Pesquisadoras	Orientador
Priscila Rebequi	**Emerson Rogério de Oliveira Junior**
Giliane Redolfi	
Lúcia Pasquato	
Universidade de Passo Fundo UPF Ciência da Computação - ICEG	Doutorando da Universidade Federal do Rio Grande do Sul - UFRGS e Professor da Universidade de Passo Fundo – UPF
{20238, 10051, 26524}@lci.upf.tche.br	emerson@upf.tche.br

COMUNICAÇÃO ENTRE OBJETOS DISTRIBUÍDOS TOLERANTES A FALHAS

RESUMO

As aplicações distribuídas estão potencialmente sujeitas a falhas. A orientação a grupos é uma boa ferramenta para garantir a continuidade da aplicação, mesmo na presença de falhas. Este artigo apresenta a implementação de um protocolo *multicast* confiável, para comunicação de objetos tolerantes a falhas. Utilizando o Java, uma linguagem de programação orientada a objetos, criou-se classes que controlam essa comunicação. A aplicação distribui objetos pela rede, sendo criados grupos de objetos que garantirão a confiabilidade da aplicação, ou seja, que haja a difusão da mensagem (*multicast*) para todos os membros do grupo.

Palavras-Chave: grupo de objetos, sistemas distribuídos, comunicação de grupo, *multicast confiável, confiabilidade.*

Abstract

Distributed applications may be in fail estate, because a fault may occurs. The group-orientation technique is a good way to guarantee that tha application can complete his commands, even in fault presence. This paper presents tha implementation of a reliable multicast protocol, to make communication between distributed objects through objects persistence. This implementation was programmed in Java, taht is a programming language object oriented. The application have objects distributed across network, controled by a group. In this group, messages are multicasts by all members.

Key-Words: object group, distributed systems, group communication.

1- Introdução

A proposta deste projeto é a implementação de classes que constituirão uma biblioteca, usando a idéia de um protocolo de comunicação *multicast* confiável permitindo que uma aplicação, executada em um ambiente distribuído, apresente alto grau de confiabilidade e segurança e, consequentemente, maior qualidade na comunicação entre objetos. A aplicação também visa maior eficiência e eficácia no trato, tanto dos objetos distribuídos pela rede, quanto nos objetos em falha. Por esta razão, utiliza-se técnicas de tolerância à falhas, além de técnicas de orientação a objetos.

Referencial Teórico

Os objetos podem ser definidos como sendo blocos de estruturas que contêm dados privados e um conjunto de operações que faz acesso a esses dados. Todo o comportamento de um determinado objeto é controlado através de suas rotinas internas e cada objeto tem suas próprias características, devido ao tipo de operação que virá a executar. A utilização de um objeto é muito simples. Solicita-se ao mesmo que efetue uma de suas operações internas, enviando-lhe uma mensagem, a qual o instrui sobre o que deve fazer. Ao receber esta mensagem, o objeto analisa o melhor método para executar esta operação.

Atualmente, os sistemas distribuídos têm sido amplamente utilizados nos sistemas de computação em geral. Este fato deve-se, principalmente, à grande utilização de redes de computadores. Um sistema distribuído pode ser considerado como um conjunto de processadores autônomos, sem compartilhamento de memória, cooperando entre si através de uma rede de comunicação. Em uma aplicação distribuída orientada a objetos, múltiplos objetos estão cooperando para executar uma tarefa.

Com o aumento do uso dos computadores, técnicas de tolerância à falhas têm se tornado uma obrigatoriedade nos atuais sistemas de computação, principalmente no que se refere a segurança, confiabilidade e disponibilidade destes sistemas. Nas aplicações distribuídas, há uma grande quantidade de recursos que precisam ser gerenciados, os quais estão potencialmente sujeitos a falhas. Para enfrentar estes problemas, estas aplicações utilizam técnicas de tolerância a falhas, visando detectar a ocorrência de erros que venham ocorrer no software ou hardware, sendo produzidos por falhas e recuperá-los, para continuar sua operação e retornar à computação normal.

A utilização do paradigma de orientação a grupos é uma boa ferramenta para garantir a continuidade da aplicação, mesmo na presença de falhas. A utilização de um grupo surge, inicialmente, com seu enfoque direcionado a processos. Segundo Birman, quando há a necessidade de realizar um serviço de forma cooperativa entre processos, pode-se reuni-los em um grupo [BIR87]. Por sua vez, Felber [FEL98] defende a idéia da utilização de objetos como sendo elementos que compõem um grupo. Segundo este contexto, a aplicação que utiliza comunicação de grupo permite a reunião de um conjunto de objetos em um grupo lógico, devendo existir primitivas para que seja possível o envio de mensagens a todos os membros do grupo, ao mesmo tempo e com diferentes garantias de ordenação [FEL98]. Neste sentido, em um sistema baseado em objetos, se existe a necessidade de ser realizada alguma ação em conjunto entre objetos, o conceito de grupo poderá ser utilizado. Algumas aplicações que justificam a utilização de grupos são: **replicação**, **migração de servidores** e aplicações **CSCW** [MOR99].

A **replicação** permite que um serviço ou um componente possua cópias redundantes espalhadas entre os nodos de uma rede, garantindo assim um aumento da disponibilidade destes elementos replicados. Neste sentido, os elementos replicados podem utilizar os serviços disponíveis pelo grupo, tais como a substituição de membros falhos e transferência de estado.

A **migração de servidores** pode ser necessária quando a aplicação está sendo executada e, por questões de falha ou de balanceamento de carga, deve-se transferir o servidor para outro nodo da rede. Para facilitar a operação de migração, os servidores poderiam utilizar a abstração de grupo, sendo membros do grupo e utilizar os serviços de transferência de estado disponibilizados.

O termo **CSCW** é utilizado para designar um conjunto de aplicações que possibilitam o trabalho cooperativo suportado pelo computador. São exemplos de sistemas CSCW o *talk* multiusuário, ferramentas de projeto compartilhado, edição cooperativa, sistemas de suporte à decisão em grupo, engenharia concorrente, sistemas de teleconferência, jogos multi-*player*, simulações distribuídas interativas e aplicações distribuídas com espaço de trabalho compartilhado [VIT98, COS94].

Implementação das Classes de Objetos

Para a implementação das classes de objetos foi utilizado Java, o qual é uma linguagem de programação orientada a objetos. Esta linguagem apresenta inúmeras características favoráveis ao projeto, pois é robusta, segura e distribuída, tornando o acesso aos objetos distribuídos pela rede simples e oferecendo programação cliente-servidor utilizando *sockets*, de forma que o objeto cliente envie a mensagem para os objetos servidores.

Muitos sistemas distribuídos são estruturados como cliente-servidor, no qual um cliente requisita um serviço através do envio de uma mensagem para o servidor, que processa esta mensagem retornando uma resposta ao cliente. Esta situação está sendo evidenciada na figura 1.

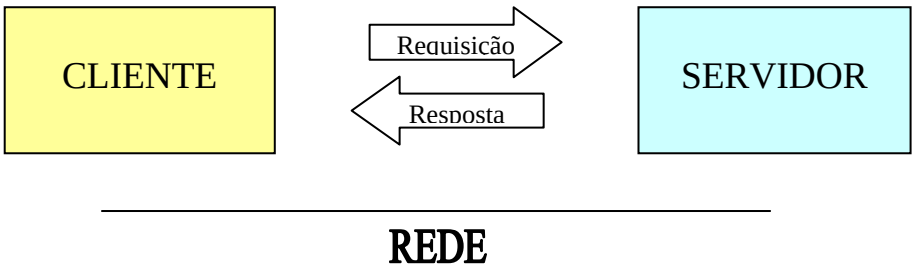

Figura 1: Ambiente Cliente-Servidor.

Na aplicação, onde um objeto necessita enviar mensagens para muitos outros objetos, a comunicação entre os objetos é feita através de troca de mensagens (*send/receive*) entre os mesmos. Esta troca de mensagens foi realizada através da utilização do protocolo de comunicação *multicast*, que é a forma utilizada para a difusão de mensagens para todos os membros (objetos) do grupo. A motivação deste projeto é devido a este comportamento dos objetos no grupo. Um grupo é uma coleção de objetos que são o destino da mesma seqüência de mensagens. Estas mensagens são originárias de um objeto cliente e os objetos destino estarão executando em um outro objeto. Cada mensagem fonte é endereçada para o grupo e o protocolo de comunicação *multicast* deve garantir que as mensagens sejam entregues aos objetos, utilizando a persistência de objetos. Será feita uma análise de aspectos como criação, manutenção e identificação do grupo; verificando o comportamento deste na ocorrência de diferentes tipos de falhas.

A aplicação possui objetos pela rede, sendo criados grupos de objetos que irão garantir a confiabilidade da aplicação, ou seja, que haja a difusão da mensagem (*multicast*) para todos os membros do grupo. A figura 2 apresenta uma aplicação de grupo de objetos, onde o objeto X está realizando uma troca de mensagens com o grupo formado pelos objetos A, B e C.

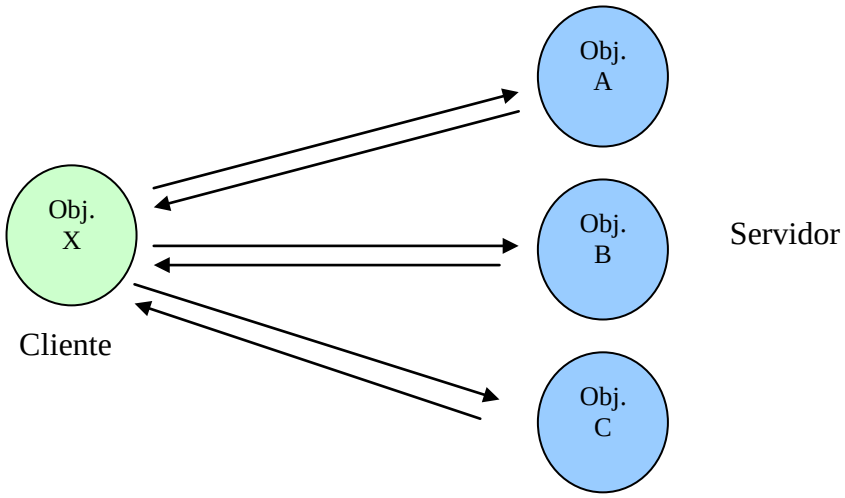

Figura 2: Troca de Mensagens entre Objetos.

Foi desenvolvida uma aplicação onde o objeto cliente envia uma mensagem ao objeto servidor e fica aguardando uma resposta. Desta forma, foi criado um programa principal chamado **serv1** que criava apenas uma *thread* com um *socket* de servidor aberto na porta **4321**. Esta *thread* fica executando em *loop* até que o cliente envie uma mensagem padrão de conexão. Neste momento, é criado um fantasma do socket cliente a é aberta a conexão com um **accept**. Essa conexão vai operar em uma nova *thread* (definida em **Conexsec**). Isso possibilita que vários clientes entrem em conexão com aquela porta, pois cada conexão será uma nova *thread*. No algoritmo apresentado na figura 3, temos a criação do servidor.

```
import java.net.*;
import java.io.*;
class serv1
{
	ConexBasica cb;

	public serv1(int port)
	{
		System.out.println("Servidor listening na porta: " + port);

		cb = new ConexBasica(port);
		cb.start();
	}

	public static void main(String args[ ])
	{
		new serv1(4321);
	}
}

class ConexBasica extends Thread
```

```
{
	int pt;
	ServerSocket ssocket;
	Conexsec csec;

	public ConexBasica (int port)
	{
		pt = port;
	}

	public void run()
	{
		try
		{
			ssocket = new ServerSocket(pt);
		}
		catch(Exception e)
		{
		System.err.println(e);
		System.exit(1);
		}

		while(true)
		{

		 try
		{
			Socket clisocket = ssocket.accept();
			csec = new Conexsec (clisocket);
			csec.start();
		}
		catch(Exception e)
		{
			System.err.println(e);
		}
		}
	}
}
class Conexsec extends Thread
{
	Socket csk;
	PrintStream soutput;

	public Conexsec (Socket s)
	{
		csk = s;
	}
	public void run()
	{
		System.out.println ("criada a conexao:"
			+ csk.getInetAddress() + ":" +csk.getPort());
		try
		{
			soutput = new PrintStream(csk.getOutputStream());
			soutput.println ("Mensagem do servidor");
			sleep (5000);
			soutput.println ("Outra mensagem");
			csk.close();
		}
		catch (Exception e)
		{
```

```
                System.err.println(e);
            }

            System.out.println ("Disconectado: "
                    + csk.getInetAddress() +":"+csk.getPort());
            stop();
        }
}
```

Figura 3: Algoritmo da criação do Servidor.

Está sendo apresentado, na figura 4, o programa cliente chamado **clix** que manda uma **mensagem padrão de conexão** para o servidor. Ela não aparece no programa pois é mandada automaticamente pelo sistema quando se cria um **socket** no cliente.

Quando criamos o *socket* no cliente, especificamos o IP do servidor e a porta que estará aberta no mesmo. Cada vez que é recebida uma mensagem ela é projetada na tela com o método **System.out.println**.

```
import java.net.*;
import java.io.*;
class clix
{
	DataInputStream cinput;
	Socket clisoc;
	public clix()
	{
		try
		{
		clisoc = new Socket("192.168.115.250",4321);
		cinput = new
DataInputStream(clisoc.getInputStream());
		while(true)
		{
			String str = cinput.readLine();
			System.out.println
			 ("Recebido do servidor: " + str);
		}
		}
		catch(Exception e)
		{
			System.err.println(e);
		}
		try
		{
			clisoc.close();
		}
		catch (Exception e)
		{
			System.err.println(e);
			System.exit(1);
		}
		System.exit(1);
	}
	public static void main(String args[])
	{
		new clix();
	}
}
```

Figura 4: Programa clix.

Em uma máquina está sendo executado o servidor que vai ficar sempre aguardando requisições dos clientes e, em outra máquina, está sendo executado o programa cliente que vai criar um socket, enviar a mensagem automática para a máquina servidora e ficar aguardando num *loop* de leitura de mensagens.

No servidor vai aparecer a mensagem informando que foi criada a conexão e dando o endereço IP do cliente. Neste momento, o servidor envia as duas mensagens e o cliente as lê e projeta na tela. O servidor, tendo transmitido as duas mensagens, encerra a conexão e apresenta, então, uma nova mensagem: **Disconectado: 192.168.115.250 (endereço IP).**

A inexistência do protocolo de comunicação *multicast* confiável pode inviabilizar as comunicações entre os objetos, pois podem ocorrer falhas de particionamento da rede, de *crash* nos nodos processadores ou até mesmo pela lentidão existente na rede. A figura 6 apresenta um cenário onde ocorre uma falha de particionamento na rede.

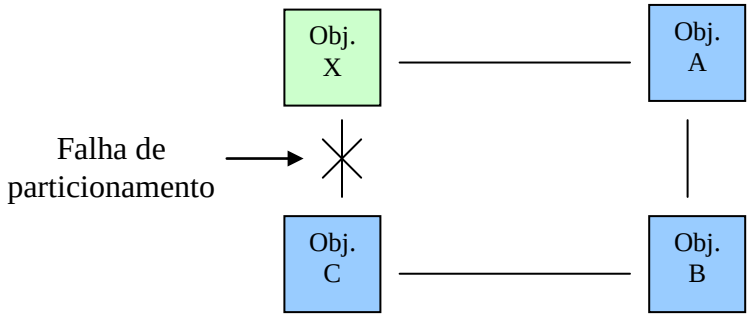

Figura 6: Falha de Particionamento na rede.

2- Conclusões Parciais

A utilização de uma troca de mensagens segura e confiável entre os objetos comunicantes de um sistema de comunicação de grupo é imprescindível. Assim sendo, a difusão das mensagens deve acontecer de forma confiável para que, mesmo na falha de particionamento de rede ou de *crash* em algum nodo processador, a aplicação continue sua execução.

Deve-se, ainda, implementar as classes necessárias para a utilização de uma troca de mensagens confiável (*multicast* confiável), juntamente com a persistência de objetos que fará a substituição de um objeto falho.

3- Referências Bibliográficas

[BIR 87] BIRMAN, K.P., JOSEPH,T.A. Reliable Communication in the Presence of Failures. **ACM Transactions on Computer Systems**, v.5, n.1, Feb. 1987.

[COS 94]] COSQUER, F.J.N. ; VERÍSSIMO, P. Survey of Selected Groupware Applications and Supporting Platforms. Lisboa-Portugal: INESC-Technical Report 21-94, Set.1994, 33p.

[FEL 98] FELBER, P. The CORBA Object Group Service: A Service Approach to Object Groups in CORBA. École Polytechnique de Lausanne, 1998. (Disponível na Internet em http://lsewww.epfl.ch/~felber). (Ph.D. Thesis).

[MOR 99] MORGAN, G. ; SHRIVASTAVA, S.K. ; EZHILCHELVAN, P.D. ; LITTLE, M.C. Design and Implementation of a CORBA Fault-Tolerant Object Group Service. 1999. (Disponível na Internet em http://arjuna.ncl.ac.uk/group/papers/p080.ps).

[VIT 98] VITENBERG, R. ; DOLEV, D. Properties of Distributed Group Communication and their Utilization. Jerusalem: Institute of Computer Science of the Hebrew University of Jerusalem, Thesis of Master of Science, Jan. 1998.