Tolerância a Faltas Adaptativa em um Modelo de Componentes

Fábio Favarim[1], Joni Fraga[1], Frank Siqueira[2]

[1]Departamento de Automação e Sistemas – DAS
[2]Departamento de Informática e Estatística – INE
Universidade Federal de Santa Catarina – UFSC
Florianópolis – SC – Brasil – CEP 88040-900
{fabio,fraga}@das.ufsc.br, frank@inf.ufsc.br

Abstract. *This paper presents a fault tolerant model based on components for building distributed applications. The TFA-CCM model allows that quality of service (QoS) requirements be used to select the configuration of replicated services during execution time, employing a set of components that deal with the non-functional aspects of the application. The characteristics of this model and its implementation are described along this paper.*

Resumo. *Este artigo apresenta um modelo de tolerância a faltas baseado em componentes para a construção de aplicações distribuídas. O modelo TFA-CCM permite que requisitos de qualidade de serviço (QoS) guiem a seleção da configuração de serviços replicados em tempo de execução, utilizando um conjunto de componentes que tratam dos aspectos não-funcionais da aplicação. As características deste modelo e a sua implementação são descritos ao longo deste artigo.*

1. Introdução

Sistemas de software atuais estão cada vez mais distribuídos e operam em ambientes dinâmicos como a Internet. Aplicações distribuídas com requisitos de tolerância a faltas são difíceis de serem construídas e mantidas, quando consideradas a complexidade e as características destes ambientes. A tolerância a faltas deve ser construída fazendo uso de suportes de *middleware* e fornecendo mecanismos que permitam a adaptação de técnicas de redundâncias às mudanças do sistema.

Modelos de desenvolvimento de software baseado em componentes distribuídos, tais como *Enterprise JavaBean*s (EJB), desenvolvido pela *Sun Microsystems* [Sun, 2001], Microsoft *.NET* [Microsoft, 2001] e *CORBA Component Model* (CCM) [OMG, 2002], estão sendo usados e tem mostrado melhora no processo de desenvolvimento e manutenção de software. Porém, tais tecnologias fornecem suporte limitado para tolerância a faltas, geralmente através de mecanismos de persistência de dados.

Neste trabalho é usada *tolerância a faltas adaptativa*, que permite que uma aplicação distribuída baseada em componentes tenha um comportamento esperado mesmo diante de mudanças no seu ambiente computacional, como falhas em componentes e em sítios (*hosts*). Diferentes níveis de confiabilidade e de disponibilidade podem ser fornecidos com diferentes configurações de replicações, alocando somente os recursos necessários para obter os requisitos desejados.

O modelo TFA-CCM - *Tolerância a Faltas Adaptativa no Modelo de Componentes CORBA*, fornece suporte a tolerância a faltas adaptativa totalmente transparente à aplicação sem a necessidade de mudanças no modelo de componentes e na sua implementação. O TFA-CCM é composto por componentes de software que são responsáveis

por implementar técnicas de tolerância a faltas, definindo e controlando o comportamento de um serviço replicado.

O modelo integra mecanismos para especificação de requisitos de QoS que guiam a seleção da configuração de serviços replicados. Deste modo, diferentes níveis de QoS podem ser especificados, visando atender diferentes requisitos de tolerância a faltas. O TFA-CCM tem mecanismos de detecção de falhas parciais que identificam a necessidade de adaptação e mecanismos para efetivar mudanças no sistema visando atender os requisitos de QoS.

Este artigo está organizado da seguinte forma: a seção 2 apresenta uma definição de componentes de software e descreve o modelo de componentes CORBA (CCM); a seção 3 faz uma pequena introdução sobre tolerância a faltas adaptativa. A seção 4 apresenta o TFA-CCM. Na seção 5 são apresentados os aspectos de implementação. A seção 6 discute trabalhos relacionados e na seção 7 o artigo é concluído.

2. Componentes de Software

A abordagem de programação baseada em *componentes de software* está fundamentada na composição de programas a partir de componentes pré-existentes. Segundo [Szypersky, 1998], "Componente é uma unidade de composição com interfaces contratualmente especificadas e apenas com dependências de contexto explícitas. Um componente pode ser instalado isoladamente e estar sujeito à composição por terceiros". O uso de componentes permite a implementação e a manutenção de programas distribuídos de maneira eficiente. Além disso, o custo de processo de desenvolvimento pode ser reduzido de forma significativa devido ao reuso de código.

A especificação de um componente é normalmente publicada separadamente de seu código fonte por meio da especificação de suas interfaces, que são os pontos de acesso aos serviços do componente. Interfaces separam a especificação dos componentes da sua implementação, não permitindo que os usuários conheçam os detalhes de implementação do componente.

2.1 CORBA *Component Model* (CCM)

O conceito de componentes CCM tem como base a extensão do modelo de objetos distribuídos do CORBA. As fases de modelagem, programação, empacotamento, implantação e execução de componentes, previstas nas especificações do CCM, organizam objetos em componentes, aportando à programação distribuída do CORBA todas as características atribuídas anteriormente à programação por componentes. As interfaces dos componentes são especificadas em CORBA IDL (*Interface Description Language*), que a partir da versão 3.0 passou a permitir a definição de componentes.

Componentes CORBA possuem atributos e portas de comunicação. Atributos são propriedades configuráveis do componente e têm como propósito à configuração do componente. As portas de comunicação são pontos de conexão entre os componentes. São definidos quatro tipos de portas [OMG, 2002], ilustradas na Figura 1.

- Facetas (*Facets*): são as interfaces IDL através das quais um componente oferece seus serviços aos seus clientes.
- Receptáculos (*Receptacles*): permitem ao componente invocar os serviços acessíveis através da interface IDL de outros componentes.
- Produtores de Eventos (*Event Sources*): são interfaces que emitem eventos de

um tipo específico para um ou mais consumidores de eventos.

- Consumidores de Eventos (*Event Sinks*): são as interfaces através das quais um componente é notificado da ocorrência de eventos de um determinado tipo.

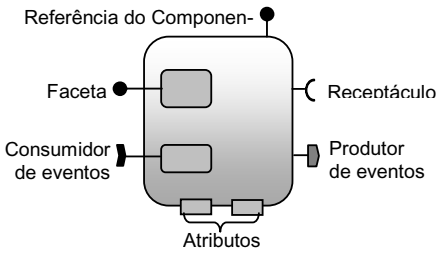


Figura 1 – Componente CORBA

Os componentes CORBA são executados em *containers*. O *container* é o responsável por fornecer um ambiente de execução para um componente, permitindo acesso transparente a serviços comuns do CORBA (aspectos não funcionais). Em outras palavras, o *container*, torna possível separar os aspectos funcionais do componente (implementação da funcionalidade do componente) dos aspectos não-funcionais (interação com o suporte e seus serviços). Com isto, a programação de aplicações se torna mais simples, pois o desenvolvedor se preocupa apenas com a parte funcional da aplicação.

Componentes são empacotados para que sejam posteriormente distribuídos e implantados. Um pacote agrupa uma ou mais implementações de um componente e um conjunto de descritores. Os descritores são escritos em XML [W3C, 1998], e especificam os arquivos binários necessários para implantar os componentes nas diferentes plataformas, assim como os serviços CORBA utilizados pelos componentes e as características de cada componente, como a definição das portas.

3. Tolerância a Faltas Adaptativa

Segundo Kim e Lawrance [1990], tolerância a faltas adaptativa é conseguida com mecanismos que satisfaçam requisitos de tolerância a faltas variáveis dinamicamente através da utilização eficiente (e adaptativa) de uma quantidade limitada e variável de recursos de processamento redundantes. Pode-se definir tolerância a faltas adaptativa como a propriedade que permite um sistema mudar a tolerância a faltas do sistema através da adaptação a mudanças no ambiente computacional ou nas políticas de tolerância a faltas.

Se considerarmos as características dinâmicas dos sistemas distribuídos atuais que ganham em escala dia a dia, a noção fixa e pré-estabelecida na coordenação de modelos de redundâncias os torna ineficientes. Sistemas distribuídos em larga escala, como os construídos com os recursos da Internet, podem se beneficiar de políticas adaptativas. Mecanismos de *tolerância a faltas adaptativa* fazem uso de políticas adaptativas na gerência de redundâncias, de maneira a se manter eficaz mesmo diante de mudanças freqüentes do seu ambiente computacional. A tolerância a faltas adaptativa pode envolver a troca de algoritmos, em tempo de execução, para atender às mudanças do ambiente [Chen et al., 2001].

4.O Modelo TFA-CCM

O TFA-CCM tira proveito da flexibilidade propiciada pela programação baseada na tecnologia de componentes. Neste projeto escolhemos o uso do CCM devido à sua rica semântica de comunicação (vários tipos de portas) e à possibilidade de usá-lo em ambi-

entes heterogêneos, nos quais diferentes linguagens, *middleware* e sistemas operacionais diferentes podem coexistir.

O TFA-CCM permite ao usuário de uma aplicação especificar requisitos de qualidade de serviço (QoS), definindo níveis desejados de disponibilidade e de desempenho do serviço, e o suporte de execução define uma configuração e emprega uma técnica de replicação de modo que os requisitos especificados sejam atendidos. O suporte monitora a ocorrência de falhas na aplicação e adapta seu comportamento de modo a manter o nível de QoS especificado pelo usuário.

A Figura 2 ilustra os diferentes componentes que compõem o TFA-CCM, em uma possível configuração. Essencialmente, estes componentes não-funcionais fazem a configuração e o monitoramento da aplicação, replicando componentes e usando uma técnica de replicação, implementada pelo *coordenador de replicação* de modo a alcançar os níveis de confiabilidade desejados. No exemplo mostrado na Figura 2, o componente localizado no sítio 2 (o servidor primário) é replicado no sítio 3 (servidor *backup*), e o estado dessas réplicas é sincronizado através dos coordenadores de replicação usando a técnica de replicação passiva[1]. A técnica de replicação pode ser facilmente trocada em tempo de execução através da substituição do coordenador de replicação por outro componente que implementa uma técnica de replicação diferente.

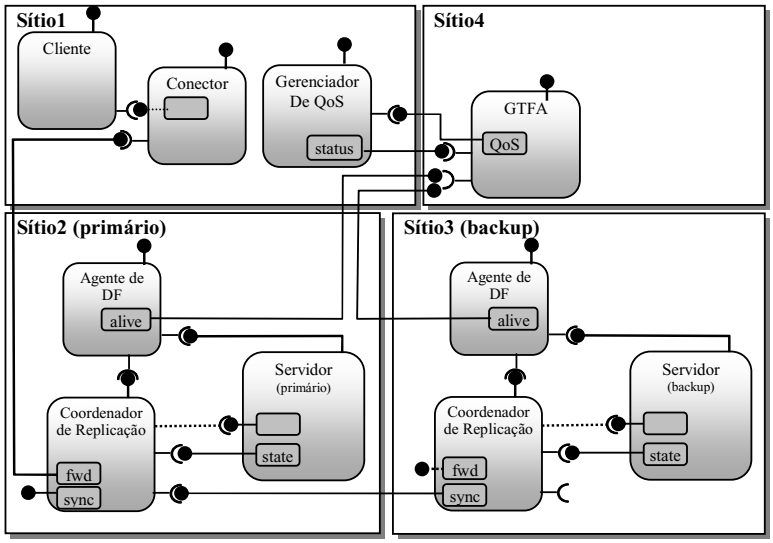


Figura 2 - Visão geral do TFA-CCM

O cliente obtém acesso aos serviços do servidor através do componente *conector*, que tem a função de tornar transparente para o cliente as possíveis mudanças de configuração da aplicação. Por exemplo, a troca de réplicas, onde a réplica *backup* substituiria a primária na configuração, não afeta o comportamento do cliente. Na falha da réplica primária, o conector é redirecionado para a réplica *backup* do servidor.

Para cada componente com requisitos de tolerância a faltas existe um *gerenciador de tolerância a faltas adaptativa* (GTFA), que define a configuração atual baseado nos requisitos de QoS exigidos pela aplicação e na freqüência de falhas parciais nos componentes do sistema. A reconfiguração da aplicação é iniciada quando dados moni-

[1] Na replicação passiva [Powell, 1991], após um envio de resultados ao cliente, a réplica primária deve sincronizar as secundárias (réplicas *backups*) ao seu estado de processamento, enviando uma cópia de seu estado às mesmas.

torados pelo GTFA mostram que a configuração atual não é mais apropriada para manter os níveis de QoS especificados, o que pode ocorrer devido à falta ou excesso de recursos (réplicas) ou pelo uso de uma técnica de replicação inadequada.

O monitoramento das falhas é feito através de *agentes de detecção de falhas* (agentes de DF, na Figura 2**)**. Cada réplica do componente é associada em sua máquina com um agente de DF, que é o responsável pelo envio dos dados de monitoramento ao GTFA. No modelo são previstos dois níveis de detecção de falhas: nível de sítio – no caso do agente de DF parar de responder – e nível de componente – quando o componente para de responder as chamadas do agente de DF.

O *gerenciador de QoS* permite que o usuário especifique seus requisitos de QoS. Através do gerenciador de QoS, diferentes níveis de QoS relacionados a tolerância a faltas podem ser especificados. Esses requisitos de QoS são repassados ao GTFA, que os interpreta e define uma configuração adequada para a aplicação, como o número de réplicas, a técnica de replicação a ser utilizada, entre outros. O GFTA tenta com os recursos disponíveis manter o nível de QoS requisitado. O GFTA volta a atuar na configuração do sistema sempre que uma mudança no sistema leva ao não-atendimento do nível selecionando. Os requisitos podem ser alterados a qualquer momento, em tempo de execução, através do gerenciador de QoS. Qualquer mudança nos requisitos de QoS é informada ao GTFA para que este se encarregue de implementar as mudanças necessárias de configuração. O gerenciador de QoS também é usado pelo GTFA para passar informações de configuração ao usuário, como por exemplo a técnica de replicação utilizada, a localização dos componentes replicados, o número de réplicas sendo utilizadas pelo TFA-CCM e estatísticas sobre a freqüência de falhas na aplicação.

5. Implementação do Modelo TFA-CCM

A implementação dos componentes do TFA-CCM tem como plataforma de desenvolvimento o OpenCCM versão 0.2 [Marvie et al., 2002], que é uma implementação parcial da especificação do CCM. O OpenCCM é totalmente desenvolvido em Java e pode ser usado com três diferentes ORBs. O ORB usado neste trabalho foi o ORBacus, versão 4.0.5 [Iona Technologies, 2002]. O OpenCCM foi escolhido por ter sido a primeira implementação de código aberto disponível da especificação CCM.

A implementação do TFA-CCM fornece três diferentes coordenadores de replicação: um primeiro que implementa a técnica de replicação passiva, um segundo que implementa a técnica de replicação semi-ativa[2] e um terceiro que é um coordenador "vazio", o qual não implementa nenhuma técnica de replicação. Este último é usado quando se deseja apenas requisitos de disponibilidade, onde informações de estado não sejam necessárias. A técnica de replicação ativa não foi implementada, pois requer mecanismos de comunicação de grupo, que não são fornecidos pelo ORB utilizado.

O componente coordenador de replicação tem duas facetas: `sync` e `fwd`. A faceta `sync` é usada pelos coordenadores de replicação passiva e semi-ativa para sincronização do estado das réplicas. A faceta `fwd` é implementada por todos os coordenadores de replicação, para que as requisições dos clientes sejam repassadas do conector para o coordenador de replicação. Além desse uso, a faceta `fwd` é usada pelo coordenador que im-

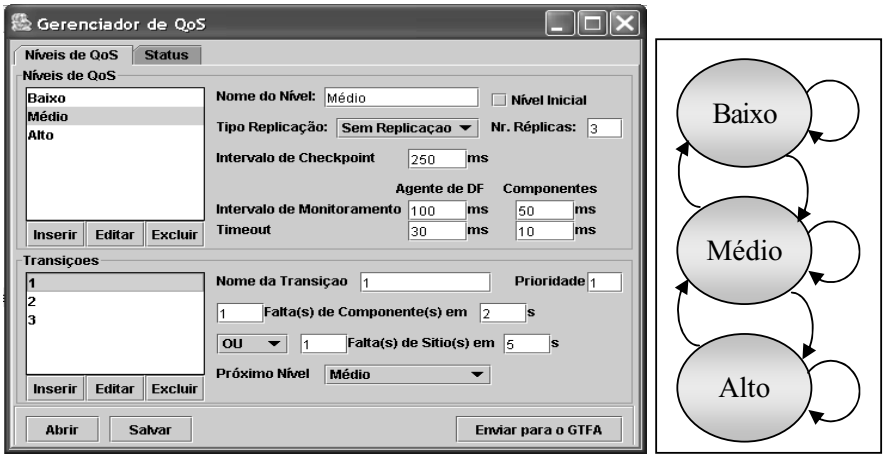
[2] Na replicação semi-ativa (ou replicação *leader/followers*) [Powell, 1991] as requisições são difundidas entre todas as réplicas (seguidoras) pela réplica líder (primária), mas somente a líder envia os resultados ao cliente.

plementa a técnica de replicação semi-ativa, para repassar as requisições recebidas dos clientes para as réplicas seguidoras. Como o coordenador não sabe antecipadamente qual a interface IDL (porta de comunicação) ao qual está associado, é usado o mecanismo de invocação dinâmica do CORBA (DII), o qual permite descobrir em tempo de execução como fazer chamadas aos métodos daquela interface.

De modo a permitir a atualização do estado de suas réplicas, o componente deve fornecer os métodos para acesso ao seu estado. Portanto, estes componentes devem implementar a faceta `state`, que provê os métodos para recuperação (`get_state`) e atualização (`set_state`) do estado do componente.

Para fazer a conexão entre componentes é necessário que as portas de interconexão sejam do mesmo tipo. De modo a tornar o conector genérico (independente do tipo de porta de comunicação), foi utilizada a interface de esqueleto dinâmico (DSI) do CORBA. Esta invocação dinâmica está representada na Figura 2 por uma linha tracejada. Quando uma requisição chega ao conector, este simplesmente a repassa para o coordenador de replicação primário através de uma conexão com a faceta `fwd` deste.

O GTFA armazena em uma estrutura de dados a visão global de todos os componentes implantados no sistema. Essa estrutura contém a referência de todos os componentes, da localização de cada componente, o tipo de replicação que está sendo utilizado, entre outras informações. Estes dados permitem que as reconfigurações necessárias no sistema sejam efetuadas, como por exemplo interligar as portas dos componentes, remover componentes de algum sítio, etc. A falha do GTFA e a conseqüente perda destes dados compromete todo o funcionamento do sistema. Para evitar isso, as informações de estado do GTFA são gravadas em um meio de armazenamento persistente.



(a) (b)

Figura 3 – Interface gráfica do Gerenciador de QoS (a) e os níveis de QoS (b)

O monitoramento de componentes é realizado periodicamente pelo GTFA em intervalos definidos pelo usuário através do gerenciador de QoS. O GTFA invoca periodicamente o método `is_alive` do agente de DF, o qual responde enviando o estado atual dos componentes que ele monitora. Se o agente de DF não responde, uma falha de sítio é assumida e a aplicação pode ser reconfigurada pelo GTFA. Falhas em componentes são detectadas pelo agente de DF através de chamadas ao método `_non_existent` (da classe `CORBA::Object`, que é implicitamente herdada por todos os componentes) e

informadas ao GTFA na próxima chamada do método `is_alive` do agente de DF. A reconfiguração da aplicação também pode ser realizada neste caso, dependendo dos níveis de QoS definidos pelo usuário através da interface gráfica do gerenciador de QoS, mostrada na Figura 3a. Através do uso desta interface, o usuário pode definir os níveis de QoS e as condições que fazem com que as transições entre níveis aconteçam, formando assim, um máquina de estado como mostrado na Figura 3b.

6. Trabalhos Relacionados

A arquitetura AQuA (*Adaptive Quality of Service Availability*) [Cukier et al., 1998] visa fornecer tolerância a faltas adaptativa para aplicações distribuídas. AQuA permite que os programadores de aplicações especifiquem os níveis de confiabilidade desejados, que são alcançados através da configuração do sistema em função da disponibilidade de recursos e de acordo com as falhas ocorridas. AQuA utiliza o QuO [Zinky, 1997] para especificar os requisitos de QoS em nível de aplicação, o gerenciador de confiabilidade do Proteus [Sabnis et al. 1998] para configurar o sistema em resposta a falhas e aos requisitos de disponibilidade. O Ensemble [Hayden 1998] é usado no AQuA para fornecer serviços de comunicação de grupo. Apesar de possuírem mecanismos de especificação de QoS com funcionalidade equivalente, enquanto QuO e AQuA exigem que os requisitos de QoS sejam definidos em tempo de projeto, o TFA-CCM permite que os requisitos de QoS sejam modificados em tempo de execução, tirando proveito da flexibilidade propiciada pela utilização de componentes de software.

Chamaleon [Bagchi et al., 1998] é uma infra-estrutura adaptativa, que permite que diferentes níveis de requisitos de disponibilidade sejam fornecidos através de ARMORs – objetos móveis adaptativos e reconfiguráveis que visam prover os requisitos exigidos pela aplicação. O *Chamaleon* pode usar seletivamente combinações de ARMORs a fim de prover diferentes níveis de disponibilidade, que podem ser introduzidos de maneira incremental no sistema. Apesar de empregar mecanismos de composição semelhantes aos existentes em modelos de componentes, o Chamaleon não segue um modelo de componentes padrão como o CCM ou o EJB.

Em [Marangozova e Hagimont, 2002] é apresentada uma abordagem para replicação de componentes CORBA. Nessa abordagem são usados objetos de interceptação, responsáveis por capturar as requisições feitas para o componente de modo a disparar as ações necessárias para o gerenciamento da replicação. Esses objetos de interceptação possuem a mesma interface do componente que será replicado, o que implica que toda a vez que se desejar replicar uma nova aplicação é necessário implementar um novo objeto de interceptação com a mesma interface do componente de aplicação. No modelo TFA-CCM, é empregado um conector genérico, que independe da interface do componente replicado e não precisa ser modificado para ser utilizado em diferentes aplicações.

7. Conclusão

Este artigo apresenta uma visão geral do modelo TFA-CCM, o qual oferece mecanismos flexíveis para a construção de software baseado em componentes CORBA com requisitos de tolerância a faltas. O modelo TFA-CCM adapta a configuração do sistema levando em conta os requisitos de QoS associados ao componente e as faltas que ocorrem no sistema. Qualquer componente CORBA pode fazer uso do TFA-CCM para oferecer requisitos de tolerância a faltas.

De modo a prover a tolerância a faltas adaptativa, o TFA-CCM foi construído a

partir de um conjunto de componentes de software, que combinados fornecem requisitos de tolerância a faltas para aplicações baseadas em componentes. Apesar de técnicas adaptativas também serem empregadas em vários outros trabalhos para prover requisitos de tolerância a faltas, estes trabalhos não tiram proveito da flexibilidade provida pela utilização de componentes de software.

Pretende-se aprimorar a implementação do TFA-CCM, com a sua evolução para novas implementações da especificação do CCM. Outra possibilidade é fornecer o suporte de tolerância a faltas adaptativa através do *container*, uma vez que este é responsável por fornecer acesso aos serviços não funcionais utilizados por um componente. Além disso, pretendemos avaliar o TFA-CCM através do seu uso em aplicações com requisitos de tolerância a faltas.

Referências Bibliográficas

Bagchi, S. et al. (1998) The Chameleon Infrastructure for Adaptive, Software Implemented Fault Tolerance. In: *17th IEEE Symposium on Reliable Distributed Systems*. p. 261–267, West Lafayette, Indiana. IEEE Computer Society.

Chen, W. K., Hiltunen, M. A. e Schlichting, R. D. (2001). Constructing Adaptive Software in Distributed Systems. In *21st International Conference on Distributed Computing Systems*. p. 635-643. Phoenix. AZ. IEEE Computer Society.

Cukier, M. et al. (1998). AQuA: An Adaptive Architecture that Provides Dependable Distributed Objects. In *17th IEEE Symposium on Reliable Distributed Systems*. p. 245-253, West Lafayette, Indiana. IEEE Computer Society.

Hayden, M. G. (1998). *The Ensemble System*. PhD thesis, Cornell University.

Marangozova, V. e Hagimont, D. (2002). An Infrastructure for CORBA Component Replication. In 1st *IFIP/ACM Working Conference on Component Deployment*. p.222-232, Berlin, Alemanha.

Marvie, R., Merle, P., e Vadet, M. (2002). *The OpenCCM Plataform*. http://corbaweb.lifl.fr/OpenCCM/

Iona Technologies (2001). *ORBacus for C++ and Java,* version 4.0.5.

Kim, K.H., Lawrence, T.(1990).Adaptive Fault Tolerance: Issues and Approaches. In *2nd IEEE Workshop on Future Trends of Distributed Computing Systems.* p. 38-46, Cairo, Egypt. IEEE Computer Society.

Microsoft (2001). *Overview of the .NET Framework*. MSDN Library White Paper.

OMG (2002). *CORBA Components*. OMG Document formal/02-06-65.

Powell, D. (1991). *Delta-4 Architecture Guide*. Esprit II P2252, Delta-4 Phase 3.

Sabnis, C. et al. (1998). Proteus: A Flexible Infrastructure to Implement Adaptive Fault Tolerance in AQuA. In *7th IFIP International Working Conference on Dependable Computing for Critical Applications*. p. 137–156. San Jose, CA, USA

Sun Microsystems, v. (2001). *Enterprise JavaBeans Specification*. v2.0.

Szyperski, C. (1998). *Component Software: Beyond Object-Oriented Programming*. ACM Press/Addison-Wesley Publishing Co.

W3C (1998). *eXtensible Markup Language (XML)* v1.0. World Wide Web Consortium.

Zinky, J. A., Bakken, D. E. e Schantz, R. E. (1997). Architectural Support for Quality of Service for CORBA Objects. *Theory e Practice of Object Systems*, v.3, n.1, p.53–73.