Substituição Dinâmica de Classes com Validação de Objetos

Werner Haetinger
wernerh@inf.ufrgs.br

Maria Lúcia Blanck Lisbôa
llisboa@inf.ufrgs.br

Universidade Federal do Rio Grande do Sul
Instituto de Informática
Curso de Pós-Graduação em Ciência da Computação - CPGCC

Av Bento Gonçalves, 9500 - Bloco IV
CEP 91501-970 Porto Alegre - RS

Resumo

A necessidade de evolução de software é uma realidade presente em todos os sistemas de computação, seja para alterar ou adicionar funcionalidades. Um importante aspecto a considerar é a possibilidade de antecipar, ainda na fase de projeto, os tipos de alterações que o software poderá sofrer durante a sua fase operacional. A antecipação permite construir componentes de software que isolem os aspectos sujeitos a alterações, de forma a facilitar a sua substituição por uma nova versão. O sistema de substituição dinâmica aqui apresentado utiliza o modelo de objetos para a construção de componentes encapsulados e emprega reflexão computacional para hospedar técnicas de tolerância a falhas, visando assegurar a manutenção da confiabilidade da nova versão do software.

Abstract

Software evolution must be considered in the context of all computational systems. Evolution includes both correction and modifications to add new capabilities. It is important to anticipate the kinds of alterations to be carried on during the software lifetime in order to promote its maintainability. This permits to encapsulate the software components prone to alteration, thus facilitating the substitution of these components by new versions.

The dynamic modification system proposed is based on the object model to structure encapsulated components and uses computational reflection to host fault-tolerant techniques. Those techniques contribute to assure the reliability of the new software version.

1 Introdução

Depois que um sistema entrou em regime de operação normal e precisa sofrer uma alteração, a abordagem mais comum é descontinuar o programa que está em execução e então substituí-lo pela nova versão. O sistema já alterado é então reinicializado, passando a estar novamente disponível ao usuário. Esta interrupção ocasiona a indisponibilidade do serviço aos usuários do software, e, também, pode implicar em diminuição de confiabilidade, pela inclusão de novo código na aplicação.

Constata-se que há necessidade de novas técnicas para manutenção que não interrompam a operação do sistema por longos períodos. Uma destas técnicas é utilizar um sistema que faz a substituição dinâmica da versão de um programa, eliminando a etapa de desativação do sistema. A substituição é facilitada quando foram previstos, ainda na fase de projeto, os tipos de alterações que o software poderá sofrer durante a sua fase operacional.

A antecipação permite construir componentes de software que isolem os aspectos sujeitos a alterações, de forma a facilitar a sua substituição. Para atender a este aspecto, o modelo de objetos é particularmente adequado: permite confinar, encapsular, esconder e proteger estruturas de dados e correspondente código de manipulação, oferecendo seus serviços por meio de interfaces bem definidas e estáveis. A estabilidade das interfaces pode ser assegurada mesmo que sejam feitas alterações em qualquer das propriedades – dados ou código – do componente.

A evolução de um software é o tema central deste trabalho, cuja preocupações são a continuidade da disponibilidade dos serviços durante a transição de uma versão de software, que se encontra em operação, e a preservação da confiabilidade, na versão mais atualizada. Técnicas de tolerância a falhas são usadas para garantir a validação da substituição, ou seja, garantir que o novo componente é funcionalmente equivalente ao componente que foi substituído. O modelo de objetos é adotado para a construção de componentes encapsulados e técnicas de reflexão computacional são usadas para separar, em um meta-programa, as atividades relacionadas ao processo de substituição e validação [LIS97].

2 Validação e Testabilidade

A validação da substituição é o elemento inovador mais importante desta proposta, e exerce um papel similar ao da testabilidade de um programa. Os projetos pesquisados na literatura [FRZ97], [GUP93], [GUP96], [SEG93] abordam principalmente os aspectos ligados à carga e substituição dinâmica de módulos em si, mas não apresentam soluções para a detecção e recuperação de falhas durante o processo de substituição.

Os mecanismos de introspecção da reflexão computacional são bastante adequados para dar suporte à validação dos componentes. A abordagem mais simples é aplicar o teste da caixa preta sobre o novo componente durante a validação. Isto é feito através da comparação entre os resultados fornecidos pela versão antiga que executa juntamente com a versão nova. Caso os resultados sejam divergentes é assumido que a nova versão apresenta falha.

2.1 Técnicas de Tolerância a Falhas

As técnicas de tolerância a falhas - TF agregam significativo conhecimento sobre gerenciamento de componentes. Através destas técnicas o comportamento dos componentes envolvidos no processo de substituição pode ser monitorado: o objeto antigo que tem um

funcionamento correto e bem conhecido pelo sistema e o objeto novo que pode apresentar algum comportamento inesperado como reação a alguma mensagem.

A técnica de TF que está presente de forma mais marcante é blocos de recuperação [RAN78] pois ela resume a idéia central da abordagem proposta. No caso de substituição de versões de classes, é usada uma adaptação da tradicional técnica de blocos de recuperação. Esta adaptação dispensa o uso de um teste de aceitação e emprega apenas duas alternativas: (a)a alternativa primária é a versão antiga, representada pelo objeto O1 e a alternativa secundária é versão nova, representada pelo objeto O2. A versão O1 é considerada correta e é usada como teste de aceitação, pois seus resultados são usados para fins de observação do comportamento da nova versão. Após a execução da versão primária, a outra alternativa é executada e seus resultados comparados com os fornecidos pela versão primária. Cada vez que a nova versão O2 apresentar um funcionamento ou resultado incorreto, é fornecido para o cliente o resultado da versão antiga.

Considerando a hipótese de a substituição estar sendo realizada para corrigir alguma falha na versão antiga, O1, existe a possibilidade de ser fornecido um resultado incorreto, gerado de forma falha pela versão O1 e que divergiu do resultado fornecido pela nova versão O2 (possivelmente correto). Para detectar este tipo de ocorrência, é gravado um arquivo contendo todos os resultados gerados por ambas as versões, durante todo o processo de transição.

3 O Processo de Substituição

Um computador servidor executa a versão antiga do objeto O1 que realiza as suas funções de serviço, supostamente de forma correta e confiável. Quer se substituir dinamicamente este objeto O1 pelo objeto O2. Temporariamente, durante o processo de validação, vai de fato ser executado um programa de validação "PV" que contém simultaneamente ambas as versões O1 e O2 e que vai passar a gerenciar estes dois objetos, além de executar as tarefas de validação, como esquematizado na Figura 1.

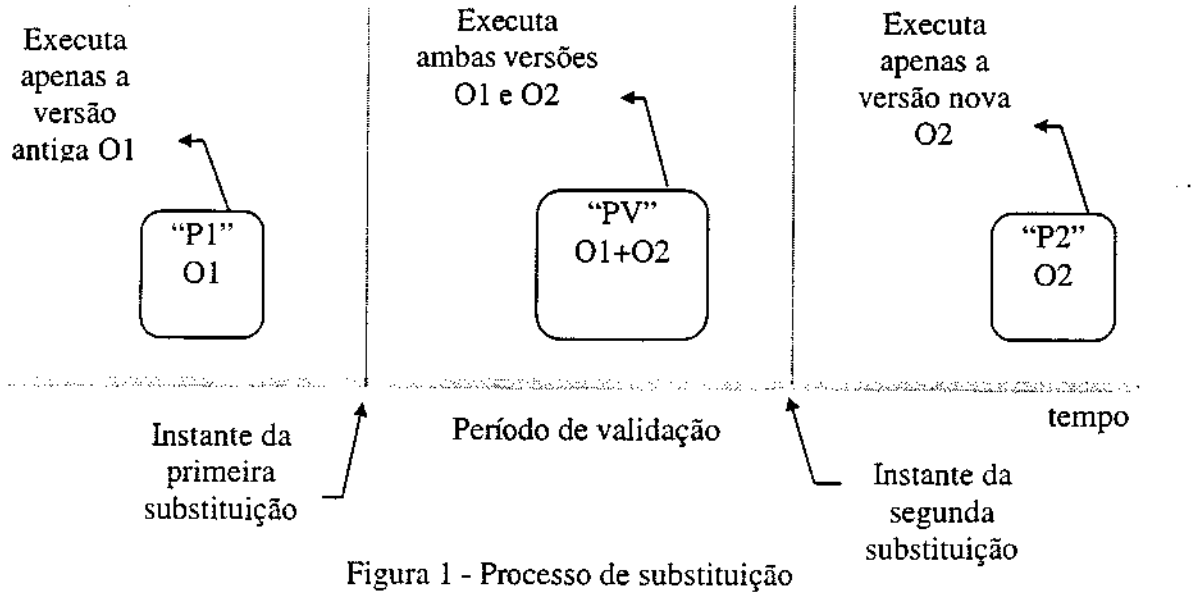

Figura 1 - Processo de substituição

3.1 Salvamento do Estado da Computação

Uma substituição no modelo de objetos apresenta possibilidade de realizar trocas no código e também nos tipos de dados, que são as propriedades definidas em cada classe. Por outro lado, um objeto possui um estado interno [BOO94] que se modifica a cada ativação. Assim, deve haver preocupação com a preservação do estado do objeto, cujos valores das variáveis instanciadas pelo objeto antigo precisam ser mapeados para os valores correspondentes no novo domínio , principalmente quando o novo objeto exigir uma reestruturação de dados, por ser oriundo de outra classe. Este estado interno deve ser salvo antes do cancelamento do objeto primário O1 e restaurado quando o objeto voltar à execução, agora instanciado dentro do programa de validação PV, que passará a gerenciar também o objeto que deverá substituir O1. O salvamento de estado de um objeto neste contexto resume-se a manter uma cópia dos dados de instância; a posterior recuperação do objeto original é feita por simples atribuição. Já a transferência de estado para outro objeto pode envolver operações de conversão, caso as estruturas de dados não sejam idênticas às do objeto que originou o salvamento de estado.

3.2 Pontos de Recuperação

Um dos serviços básicos de técnicas de TF é restaurar o sistema a um estado consistente [JAL94]. Baseada nesta afirmação está a estratégia de recuperação adotada neste modelo.

O programa PV contém meta-objetos associados a O1 e a O2 e que fazem o papel de árbitro da substituição. O meta-objeto (MO) associado a O2 toma conhecimento da falha de O2 através da comparação dos seus resultados com os resultados corretos fornecidos por O1. O meta-objeto assume as funções de TF, fazendo a detecção de falhas, o confinamento de danos, registro de erros num arquivo de log, e envia uma mensagem para O1 transferir o seu estado para que O2 consiga restaurar-se e novamente apresentar um estado correto e consistente, podendo retornar à operação normal e ser submetido a novas ativações e continuar sendo validado sob novas circunstâncias. Assim o estado incorreto de O2 é substituído pelo estado correto de O1. A Figura 2 ilustra a seqüência de operações de salvamento e restauração de estados.

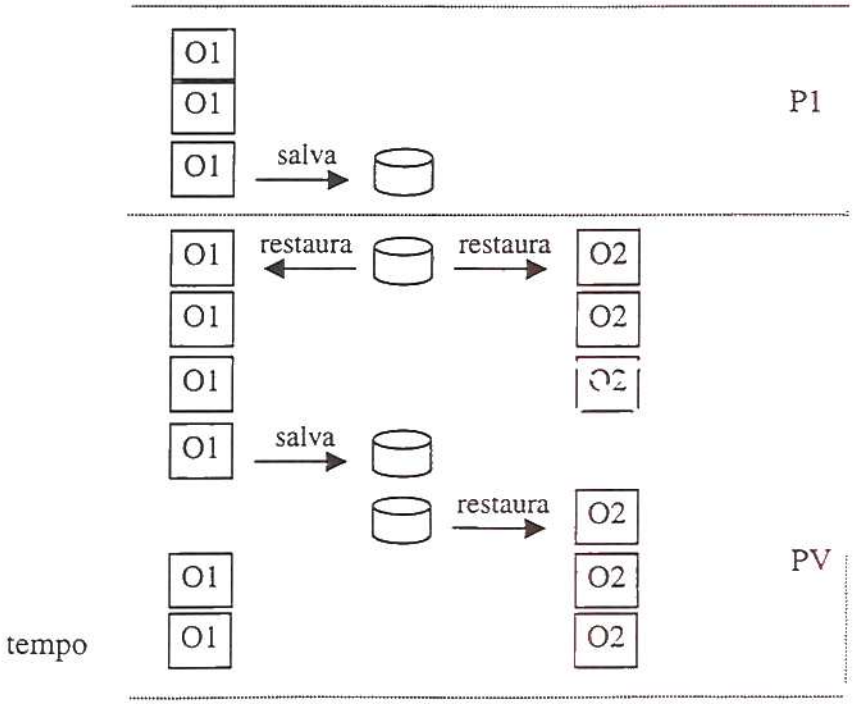

Figura 2 - Pontos de salvamento e restauração

4 Estudo de Caso

O cenário básico escolhido para desenvolver a aplicação com a qual foram feitos os testes de substituição de versão, comumente aparece na literatura e compreende um sistema de tempo-real adaptado de [BUR96] Consiste de um modelo de um sistema de drenagem para uma mina, que controla o nível de água de um reservatório através de um sensor. Uma bomba faz a coleta de água de um recipiente no fundo de um poço e a transporta para a superfície. O sistema possui as características tipicamente embutidas nos sistemas de tempo-real e pode ser implantado em uma arquitetura distribuída ou centralizada. Para este estudo de caso, optou-se pela substituição do objeto controlador do sensor. O sensor controlado pelo objeto O1 deve ser substituído por um novo sensor físico, controlado pelo sensor abstrato representado pelo objeto O2.

A linguagem de programação utilizada para a implementação foi Java e o modelo de reflexão empregada foi inspirado no protocolo de MetaJava [GOL97], visto que o protocolo de reflexão do ambiente JDK1.1® á basicamente introspectivo, não oferecendo muitas facilidades para intervenção no estado da computação. Java foi selecionada como linguagem de implementação por permitir carga de classes durante o processo de execução e possuir facilidades para programação concorrente e distribuída.

A aplicação é do tipo duas camadas ('two tiers'). O cliente e o servidor comunicam-se através de 'sockets', usando as classes Socket e ServerSocket importadas da biblioteca java.net. O 'socket' faz uma ligação virtual entre o servidor e o cliente. Ao ser efetuada uma substituição do objeto servidor antigo pelo novo, a conexão entre o cliente e o objeto servidor antigo é temporariamente desfeita e em seguida restabelecida pelo objeto que executa a nova versão. O programa cliente não é interrompido durante este processo de substituição.

Neste trabalho, para simular a ocorrência de falhas nas versões candidatas à substituição, foi usada a injeção de falhas por software, através de alterações no conteúdo de registros e variáveis. Para testar a implementação foram simuladas as seguintes situações de falha: (a) falha na leitura fornecida pelo sensor O2; (b) falha no algoritmo que realiza o cálculo da média das leituras.

Os testes realizados mostraram a viabilidade da solução proposta. O objeto servidor teve sua versão substituída sem descontinuar o cliente. As divergências entre os resultados fornecidos por O1 e O2 foram registradas no arquivo de log e o cliente sempre recebeu a resposta correta fornecida por O1, mesmo na presença de falhas em O2. Assim o objeto O2 foi testado e validado sem propagar erro para o sistema como um todo.

5 Conclusões

O objetivo deste trabalho foi de estudar e propor uma estratégia para substituição dinâmica de versões de componentes de software tolerante a falhas orientado a objetos. Este objetivo foi atingido e foi demonstrada a viabilidade de construção de um sistema com esta finalidade. O trabalho apresenta uma solução genérica para o problema da substituição sendo aplicável, em sua essência, a sistemas centralizados, distribuídos, ou tempo real, desde que o sistema seja concebido com as características propostas: adote o modelo de objetos, com componentes sujeitos a alterações estruturados na forma de classes e que o ambiente de execução ofereça a possibilidade de carga dinâmica de classes.

® Sun Microsystems

O problema apresentado é relevante visto que a manutenção de software é uma realidade e causa interrupções inconvenientes aos usuários dos sistemas. A solução proposta é inovadora no sentido de ser voltada para o modelo de objetos, utiliza técnicas de tolerância a falhas e adota uma arquitetura reflexiva. As técnicas de tolerância a falhas são usadas para validar a substituição, fazendo o monitoramento de ambas as versões através da comparação dos resultados por elas fornecidos.

O emprego da reflexão computacional permite separar o código das rotinas de substituição e validação dos demais componentes da aplicação, o que torna mais fácil compreender, manter e depurar o programa em si.

5 Referências Bibliográficas

[BOO94] BOOCH, G., "Object-Oriented Analysis and Design With Applications", The Benjamin/Cummings Publishing Co., USA, Second edition, 1994.

[BUR96] BURNS, Alan, and Wellings Andy, "Real-Time Systems and Programming Languages", Second Edition, Addison Wesley, 1996.

[FRZ97] FRANZ, Michael, "Dynamic Linking os Software Components", Computer, IEEE, pp. 74-81, March, 1997.

[GOL97] GOLM, Michael, "Design and Implementation of a Meta Architecture for Java", Institut für Mathematische Maschinen und Datenverarbeitung der Friedrich-Alexander-Universität, Erlangen-Nürnberg, Diplomarbeit, Jan, 1997.

[GUP93] GUPTA, Deepak; JALOTE, Pankaj; "Dynamic Software Version Change Using State Transfer Between Processes", Software Practice and Experience, Vol 23, pp. 949-964, September, 1993.

[GUP96] GUPTA, Deepak; JALOTE, Pankaj; and BARUA, Gautam; "A Formal Framework for Dynamic Software Version Change", IEEE Transactions on Software Engineering, Vol 22, No. 2, pp. 120-131, February, 1996.

[JAL94] JALOTE, Pankaj., "Fault-Tolerance in Distributed Systems", Prentice-Hall, New Jersey, 1994.

[LIS97] LISBÔA, M.L.; HAETINGER, W. "Troca Dinâmica de Componentes: problemas e soluções no modelo OO". Argentine Simposium on Object-Orientation, Buenos Aires, anais pp.67-75, setembro, 1997.

[RAN78] RANDELL, B.; LEE, P. A. e TRELEAVEN, P. C., "Reliability Issues in Computing System Design", ACM Computing Surveys, New York, v.10, n. 2, p. 123-166, Jun, 1978.

[SEG93] SEGAL, M., Mark E., Frieder O., "On-the-fly Program Modification: Systems for Dynamic Updating", IEEE Software, vol. 10, num. 3, pp. 53-65, March, 1993.