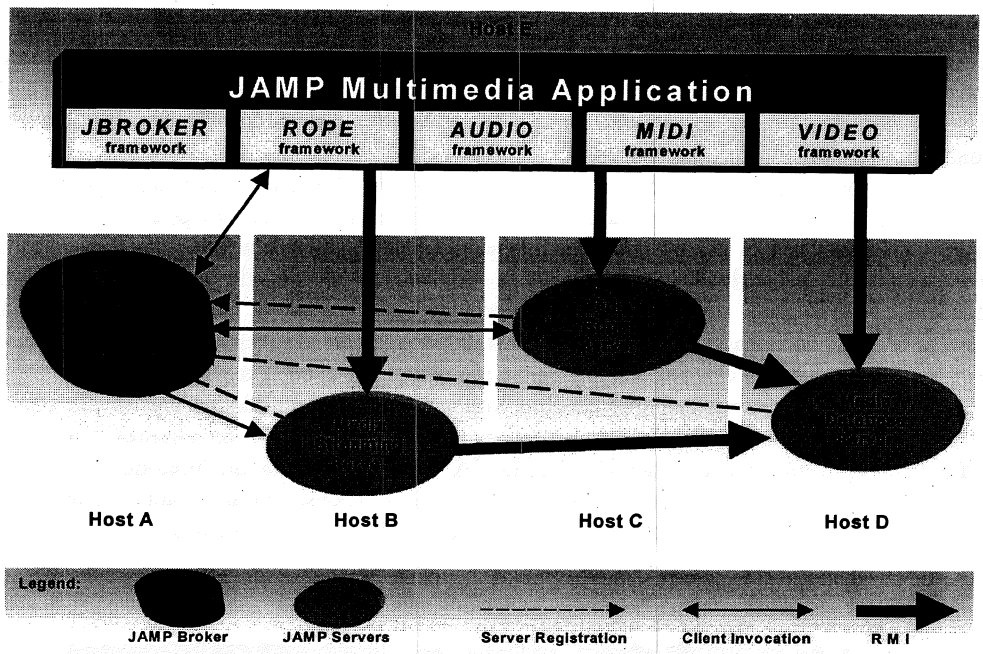

Figura 4 - Arquitetura Distribuída JAMP

A seguir será mostrado um breve estudo sobre a comunicação multicast, seu conceito e características e sobre os protocolos que serão utilizados neste trabalho (IGMP – Protocolo de gerenciamento de grupos e o TCP/IP).

4 Multicast

IP m*ulticast* é a transmissão de um datagrama IP para um "grupo de *hosts*", representado por um conjunto de zero ou mais *hosts*. Como no *IP-unicast,* não há garantia que no *multicast* o datagrama chegue intacto a todos os membros do grupo destino ou que cheguem na mesma ordem em que foram enviados. Porém isto é útil para certas aplicações multimídia, as quais são mais preocupadas com a performance que com a própria confiabilidade.

A associação a um grupo é dinâmica; *hosts* podem participar ou abandoná-los a qualquer momento. Não há restrição quanto ao posicionamento geográfico ou número de membros em um grupo de *hosts*. Um *host* pode ser membro de um ou mais grupos ao mesmo tempo. Um *host* não precisa ser membro de um grupo para enviar datagramas a ele. Um grupo de *hosts* pode ser permanente ou transiente. Um grupo permanente tem um IP bem conhecido. É o endereço e não a associação que é permanente; a qualquer momento um grupo permanente pode ter um número qualquer de membros, até mesmo zero.

O roteamento de datagramas IP *Multicast* na Internet é manipulado por roteadores *Multicast* que podem coexistir ou não com os *gateways* Internet. Quando o datagrama é transmitido se este possui um *time-to-live* maior que 1, o roteador *Multicast* associado a esta rede tem a responsabilidade de repassar este datagrama a todas as outras redes que possuam membros deste grupo destino. Nas redes alcançáveis, o roteador *Multicast* a elas associado completa a entrega transmitindo o datagrama como um *Multicast* local aos membros do grupo.

4.1 Protocolos Utilizados

A implementação da técnica *multicast* exige funções básicas que devem ser incorporadas às estações (computadores e roteadores) em uma rede TCP/IP. Elas são incorporadas através do *Internet Group Management Protocol* (IGMP) o protocolo de gerenciamento de grupos. No desenvolvimento do serviço *multicast* para plataforma JAMP foi utilizado o protocolo TCP(*Transmission Control Protocol*) como protocolo

de Transporte, o modelo de endereçamento IP(*Internet Protocol*) os quais serão descritos resumidamente a seguir.

4.1.1 TCP/IP

A arquitetura Internet TCP/IP, provê basicamente dois conjuntos de serviços: no nível inferior, um serviço de rede não orientado à conexão, fornecido pelo (IP); no nível superior, um serviço de transporte orientado à conexão, fornecido pelo *Transmission Control Protocol* (TCP), ou um serviço sem conexão que utiliza o protocolo IP para transportar as mensagens, fornecido pelo *User Datagram Protocol* (UDP)[COM91, TAN96].

Na Internet cada computador é identificado por um endereço inteiro de 32 bits, denominado endereço Internet ou IP. Cada endereço IP é formado por um par de campos: o identificador da sub-rede e o identificador do *host*. O formato dos endereços IP são divididos em cinco classes: A, B, C, D sendo as classes A, B, C são *unicast* e a classe D é *multicast*.

Grupos de hosts são identificados pelos endereços IP classe D (possuem "1110" associados aos quatro bits de mais alta ordem. Os endereços de grupo variam de 224.0.0.0 a 239.255.255.255, o endereço 224.0.0.0 não pode ser atrubuído a nenhum grupo, e 224.0.0.1 á atribuído ao grupo permanente de todos os hosts IP (incluindo gateways) usado para endereçar todos os hosts multicast na rede diretamente conectada

4.1.2 IGMP(Internet Group Management Protocol)

O protocolo de gerenciamento de grupo (IGMP) é usado por *hosts* para reportar seus participantes de grupos de *hosts* a roteadores multicast vizinhos. Como o ICMP(*Internet Control Message Protocol)*, IGMP é uma parte integral do IP. É um requisito básico de implementações a todos os *hosts* que desejem enviar e receber pacotes multicast. As mensagens IGMP são encapsuladas em datagramas IP, com um número de protocolo IP igual a 2

Roteadores *Multicast* enviam mensagens *Host Membership Query* para decobrir que grupos de hosts têm membros nas suas rede locais associadas. As requisições são endereçadas a todos os grupos de host (endereço 224.0.0.1) e carregam um *time-to-live* igual a 1.

Os *hosts*, por sua vez, respondem à consulta gerando *Host MemberShip Reports*, reportando cada grupo de *host* para o qual ele pertence na interface de rede a partir da qual a consulta foi recebida

A seguir é apresentado sockets os quais são utilizados na implementação do serviço de *multicasting* na plataforma JAMP.

5 Sockets

A camada de transporte tem um papel fundamental na hierarquia de protocolos. Seu principal objetivo é fornecer um serviço eficiente, confiável e efetivo a seus usuários (as aplicações).

As primitivas de serviço desta camada permitem seus usuários acessarem o serviço de transporte (prestado pela camada de rede). Cada serviço de transporte tem suas próprias primitivas de acesso, podendo variar bastante de rede para rede. Com as primitivas da camada de transporte é possível ter programas funcionando em diferentes redes, lidando com diversas interfaces da rede.

Primitiva	Significado
SOCKET	Cria um novo ponto final de comunicação.
BIND	Atribui um endereço local a um socket.
LISTEN	Abre uma porta para conexões, especificando um tamanho para a fila.
ACCEPT	Bloqueia o servidor até que uma conexão seja solicitada.
CONNECT	Tenta estabelecer uma conexão.
SEND	Envia dados pela conxão.
RECEIVE	Recebe dados da conexão.
CLOSE	Encerra a conexão.

Tabela 1 – Primitivas Sockets

As primeiras primitivas mostradas na tabela 1 são executadas pelo servidor. Uma chamada bem sucedida à primitiva SOCKET retorna um descritor de arquivo, que mais tarde será utilizado em outras chamadas e para a transmissão de dados, aqui alguns parâmetros especificam o tipo de serviço desejado e o protocolo (TCP/UDP). O socket criado ainda não possui um endereço, para atribuir um endereço, utilizamos a primitiva BIND. Uma chamada à primitiva LISTEN cria uma fila para o caso de vários clientes solicitarem conexões simultaneamente. Para esperar por conexões, o servidor executa a primitiva ACCEPT.

Um cliente precisa criar um socket usando a primitiva SOCKET, mas não precisa executar um BIND. Depois que o socket é criado, o cliente precisa executar um CONNECT para estabelecer a conexão com o servidor. Se o servidor aceitar a solicitação do cliente, uma conexão é estabelecida e ambos podem executar as primitivas SEND e RECEIVE para trocar informações. Quando a conexão não for mais necessária, ambos os lados precisam executar a primitiva CLOSE para liberá-la.

Para trabalharmos com sockets em Java, utilizamos as classes implementadas no pacote java.net. Veremos como utilizar as classes Socket e ServerSocket para criarmos clientes e servidores.

Para transmitir e receber dados, utilizaremos duas classes do pacote java.io: InputStream e OutputStream, que são utilizadas neste contexto da mesma forma que quando são utilizadas para ler e/ou escrever dados em arquivos

6 O Serviço *Multicast* na Plataforma JAMP

A plataforma JAMP tem como objetivo tratar as aplicações distribuídas multimídia cooperativas, em que encontra-se diversos níveis de necessidades e qualidades de informações as quais são entregues a diferentes grupos de usuários ligados a equipamentos com capacidade heterogêneas. De acordo com as exigências encontradas nas aplicações alvo da plataforma JAMP, o tratamento da comunicação em grupo está sendo feito por difusão seletiva ou *multicasting* Utilizando o *multicasting*, apenas uma cópia de cada pacote é enviada por enlance, sendo copiada apenas quando houver um braço na árvore lógica dos destinos (figura 5). A utilização do *multicasting* fornece um ganho de processamento de CPU e largura de banda quando vários hots estão envolvidos simultaneamente.

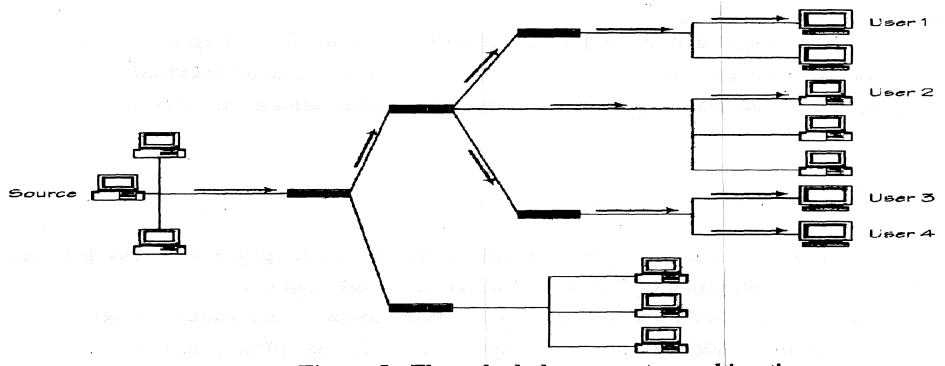

Figura 5 - Fluxo de dados e pacotes *multicasting*

Será mostrado nesta seção o serviço *multicast* criado para plataforma JAMP, tendo em vista a disponibilização da comunicação *multicast,* para utilização em aplicações distribuídas que utilizam a plataforma JAMP, particularmente as de trabalho cooperativo, objetivo do projeto MultiEng.

A implementação do serviço *multicast* na JAMP será feito por sockets Java, porém manipulados remotamente por Java/RMI, devido ainda não existir o suporte multicast para RMI.

O serviço *multicast* na JAMP é disponibilizado em duas formas: como um servidor de comunicação, qual trata todo o gerenciamento de grupos entre as aplicações que utilizem a JAMP, e um *framework,* o qual tem o propósito de facilitar o desenvolvimento das aplicações clientes, e para utilização do servidor disponibilizando assim uma forma fácil e eficiente do uso da comunicação *multicast* às aplicações desenvolvidas na plataforma.

Além dos tratamentos gerais do *multicast*, esta sendo implementado um protocolo *multicast* que ofereça as outras garantias como a de entrega, exigidas pelas aplicações, pois existem aplicações as quais requerem que a comunicação seja confiável e outras já exigem uma comunicação não-confiável as quais são mais preocupadas com a performance que com a própria confiabilidade. Assim serão implementados no protocolo o uso de reconhecimento de recepção de pacotes do tipo positivo e negativo respectivamente, para tratamento de aplicações que exigem comunicação confiável.

Existem características inerentes à criação de grupos *multicast* através de uma aplicação JAMP do tipo servidor, como por exemplo solicitar uma junção de vários receptores a seu grupo, podendo estes receptores do grupo estarem distanciados fisicamente. Problemas de rede como quebra e remontagem de pacotes, tamanho de *buffers* a serem utilizados e ordenação de pacotes recebidos também serão levados em consideração na modelagem e implementação do presente sistema.

6.1 Framework Multicast

O *framework multicast* faz parte da camada básica da plataforma JAMP que oferece além deste vários outros *frameworks* para suporte a aplicações distribuídas multimídia cooperativa (veja Figura 3).

Partido do framework *multicast*, é possível implementar serviços de multicast específicos de forma organizada e rápida através do reuso de toda estrutura genérica, aliada a configuração de determinados componentes de serviço. Esses pontos de configuração do framework são denominados *pontos de flexibilização* ou *hot spots*. Os pontos de flexibilização do framework para gerenciamento de grupo dizem respeito a escala de distribuição das informações do grupo, alem da estrutura do esquema de endereçamento utilizado pelo sistema de comunicação especifico.

Utilizando *framework* herda-se os benefícios do paradigma de orientação a objetos . Sua utilização na comunicação distribuída da plataforma JAMP aumenta a dimensão da aplicação e facilita a implementação do serviço multicast nas aplicações da plataforma, sendo que este funciona como um molde facilmente adaptável as exigências das mesmas. O *framework multicast* tem o intuito de facilitar nas aplicações JAMP a implementação da comunicação, a sua manutenção, e de torná-las mais consistente. A funcionalidade do *framework multicast* esta presa nas aplicações pela compilação, a qual permite o uso do serviço de comunicação *multicast* oferecido remotamente pelo servidor.

Estão sendo feitas para o framework *multicast* as devidas documentações, com informações sobre o projeto, restrições e regras a serem seguidas pelos desenvolvedores. Em um estudo inicial esta sendo utilizado o método de Pree[PRE94]. Esta abordagem, consiste na identificação dos pontos fixos e dos pontos adaptáveis do domínio de uma aplicação. Os pontos fixos correspondem as partes comuns das aplicações correlatas, enquanto que os pontos adaptáveis são as partes que podem ser estendidas para cada aplicação específica, dando ao *framework* a capacidade de ser flexível e se moldar a diferentes aplicações. Na figura 6 é mostrado o algumas funções do framework que esta sendo desenvolvido, e os métodos disponibilizados.

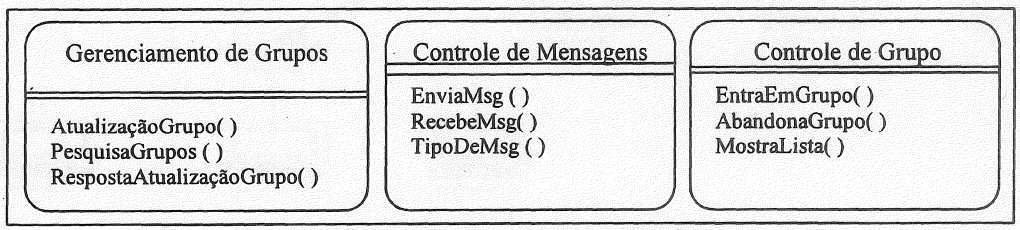

Figura 6 – Framework Multicast

6.2 *Servidor Multicast*

O Serviço é disponibilizado através de uma requisição do JBroker ao serviço *multicast,* o qual, utilizando o processo de *trading process* estabelecerá uma conexão direta com servidor. As aplicações JAMP utilizam métodos do *framework multicast* como suporte à implementação da comunicação e manuseio do servidor.

Foram analisadas inicialmente três propostas de projeto para o servidor multicast. Os projetos propostos foram criados de acordo com o serviço, o tráfego de rede, uso do RMI de acordo com sua vantagens e desvantagens e número médio de máquinas. Através de testes de desempenho e exigências da plataforma JAMP e suas aplicações, foi selecionada uma para ser implementada a seguinte proposta:

Sempre quando o servidor multicast for ativado é enviado ao Jbroker um registro com sua localização, disponibilizando o serviço aos usuários da plataforma. O processo de conexão e disponibilização do servidor *multicast* é feito através do processo de *Trading Process (seção 3.2).*

O servidor é responsável pelo gerenciamento da comunicação, distribuição dos pacotes, e formação dos grupos. As aplicações clientes e os servidores destas aplicações manipulam a comunicação em grupo através de métodos diretamente ligados via RMI ao servidor multicast. Assim o servidor multicast é responsável por toda comunicação de grupos, recebendo e enviando as mensagens por difusão seletiva as máquinas interessadas, que participam do grupo. As vantagens desta arquitetura é que o servidor multicast tem uma maior flexibilidade e maior independência dos servidores de aplicação e tem um controle total do gerenciamento da comunicação. Este tipo de estrutura também beneficia a implementação das aplicações pois a preocupação com a comunicação por parte das aplicações fica sendo mínima.

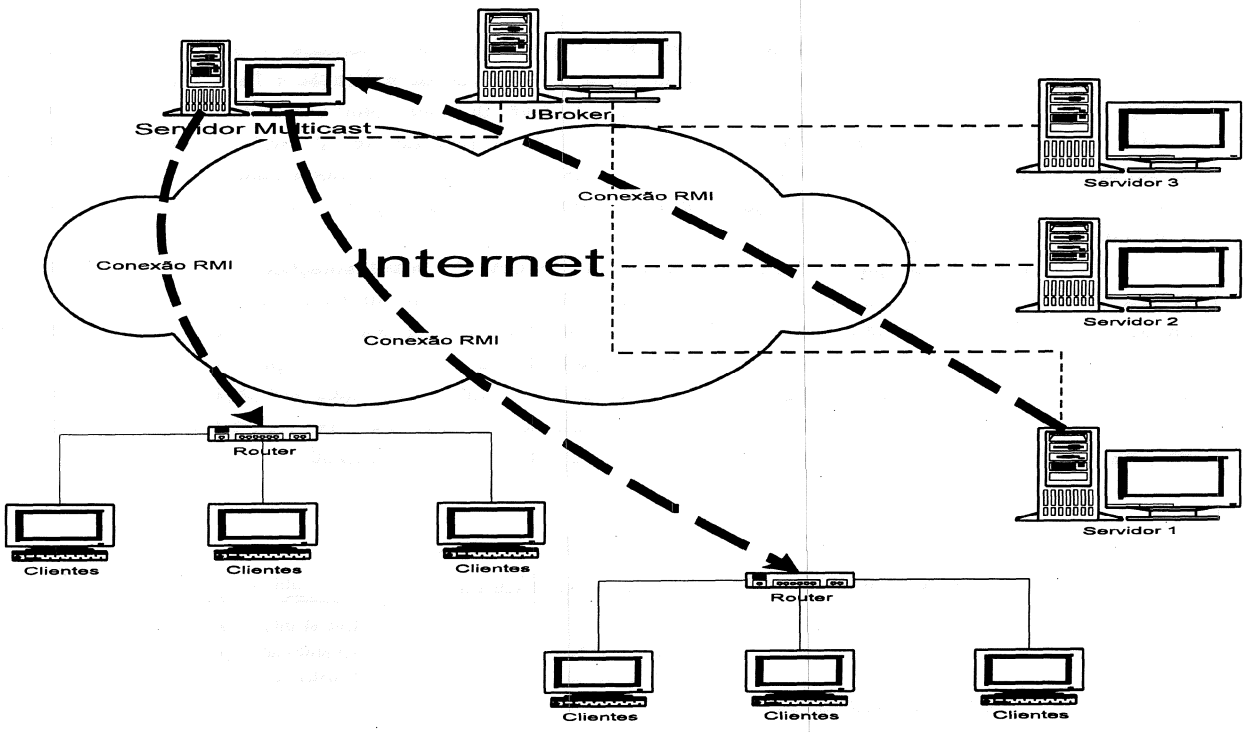

Figura 7 - Servidor *Multicast*

6.3 *As Aplicações que utiliza o* Multicasting *JAMP*

As aplicações desenvolvidas pela plataforma JAMP, tem a sua disposição os *frameworks* e servidores *multicast*. A preocupação de implementação quanto a comunicação *multicast* é mínima, sendo que apenas faz reuso do *framework*, o qual é responsável pelas devidas negociações com o servidor.

Figura 8 – Exemplo do Uso de Métodos em Interface *Multicast*

O desenvolvedor das aplicações JAMP pode criar interface para controle do gerenciamento de grupos através de métodos disponibilizados no *framework*. Os grupos são separados de acordo com o tipo de aplicação, ou seja um usuário de uma aplicação não terá acesso nem visibilidade de outras aplicações diferentes. O usuário que cria o grupo define o nome do grupo e o numero de máximo de participantes, torna-se o gerente daquele grupo. E o usuário cliente quando deseja entrar em algum grupo, este solicita lista dos grupos existentes, seleciona o grupo, e solicita entrada. Quando o usuário se junta a um projeto, o servidor da aplicação (do projeto) informa ao servidor de multicast sobre este novo usuário, e a que grupo deve ser incluído. O nome e endereço do host da aplicação é detectado pelo gerenciador de *multicast,* como também toda negociação de informações com o servidor *multicast.*

Para maior entendimento do funcionamento das aplicações clientes é apresentado na figura 9 o envio de um pacote multicast para difusão

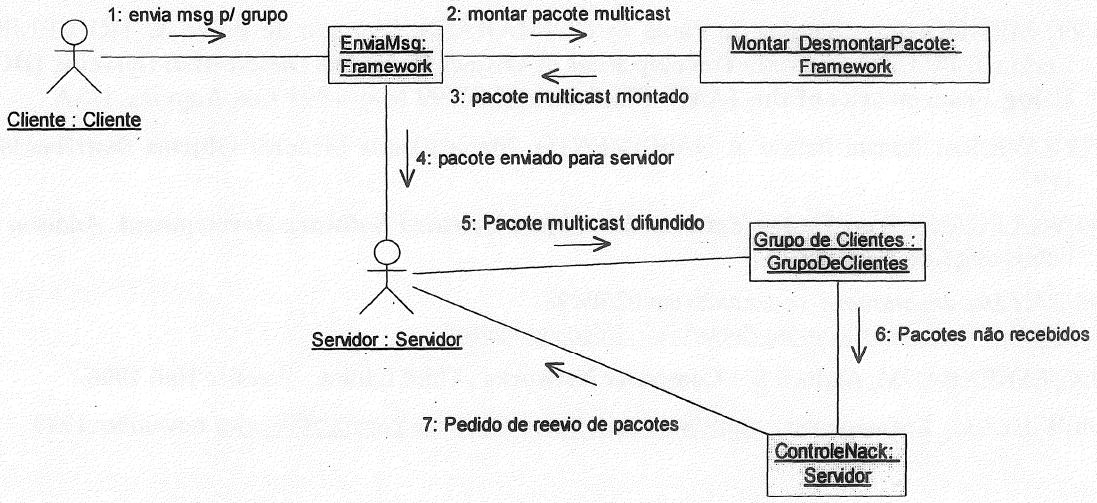

Figura 9 – Envio de difusão de pacote

7 Conclusão

Neste presente trabalho apresentou-se um suporte a comunicação *multicast* para plataforma distribuída JAMP , o qual está sendo implementado em Java/RMI. O trabalho é mostrado em duas partes: em um servidor, que faz a disponibilização do serviço à plataforma através do JBroker, e um *framework* o qual facilita o desenvolvimento, e a padronização das aplicações distribuídas desenvolvidas.

O serviço multicast proposto aqui, é somado com vários outros suportes a aplicações distribuídas oferecidos pela plataforma, aumentando sua utilidade e força principalmente a comunicação de grupos. O multicast JAMP implementado em Java/RMI oferecerá diversos outros benefícios, principalmente no uso de comunicação remota multicast em Intranet e Internet(Web design).

8 Bibliografia

[COM91]COMER, D. E. **Internetworking with TCP/IP** - Englewood Cliffs: Prentice Hall - Vol. 1 - 1991.

[DEE89]DEERING, S. **Host Extensions for IP Multicast** – RFC 1112 - ago. 1989.

[DIS98]Discover Tecnology - **Java RMI** – http://discover.com.br/Discover/javarmi.htm consultado em 05/10/98.

[FER97]FERREIRA,M. M. & TREVELIN, L. C. **A Platform for Developing Distributed Multimedia Application**.Project Report, DC. UFSCar1997.

[FER98]FERREIRA,M. M. & TREVELIN, L. C. **The Java Broker System: Concepts &Java Programming Guide.** Project Report. DC. UFSCar 1998.

[MEL96]MELCHIORS, Cristina **Tutorial sobre Mbone**, UFRGS, dezembro 1996 http://penta.ufrgs.br/redes296/mbone/capa.htm. consultado em 01/02/99

[MEN99] MENDONÇA, Sandro de Paula & GUIMARÃES, Marcelo de Paiva & TREVELIN, Luis Carlos & PRADO, Antonio **Development of Object Oriented Distributed Systems (DOODS) Using Frameworks of the JAMP Platform,** ICSE'99 May 1999 Los Angeles, USA

[OLI97]OLIVEIRA, Renata Peluso & NASCIMENTO, Maria Elenita Menezes **Objetos Distribuídos,** UnB 1997.

[PRE94]WOLFGANG Pree. **Design Patterns for Object-Oriented Software Development**. Addison Wesley Publishing Company, 1994.

[RMI98]***RMI Documentation***; consultado em 02/09/98 http://www.javasoft.com/products/jdk/1.1/docs/guide/rmi.

[TAN96]TANENBAUM, Andrew S. - **Computer Networks** , Third Edition, Prentice Hall 1996.

[WIL98]Wills,A.C., **Frameworks.** http://www.trireme.com/trireme/papers/50fworks/ november,1998.