Suporte de Tolerância a Faltas Adaptativa para Aplicações Desenvolvidas em CCM*

Fábio Favarim[1†], Joni Fraga[1‡], Lau Cheuk Lung[2], Frank Siqueira[3]

[1]Departamento de Automação e Sistemas – DAS
[3]Departamento de Informática e Estatística – INE
Universidade Federal de Santa Catarina – UFSC
Florianópolis – SC – Brasil – CEP 88040-900
[2]Programa de Pós-Graduação em Informática Aplicada – PPGIA
Pontifícia Universidade Católica do Paraná – PUCPR
{fabio,fraga}@das.ufsc.br, lau@ppgia.pucpr.br, frank@inf.ufsc.br

Resumo. *Este artigo propõe um suporte para desenvolvimento de aplicações baseadas em componentes de software CORBA (CCM) com requisitos de tolerância a faltas. O modelo TFA-CCM permite que requisitos de QoS norteiem 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 os resultados obtidos com a sua implementação são descritos ao longo deste artigo.*

Abstract. *This paper proposes a support for building distributed applications with fault-tolerance requirements by using software components. The TFA-CCM model allows that QoS requirements be used to select the configuration of replicated services at execution time, employing a set of components that deal with the non-functional aspects of the application. The characteristics of this model and the results obtained with its implementation are described along this paper.*

1. Introdução

Os sistemas de software atuais estão cada vez mais distribuídos, operando em ambientes dinâmicos e sujeitos a falhas, como a rede mundial – a Internet. Aplicações distribuídas com requisitos de tolerância a faltas são difíceis de serem construídas e mantidas, se consideramos a complexidade e as características de ambientes de larga escala. Modelos adaptativos de processamento têm se mostrado bastante adequados em sistemas distribuídos de larga escala quando o objetivo é manter requisitos de qualidade de serviço. Se o requisito é a tolerância a faltas, o uso de técnicas adaptativas pode também se mostrar interessante em sistemas onde as condições de carga e a própria evolução das aplicações não são de todo previsíveis. A tolerância a faltas adaptativa é recente e o seu uso pode trazer benefícios na gerência dos recursos computacionais envolvidos [KIM, 1990; CHANG, 1998; CHEN AT AL., 2001].

Um dos objetivos do emprego de técnicas adaptativas na tolerância a faltas é tentar fazer com que os serviços de um sistema sejam mantidos em um nível de confiabilidade requisitado pela aplicação. A configuração dinâmica pode ser o instrumento provedor de tolerância a faltas adaptativa, ou seja, trocas de configuração de uma aplicação distribuída fariam face a variações no seu ambiente de execução ou a mudanças nas políticas de confiabilidade. O uso destas técnicas permite, por exemplo, que em uma replicação de software se possa adicionar ou remover réplicas de um serviço de acordo com as variações do seu ambiente de execução, alcançando assim a otimização no uso dos recursos do sistema.

O paradigma de programação baseada em componentes de software, fundamentado na composição de programas a partir de componentes pré-existentes, sempre foi apontado como

* Parcialmente financiado pelo Projeto CNPQ CT-INFO Nº 401802/2003-5
† Parcialmente financiado pelo CNPQ através do processo Nº 140848/2003-7
‡ Parcialmente financiado pelo CNPQ através do processo Nº 51948/02-7

base de ferramentas automatizadas ou técnicas de sistematização da produção de software. Várias das características apontadas em outros estilos de programação podem ser encontradas nesta abordagem [SZYPERSKI,1998]. Entre estas, uma das mais importantes é a flexibilidade. A utilização de elementos auto-contidos e a interação através de interfaces bem definidas fazem de componentes unidades de configuração que permitem a implementação e a manutenção de programas em sistemas distribuídos de maneira muito eficiente.

Nos últimos anos tem sido grande a oferta de ferramentas e de plataformas de *middleware* próprias para o suporte de aplicações distribuídas. Estes suportes de *middleware* podem ser usados no suporte a tolerância a faltas, fornecendo mecanismos que permitam, por exemplo a adaptação de técnicas de redundâncias às mudanças do ambiente de execução da aplicação. Além disso, o conceito de componentes vem sendo integrado a várias tecnologias de *middleware* existentes. Exemplos desses esforços são: CCM (CORBA *Component Model*) [OMG, 2002b], parte integrante das especificações CORBA 3.0 da OMG [OMG, 2002a]; EJB (*Enterprise JavaBeans*) desenvolvido pela Sun [SUN, 2001]; e o .NET, da Microsoft [MICROSOFT, 2000]. Estas tecnologias fornecem um suporte limitado para tolerância a faltas, geralmente na forma de mecanismos de persistência de dados.

Na literatura são mencionadas várias experiências com o uso de componentes em aplicações distribuídas com requisitos de tolerância a faltas [BATISTA E CARVALHO, 2002; KRAMER, 1990; LOQUES ET AL., 1999; MAGEE E KRAMER, 1997]. A idéia que orienta estas experiências é justamente a flexibilidade que esta abordagem de programação oferece. Uma das possibilidades é o uso de mecanismos de configuração dinâmica para isolar elementos faltosos de um programa distribuído.

O presente artigo apresenta uma proposta de utilização de técnicas adaptativas de tolerância a faltas tomando como base a tecnologia de componentes. O modelo proposto, denominado TFA-CCM – Tolerância a Faltas Adaptativa através do uso de Componentes CORBA – implementa um suporte de tolerância a faltas adaptativa totalmente transparente à aplicação [FAVARIM, 2003]. 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 proposto apresenta soluções práticas para integrar requisitos de qualidade de serviço (QoS) que devem nortear 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 possui mecanismos capazes de reconhecer a necessidade de reconfigurar e de efetivar mudanças no sistema visando atender os requisitos de QoS, sem que aspectos como desempenho e estabilidade sejam duramente comprometidos. Ou seja, a configuração dinâmica no TFA-CCM faz com que um nível de confiabilidade ou de disponibilidade requisitado pela aplicação possa ser mantido com diferentes configurações de replicações, alocando somente os recursos necessários para obter os requisitos desejados.

Este artigo está organizado da seguinte forma: a seção 2 faz uma pequena introdução sobre tolerância a faltas adaptativa; a seção 3 apresenta uma visão geral sobre componentes de software e descreve o modelo de componentes CORBA (CCM). A seção 4 apresenta o modelo proposto para tolerância a faltas adaptativa usando componentes de software. Na seção 5 são apresentados alguns aspectos de implementação, e medidas de desempenho são apresentadas na seção 6. A seção 7 discute trabalhos relacionados, e na seção 8 o artigo é concluído.

2. Tolerância a Faltas Adaptativa

Usualmente, tolerância a faltas em sistemas distribuídos tem sido provida através de combinações de redundâncias de software e hardware, definidas e configuradas estaticamente. Se considerarmos as características dinâmicas dos sistemas distribuídos atuais, que crescem

em escala dia a dia, a noção fixa e pré-estabelecida na coordenação destes modelos de redundâncias os torna mesmos potencialmente ineficientes e custosos [POWELL, 1991; SCHNEIDER, 1990].

Sistemas distribuídos de larga escala, como os construídos com os recursos da Internet, podem se beneficiar de políticas adaptativas. Mecanismos de tolerância a faltas adaptativa fazem uso dessas políticas na gerência de redundâncias, de maneira a se manterem eficazes. Um sistema adaptativo permite que sua configuração seja modificada no sentido de responder às variações no seu ambiente de execução. Alterações na configuração baseada em tolerância a faltas adaptativa podem ser conduzidas a partir da detecção dessas variações, tais como: padrões de comunicação de um sistema distribuído, freqüência de ocorrências de falhas parciais, carga de processamento, ou mesmo por novas necessidades da aplicação [HILTUNEN, 1996]. Para isso, sistemas adaptativos devem ter disponíveis mecanismos para obter informações sobre as condições do ambiente em que está executando.

A 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 [KIM,1990]. A tolerância a faltas adaptativa pode ainda envolver a troca de algoritmos, em tempo de execução, para atender às mudanças do ambiente [CHEN at al., 2001].

3. Componentes de Software

O desenvolvimento de aplicações baseadas em componentes consiste em fazer a composição das aplicações a partir de códigos pré-existentes, os quais são denominados componentes, permitindo desta forma o reuso de software [SZYPERSKI, 1998]. As interfaces do componente definem os pontos de acesso aos serviços oferecidos pelo componente, separando a especificação da própria implementação dos componentes, não permitindo que os usuários conheçam os detalhes de implementação dos mesmos. A programação deve ser fundamentada por um modelo de componentes que define a forma como publicar as interfaces de um componente, e a maneira de especificar métodos e eventos que tornam seus serviços disponíveis aos seus clientes; o modelo também deve prover diretrizes para a criação e a implementação de componentes [BACHMAN ET AL., 2000].

A reutilização de componentes reduz o tempo de desenvolvimento de software, elevando o nível de produtividade. Além disso, uma aplicação construída a partir da composição de componentes pré-existentes testados durante seu desenvolvimento tende a levar à redução dos gastos em manutenção. A manutenção é facilitada, pois será feita de forma localizada, envolvendo somente os componentes necessários.

A abstração de componentes promove a separação entre a lógica das implementações (parte funcional) e o gerenciamento dos serviços de sistema utilizados (parte não funcional). Esta separação habilita o programador a somente se concentrar nos aspectos de implementação das funcionalidades do componente. Os aspectos não funcionais (políticas, suporte de sistema, etc.) ficam a cargo da plataforma de execução dos componentes (ou seja, *containers* e servidores de aplicação).

Considerando as tecnologias de *middleware* existentes, o *modelo de componentes do CORBA* (CCM) está entre os mais promissores. Devido ao fato do CCM ser totalmente baseado no CORBA, traz todas as vantagens deste, como interoperabilidade e portabilidade. Isto não ocorre com os demais modelos de componentes, normalmente restritos a uma linguagem (ex: EJB da Sun) ou a um sistema operacional (ex: .NET da Microsoft). O CCM foi adotado neste trabalho devido aos serviços oferecidos pelo CORBA (CORBAServices [OMG, 2002a]) e devido à flexibilidade apresentada pelo seu modelo quando comparada com outras tecnologias disponíveis (EJB e .NET).

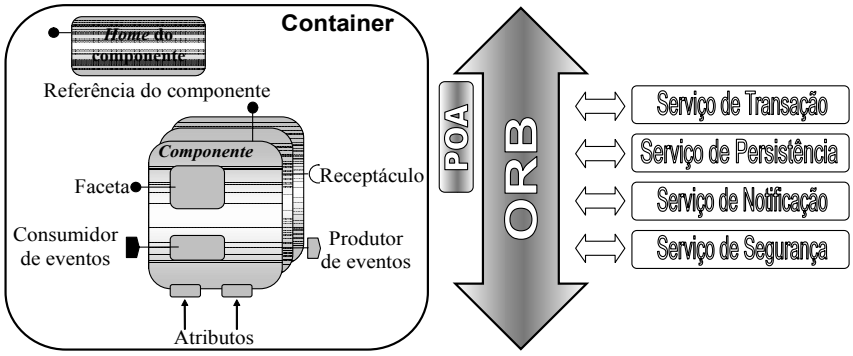

Figura 1. Modelo de Componentes CORBA

3.1 CCM - O Modelo de Componentes CORBA

O CCM foi criado como uma extensão ao modelo de objetos distribuídos do CORBA para adicionar suporte a componentes. As fases de modelagem, programação, empacotamento, implantação e execução de componentes, previstas nas especificações do CCM, organizam objetos em componentes, incorporando à arquitetura CORBA as vantagens atribuídas anteriormente à programação por componentes.

Na figura 1, é apresentado o modelo de componentes CORBA. Um componente possui atributos e portas de comunicação. Atributos são propriedades configuráveis do componente e têm como propósito a configuração do componente. As portas de comunicação são os pontos para a conexão entre componentes, através das quais componentes interagem. São definidos quatro tipos de portas [OMG, 2002B]:

- **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 determinado tipo.

As interfaces dos componentes são especificadas em CORBA IDL (*Interface Definition Language*), que a partir da versão 3.0 (também chamada IDL3) passou a permitir também a definição de componentes.

Cada componente é associado a uma interface *home*. A interface *home* é responsável pelo gerenciamento do ciclo de vida dos componentes; fornece as operações para criação e remoção de instâncias de componentes e para localização das mesmas (Figura 1).

Os componentes CORBA são executados dentro de *containers*. O *container* é responsável por fornecer um ambiente de execução para um componente; ele abstrai os aspectos não funcionais da aplicação do componente, permitindo, por exemplo, acesso transparente aos serviços comuns do CORBA. Em outras palavras, o *container* torna possível separar os aspectos funcionais do componente (implementação de suas funcionalidades) dos aspectos não-funcionais (interação com o suporte e seus serviços).

Na especificação do CCM é definido um suporte de execução padrão para componentes. Este suporte concentra a parte não-funcional da aplicação que pode fazer uso, por exemplo, dos serviços comuns do CORBA: serviço de transação, de persistência, de ciclo de vida, de segurança e de notificação de eventos (Figura 1).

Assim como objetos CORBA, as instâncias de componentes são caracterizadas por suas interfaces e por uma referência IOR (*Interoperable Object Reference* [OMG, 2002a]). As

portas dos componentes são implementadas na forma de objetos CORBA, acessíveis através de suas interfaces IDL3, usando invocação estática ou dinâmica, com as requisições sendo conduzidas pelo ORB. Tal como no CORBA, o POA é responsável por gerar as referências e conduzir as requisições, que chegam através do ORB, à porta desejada.

A referência do componente permite que os clientes acessem as portas da instância do componente e as conectem às suas portas compatíveis – ou seja, facetas com receptáculos de mesmo tipo IDL, e produtores com consumidores de eventos de mesmo tipo IDL. Portas compatíveis podem ser conectadas em tempo de implantação (configuração estática) ou mesmo em tempo de execução, caracterizando uma configuração dinâmica.

Quatro tipos de componentes são definidos pelo CCM:

- **Serviço**: não possui estado e nem identidade. Seu ciclo de vida está limitado à duração de uma invocação;
- **Sessão**: possui estado transiente, e sua identidade não é persistente;
- **Processo**: possui estado e identidade persistentes. Seu estado não é visível para o cliente. Já a identidade pode ser visível para os clientes através de operações definidas pelo desenvolvedor do componente;
- **Entidade**: possui estado e identidade persistentes. Seu estado é visível ao cliente, e a sua identidade é visível através da chave primária especificada na sua interface *home*.

Algumas facilidades de programação são definidas no CCM de modo a permitir a descrição de como a parte funcional deve interagir com a parte não-funcional dos componentes. No CCM é introduzido o *Component Implementation Framework* (CIF). O CIF utiliza a CIDL (*Component Implementation Definition Language*): uma linguagem declarativa usada para descrever as estruturas de implementação do componente. A partir das descrições em CIDL, juntamente com a descrição em IDL3 das interfaces do componente, são gerados através do compilador CIDL, os esqueletos de implementação do componente.

4. Descrição do Modelo TFA-CCM

A idéia fundamental do modelo TFA-CCM consiste em permitir que o programador especifique requisitos de qualidade de serviço (QoS) definindo níveis desejados de disponibilidade e de desempenho. Para isso, os suportes de gerenciamento e de QoS selecionam configurações com a técnica de replicação necessária para atender os requisitos especificados. O suporte de gerenciamento atua nas partes que suportam a tolerância a faltas, respondendo, por exemplo, a mudanças de carga e de freqüência na ocorrência de falhas nos componentes que levam o serviço a se distanciar das expectativas de QoS do usuário.

A Figura 2 ilustra os diferentes componentes que compõem o suporte de TFA-CCM [FAVARIM, 2003] (por simplicidade o ORB e o POA foram suprimidos da figura). O TFA-CCM é composto pelos componentes: *Gerenciador de Tolerância a Faltas Adaptativa* (GTFA), *Gerenciador de QoS* (GQoS), *Coordenador de Replicação* (CR), *Agente de Detecção de Falhas* (ADF) e *Conector*. Essencialmente, estes componentes não-funcionais fazem a configuração e o monitoramento da aplicação.

Na configuração mostrada na Figura 2, os componentes estão dispostos em 4 sítios de um sistema distribuído. A configuração do serviço usa a técnica de *replicação passiva*[§] com duas réplicas do servidor da aplicação. O componente cliente situado no sítio 1 utiliza os serviços fornecidos pelo componente servidor situado no sítio 2 (réplica primária). Cliente e conector podem opcionalmente conviver num mesmo espaço de endereçamento, e com isso a falha de um deles implica na falha de ambos.

[§] Na replicação passiva [BUDHIRAJA et al, 1993], após concluída a chamada, a réplica primária deve sincronizar as secundárias (réplicas *backups*) ao seu estado de processamento, enviando uma cópia de seu estado às mesmas.

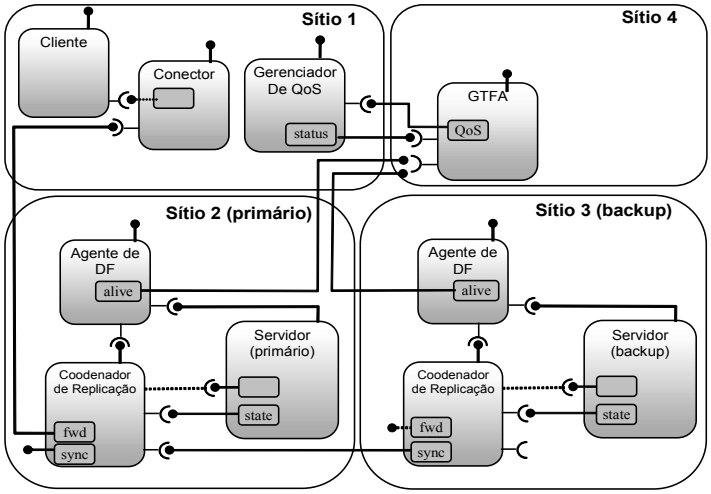

Figura 2. Visão geral do TFA-CCM

O cliente obtém acesso ao servidor através do componente *conector*, que torna transparente para o cliente as possíveis mudanças de configuração do servidor. Por exemplo, a troca de réplicas, onde a réplica *backup* substituiria a primária na configuração, não afeta em nada o comportamento do cliente. Na falha da réplica primária, o *conector* é redirecionado para a réplica *backup* do servidor. A seguir, serão descritos em detalhes os componentes de suporte do TFA-CCM.

GTFA - *Gerenciador de Tolerância a Faltas Adaptativa*

A principal função do GTFA é escolher a configuração adequada ao nível de QoS desejado pela aplicação, que consiste de requisitos de tolerância a faltas, tais como a técnica de replicação, o número e a localização das réplicas, entre outros. Os níveis de QoS aceitos pelo usuário são explicitados através do gerenciador de QoS, que os repassa ao GTFA.

A segunda função do GTFA é efetuar as ações necessárias para que uma nova configuração do sistema seja instalada. Algumas ações que podem ser tomadas são:

- Implantar novas réplicas (componentes, agentes de DF e coordenadores de replicação) ou restaurar réplicas já existentes que tenham sofrido falha;
- Determinar as máquinas onde serão implantadas novas réplicas;
- Trocar a técnica de replicação, substituindo o coordenador de replicação;
- Definir qual o componente primário, caso a técnica de replicação o possua;
- Modificar os intervalos de monitoramento e de checkpoint das réplicas.

A terceira função do GTFA é a manutenção do nível de QoS, que pode envolver a substituição da técnica de replicação utilizada, a restauração do número de réplicas, a migração de uma réplica para outra máquina, a alteração do intervalo de *checkpoint*, etc. Para isso, o GTFA monitora os ADFs para detectar eventuais falhas que podem degradar o nível de QoS do sistema. Quando isso ocorre, o GTFA atua dinamicamente, reconfigurando o sistema de modo a manter o nível de QoS requisitado pelo usuário.

O GTFA fornece também informações de *status* do sistema para o gerenciador de QoS, que disponibiliza estas informações para o usuário. Estas informações envolvem a técnica de replicação que está sendo utilizada, a localização dos componentes replicados, e estatísticas das falhas ocorridas, como o número de falhas nas máquinas, a carga da CPU de cada máquina, etc. O envio periódico destas informações também permite a detecção de falhas no próprio GTFA a partir do gerenciador de QoS, que pode recuperar o GTFA na mesma ou em outra máquina. O estado do GTFA pode ser facilmente recuperado, por ser salvo em *checkpoints* utilizando o serviço de persistência do CORBA.

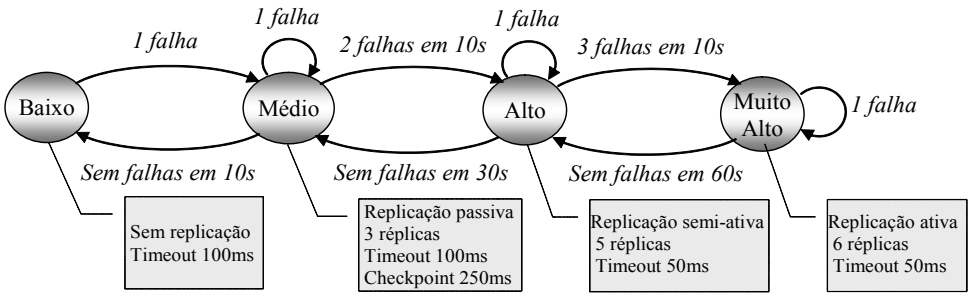

Figura 3. Exemplo de especificação de QoS

GQoS - Gerenciador de Qualidade de Serviço

Para especificar os requisitos de QoS desejados, relacionados diretamente à tolerância a faltas, o TFA-CCM provê o gerenciador de QoS. Os requisitos de tolerância a faltas são especificados na forma de níveis de QoS, onde cada nível especifica uma configuração desejada do sistema. Em cada nível de QoS são especificados os seguintes requisitos: o número de réplicas; a técnica de replicação utilizada; o intervalo de monitoramento das réplicas; o tempo máximo de espera (*timeout*) das respostas ao monitoramento dos componentes; e o intervalo de *checkpoint*, que determina o número de requisições e o intervalo de tempo entre cada atualização de estado (na replicação passiva).

Diferentes níveis de QoS podem ser especificados através do gerenciador de QoS. De modo a manter o nível de confiabilidade desejado, devem se definir as transições entre os níveis de QoS. A transição de um nível de QoS para outro se dá quando o nível de QoS corrente não atende mais aos requisitos de confiabilidade desejados. O GQoS permite que sejam especificadas as ações a serem tomadas nas seguintes situações: falha de uma réplica de um componente; falha de um sítio utilizado pela aplicação; e falha do GTFA.

A detecção de um determinado número de falhas ou a ausência de falhas em um certo intervalo de tempo pode ocasionar mudanças de nível de QoS. Neste caso, condições de transição se tornarão válidas e ocasionarão a mudança da configuração do sistema. Várias condições de transição podem ser especificadas ligando níveis de confiabilidade diferentes. São atribuídos graus de prioridade às condições de transição. Com isto, caso mais de uma condição de transição seja satisfeita em um determinado instante, será escolhida aquela que tiver o maior grau de prioridade. Ao definir os níveis de QoS desejados e transições entre estes níveis, o programador cria uma máquina de estado, conforme exemplo na Figura 3.

Quando ocorrer uma transição entre dois níveis de QoS, o GTFA irá adequar a configuração da aplicação de acordo com os novos requisitos de QoS em vigor. Durante essas mudanças, os conectores são bloqueados, e passam a armazenar as invocações recebidas até que a nova configuração esteja funcional. Através do gerenciador de QoS, o usuário pode ainda modificar manualmente os requisitos de QoS ou ativar transições entre níveis, levando o GTFA a reconfigurar a aplicação. O GQoS é usado também pelo GTFA para passar informações de configuração ao usuário, como por exemplo a técnica de replicação utilizada no momento, a localização dos componentes replicados, entre outras informações pertinentes.

CR - Coordenador de Replicação

O coordenador de replicação implementa os algoritmos referentes à coordenação da técnica de replicação selecionada. Os componentes de coordenação de uma técnica de replicação trocam informações de modo a manter a consistência entre os componentes replicados do servidor. Portanto, operações de salvamento de estado, definição de réplica privilegiada (réplica primária ou líder), construção de *logs* de operações, e outros aspectos não funcionais referentes à técnica de replicação utilizada são executados pelos algoritmos implementados a partir destes componentes de coordenação. No TFA-CCM, a separação entre

aspectos funcionais e não funcionais é feita de modo que a simples troca do coordenador resulta na troca da técnica de replicação na configuração. Nos experimentos realizados no TFA-CCM foram implementados quatro coordenadores: um sem replicação, um para replicação passiva, um para replicação semi-ativa[**] e outra para replicação ativa (replicação máquina de estado [SCHNEIDER, 1990]). A substituição de coordenadores não afeta os componentes replicados nos aspectos funcionais da aplicação.

ADF - Agente de Detecção de Falhas

O modelo possui entre suas premissas faltas de *crash* (faltas de parada) para processadores ou componentes. A comunicação no TFA-CCM é assumida como confiável tomando como base as propriedades do protocolo IIOP (*Internet Inter-ORB Protocol*) do CORBA. O IIOP faz uso da pilha TCP/IP como serviço subjacente e está fundamentado na semântica *at most once*, permitindo que eventuais retransmissões de mensagens de requisição ou respostas provocadas por perdas sejam filtradas no sentido de manter uma só invocação do método requisitado. O suporte de exceções do CORBA é usado no modelo para notificar que o alvo de uma invocação não está respondendo e, portanto, está falho (detecção de *host* ou componente em parada).

Dois níveis de detecção são implementados a partir dos agentes de detecção, ambos baseados no suporte de invocação de métodos do CORBA. Na detecção de falhas de *host*, o GTFA invoca periodicamente o método `is_alive` do ADF (seguindo o modelo *pull* de monitoração). A falha de um *host* (e conseqüentemente de seus componentes) é assumida quando retornar ao GTFA uma exceção do ORB referente à invocação sobre um ADF. Em situação normal, informações de estado da configuração local são retornadas ao GTFA.

O segundo nível (componente) é concretizado na configuração da Figura 2 pelas ligações de cada ADF a um componente replicado e a um coordenador de replicação. Eventuais falhas dos componentes replicados e coordenadores são detectados quando ocorrer uma exceção referente a chamadas periódicas do método `_non_existent` da classe `CORBA::Object`, que é implicitamente herdada por todos os componentes. As falhas de componentes são reportadas ao GTFA pelo ADF nas chamadas periódicas de monitoramento.

Conector

Para fazer uso dos serviços de um componente servidor tolerante a faltas, os componentes clientes devem se conectar ao conector que o representa. As requisições dos clientes são manipuladas através do componente conector, permitindo que seja utilizada uma técnica de replicação para enviar a requisição ao componente servidor, sem no entanto modificar a abstração de uma chamada sobre um componente servidor único, não replicado. Assim sendo, a transparência das mudanças de configuração no servidor é total para os componentes clientes, pois as interfaces do conector e dos componentes servidores são equivalentes.

5. Implementação do Modelo TFA-CCM

O modelo TFA-CCM foi implementado utilizando o OpenCCM versão 0.8 [MARVIE et al., 2002], que segue a especificação CCM. O OpenCCM foi escolhido por ter sido a primeira implementação não comercial de código aberto. A atual versão roda sobre quatro diferentes ORBs. Neste trabalho utilizamos o ORBacus, versão 4.0.5 [IONA TECHNOLOGIES, 2002].

Conforme discutido na seção anterior, o modelo TFA-CCM é composto por um conjunto de componentes de software, que juntos visam fornecer suporte de tolerância a faltas a serviços de aplicação em ambientes distribuídos.

[**] Na técnica de replicação semi-ativa (ou replicação *leader/followers*) [POWELL, 1991] as requisições são difundidas entre todas as réplicas pela réplica líder. As réplicas seguidoras executam também a requisição, mas apenas a líder envia a resposta ao cliente. Os resultados podem ser enviados também às replicas seguidoras no sentido de testar o funcionamento do líder.

```
module tfa-ccm {
   struct NamedValueFT {
      string         name;       // argument name
      any            value;      // argument
      unsigned long  flags;      // argument mode flags
   };
   typedef sequence <NamedValueFT> NVListFT;
   interface ifwd {
      any forwardRequest(in string op, in NVListFT params,
                         in CORBA::TypeCode resultTypeCode);
   };
   interface isync {
     void syncState(in State state);
   };
   component Coordinator {
      attribute      string         replType;
      provides       ifwd           facet_fwd;
      provides       isync          facet_sync;
      uses           MgtState       state;
      uses           isync          sync;
}; };
```

Figura 4. Descrição em IDL3 do Coordenador de Replicação

O conector tem a função de garantir transparência de localização dos componentes replicados. 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. 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 (especificada na Figura 4).

Os coordenadores de replicação têm a função de implementar a técnica de replicação utilizada. Foram implementados os coordenadores que implementam as técnicas de replicação passiva, semi-ativa, ativa e um coordenador que não implementa nenhuma técnica de replicação. Este último é usado quando se deseja apenas requisitos de disponibilidade, onde informações de estado não sejam necessárias. Este tipo de coordenador é utilizado quando só existe uma instância do componente, que ao falhar será apenas re-instanciada para que a aplicação continue funcionando normalmente. A técnica de replicação ativa foi implementada usando mecanismos de comunicação de grupo fornecidos pelo ORB MJaco [BESSANI, 2003].

Todo componente coordenador de replicação possui duas facetas: a faceta `sync` e a faceta `fwd` (Figura 4). A faceta `sync` é usada pelos coordenadores de replicação passiva e semi-ativa para sincronização do estado de uma nova réplica com a réplica primária. Além disso, o coordenador de replicação passiva usa essa faceta para sincronizar o estado das réplicas em caso de mudança do estado do componente primário. A faceta `fwd` é implementada por todos os tipos de coordenadores de replicação, para que as requisições dos clientes sejam repassadas do conector para o coordenador primário (nos dois tipos de replicação citados acima). Além desse uso, essa faceta também é usada pelo coordenador primário (replicação semi-ativa), para repassar as requisições recebidas dos clientes para os coordenadores acoplados às réplicas seguidoras (*followers*). A faceta `fwd` possui apenas um método, que é responsável por efetuar a invocação do método requerido pelo cliente no componente replicado. 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 em tempo de execução descobrir como fazer as chamadas aos métodos daquela interface. Esta invocação é representada na Figura 2 por uma linha tracejada.

Para fornecer um mecanismo padrão para a atualização de estado nos componentes replicados, é definido um componente base (`baseComp`) que todo componente que deseja ser replicado deve herdar. O componente `baseComp`, fornece uma faceta (`state`), que provê os métodos para recuperação (`getState`) e atualização (`setState`) do estado do componente.

```
module tfa-ccm {
   struct alive_status {
      short faults_in_coordinator, faults_in_component;
   };
   interface ialive {
      alive_status is_alive();
   };
   enum conectiveMode{ none, or, and };
   struct QoS_Transition {
      string   transitionName, nextLevel;
      short    priority, componentFault, componentTime, hostFault, hostTime;
      conectiveMode connective;
   };
   typedef sequence <QoS_Transition> QoS_Transitions;
   enum replModel{ none, passive, semi-active, active };
   struct QoS_Level {
      string    levelName;
      Boolean   initialLevel;
      replMode  replType;
      short     numberReplicas, intMonitorAgent, timeoutAgent,
                intMonitorComp, timeoutComp;
      QoS_Transitions transitions;
   };
   typedef sequence <QoS_Level> requirements;
   interface iQoS {
      void set_QoS (in requirements QoS_Requirements);
      void LogInformation (in boolean on);
   };
   component GTFA {
      provides      iQoS           QoS;
      uses          istatus        status;
      uses          multiple       ialive alive;
   };};
```

Figura 5. Descrição em IDL3 do GTFA

Para a implantação de componentes, o TFA-CCM usa as APIs de implantação fornecidas pelo OpenCCM. Porém, nem todas as APIs estão implementadas pelo OpenCCM, como o caso da API para configuração dos atributos dos componentes em tempo de execução. Neste caso, foi necessário implementar funções que suprem a ausência desta API.

O GTFA mantém uma estrutura de dados com 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 interligar as portas dos componentes ou remover componentes de algum sítio. O GTFA também usa esses dados para fornecer informações sobre o funcionamento do sistema ao gerenciador de QoS, como o tipo de replicação que está sendo utilizada, a localização dos componentes replicados, e falhas que acontecem nos componentes (Figura 5).

A falha do GTFA e a conseqüente perda destes dados comprometeriam todo o funcionamento do sistema. Para evitar que as informações de estado do GTFA sejam perdidas, essas informações são gravadas em um meio de armazenamento persistente. Assim, caso esse componente apresente falha e tenha que ser re-implantado em outro sítio, seu estado pode ser recuperado. Para isso, o GTFA é um componente do tipo entidade (subseção 3.1), cujo estado é persistente e é gerenciado de forma transparente pelo *container*. Os outros componentes que compõem o TFA-CCM são do tipo sessão (subseção 3.1), visto que a perda do estado destes componentes não compromete a consistência do sistema.

O Gerenciador de QoS recebe do GTFA informações de *status* a respeito do sistema através da faceta `status` (Figura 5). Estas informações são mostradas ao usuário através da interface gráfica do gerenciador (Figura 6). Através desta interface gráfica, também é possível especificar dos requisitos de QoS Quando o gerenciador de QoS percebe que não está mais recebendo informações do GTFA, ele poderá então re-instalar o GTFA em outro sítio.

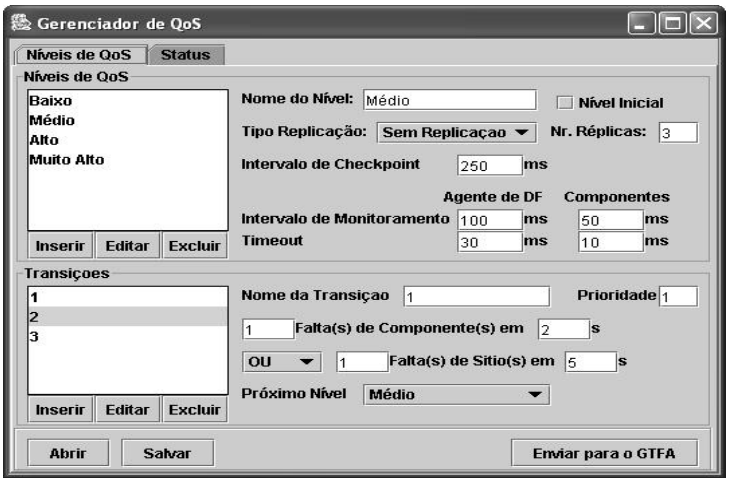

Figura 6. Interface Gráfica do Gerenciador de QoS (Edição dos Níveis de QoS)

6. Medidas de Desempenho

Visando verificar o desempenho da implementação do TFA-CCM, foram executados testes[††] em uma rede local Ethernet de 100Mbps composta por computadores Pentium IV 1.6Ghz com 256Mb de memória RAM, sistema operacional Linux Mandrake versão 9.1 e JDK 1.4.

O primeiro teste mensurou o tempo de resposta das chamadas ao componente replicado em diferentes configurações, isto é, com diferentes técnicas de replicação. As seguintes configurações foram montadas: sem a adição do TFA-CCM, com coordenador que não implementa nenhuma técnica de replicação, com coordenador de replicação passiva, com coordenador de replicação semi-ativa, e com coordenador de replicação ativa. Foram usadas duas réplicas nos três últimos casos. Foi utilizado nos testes um componente com apenas uma faceta, com um método que não executa nenhuma instrução (tamanho do pacote 0KB, na figura 7). Deste modo, foi minimizada a influência do tempo de processamento do método nos resultados dos testes, avaliando apenas a sobrecarga gerada pela adição do TFA-CCM. Esses testes não visam avaliar/analisar o desempenho das técnicas de replicação de acordo com o grau de redundância, pois existe uma vasta e exaustiva literatura sobre o assunto [BUDHIRAJA, 1993, SCHNEIDER, 1990, POWELL, 1991]. O nosso objetivo é medir o custo de cada técnica.

É possível observar no gráfico apresentado na Figura 7 a sobrecarga imposta pelo TFA-CCM sem replicação, com a técnica de replicação passiva, com a técnica de replicação semi-ativa e com a técnica de replicação ativa. Nota-se que a sobrecarga cresce de acordo com a técnica de replicação utilizada. O desempenho bem menor da replicação ativa é devido ao uso de comunicação de grupo com ordenação total fornecido pelo MJaco [BESSANI, 2003] – as outras técnicas usam apenas ordenação FIFO. O aumento no tempo de resposta era esperado, tendo em vista que os componentes do TFA-CCM devem interceptar as chamadas e coordenar as réplicas. Esta sobrecarga pode ser considerada aceitável para o fornecimento de requisitos de tolerância a faltas.

O segundo experimento realizado verificou o tempo médio gasto para criar uma réplica do componente em um novo sítio, juntamente com o seu coordenador de replicação e o seu agente de TFA; e o tempo médio para trocar o tipo de replicação[4]. Neste experimento foi observado que o tempo médio gasto para a implantação de uma nova réplica foi de **330ms**, enquanto o tempo médio de estabilização gasto para fazer a troca de coordenadores,

[††] Os resultados foram obtidos a partir da média de 1000 execuções.

alternando entre replicação passiva e semi-ativa, foi de **260ms**. O tempo médio de estabilização da replicação semi-ativa para ativa foi de aproximadamente **280ms**.

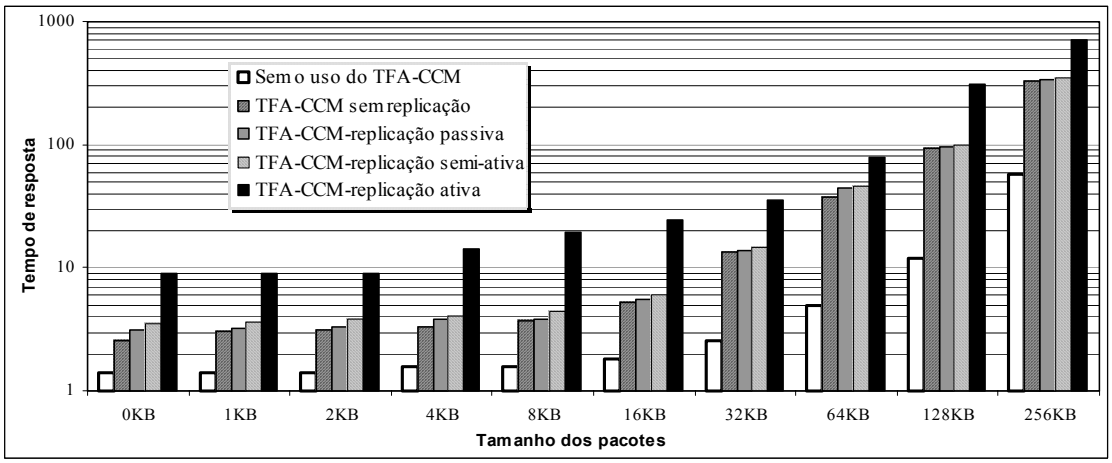

Figura 7. Desempenho do TFA-CCM com diferentes técnicas de replicação

7. Trabalhos Relacionados

Embora os primeiros trabalhos na área de componentes tenham surgido já na década de 60 (SZYPERSKI, 1998), a incorporação deste estilo de programação em *middlewares* é recente. Implementações do modelo CCM estão apenas começando a surgir. Adicionalmente, apesar das vantagens trazidas pela utilização de componentes de software para desenvolver aplicações distribuídas, uma aplicação que possua requisitos temporais, de tolerância a faltas, balanceamento de carga, entre outros, não encontra suporte para atendimento destes requisitos nos modelos de componentes atuais. Isto se deve ao fato do ambiente de execução – ou seja, o *container* – não oferecer suporte de qualidade de serviço (QoS) aos componentes de software. Tendo em vista tais limitações, este artigo apresenta um modelo para construção de aplicações baseadas no CCM onde se procura estruturar a aplicação usando componentes replicados.

Vários trabalhos que visam fornecer requisitos de QoS a aplicações, em alguns casos empregando tolerância a faltas adaptativa, são descritos na literatura. QuO (*Quality Objects*) [ZINKY, 1997] é uma arquitetura que dá suporte ao desenvolvimento de aplicações distribuídas, baseadas em CORBA, com requisitos de QoS. QuO fornece meios para especificar, monitorar e controlar aspectos de QoS em uma aplicação, assim como meios para adaptar o seu comportamento com base em mudanças no ambiente. AQuA (*Adaptive Quality of Service Availability*) [CUKIER et al., 1998] visa fornecer tolerância a faltas adaptativa para aplicações distribuídas. Programadores de aplicações especificam os níveis de confiabilidade desejados, que são alcançados por AQuA através da configuração do sistema, levando em conta a disponibilidade de recursos e as falhas ocorridas. AQuA utiliza o QuO para especificar os requisitos de QoS em nível de aplicação, e 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 pelo AQuA para fornecer serviços de comunicação de grupo.

Enquanto QuO e AQuA exigem que os requisitos de QoS sejam definidos em tempo de compilação, o TFA-CCM permite que os requisitos de QoS sejam modificados em tempo de execução, tirando proveito da flexibilidade propiciada pelo uso de componentes de software.

Vários trabalhos usam mecanismos de configuração dinâmica para tolerar faltas em um sistema distribuído, e algumas destas experiências usam também o conceito de conectores. Em [BATISTA e CARVALHO, 2002], um conector é antes de tudo um suporte para configuração, que busca entre várias opções de componentes servidores, aquele que possui a assinatura mais

adequada para atender a invocação de um cliente. Se na chamada do componente – que é executada pelo conector – ocorrer alguma exceção, a procura de uma nova alternativa de componente servidor é imediatamente iniciada. Já os conectores do TFA-CCM são mais próximos conceitualmente dos conectores usados em [SZTAJNBERG, 2002]. Neste trabalho os conectores concentram todos os aspectos não funcionais de uma replicação. No TFA-CCM, o conector assume um papel mais simples, que é o de um elemento *redirecionador de invocações* para um componente replicado na configuração. A opção por um conector simples foi determinada pelo uso do CCM onde, por definição, o *container* já concentra muitos dos aspectos não funcionais de uma aplicação.

Em [MARANGOZOVA e HAGIMONT, 2002] é apresentada uma abordagem para replicação de componentes CORBA. Nessa abordagem são usados objetos de interceptação, que são 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. Já 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.

8. Conclusão

Este artigo apresentou o modelo TFA-CCM, que oferece mecanismos flexíveis para a construção de software baseado em componentes CORBA com requisitos de tolerância a faltas. O 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, que combinados atendem requisitos de tolerância a faltas. 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. Testes realizados com o protótipo do modelo mostraram que seus custos de desempenho são aceitáveis, tendo em vista o ganho em confiabilidade propiciado pelo TFA-CCM.

A implementação do modelo TFA-CCM será em breve disponibilizada como um módulo do OpenCCM [MARVIE et al., 2002], com código aberto e licença pública GNU. Em versões futuras, pretendemos disponibilizar um *container* com suporte a tolerância a faltas adaptativa, que seria então fornecida transparentemente aos componentes. Também antevemos a necessidade de adequar o modelo às especificações da OMG para tolerância a faltas [OMG, 2000], que devem ser estendidas de modo a serem aplicadas também ao CCM. Dentro da perspectiva de fornecer suporte a serviços não funcionais não previstos pelo CCM, pretendemos adicionar suporte a outros requisitos de QoS, além de tolerância a faltas, como requisitos de tempo real e balanceamento de carga. Além disso, pretendemos permitir a especificação dos níveis de QoS através de uma linguagem declarativa baseada em XML.

Referências Bibliográficas

Bachman, F. et al. (2000). "Volume II: Technical Concepts of Component Based Software Engineering". Technical Report CMU/SEI-2000-TR-08. Software Engineering Institute, Carnegie Mellon University.

Batista, T. V., Carvalho, M. G. (2002). "Component-Based Applications: A Dynamic Reconfiguration Approach with Fault Tolerance Support". In: Electronic Notes in Theoretical Computer Science, vol.65. Elsevier Science.

Bessani, A.N., Lung, L.C., Fraga, J.S. (2003). "MJaco: Middleware CORBA para Comunicação de Grupo". In: XXI Simpósio Brasileiro de Redes de Computadores, Natal-RN.

Budhiraja, N., Marzulo, K., Schneider, F. B., Toueg, S. (1993). "The Primary-Backup Approach". In: Distributed Systems. 2nd edition. Addison Wesley.

Chang, I.; Hiltunen, M.A.; Schlichting, R. D. (1998) "Affordable Fault Tolerance through Adaptation". In: IPPS/SPDP Workshops 1998.

Chen, W. K., Hiltunen, M. A., Schlichting, R. D. (2001). "Constructing Adaptive Software in Distributed Systems". In: 21st International Conference on Distributed Computing Systems.

Cukier, M. et al. (1998). "AQuA: An Adaptive Architecture that Provides Dependable Distributed Objects". In: 17th IEEE Symposium on Reliable Distributed Systems.

Favarim, Fábio (2003). "Componentes em um Esquema de Tolerância a Faltas Adaptativa". Dissertação de Mestrado, PGEEL-UFSC.

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

Hiltunen, M. Schlichting, R. (1996). "Adaptive Distributed e Fault-Tolerant Systems". In: International Journal of Computer Systems Science and Engineering, 11(5): 125–133.

Kim, K.H., Lawrence, T. (1990). "Adaptive Fault Tolerance: Issues and Approaches". In: Second IEEE Workshop on Future Trends of Distributed Computing Systems.

Kramer, J. (1990). "Configuration Programming – A Framework for Development of Distributed Systems". In: IEEE International Conference on Computer Systems and Software Engineering.

Loques, O. G., Leite, J., Sztajnberg, A., Lobosco, M.(1999). "Towards Integrating Meta-Level Programming and Configuration Programming". In: XIII Simpósio Brasileiro de Engenharia de Software.

Magee, J. e Kramer, J. (1997). "Composing Distributed Objects in CORBA". In: 3rd International Symposium on Autonomous Decentralized Systems.

Marangozova, V. e Hagimont, D. (2002). "An Infrastructure for CORBA Component Replication". In: 1st IFIP/ACM Working Conference on Component Deployment.

Marvie, R., Merle, P., Vadet, M. (2002). "The OpenCCM Platform". http://openccm.objectweb.org/

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

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

OMG (2000). "Fault-Tolerant CORBA Specification v.1.0". OMG Document ptc/2000-04-04.

OMG (2002a). "The Common Object Request Broker Architecture" v3.0. OMG Document 02-06-33.

OMG (2002b). "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 Int. Working Conference on Dependable Computing for Critical Applications.

Schneider, F. B. (1990). "Implementing Fault-Tolerant Service Using the State Machine Approach: A Tutorial". In: ACM Computing Survey, 22(4):299-319.

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

Sztajnberg, A. (2002). "Flexibilidade e Separação de Interesses para Concepção e Evolução de Sistemas Distribuídos". Tese de doutorado, COPPE/UFRJ.

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

Zinky, J. A., Bakken, D. E., Schantz, R. E. (1997). "Architectural Support for Quality of Service for CORBA Objects". In: Theory and Practice of Object Systems, 3(1):53–73.