Sincronização em uma Aplicação Distribuída para a Educação à Distância Utilizando Recursos do *Middleware* ProActive

Berto de Tácio P. Gomes
Omar A. Carmona Cortes
Centro Federal de Educação Tecnológica do Maranhão
Departamento Acadêmico de Informática - DAI
São Luis - MA - Brasil
bertodetacio@gmail.com, omar@cefet-ma.br

Resumo

O objetivo deste trabalho é mostrar recursos de um middleware *open source, que foram utilizados no desenvolvimento de um Chat que possui mecanismos de sincronização análogos aos algoritmos de exclusão mútua em sistemas distribuídos. Foram utilizados em especial os algoritmos centralizado, distribuído e token-ring. A aplicação, batizada com o nome de SynchroTalk, foi desenvolvida em linguagem Java e o* middleware *é denominado ProActive.*

1. Introdução

Ferramentas de bate papo são exemplos de aplicações distribuídas muito populares. No entanto, os trabalhos de Oeiras [5], Pimentel [7], entre outros, mostram as dificuldades encontradas em ferramentas de bate-papo, quando vários usuários enviam mensagens simultaneamente, pois vários núcleos de conversa podem ser gerados, dificultando a identificação de quem está falando, com quem e sobre o que. Isso torna crítico o uso dessa ferramenta no ensino distância, pois a quantidade de alunos pode ser muito grande. Observa-se que esse tipo de ferramenta carece de coordenação, que pode ser obtida através de mecanismos de sincronização do recurso de envio de mensagens.

Os mecanismos propostos por Smith [9] e Pimentel, abordam linhas de diálogo dentro do bate-papo, onde cada usuário indica a mensagem que deseja responder. Vahl Junior [11] utiliza tipos de entonação para o ato de pedir palavra, onde tem mais chances de falar o usuário que escolher a entonação de maior nível. Tais soluções tiveram dificuldades de aceitação pelos usuários, os quais tiveram problemas de adaptação aos mecanismos de sincronização propostos, uma vez que estes não se mostraram transparentes e exigiam mais esforço cognitivo.

Durante este projeto está sendo elaborado um *Chat Desktop*, chamado *SynchroTalk* (Figura 1), que possui mecanismos de sincronização transparentes aos usuários e deverá ser utilizado em cursos de educação à distância.

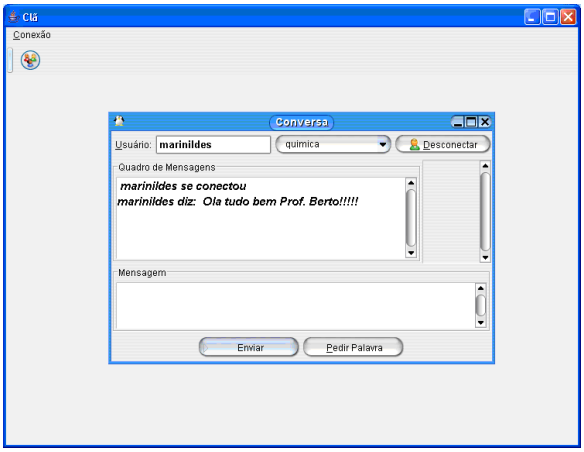

Figura 1. SynchroTalk

Este artigo tem o objetivo de mostrar recursos do *middleware* ProActive [4], que auxiliaram na implementação das soluções. Os mecanismos de sincronização são análogos aos algoritmos de sistemas distribuídos [2], para o tratamento de exclusão mútua [10]. Na literatura consultada não foram encontrados registros de tentativas que também tenham se baseado nesses algoritmos.

O artigo encontra-se organizado da seguinte forma: a Seção 2 mostra os motivos que levaram a escolha

do *middleware* ProActive; a Seção 3 explica a relação entre os algoritmos de exclusão mútua e o SynchroTalk; a Seção 4 trata das implementações dos mecanismos de sincronização; a Seção 5 contém as considerações finais; finalmente, a Seção 6 apresenta os trabalhos futuros.

2. Motivações para o uso do Proactive

ProActive é um framework Java para computação concorrente, paralela, distribuída e em grade (*GRID*). Entre os motivos de sua escolha pode-se citar o fato de que, ao contrário de outras tecnologias como CORBA[6] e JavaRMI [3], esse *middleware* não requer que classes que originam objetos remotos sejam modificadas, simplificando o desenvolvimento de aplicações distribuídas e permitindo o polimorfismo entre objetos locais e remotos.

A comunicação distribuída ocorre através de objetos ativos. Objetos ativos são unidades de atividade que podem ser acessadas remota e concorrentemente, com capacidade de migrar entre computadores através da rede, possibilitando o uso de agentes móveis. Cada objeto ativo possui sua própria *thread*, que executa métodos invocados dentro do objeto ativo. Com o ProActive o programador não precisa manipular explicitamente objetos de linha de execução, e ainda pode definir políticas FIFO ou LIFO para a execução de chamadas.

Entre outros recursos interessantes no ProActive estão os grupos de comunicação, que proporcionam melhorias de desempenho, eficiência e flexibilidade em aplicações distribuídas [1]. O *middleware* utilizado contém ainda mecanismos de tolerância à falhas, checkpoint, segurança e monitoramento remoto.

Além das vantagens mencionadas, ProActive é open source sob licença LGPL, o que o tornou uma boa escolha em relação a tecnologias proprietárias de mesma finalidade, como por exemplo, o *middleware* Voyager [8].

Aliado a esses fatores, o ProActive disponibiliza também o software IC2D, que permite o monitoramento da aplicação, visualização de chamadas remotas, detecção de falhas e ainda acompanhar o movimento de agentes através da rede.

3. Algoritmos de exclusão mútua e a sincronização na ferramenta de *Chat*

Para realizar a sincronização na ferramenta SynchroTalk fez-se uma analogia ao problema de exclusão mútua em sistemas distribuídos, onde apenas um processo por vez pode ter acesso a um mesmo recurso, tido como região crítica. O recurso de envio de mensagens foi considerado a região crítica do SynchroTalk, para o qual foram adotados algoritmos de sincronização similares aos que podem ser usados em sistemas distribuídos. Os algoritmos utilizados foram: centralizado, distribuído e *token-ring*. Para cada algoritmo criou-se um mecanismo análogo.

No algoritmo centralizado há um coordenador para controlar o acesso à região crítica. Um processo, antes entrar na região, envia uma mensagem de solicitação de recurso ao coordenador que coloca todas as mensagens recebidas em uma fila. O processo solicitante deverá aguardar a resposta do coordenador, que verifica se algum outro processo ocupa o recurso solicitado. Quando um processo deixa a região crítica, envia uma mensagem ao coordenador informando que liberou o recurso, e o coordenador poderá atender o próximo pedido. Baseado neste algoritmo, o SynchroTalk mantém um mecanismo centralizado de controle, onde há um coordenador humano e/ou processo computacional para controlar o acesso ao recurso de envio de mensagens. Os participantes agem como se fossem processos. Para qualquer que seja o coordenador, pode-se determinar o tempo que os participantes terão para postar suas mensagens, neste caso, o recurso estará liberado automaticamente após o término do tempo, permitindo que o coordenador atenda outra solicitação. No contexto educativo, o coordenador poderá ser um professor ministrando uma aula à distância. Com o apoio deste mecanismo, o professor poderá definir um aluno para fazer perguntas ou permitir exposições sobre o assunto discutido.

O segundo mecanismo de sincronização do SynchroTalk é semelhante a um algoritmo distribuído. Neste algoritmo, quando um processo deseja entrar na região crítica, ele envia uma mensagem com seu pedido e o tempo atual para todos os outros e para si mesmo. Quando um processo recebe uma mensagem desse tipo, ele reagirá de acordo com o seu estado podendo:

1. Enviar uma mensagem de OK ao processo emissor, caso não esteja na região crítica e não queira entrar nela;

2. Não responder a mensagem recebida e inserir-la numa fila, caso esteja na região crítica;

3. Caso o processo deseje entrar na região crítica, e ainda não o tiver feito, ele compara o tempo da mensagem recebida com o tempo constante da mensagem que ele enviou aos demais processos, enviando uma mensagem de OK como resposta a mensagem recebida se esta possuir um tempo menor que a mensagem enviada, caso contrário, ele não responde a mensagem e insere-a numa fila.

Na implementação do mecanismo distribuído análogo a esse algoritmo, os participantes enviam mensagens especiais (diferentes das mensagens normais de um *Chat*) para os outros e para eles mesmos, quando querem ter acesso ao recurso. Quando um participante recebe um pedido, seu software de *Chat* reage automaticamente de forma semelhante a um processo na mesma situação. Quando um participante envia seu pedido aos demais, ele só poderá ter acesso ao recurso quando receber de todos os outros uma mensagem de OK, devendo enviar uma mensagem de liberação quando não quiser mais enviar mensagens (mensagens normais), ou caso o tempo de envio termine (essa possibilidade foi adotada caso se queira evitar que um participante monopolize o recurso).

No algoritmo *token-ring* cria-se um anel de processos pelo qual circula um *token*. Quando um processo recebe o *token*, ele poderá entrar na região crítica caso queira (só poderá fazê-lo quem estiver com o token). O processo deverá *soltar* o *token* ao deixar a região crítica, ou passá-lo imediatamente caso não deseje acessá-la. No SynchroTalk, o modelo análogo a esse algoritmo, organiza todos os participantes em um anel, pelo qual circula uma permissão. Quando a permissão chega a um participante, ele a repassa ao próximo participante caso não queira postar mensagens, ou, a retém caso queira, e deverá solta-lá ao termino da utilização do recurso. Semelhante aos demais mecanismos, pode-se determinar o tempo de retenção da permissão, o que permite que a mesma seja repassada periodicamente.

4. Implementação

4.1. Mecanismo centralizado

No mecanismo centralizado um objeto ativo (localizado no servidor) faz o papel de coordenador da região crítica. Esse objeto recebe todas as solicitações dos participantes do SynchroTalk e as armazena em uma fila de mensagens. Este objeto ativo verifica se algum participante está com permissão para enviar mensagens, caso contrário atende a próxima solicitação da fila assim que o participante que estiver com a palavra liberar o recurso.

O ProActive adiciona um conjunto de exceções que vão além das exceções normais (originadas das regras de negocio), esse novo conjunto é chamado de *Exceções Não Funcionais* (NFE), que indicam problemas durante a comunicação entre objetos distribuídos. Desta forma, quando um objeto ativo percebe um problema na comunicação com o participante que estava enviando mensagens, ele pode imediatamente atender a solicitação de outro. Quando o tempo de postagem de mensagens for determinado, o coordenador atende outra solicitação assim que o mesmo acabar. Um coordenador humano poderá manipular o objeto ativo, de forma a fazer concessões de recurso de acordo com a conveniência, permitindo maior flexibilidade neste mecanismo.

4.2. Mecanismo distribuido

No mecanismo distribuído utilizou-se o recurso de formação de grupos de comunicação. Isso permitiu reunir todas as referências remotas dos participantes, e fazer o *broadcast* das mensagens que contém os pedidos de recurso. As chamadas efetuadas aos membros do grupo são realizadas de forma paralela, melhorando o desempenho. A verificação das *Exceções Não Funcionais* é um fator importante para o uso deste algoritmo, já que cada nó da rede pode ser um ponto de falhas. Quando o algoritmo percebe uma exceção desse tipo (lançada por problemas de comunicação na rede) deixa-se de esperar sua resposta. O recurso de envio de mensagens é liberado caso o host onde a falha ocorreu estiver usando o recurso. O participante cujo host falhou é retirado do grupo, permitindo a continuidade do algoritmo.

Outra característica dos grupos de comunicação do ProActive é sincronização de chamadas com espera por necessidade, que possibilitou fazer com que um participante só tivesse acesso ao envio de mensagens assim que todas as chamadas efetuadas ao grupo retornassem. Para evitar problemas com sincronização de relógios, o tempo base para comparação de mensagens é tempo de chegada no servidor.

4.3. Mecanismo *token-ring*

O mecanismo *token-ring* utilizou uma classe do ProActive denominada *Ring*. Graças a essa classe é possível criar uma estrutura de coleção capaz de organizar as referências remotas dos participantes do SynchroTalk em forma anel. Nessa coleção existem métodos para acessar os objetos nas curvas a direita e a esquerda do anel, o que torna fácil saber para qual participante o token deve ir e ainda permite mudá-lo de direção.

Utilizou-se a tecnologia de agentes móveis para criar um agente que se move pelo anel como se fosse um *token*. Ao chegar a um host remoto, o agente interage com o participante perguntando se este deseja enviar mensagens para o *Chat*. O agente se move para o próximo host caso o participante não queira usar o recurso, ou, assim que este terminar de enviar mensagens. A aplicação possui a flexibilidade de estabelecer um limite de tempo para a permanência do agente com um

participante. Para evitar problemas em caso de falha em algum host, o agente informa periodicamente sua localização.

5. Considerações finais

Os mecanismos de sincronização implementados no SynchroTalk buscaram os algoritmos de exclusão mútua como referenciais teóricos, ajudando a consolidar teoria e prática. Apesar das adaptações feitas em função da arquitetura e do objetivo da aplicação, os princípios teóricos foram mantidos.

Assim como na teoria dos sistemas distribuídos, o mecanismo centralizado foi mais simples de programar e tende a ser mais eficiente. No entanto, um coordenador pode ser um ponto crítico de falhas.

Já o mecanismo distribuído elimina o *starvation*, e pode ser uma solução alternativa caso não se deseje um coordenador central, mas o grande fluxo de mensagens gerado requer mais processamento. Além disso, cada host pode ser um ponto crítico de falhas.

No mecanismo token-ring, não é necessário solicitar um recurso, mas o (*token*) pode ser perdido. Nesse contexto, um agente de emergência pode percorrer o anel em busca do *token* e gerar um novo caso o mesmo não seja encontrado.

Espera-se que os mecanismos do SynchroTalk sejam úteis no contexto educacional, especialmente por serem transparentes aos usuários, pois independentemente do mecanismo utilizado, o usuário necessita apenas enviar um pedido de recurso e esperar a concessão. Apenas no caso do mecanismo *token-ring* é necessário que o usuário confirme se deseja ou não enviar mensagens.

6. Trabalhos Futuros

Devido às vulnerabilidades que cada algoritmo pode apresentar, e ainda pelo fato de aplicações distribuídas estarem sujeitas a problemas na comunicação em rede, pretende-se explorar os recursos de tolerâncias às falhas, disponibilizados pelo ProActive.

No mecanismo centralizado espera-se conseguir eleger um host para coordenar o envio de mensagens, em caso de problemas com o coordenador anterior. Ainda com os recursos de tolerância à falhas, espera-se conseguir gerar um agente móvel substituto, caso o anterior seja perdido na rede. Pode-se ainda adicionar Inteligência à esse agente. Espera-se ainda minimizar possíveis falhas no algoritmo distribuído.

Futuramente, após melhorias na interface gráfica, o SynchroTalk será testado por meio de aulas à distância, para verificar a eficiência das soluções.

Agradecimentos

Os autores agradecem o apoio financeiro do CNPq sem o qual este trabalho não seria possível.

Referências

[1] L. BADUEL, F. Baude, and D. Caromel. Efficient, flexible, and typed group communications in java. In *Joint ACM Java Grand - ISCOPE 2002 Conference*, pages 28–36, ACM Press, 2002.

[2] G. COULOURIS and T. DOLLIMORE, J. ans KINDBERG. *Distributed Systems: Concepts and Design*. Adson Wesley, 3 edition, 2000.

[3] S. Microsystems. Java rmi. http://java.sun.com/products/jdk/rmi/, Visitado em 25 de junho de 2006.

[4] OBJECTWEB. Proactive. http://www.objectweb.org/ProActive, Visitado em 25 de maio de 2006.

[5] J. Y. Y. OEIRAS and H. V. ROCHA. Uma modalidade de comunicação mediada por computador e suas várias interfaces. In *Workshop Sobre Fatores Humanos em Sistemas Computacionais*, pages 151–160, UFRGS - Porto Alegre - RS, 2000.

[6] OMG. Corba website. http://www.corba..com, Visitado em 20 de junho de 2006.

[7] M. G. PIMENTEL and F. F. SAMPAIO. Hiperdiálogo uma ferramenta de bate-papo para diminuir a perda de co-texto. In *SIMPÓSIO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO*, pages 21–23, Vitória-ES, 2002.

[8] V. W. Site. Voyager. http://www.recursionsw.com/voyager, Visitado em 25 de junho de 2006.

[9] M. SMITH, J. J. CADIZ, and B. BURKHALTER. Conversation trees and threaded chats. *SIGCHI Bulletin*.

[10] A. S. TANENBAUM. *Sistemas Operacionais Modernos*. Pretince Hall, São Paulo, 2003.

[11] J. C. VAHL JÚNIOR. Uso de agentes de interface para adequação de bate-papos ao contexto de educação a distância. Master's thesis, Instituto de Computação, Universidade Estadual de Campinas - UNICAMP, Campinas-SP, 2003.