Adicionando Replicação utilizando Componentes de Software e um Ambiente Interativo

João Carlos Filho, Silvia de Castro Bertagnolli, Maria Lúcia Blanck Lisbôa

e-mail: jfilho@msinternet.com.br, {silviacb, llisboa}@inf.ufrgs.br

Universidade Federal do Rio Grande do Sul

Instituto de Informática

PPGC – Programa de Pós-Graduação em Computação

Caixa Postal 15064 CEP 91501-970 BRASIL

RESUMO

Nosso objetivo é simplificar o desenvolvimento de aplicações tolerantes a falhas e minimizar o tempo despendido na fase de implementação. Para isto, foi utilizado um ambiente reflexivo de apoio à programação JReflex e um conjunto de componentes que implementam replicação usando RMI - Remote Method Invocation em Java. Descrevemos brevemente esses componentes e como usar JReflex para selecionar as classes a serem por eles replicadas.

Palavras-chave: replicação de objetos, chamada remota de métodos, ambientes interativos de desenvolvimento de software, linguagem de programação Java.

ABSTRACT

Our purpose is to simplify the development of fault tolerant applications and to reduce the amount of time spent in their implementation phase. To do this, we use a reflexive programming environment (JReflex) and a set of components that implement replication, using RMI - Remote Method Invocation in Java. We briefly describe those components and how to use JReflex to select the classes that must be replicated by such components.

Keywords: replication of objects, remote method invocation, software development environments, Java programming language.

1 INTRODUÇÃO

A base da tolerância a falhas é a redundância. No caso de mecanismos de tolerância a falhas por software, esta redundância significa replicação de dados ou processos. A replicação permite que, caso a máquina onde uma das réplicas está situada falhe ou torne-se indisponível temporária ou permanentemente, o serviço continue disponível utilizando-se de outra réplica acessível.

Uma aplicação no modelo de objetos é estruturada a partir de classes, as quais definem as funcionalidades esperadas da aplicação e determinam os serviços e interações dos objetos instanciados durante o processo de execução. Alguns desses serviços podem ser considerados críticos pela aplicação, exigindo, por exemplo replicação.

Assim, para agilizar e incentivar a programação de aplicações tolerantes a falhas desenvolveu-se este trabalho de integração entre o ambiente JReflex e um conjunto de componentes de replicação.

O ambiente de programação JReflex foi desenvolvido por Bertagnolli em [BER 00a] com a finalidade de propiciar ao programador facilidades para desenvolvimento interativo de aplicações reflexivas na linguagem Java. JReflex utiliza, fortemente, reflexão computacional para fornecer ao programador informações sobre classes e objetos de seu programa, bem como possibilita a geração de código para permitir que algumas destas informações reflexivas sejam incorporadas ao programa do usuário [BER 00a]. Neste trabalho, JReflex realiza a seleção das classes do usuário a serem replicadas e gera, automaticamente, o código responsável pela replicação destas classes.

Os componentes foram desenvolvidos por Ferreira [FER 00] com o intuito de simplificar a tarefa de obtenção de réplicas a partir de um ou mais objetos da aplicação considerados críticos. Ou seja, um ou mais objetos da aplicação original podem ser replicados, bastando que o programa indique a classe de cada um desses objetos. Os componentes de replicação de JReflex realizam os seguintes serviços: criam idênticos objetos (mesmo estado), distribuem as réplicas em diferentes máquinas e, cada vez que o objeto original da aplicação recebe uma mensagem, a mesma é enviada para todas as réplicas. Este envio é realizado utilizando-se o mecanismo de invocação remota de métodos cuidando assim de preservar a consistência das réplicas.

A seleção da linguagem Java para desenvolver tanto os componentes quanto o ambiente deve-se aos seguintes fatores: portabilidade, segurança, eficiência e facilidades de uso de componentes de software. Outro fator determinante é o sistema de chamada remota de métodos (RMI) que essa linguagem oferece. Em Ferreira [FER 99] e Bertagnolli [BER 00b] realizaram-se estudos sobre implementação de replicação na linguagem Java utilizando-se RMI.

Este trabalho prossegue descrevendo os componentes desenvolvidos (seção 2). A seção 3 descreve o uso de JReflex e a seção 4 dedica-se às conclusões.

2 COMPONENTES DESENVOLVIDOS

Os componentes de replicação apresentados neste trabalho surgiram como uma extensão do trabalho de Amaral [AMA 99], o qual utiliza classes genéricas para a obtenção de cópias de objetos, fazendo a seguir a sua serialização para seu transporte em redes de comunicação. Além disso, é fácil compreender que no momento que o programador está desenvolvendo uma aplicação que envolve aspectos não funcionais (tais como concorrência, replicação, serialização, etc.), ele é forçado a desviar sua atenção do seu objetivo principal e preocupar-se com aspectos de implementação. Por exemplo, o programador está escrevendo uma aplicação de contas bancárias. Para inserir replicação nesta classe, ele necessitaria

implementar algoritmos para criar as réplicas, distribuí-las e implementar alguma técnica para o gerenciamento das mesmas, e finalmente, desenvolver sua aplicação que desde o princípio deveria ser o foco central de sua atenção.

Os componentes desenvolvidos por [FER 99] realizam a adição de replicação em aplicações centralizadas através da implementação de grupos de objetos compostos por coleções de objetos remotos replicados. Estes podem ser acessados pelos clientes através da chamada remota de seus métodos e com o mesmo nível de simplicidade de objetos não replicados [BAB 98]. A replicação destes objetos exige [BIR 87, BIR 96, FRI 95, GUO 96], entre outras coisas, gerenciamento de grupos [LIA 90] e controle da replicação de dados [GUE 96].

Para a implementação do serviço de controle da replicação é utilizada a abordagem de replicação primário-backup. O objeto primário está encarregado de criar as réplicas e distribuí-las em diferente processadores, receber as mensagens de solicitação de serviços e disseminá-las entre as réplicas e assegurar a consistência de estado entre as réplicas. Deve-se observar que todas as réplicas são instâncias de uma mesma classe, ou seja, possuem idênticas propriedades e se distinguem apenas por seu estado. Portanto, ao criar uma réplica de um objeto em um determinado instante, o componente de replicação deve ser capaz de fornecer uma cópia da parte estática (estrutura de dados e código) juntamente com uma cópia da parte dinâmica (variáveis de estado). A obtenção de cópias de objetos é feita num determinado momento da execução e, portanto, cada cópia de objeto contém o estado da computação que pode ser usado pelos algoritmos de controle de consistência de estado de componentes replicados.

A idéia básica dos componentes de replicação desenvolvidos é que qualquer programador possa replicar classes consideradas críticas para sua aplicação. Além de atuar sobre os objetos da classe crítica o sistema deve fazer a replicação de forma transparente ao programador; para esta finalidade, foi usado o ambiente JReflex.

O mecanismo de replicação adotado baseia-se em um modelo onde um sistema distribuído consiste de um conjunto de sites interconectados por uma rede de comunicação Assume-se também que o sistema é síncrono, ou seja, existem limites para o atraso das mensagens. O modelo admite dois estados: operante e inoperante. Enquanto está operante cada réplica fornece exatamente o serviço esperado. Uma réplica pode sofrer alguma falha, tornar-se inoperante e não executar mais nenhuma ação. Este é o único tipo de falha admitido por este modelo e é chamada de *crash*. Uma réplica pode também, após sofrer uma falha e perder todo seu estado local, recuperar-se através de um protocolo de recuperação. Além disso, para determinar se uma réplica está falha ou não, é utilizado o protocolo de *commitment*. Do ponto de vista da comunicação, o modelo não admite falhas, ou seja, a comunicação é confiável.

Os principais serviços da replicação são: criação das réplicas, disseminação de mensagens e a consistência das réplicas. Para atender aos aspectos anteriormente mencionados, vários algoritmos foram desenvolvidos, mas neste trabalho apenas os mais importantes serão brevemente explicados:

- `ClienteCriaReplicas`: responsável por criar as cópias, este algoritmo determina se deve ser criada uma réplica ou o cliente deve ser atualizado com uma já existente, isto é, deve ser recuperado após uma falha.
- `ClienteAtualizaReplica`: responsável pelo início da transação que atualiza o estado do primário e dos backups, este algoritmo baseia-se no conceito de transação [BAB 98] para garantir a consistência das réplicas.
- `ReplicaCommit`: faz parte do esquema de consistência e tem o objetivo de iniciar o processo de decisão das réplicas para garantir atomicidade na entrega das mensagens.

- `Broadcast`: finalizando o processo de consistência este envia a mensagem de confirmação para todos os membros do grupo.
- `ValidaGrupo`: utilizado por `ClienteAtualizaReplica` para testar as réplicas e tentar recuperar os membros que inativos.

A idéia central para manter o grupo sempre com a informação correta de quem está ativo ou não, é atualizar o estado das réplicas uma a uma e marcar como inativas aquelas que não responderem a mensagem de atualização. Uma réplica que está marcada como inativa não é considerada quando houver a necessidade de escolher um novo primário. No entanto, ela fica no grupo e, a cada operação de atualização realizada pelo primário, é requisitada a responder com um sinal de vida. Quando o fizer, ela é marcada como ativa e é atualizada pelo primário.

O objeto original da aplicação é considerado como cliente em relação às suas réplicas, sendo estas os servidores. Entre os servidores, um deles é considerado como primário e os demais como backups.

Quando o primário falha, o cliente e os backups não são avisados. O cliente, aplicação que utiliza o componente de replicação, irá perceber a falha quando não receber resposta a uma invocação. Neste momento, o algoritmo `ProcuraNovoPrimario` é executado pelo cliente que percorre a matriz de membros para marcar o antigo primário como um backup inativo. O algoritmo seleciona o novo primário e pede um sinal de vida. Se receber resposta, a matriz é atualizada para representar a nova situação. Finalmente, após escolher o novo primário o algoritmo comunica a ele sua promoção, atualiza sua matriz de grupo, solicita sua conexão aos backups restantes e atualiza seu estado. Quando o novo primário conecta-se aos backups ele os informa das alterações ocorridas no grupo. Quando o antigo primário recuperar-se ele será reintegrado ao grupo porém, agora como um backup. Se o cliente não conseguir obter nenhum primário, o algoritmo continuará tentando a cada vez que houver uma alteração no estado do objeto crítico até que alguma das réplicas recupere-se e possa assumir o lugar do primário.

Em resumo, quem procura o novo primário é o componente de replicação que é utilizado pela aplicação cliente. Deve-se observar que o fato da aplicação cliente não receber resposta a uma invocação não tira a transparência do mecanismo, visto que, a invocação não respondida será submetida à outro primário de forma imperceptível à aplicação cliente.

O algoritmo `PrimarioConectaBackups` percorre a matriz de grupo e para os elementos que são backups e estão ativos obtém o acesso remoto. Se o primário não tem replica criada ele testa se algum de seus backups tem. Se encontrar, ele coloca o valor 2 em sua variável de estado `ReplicaCriada` significando que o primário está desatualizado. Sempre que o cliente se conecta a um primário, ele verifica este valor e, uma vez detectada a desatualização do primário, ele executa o algoritmo de seleção de um novo primário.

3 AMBIENTE JREFLEX

Nesta seção é brevemente descrito o ambiente interativo de programação JReflex, no que tange aos seguintes serviços:

- disponibilizar uma biblioteca de componentes que simplifiquem a utilização de classes de bibliotecas especializadas da linguagem Java, por exemplo componentes de replicação, distribuição, concorrência, etc.;
- oferecer uma interface interativa, baseada em janelas, para permitir ao programador incorporar ao seu programa componentes identificados ou selecionados como os que atenderam a solicitação realizada.

Ao ser ativado, JReflex exibe uma interface gráfica que contém os seus principais componentes: um editor de programas, um visualizador de classes e uma área de mensagens. Através de menus, são oferecidos os serviços: inspeção de classes, inserção de código, e bibliotecas de componentes. Os componentes de replicação foram integrados ao JReflex como um destes serviços.

Para desenvolver uma aplicação com replicação, o desenvolvedor segue um roteiro simples: (a) abre sua aplicação na janela de edição do menu principal; (b) seleciona os serviços de replicação; (c) indica a classe de seu programa a ser replicada; (d) especifica as máquinas onde as réplicas devem ser instaladas. A seguir, os componentes de replicação são incorporados pelo ambiente ao programa do usuário, já adaptados ao contexto. O programa é então compilado (e pode ser executado) através do ambiente, e, em caso de erros, as mensagens são exibidas na área de mensagens. A figura 1 exibe a janela do ambiente responsável pela ativação dos componentes de replicação.

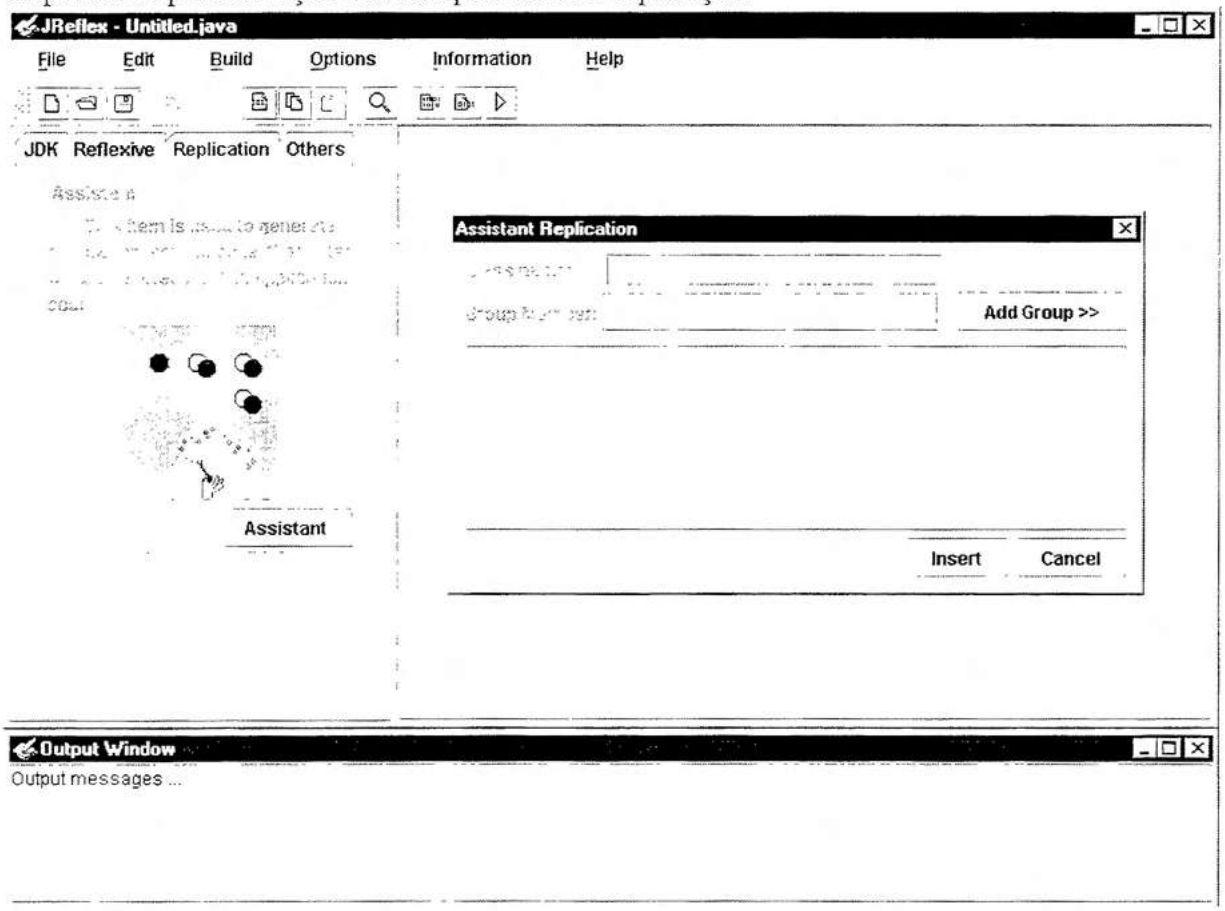

Figura 1: Janela para ativação dos componentes de replicação

A idéia de aplicar um ambiente para apoiar o desenvolvimento de aplicações reflexivas tolerantes a falhas surgiu a partir do trabalho de Lisboa [LIS 95]. Bertagnolli [BER 00a] desenvolveu o ambiente JReflex com o intuito de facilitar o desenvolvimento de software reflexivo na linguagem Java. Utilizando como base os componentes desenvolvidos por Ferreira [FER 00](seção 2), este ambiente possibilita ao programador compilar e executar, de forma simples, programas que utilizam o mecanismo RMI. Além disso, todas as configurações de segurança, de inicialização, de compilação e execução são geradas automaticamente pelo ambiente. Isto deve-se ao fato de que o mecanismo de RMI exige parâmetros de compilação distintos dos exigidos para a compilação de outros programas.

4 CONCLUSÕES

Em resumo, a utilização do ambiente JReflex aliada ao conjunto de componentes que implementam serviços de replicação, auxiliam o programador de várias maneiras:

- a partir de uma aplicação não tolerante a falhas, é possível transformá-la em uma aplicação que usa a técnica de replicação, sem que o programador implemente código específico de tolerância a falhas;
- os componentes a serem replicados - classes da aplicação - são selecionados, replicados e distribuídos, sem que o programador escreva uma única linha de código; a interação é feita através de menus;
- o programador utiliza um código que já foi testado, validado e está documentado, diminuindo assim a taxa de erros, falhas e o tempo para a finalização de sua aplicação

Este trabalho de integração pode ser feito também para outras categorias de bibliotecas de componentes, visando adicionar requisitos não funcionais a aplicações já existentes. Reduzindo, assim, a quantidade de conhecimento e de código normalmente exigida do desenvolvedor de aplicações tolerantes a falhas.

AGRADECIMENTOS

Este trabalho foi desenvolvido com o apoio da FAPERGS e CAPES.

5 REFERÊNCIAS

[AMA 99] AMARAL, J.B.; BERTAGNOLLI, S. C; LISBÔA, M.L.B. Componentes para Apoiar o Desenvolvimento de Aplicações Tolerantes a Falhas. In: SIMPÓSIO DE COMPUTAÇÃO TOLERANTES A FALHAS, 8., 1999, Campinas, SP. **Anais...** Campinas: SBC, 1999. p. 142-146.

[BAB 98] BABAOGLU, O.; DAVOLI, R.; MONTRESOR, A. **Group Communication in Partitionable Systems: Specification and Algorithms**. Bologna: Dept. of Computer Science, University of Bologna. April, 1998. Technical Report UBLCS-98-01.

[BER 00a] BERTAGNOLLI, S. C. **Ambiente Visual para o Desenvolvimento de Aplicações Java Reflexivas**: dissertação de mestrado. Porto Alegre: PPGC da UFRGS, maio, 2000.

[BER 00b] BERTAGNOLLI, S. C. **Integrando Componentes Tolerantes a Falhas em um Ambiente de Programação Reflexivo**: trabalho individual. Porto Alegre: PPGC da UFRGS, 2000.

[BIR 87] BIRMAN, K.; JOSEPH, T. Reliable Communication in the Presence of Failures. **ACM Trans. On Computer Systems**, v.5, n.1, Feb. 1987

[BIR 96] BIRMAN, K.; **Building Secure and Reliable Network Applications**. Greenwich: Manning Publ., 1996, 59p.

[FER 99] FERREIRA FILHO, João Carlos. **Implementação de Objetos Replicados usando Java RMI :** trabalho individual. Porto Alegre: PPGC da UFRGS, 1999.

[FER 00] FERREIRA FILHO, João Carlos. **Implementação de Objetos Replicados usando Java:** dissertação de mestrado. Porto Alegre: PPGC da UFRGS, 2000.

[FRI 95] FRIEDMAN, Roy. **Using virtual synchrony to develop efficient fault tolerant distributed shared memories**. Department of Computer Science - Cornell University. Ithaca-NY, 1995.

[GUE 96] GUERRAOUI, Rachid; SCHIPER, A. Fault-Tolerance by Replication in Distribuited Systems. Reliable Software Technologies In: ADA EUROPE'96, p.38-57, 1996.

[GUO 96] GUO, K.; VOGELS, W.; RENESSE, R.V. Structured Virtual Synchrony: Exploring the Bounds of Virtual Synchronous Group Communication. In: ACM SIGOPS EUROPEAN WORKSHOP, **Proceedings...**, Connemaran-Ireland, Sep. 1996.

[LIA 90] LIANG, L.; CHANSON, S., NEUFELD, G.; Process Groups and Group Communication: Classifications and Requirements, **IEEE Computer**, p. 56-66, Feb. 1990.

[LIS 95] LISBOA, M. L. B. **MOTF: Meta-objetos para Tolerância a Falhas:** tese de doutorado. Porto Alegre: CPGCC da UFRGS, 1995.