Uma Arquitetura para Tolerância a Faltas em Serviços Web*

Jeferson L. R. Souza , Frank Siqueira

Departamento de Informática e Estatística (INE)
Laboratório de Pesquisa em Sistemas Distribuídos (LaPeSD)
Universidade Federal de Santa Catarina (UFSC)
Florianópolis, Brasil

{jeferson,frank}@inf.ufsc.br

Abstract. *Web services have been widely used to solve problems of interoperability between applications and/or technologies. However, the specifications and standards defined for Web Services do not address solutions for problems related to fault tolerance and high availability of services. To solve this problem, this paper proposes a software architecture for fault tolerance in Web Services. This architecture is responsible for increasing service dependability and maintaining all replicas of a service in a consistent state, having as main characteristic the separation of these replicas in two replication layers.*

Resumo. *Serviços Web têm sido amplamente utilizados para solucionar problemas de interoperabilidade entre aplicações e/ou tecnologias. Porém, as especificações e padrões definidos para Serviços Web não solucionam problemas relativos a tolerância a faltas e a alta disponibilidade dos serviços. Para solucionar esse problema, esse artigo apresenta uma proposta de arquitetura de software para tolerância a faltas em Serviços Web. Essa arquitetura é responsável por aumentar a confiabilidade do serviço e por manter a consistência de estado de todas as suas réplicas, tendo como principal característica a separação dessas réplicas em duas camadas de replicação.*

1. Introdução

A grande vantagem na utilização de Serviços Web para concepção de sistemas distribuídos é a interoperabilidade que essa tecnologia proporciona [Ayala et al. 2002]. Essa interoperabilidade facilita de forma significativa o desenvolvimento de aplicações distribuídas, já que problemas tais como a diversidade de *hardwares* e *softwares* utilizados são solucionados pelos protocolos utilizados por esses serviços [Newcomer 2002].

[Cerami 2002] define um Serviço Web como sendo qualquer serviço, disponível sobre a Internet, que se comunique utilizando um protocolo padrão baseado em XML e não seja específico para um determinado sistema operacional e/ou linguagem de programação. Para a interação com Serviços Web, foram definidos alguns protocolos e padrões para publicação, descrição e troca de mensagens. Esses protocolos e padrões são o UDDI (*Universal Description, Discovery and Integration*) [Uddi 2004], WSDL (*Web Services Description Language*) [Wsdl 2001] e o SOAP [Soap 2003].

Cada Serviço Web é identificado por uma URI (*Unique Resource Identifier*) e pode ser acessado diretamente por qualquer aplicação que tenha conhecimento dessa

*Esse trabalho tem o apoio do Conselho Nacional de Pesquisa (CNPq) em conjunto com o Programa de Pós Graduação em Ciência da Computação (PPGCC) da Universidade Federal de Santa Catarina (UFSC)

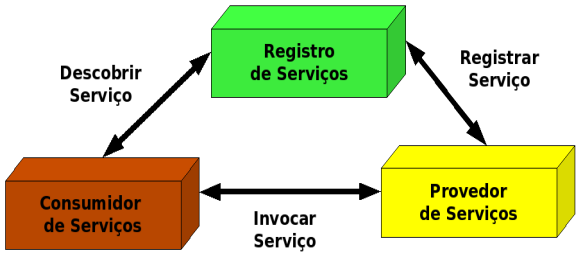

Figura 1. Elementos Fundamentais da Arquitetura Orientada a Serviços

identificação. Conceitualmente, Serviços Web adotam a arquitetura SOA (*Service Oriented Architecture*) [Ayala et al. 2002] que é composta por três elementos fundamentais, como mostra a figura 1.

O **Registro de Serviços** é a entidade que mantém informações sobre a descrição do serviço e sua localização. É através desse registro que **Consumidores do serviço** podem realizar uma busca e obter o arquivo de descrição do serviço que contém informações para que esse consumidor possa utilizá-lo. Já o **Provedor de Serviços** é a entidade que fornece o serviço propriamente dito. O provedor registra seu serviço no registro de serviços, para que consumidores do serviço possam encontrá-lo e utilizar suas funcionalidades.

Percebe-se que nesse modelo fundamental não há nenhuma preocupação com a tolerância a faltas do provedor de serviços. Isto é, se por acaso o provedor de serviços parar de funcionar, os consumidores só poderão utilizar o serviço novamente quando a falha do serviço for solucionada.

Para resolver esse problema, esse artigo propõe uma arquitetura de *software* que tem o objetivo de fornecer tolerância a faltas para Serviços Web. As seções estão organizadas da seguinte forma: a seção 2 apresenta uma visão geral de tolerância a faltas para Serviços Web, abordando alguns trabalhos relacionados com o tema. A seção 3 descreve a arquitetura proposta. A seção 5 apresenta uma análise comparativa entre a arquitetura proposta e cada um dos trabalhos relacionados descritos na seção 2. Por fim, a seção 6 apresenta algumas conclusões e trabalhos futuros.

2. Tolerância a Faltas em Serviços Web

Serviços que são executados sobre uma infra-estrutura de rede podem sofrer com os problemas inerentes a essa infra-estrutura. Esses problemas podem causar uma indisponibilidade do serviço, podendo ser resumidamente de três tipos: problemas no serviço, na plataforma de execução (hardware e software) e na rede de comunicação.

A tecnologia base para implementação de Serviços Web se tornou uma ferramenta atrativa para a construção e concepção de sistemas distribuídos [Ye and Shen 2005]. Uma das principais características fornecidas pelos padrões adotados na construção de Serviços Web é a interoperabilidade [Ayala et al. 2002]. Os Serviços Web, por serem serviços executados sobre uma infraestrutura de rede, sofrem dos mesmos problemas que foram introduzidos no início dessa sessão. Portanto, técnicas de tolerância a faltas devem ser aplicadas com o objetivo de minimizar o impacto negativo da falha de fornecimento de

um determinado serviço, causada pela manifestação desses problemas.

Com base em diversas fontes da literatura [Salas et al. 2006, Santos et al. 2005, Ye and Shen 2005, Li et al. 2005, He 2004, Aghdaie and Tamir 2002, Dialani et al. 2002] pode-se definir que tolerância a faltas (TF) em Serviços Web é a capacidade de fornecimento do serviço requisitado, mesmo na presença de faltas. O fornecimento de TF está fundamentado na utilização de técnicas que forneçam um alicerce sólido para que uma determinada falta possa ser tolerada da melhor maneira possível, de modo a não afetar o funcionamento do *software* quando esta se manifestar. Nessa base fundamental, técnicas tais como Replicação de estado [Jalote 1994], detecção de falha [Chandra and Toueg 1996], ordenação de eventos [Veríssimo and Rodrigues 2001, Sousa et al. 2002], comunicação confiável [Chockler et al. 2001], entre outras técnicas, se tornam importantíssimas no intuito de fornecer TF a um nível aceitável.

Já que uma padronização em torno dos aspectos relativos a TF para Serviços Web não foi formulada, a próxima subseção descreve alguns trabalhos relacionados que apresentam propostas de arquiteturas e infra-estruturas de *software* que têm o objetivo de fornecer TF para Serviços Web.

2.1. Trabalhos Relacionados

[Ye and Shen 2005] desenvolveram um *middleware* para dar suporte à concepção de Serviços Web confiáveis, utilizando a técnica de replicação ativa. Esse *middleware* exige que consumidores do serviço sejam implementados utilizando um conjuntos de classes que permitam o acesso ao serviço com suporte a tolerância a faltas. Um detalhe importante é que somente operações que realizem alteração de estado nas réplicas têm garantia de ordenação total de execução. Para tal, é necessário que o administrador do serviço forneça uma lista que indique essas operações.

[Li et al. 2005] definiram um *framework* denominado SWS para o desenvolvimento de Serviços Web tolerantes a faltas. Para isso, esse *framework* utiliza como conceitos base esquemas de replicação e redundância N-Modular [Xu and Bruck 1998]. Seu principal objetivo é possibilitar a utilização de Serviços Web em aplicações consideradas críticas, já que ataques de segurança foram modelados como falhas bizantinas. Essa abordagem divide as réplicas em vários grupos, possuindo um protocolo de comunicação inter-grupo (SWS-IGC) e um mecanismo de ordenação de mensagens (SWS-MO) para garantir eficiência e corretude na comunicação de grupo. Para dar suporte à descrição de grupos de serviços, foram realizadas modificações no registro UDDI com o objetivo de manter informações sobre todas as réplicas de um determinado grupo. Consumidores do serviço devem ser desenvolvidos utilizando um pacote específico fornecido pelo *framework*.

[Santos et al. 2005] definiram uma infra-estrutura de tolerância a faltas para Serviços Web denominada FTWEB. Essa infra-estrutura é baseada nos conceitos e padrões definidos pela especificação FT-CORBA [Omg 2002]. As réplicas do serviço são organizadas em um único grupo, sendo que as requisições dos consumidores são enviadas para um componente denominado *WSDispatcher*. Esse componente é responsável por gerenciar as réplicas do serviço e enviar as requisições recebidas dos consumidores para essas réplicas. Nessa infra-estrutura Serviços Web são implementados como objetos CORBA, possuindo um componente denominado *WSWrapper* responsável pela

integração desses objetos com o restante da infra-estrutura. Consumidores do serviço também necessitam de um componente específico denominado *WSClientDriver*, para se beneficiar das características de tolerância a faltas fornecidas pela infra-estrutura.

[Salas et al. 2006] especificam um *framework* para alta disponibilidade de Serviços Web denominado WS-Replication. O principal objetivo desse *framework* é a replicação de Serviços Web em uma WAN (*Wide Area Network*). Esse *framework* utiliza replicação ativa para organizar seu grupo de réplicas do serviço ao longo de uma WAN. Sua composição tem como base a definição de dois componentes principais: um componente de replicação e um componente de multicast confiável. O componente de replicação trata de todas as características envolvidas na replicação ativa e utiliza o componente de multicast confiável para disseminar informação para todas as réplicas. Esse componente de multicast confiável é baseado no protocolo SOAP.

Apesar de propiciarem um aumento na disponibilidade de Serviços Web, estes trabalhos apresentam restrições como a incompatibilidade com consumidores legados[1] e a exigência de alterações nos padrões existentes. A arquitetura proposta neste artigo busca atingir o mesmo objetivo eliminando tais restrições.

3. Arquitetura de Tolerância a Faltas para Serviços Web

A especificação dessa arquitetura de *software* descreve de forma abstrata quais os componentes necessários para o fornecimento de TF para Serviços Web. Desse modo, a arquitetura proporciona uma separação implícita do problema, dividindo assim as responsabilidades e funcionalidades ao longo de seus componentes. Esses componentes são:

- Componente de comunicação de grupo (CG): responsável pela abstração do protocolo de transporte utilizado na troca de mensagens;
- Componente de ordenação de mensagens (OM): responsável pela garantia de ordenação total das mensagens em todas as réplicas do serviço;
- Componente de Recuperação (REC): responsável pela recuperação de estado de uma réplica faltosa;
- Componente de *log* (LOG): responsável por armazenar informações sobre os dados que trafegam pelos componentes da arquitetura;
- Detector de falha de réplica (DFR): responsável pela detecção de falha de uma réplica pertencente à arquitetura proposta;
- Detector de falha de grupo (DFG): responsável pela detecção de falha de um grupo pertencente à arquitetura;

Antes de descrever com maiores detalhes cada um dos componentes pertencentes à arquitetura, é necessário um entendimento geral do seu funcionamento. A arquitetura proposta nesse artigo utiliza a replicação como conceito base no fornecimento de TF para Serviços Web. A principal característica da arquitetura proposta em relação a replicação é a forma como as réplicas do serviço são organizadas. A figura 2 exibe a representação da arquitetura de TF proposta para Serviços Web.

Na arquitetura proposta, réplicas do serviço são organizadas em duas camadas de replicação. Dentro dessas duas camadas, as réplicas são subdivididas em grupos. Um dos objetivos dessa subdivisão em camadas e grupos é fornecer a possibilidade de utilização

[1]Consumidores que foram desenvolvidos sem qualquer conhecimento de uma arquitetura de TF

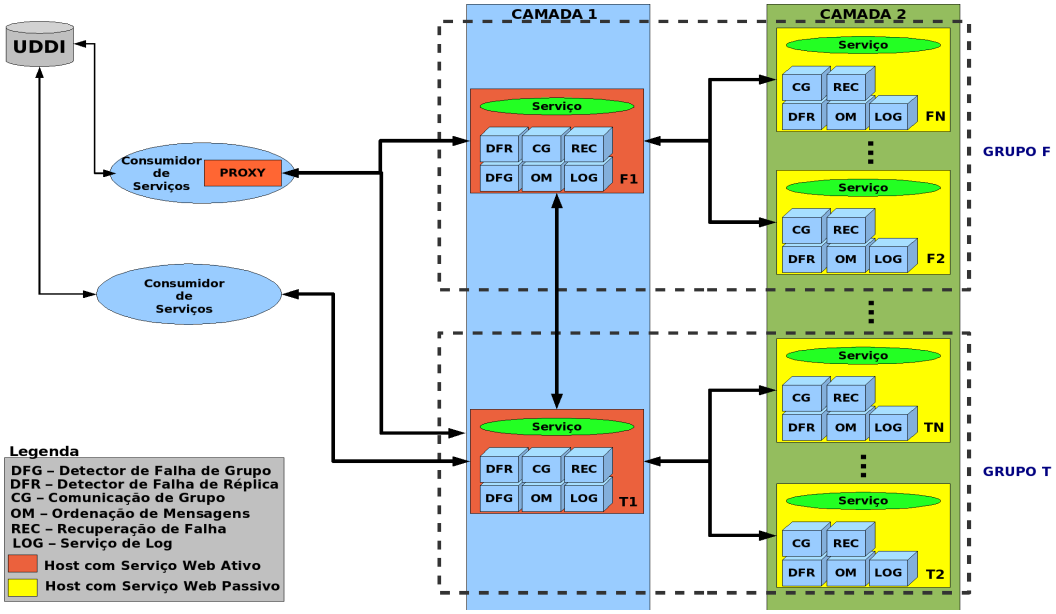

Figura 2. Arquitetura de TF para Serviços Web

de técnicas de replicação distintas entre os grupos. Cada grupo possui um líder, sendo esse líder integrante da primeira camada. Com exceção dos líderes, todas as outras réplicas fazem parte da segunda camada de replicação.

Essa primeira camada é composta por Serviços Web que são acessados diretamente pelos consumidores do serviço. Assim, consumidores somente podem enviar requisições para os Serviços Web pertencentes a essa camada. Do ponto de vista do consumidor de serviços, esses Serviços Web da primeira camada são serviços visíveis, ou seja, que recebem e respondem as requisições desse consumidor. Já a segunda camada de replicação é constituída por Serviços Web invisíveis para os consumidores do serviço, ou seja, que não recebem diretamente requisições enviadas por consumidores. Sendo assim, a segunda camada é subordinada às decisões efetuadas pelos Serviços Web da primeira camada.

Além de ser o contato direto com os consumidores do serviço, os Serviços Web da primeira camada utilizam o OM e o CG para realizar a ordenação das requisições dos consumidores e encaminhar essas requisições (na mesma ordem) para seu grupo. Caso a técnica de replicação utilizada no grupo exija que todas as réplicas processem a requisição, esse Serviço Web da primeira camada utiliza seu CG para receber as respostas das réplicas desse grupo, processa a requisição e realiza uma votação que elegerá a resposta que será encaminhada para o consumidor. É importante salientar que essas atividades podem ser executadas de forma paralela em cada um dos Serviços Web da primeira camada.

Assim, os Serviços Web da primeira camada podem estar distribuídos, juntamente com seu grupo, em redes distintas interligadas através da Internet, ou seja, pode-se ter, por exemplo, um grupo alocado em uma rede no Brasil e outro grupo alocado em uma rede na Argentina. A possibilidade de alocação de grupos distintos que se comunicam por uma infra-estrutura de rede WAN resgata todos os conceitos já arduamente descritos na

SOA [Ayala et al. 2002]. A interoperabilidade entre os Serviços Web da primeira camada segue o mesmo princípio da interoperabilidade descrito na SOA. Isso quer dizer que mensagens são trocadas utilizando um protocolo único que seja independente de plataforma e protocolo de transporte utilizado (por exemplo, o protocolo SOAP). Adicionalmente, essa troca de mensagens deve ser confiável, já que são essas mensagens que irão garantir a consistência de estado do serviço como um todo.

A abstração imposta pela arquitetura proporciona que seja utilizado qualquer protocolo de comunicação em qualquer uma das camadas. Por questões de interoperabilidade, a primeira camada atualmente utiliza o protocolo padrão para comunicação entre Serviços Web, ou seja, o protocolo SOAP. Além de manter suas características nativas, aspectos que garantam a entrega das mensagens a seus destinatários devem ser adicionados. Nesse sentido, já existe uma especificação para troca de mensagens SOAP confiáveis denominada WS-Reliability [WS-Reliability 2004] que é adotada como padrão de troca de mensagens SOAP confiáveis na arquitetura proposta.

A segunda camada insere uma maior flexibilidade no tratamento das classes de falhas [Chandra and Toueg 1996] por permitir um grau de independência entre os diversos grupos. Isso possibilita que em um ambiente totalmente distribuído, cada um desses grupos esteja alocado em uma rede distinta, permitindo a utilização de diferentes técnicas de replicação, protocolos de transporte e esquemas de detecção e recuperação de falhas.

Isso permite que nossa arquitetura forneça a flexibilidade de configuração de técnicas de replicação e protocolos de comunicação intra-grupo diferentes, possibilitando o tratamento de classes de falhas de forma adaptativa aos requisitos de tolerância a faltas necessários ao serviço fornecido. Com essa flexibilidade, pode-se ter grupos implementados em diferentes tecnologias, utilizando diferentes técnicas de replicação que podem utilizar diferentes protocolos de comunicação intra-grupo, permitindo que a composição da arquitetura seja heterogênea. A ligação entre esses grupos é efetuada pelos líderes de cada grupo, que formam o conjunto de Serviços Web da primeira camada. Através dos líderes é possível que todos os grupos troquem informações utilizando um protocolo de comunicação único.

Como cada um dos líderes representa unicamente seu grupo na primeira camada da arquitetura, esse Serviço Web se torna um ponto de falha do grupo. Por isso, é importante que as demais réplicas pertencentes ao grupo realizem o monitoramento desse Serviço Web para que, em caso de sua falha, uma dessas réplicas assuma seu lugar, notificando as demais réplicas na primeira camada que ela é a nova representante do grupo. No instante subseqüente à detecção da falha, mecanismos de recuperação devem ser acionados na tentativa de retomar o estado da réplica faltosa ou, em caso de impossibilidade de recuperação, providenciar a criação de uma nova réplica do serviço no grupo em questão.

Em caso de falha de um grupo inteiro, consumidores legados que realizavam requisições para esse grupo não terão transparência da falha. Para continuar usufruindo do serviço, esses consumidores precisarão obter uma referência para outro Servidor Web da primeira camada. Para tolerar falhas de grupo, consumidores podem ser implementados com auxílio de um *proxy* (opcional). O *proxy* submete as requisições do consumidor para todos os Serviços Web da primeira camada, e em caso de falha de algum desses serviços, essa falha fica transparente para esse consumidor.

3.1. Detecção de Falha

Os detectores de falhas são responsáveis pelo monitoramento das réplicas ativas, realizando essa tarefa periodicamente. A figura 2 mostra que a arquitetura possui réplicas nas duas camadas. Como há diferentes grupos na arquitetura, é necessário ter detectores que atuem na detecção de falha de réplicas e de grupos independentemente. Assim, os detectores de falhas são classificados em duas classes distintas: detectores de falha de réplica (DFR) e detectores de falha de grupo (DFG).

3.1.1. Detectores de Falha de Réplica - DFR

DFRs são responsáveis pelo monitoramento das réplicas ativas alocadas em um grupo específico. Cada réplica possui um DFR próprio com a responsabilidade de monitorar periodicamente seu estado, recebendo também informações dos outros DFRs sobre o estado de todas as réplicas do grupo. Com isso, as trocas de mensagens necessárias para a execução dessa tarefa ficam restritas a cada rede que contenha um grupo. O tempo gasto para realizar todo o processo de detecção de falhas pode ser mensurado e reduzido para cada rede, onde é possível especificar qual o algoritmo de detecção é mais adequado para cada caso, ou seja, que pode levar à redução do tempo de detecção de falhas. A escolha desse algoritmo permite especificar quais os tipos de falhas que serão tratadas pelo DFR. A figura 3 mostra como são organizados os DFRs. Nota-se nessa figura que cada Serviço Web possui um detector de falha associado a ele, onde cada um desses detectores tem a responsabilidade de realizar o monitoramento do estado de cada serviço. Também é possível visualizar na figura 3 que os detectores de falhas possuem ligações entre si, para que a detecção da falha de um serviço seja notificada para todas as outras réplicas do grupo.

Os DFRs são úteis na detecção de falhas internas nos serviços e de falhas de software na plataforma de execução. Juntamente com o DFR, a utilização de sistemas heterogêneos e de múltiplas versões [Xu and Bruck 1998] de um mesmo serviço pode garantir que uma mesma falha não ocorrerá simultaneamente em todas as réplicas. Dentro do processo de detecção de falhas ocorrem duas situações que necessitam de tratamento diferenciado. A primeira situação ocorre quando a falha de alguma das réplicas pertencentes à segunda camada é detectada. Nesse caso, é necessário primeiramente excluir essa réplica do grupo e acionar o mecanismo para recuperação do seu estado, possibilitando que essa réplica possa retornar ao grupo posteriormente. A segunda situação diz respeito à possível falha da réplica que pertence à primeira camada. Em caso de falha dessa réplica, a primeira ação que deve ser tomada é o início de um processo de eleição com a responsabilidade de eleger a réplica pertencente ao grupo que assumirá seu lugar. A réplica eleita utiliza seu CG para notificar as outras réplicas da primeira camada que a partir desse momento ela é a nova representante do grupo, ou seja, todas mensagens trocadas na primeira camada devem ser encaminhadas para ela. Em paralelo com a eleição, assim como no caso anterior, o mecanismo de recuperação é acionado para recuperar o estado da réplica faltosa.

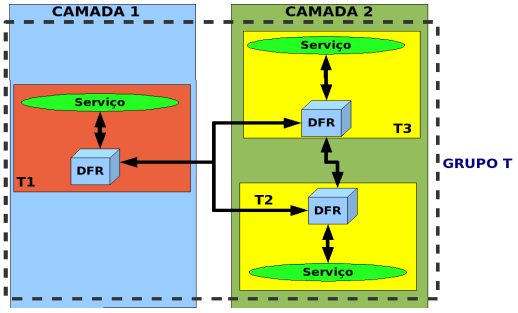

Figura 3. Organização dos DFRs

3.1.2. Detectores de Falha de Grupo - DFG

DFGs têm a responsabilidade de detectar a falha de um grupo inteiro de réplicas. Pode-se observar na figura 2 que os membros da primeira camada realizam a interconexão entre os grupos da arquitetura. Sendo assim, o processo de detecção de falhas inter-grupos tem como principal objetivo a detecção da falha da réplica da primeira camada que representa cada um dos grupos.

Após a detecção da falha dessa réplica, o grupo ao qual ela pertence se torna inacessível; logo, sua falha é extremamente crítica e deve ser tratada pela arquitetura. A figura 4 fornece a visualização da organização dos DFGs. Observa-se na figura 4 que cada uma das réplicas da primeira camada possui dois detectores de falhas, um DFR e outro DFG. DFGs se comunicam constantemente de maneira similar aos DFRs. A diferença dessa comunicação está principalmente no protocolo utilizado, já que grupos podem estar localizados em redes distintas, necessitando de um protocolo de comunicação adequado para ambientes heterogêneos como o SOAP, por exemplo.

Apesar da arquitetura proposta permitir que DFRs e DFGs utilizem protocolos de comunicação distintos, essas duas classes de detectores ainda continuarão possuindo uma relação entre si. Como mencionado anteriormente, a falha de um grupo é detectada tomando como base a falha da réplica da primeira camada que representa o grupo. Relembrando os conceitos apresentados para DFRs, percebe-se que a falha da réplica da primeira camada pode ser detectada tanto por um DFR quanto por um DFG. Quando a falha é detectada pelo DFR, existe a possibilidade da falha de um grupo se tornar transparente para os demais grupos ativos. Essa transparência é alcançada com auxílio da nova réplica, que substitui a réplica faltosa na primeira camada e notifica todos dessa camada que ela é a nova integrante, com responsabilidade de realizar e intermediar toda e qualquer troca de mensagens para seu grupo.

Já quando a falha é detectada pelo DFG, é assumido que o grupo falhou e não receberá mais mensagens, ficando inativo até que uma nova réplica pertencente ao grupo assuma sua liderança e notifique as demais réplicas da primeira camada que o grupo está ativo novamente. Porém, nesse caso, antes de voltar ao funcionamento normal, o grupo deve sincronizar seu estado com o estado dos outros grupos, caso seja necessário. Se a falha detectada por DFGs for originada de uma falha física da rede de comunicação na qual o grupo está alocado, será necessário o acionamento de mecanismos que possibilitem a criação, inicialização e integração de um novo grupo de réplicas ao serviço.

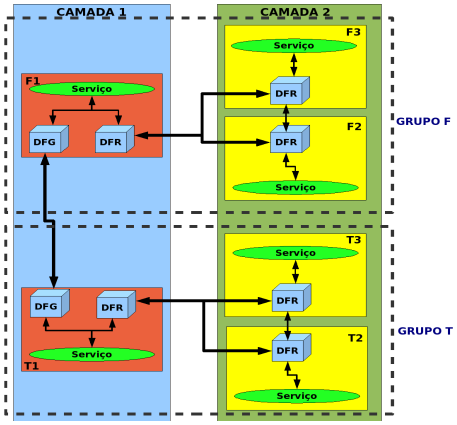

Figura 4. Organização dos DFGs

Falhas na comunicação no domínio no qual um grupo está situado podem deixá-lo inativo por um tempo indeterminado, sendo necessário o procedimento de substituição desse grupo por outro, em outro domínio, de modo a manter o nível de tolerância a faltas fornecido pela arquitetura. Esse procedimento é realizado pela arquitetura sem afetar o funcionamento dos demais grupos, que continuam a atender requisições normalmente.

3.2. Comunicação de Grupo

O componente de comunicação de grupo (CG) é na verdade uma abstração do protocolo de comunicação de grupo utilizado pelas réplicas da arquitetura. Ele contém informações sobre o *membership* e funciona como o canal de comunicação entre todos os membros da arquitetura proposta. Esse componente foi criado com o objetivo de manter a flexibilidade de utilização de diferentes protocolos de comunicação inter-grupos e intra-grupos. Com esse componente a comunicação entre as réplicas é realizada de forma homogênea, apesar de existir a possibilidade de utilização de protocolos de comunicação diferentes, abstraindo dos outros componentes da arquitetura detalhes do protocolo de comunicação utilizado.

3.3. Ordenação de Mensagens

A arquitetura proposta possui um componente para ordenação de mensagens (OM), que tem o objetivo de ordenar as mensagens para manter a consistência de estado entre todas as réplicas do serviço. Esse componente visa garantir a consistência entre as réplicas do serviço, ou seja, certificando que todas as réplicas recebam as mensagens de maneira ordenada, seja para processamento imediato ou posterior. Essa ordenação garante que todas as réplicas ativas compartilharão de um mesmo estado, ou seja, que estarão em um estado consistente. A figura 5 demonstra conceitualmente a base de funcionamento da ordenação de mensagens na arquitetura proposta.

No exemplo ilustrado pela figura 5, dois consumidores estão enviando requisições para um serviço replicado. Após receber as requisições, os componentes OM de cada uma das réplicas pertencentes à primeira camada definem a ordem de execução das requisições recebidas por todas as réplicas da primeira camada. A definição efetiva da ordem de execução das requisições vai depender do algoritmo de ordenação utilizado. Pode-se visualizar na figura 5 que a ordenação das mensagens enviadas pelos consumidores, após

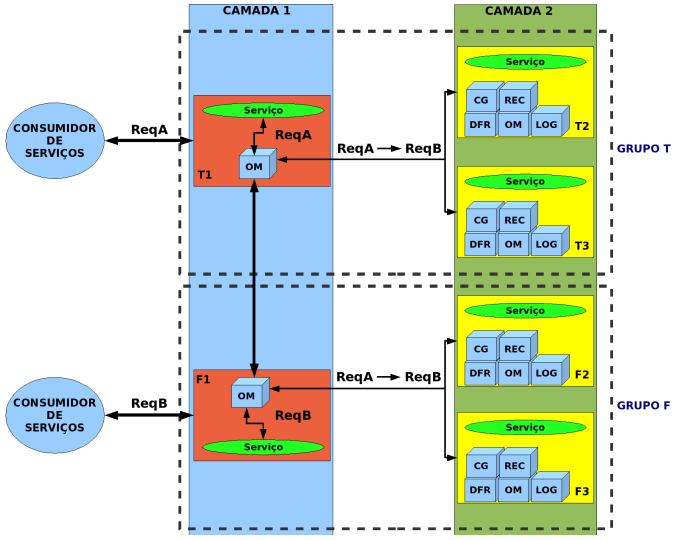

Figura 5. Ordenação de Mensagens

a execução do algoritmo de ordenação, resulta na fila de entrega ordenada 'ReqA→ReqB' em todos os membros, demonstrando conceitualmente a ordenação total das requisições em todas as réplicas da arquitetura. Após definida a ordem, as mensagens ordenadas são encaminhadas pelos líderes para as outras réplicas de cada um dos grupos.

3.4. Mecanismo de Recuperação e Log

O mecanismo de recuperação (REC) depende diretamente do estágio de detecção de falhas. Ou seja, assim que é identificada a falha de uma determinada réplica, o REC dessa réplica deve ser ativado o mais rápido possível, objetivando a recuperação e reintegração dessa réplica faltosa ao grupo. Caso não seja possível realizar essa recuperação, procedimentos para criação e inicialização de uma nova réplica que será integrada ao grupo, substituindo a réplica faltosa, devem ser executados.

Os registros de *log* do serviço se tornam peças fundamentais na etapa de recuperação de falhas. É com base nesses registros que a recuperação de estado de uma réplica é realizada. O componente de log realiza a etapa de armazenamento de logs logo após a etapa de ordenação das mensagens. Essas mensagens devem ser armazenadas em uma fonte de dados confiável, permitindo o acesso futuro às informações persistidas anteriormente.

4. Implementação de Referência

Conforme citado na seção anterior, a arquitetura proposta é uma descrição abstrata dos componentes necessários para o fornecimento de TF para Serviços Web. Essa abstração permite que o desenvolvimento da arquitetura possa ser realizado em diferentes tecnologias.

Uma implementação de referência foi desenvolvida na linguagem Java utilizando a versão 1.6 do *Java Development Kit* (JDK). Sendo assim, toda a descrição abstrata dos componentes da arquitetura foi implementada na forma de classes Java que interagem de acordo com a descrição apresentada anteriormente.

Foram utilizadas diversas APIs (*Application Programming Interface*) fornecidas pela linguagem Java para implementação dos componentes da arquitetura. O JAX-WS (*Java API for XML Web Services*) [Sun 2006], um *framework* para desenvolvimento de Serviços Web, foi empregado para suporte e manipulação de mensagens SOAP. Além disso, um servidor Web leve, fornecido pelo JAX-WS, foi utilizado para processar requisições SOAP sobre o protocolo HTTP (*HyperText Transfer Protocol*), fazendo com que os serviços Web desenvolvidos não necessitem de um servidor de aplicação para serem executados. Manipuladores *(handlers)* foram utilizados para a inserção dos componentes da arquitetura nos Serviços Web desenvolvidos.

5. Análise Comparativa

Depois de realizada a descrição da arquitetura de tolerância a faltas para Serviços Web, é necessário realizarmos uma análise comparativa com os trabalhos relacionados que foram descritos na subseção 2.1. Pela diferença de características de cada um dos trabalhos, se torna inviável a elaboração de uma simples tabela comparativa, sendo necessária a comparação de cada um dos trabalhos com a arquitetura proposta.

O *middleware* para concepção de Serviços Web confiáveis proposto por [Ye and Shen 2005] não suporta consumidores legados, ou seja, esses consumidores legados teriam conhecimento de apenas uma réplica que faz o *multicast* das requisições que recebe para as demais réplicas do serviço, já que o *middleware* atua com base na replicação ativa. Caso essa réplica falhe, os consumidores legados não têm nenhum tratamento dessa falha, deixando assim de usufruir das funcionalidades do serviço. Na arquitetura proposta, esse problema é resolvido pelos Serviços Web presentes na segunda camada. Consumidores legados conhecem somente uma das réplicas pertencentes à primeira camada de replicação. Caso essa réplica falhe, uma outra réplica do seu grupo assume o seu lugar de forma transparente para o consumidor legado. Além disso, a arquitetura proposta possui um componente de *proxy* que pode ser utilizado pelos consumidores do serviço para tolerar falhas que possam ocorrer com o grupo.

Outro problema é em relação à ordenação total das requisições recebidas por consumidores. Como não existe uma réplica central que define a ordem de execução das requisições para as outras réplicas, o aumento do tempo de definição da ordem de execução das requisições é diretamente proporcional ao aumento do número de réplicas do serviço. Em nossa arquitetura, cada grupo possui um líder, sendo que somente esses líderes definem a ordem de execução das mensagens para todas as réplicas. Isso permite que o número de réplicas aumente, sem que o tempo para definição da ordem de execução das requisições aumente de forma proporcional.

Consumidores de serviços que utilizem o SWS [Li et al. 2005] devem ser implementados com auxílio de um pacote específico que possibilita o conhecimento dos grupos de replicação, podendo assim usufruir das vantagens da tolerância a faltas do serviço. Nossa arquitetura fornece tolerância a faltas para o serviço em questão, mesmo que o consumidor do serviço seja um consumidor legado. O SWS também utiliza a idéia de mais de um grupo de replicação, possuindo protocolos específicos de comunicação de grupo e ordenação de mensagens. Porém, para definir a ordem de execução das requisições, todas as réplicas de todos os grupos devem participar dessa definição. A vantagem da nossa arquitetura é que a definição da ordem é realizada somente pelos líderes de cada grupo,

possibilitando um menor tempo de processamento e utilização da rede para realizar essa tarefa.

Outro fator relevante é em relação à comunicação de grupo. Como o SWS utiliza a replicação ativa, todas as réplicas de todos os grupos devem executar as requisições recebidas. Quando um grupo recebe uma requisição de um determinado consumidor, uma réplica do grupo é eleita para encaminhar essa requisição para as demais réplicas dos outros grupos. Porém, cada réplica do outro grupo que recebe essa requisição para processamento deve encaminhar a resposta para todas as réplicas do grupo de origem dessa requisição. Dependendo do volume de mensagens trocadas, essa estratégia pode gerar um tráfego excessivo na rede. Já na arquitetura proposta nesse artigo, toda troca de informações é particionada entre os grupos. Assim que a ordem de execução das requisições é definida, cada líder encaminha a requisição para seu grupo. Se for o caso, cada líder aguarda a resposta das réplicas para, juntamente com os demais líderes, realizar a votação da resposta que será enviada ao consumidor. Isso permite um confinamento do tráfego em cada um dos grupos pertencentes à arquitetura.

No FTWEB [Santos et al. 2005], consumidores do serviços interagem com um componente denominado *WSDispatcher*. Esse componente é responsável pelo gerenciamento das réplicas e pelo sequenciamento das requisições, fornecendo assim características de ordenação total na execução das requisições por parte das réplicas do serviço. Caso o *WSDispatcher* falhe, consumidores legados não têm transparência da falha. Na arquitetura proposta nesse artigo esse problema é resolvido pelos Serviços Web da segunda camada e sua constituição hierárquica com os Serviços Web da primeira camada. Outra desvantagem é a recepção das requisições dos consumidores por um único ponto (*WS-Dispatcher*). Em nossa arquitetura, qualquer réplica pertencente à primeira camada de replicação pode receber requisições de consumidores do serviço, permitindo um balanceamento natural das requisições recebidas pela arquitetura. O FT-SOAP [Fang et al. 2006] apresenta os mesmos aspectos negativos presentes no FTWEB.

Por fim, o WS-Replication [Salas et al. 2006] tem o objetivo de ser utilizado para replicação de serviços em uma WAN. Consumidores legados têm conhecimento de apenas uma das réplicas do WS-Replication, portanto, em caso de falha dessa réplica, o *framework* não consegue tolerar essa falha, justamente pelo fato das réplicas formarem um grupo único distribuído em uma WAN. A arquitetura proposta nesse artigo pode ser utilizada perfeitamente em uma WAN. Nesse tipo de rede, cada um dos grupos é instanciado em uma rede distinta, sendo que somente os líderes se comunicam pela WAN. Isso permite que a arquitetura ofereça tolerância a faltas para consumidores legados, já que estes acessam uma das réplicas pertencentes à primeira camada e a falha dessa réplica é tolerada pelas réplicas do seu grupo, que pertencem à segunda camada de replicação. Além disso, os consumidores podem ter falhas de grupo toleradas utilizando um componente de *proxy* da arquitetura proposta.

6. Conclusão e Trabalhos Futuros

Esse artigo apresentou uma proposta de arquitetura de *software* para fornecimento de tolerância a faltas em Serviços Web. Essa arquitetura descreve de forma abstrata todos os componentes que fazem parte da solução, possibilitando assim a compatibilidade com as especificações e padrões utilizados pelos Serviços Web.

A arquitetura proposta, também permite que grupos de replicação sejam desenvolvidos em diferentes tecnologias, incorporando características da programação N-Versão [Xu and Bruck 1998]. O fato das réplicas serem alocadas em 2 camadas e em diversos grupos de replicação permite uma maior flexibilidade e confinamento do tráfego da rede, possibilitando que um menor número de mensagens sejam trocadas entre as réplicas do serviço. A subdivisão em grupos de replicação permite ainda um fornecimento mais adequado de tolerância a faltas do serviço para consumidores legados, fazendo com que serviços incorporem propriedades de tolerância a faltas e forneçam essa vantagem para consumidores do serviço já existentes.

Pela constituição da arquitetura, a replicação de Serviços Web em WANs se torna factível, sendo possível a distribuição de grupos em diferentes redes. A comunicação entre os grupos é realizada somente entre seus líderes, diminuindo o tráfego de informações em uma rede de larga escala, na qual o tempo de resposta é um fator crítico para aplicações distribuídas.

Por fim, tem-se diversos trabalhos futuros a serem realizados na arquitetura proposta neste artigo, tais como a elaboração e execução de testes de desempenho para verificar o *overhead* causado pela arquitetura proposta, a inserção de características de qualidade de serviço (QoS) para solucionar problemas tais como balanceamento de carga e tempo de resposta, a descoberta dinâmica de novos grupos de replicação, a adaptação da arquitetura para fornecimento de tolerância a faltas para Serviços Web em grids computacionais e a implementação de diversos algoritmos de ordenação e detecção de falha para que a arquitetura possa realizar o tratamento dos mais variados tipos de falhas que possam ocorrer em sistemas distribuídos.

Referências

Aghdaie, N. and Tamir, Y. (2002). Implementation and evaluation of transparent fault-tolerant web service with kernel-level support. In *11th IEEE International Conference on Computer Communications and Networks*, Miami, Florida, USA. IEEE Computer Society.

Ayala, D., Browne, C., Chopra, V., Sarang, P., Apshankar, K., and McAllister, T. (2002). *Professional Open Source Web Services*. Wrox Press Ltd.

Cerami, E. (2002). *Web Services Essentials: Distributed Applications with XML-RPC, SOAP, UDDI and WSDL*. O'Reilly.

Chandra, T. D. and Toueg, S. (1996). Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267.

Chockler, G. V., Keidar, I., and Vitenberg, R. (2001). Group communication specifications: A comprehensive study. *ACM Computing Surveys*, 33(4):427–469.

Dialani, V., Miles, S., Moreau, L., Roure, D. D., and Luck, M. (2002). Transparent fault tolerance for web services based architectures. In *8th International 12 Europar Conference (EURO-PAR'02)*, Padeborn, Germany.

Fang, C.-L., Liang, D., Lin, F., and Lin, C.-C. (2006). Fault tolerant web services. *Journal of System Architecture*.

He, W. (2004). Recovery in web service applications. In *IEEE International Conference on e-Technology, e-Commerce and e-Service*, pages 25–28, Taipei, Taiwan. IEEE Computer Society.

Jalote, P. (1994). *Fault tolerance in distributed systems*. Prentice-Hall, Inc, Upper Saddle River, NJ, USA.

Li, W., He, J., Ma, Q., Yen, I.-L., Bastani, F., and Paul, R. (2005). A framework to support survivable web services. In *19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05)*, volume 01, page 93b, Denver, Colorado, USA. IEEE Computer Society.

Newcomer, E. (2002). *Understanding Web Services: XML, WSDL, SOAP, and UDDI*. Addison-Wesley Professional.

Omg (2002). *Common Object Request Broker Architecture: Core Specification Chapter 23*. OMG. http://www.omg.org.

Salas, J., Perez-Sorrosal, F., Patiño-Martinez, M., and Jímenez-Peris, R. (2006). Ws-replication: A framework for highly available web services. In *15th International World Wide Web Conference*, Edinburgh, Scotland.

Santos, G. T., Lung, L. C., and Montez, C. (2005). Ftweb: A fault tolerant infrastructure for web services. In *9th IEEE International EDOC Enterprise Computing Conference (EDOC'05)*, pages 95–105, Enschede, Netherlands. IEEE Computer Society.

Soap (2003). *SOAP Specification Version 1.2*. W3C. http://www.w3.org/TR/soap12-part1/.

Sousa, A., Pereira, J., Moura, F., and Oliveira, R. (2002). Optimistic total order in wide area networks. In *21st IEEE Symposium on Reliable Distributed Systems (SRDS'02)*, page 190, Suita, Japan. IEEE Computer Society.

Sun (2006). *Java API for XML Web Services*. Sun Microsystems, Inc. https://jax-ws.dev.java.net/jax-ws-ea3/docs/.

Uddi (2004). *Universal Description Discovery and Integration Specification 3.0.2*. OASIS. http://uddi.org/pubs/uddi_v3.htm.

Veríssimo, P. and Rodrigues, L. (2001). *Distributed Systems for System Architects*. Kluwer Academic Publishers.

WS-Reliability (2004). *WS-Reliability Specification Version 1.1*. OASIS. http://docs.oasis-open.org/wsrm/ws-reliability/v1.1/wsrm-ws_reliability-1.1-spec-os.pdf.

Wsdl (2001). *Web Services Description Language (WSDL) 1.1*. W3C. http://www.w3.org/TR/wsdl.

Xu, L. and Bruck, J. (1998). Deterministic voting in distributed systems using error-correcting codes. In *IEEE Transactions on Parallel and Distributed Systems*, volume 09, pages 813–824. IEEE Computer Society.

Ye, X. and Shen, Y. (2005). A middleware for replicated web services. In *IEEE International Conference on Web Services (ICWS'05)*, pages 631–638, Orlando, Florida, USA. IEEE Computer Society.