Uma Infra-estrutura para Replicação Semi-Ativa em Arquiteturas Orientadas a Serviços[1]

Giuliana Teixeira Santos, Lau Cheuk Lung

[1]Programa de Pós-Graduação em Informática Aplicada - PPGIA
Pontifícia Universidade Católica do Paraná – PUCPR

and

Joni da Silva Fraga, Carlos Barros Montez
DAS – Departamento de Automação e Sistemas
UFSC – Universidade Federal de Santa Catarina
Campus Universitário, Caixa Postal 476 – CEP 88040-900 – Florianópolis – SC.

Abstract

The web services architecture appeared as a reply to the interoperability search between applications. The last years has seen an increase interest in executing in the Internet environment applications with high availability and reliability requirements. However the technologies associates to this architecture still do not offer adequate support to these requirements. The proposed infrastructure in this paper is inserted in this context and provides a new software layer that acts as a proxy between the client request and suppliers services. The main goal is to guarantee transparent fault tolerance for the customer through semi-active replication technique. This model supports the following faults: value, crash and omission. The characteristics and the results gotten with the infrastructure implementation are described in elapsing of this paper.

Keywords: Distributed Systems, Web services, Fault-Tolerance, FT-CORBA.

Resumo

A arquitetura de serviços web surgiu como uma resposta à busca da interoperabilidade entre aplicações. Nos últimos anos existe um interesse crescente em executar na Internet aplicações com requisitos de alta disponibilidade e confiabilidade, contudo as tecnologias associadas a essa arquitetura ainda não oferecem suporte adequado a esses requisitos. A infra-estrutura proposta, neste artigo, se situa neste contexto e provê uma nova camada de software que atua como um proxy entre as requisições do cliente e os serviços nos provedores. O objetivo principal é garantir tolerância a faltas transparente para o cliente através da técnica de replicação semi-ativa. Este modelo suporta as seguintes faltas: valor, omissão e parada. As características e os resultados obtidos com a implementação desta infra-estrutura são descritos no decorrer deste artigo.

Palavras chaves: Sistemas Distribuídos, Web services, Tolerância a Faltas, FT-CORBA.

1. INTRODUÇÃO

A palavra-chave dos serviços web é interoperabilidade, componentes de software podem ser acessados através de protocolos consolidados e amplamente utilizados como o HTTP (*Hypertext Transfer Protocol*) e o XML (*Extensible Markup Language*) [XML, 2000]. A sua principal vantagem está em permitir a integração de componentes já desenvolvidos, esta flexibilidade permite explorar as melhores características de cada tecnologia envolvida no processo de desenvolvimento de uma aplicação.

Por possuírem uma arquitetura aberta, os serviços web têm se mostrado uma excelente opção para a programação de sistemas distribuídos, permitindo o desenvolvimento de soluções adaptadas à heterogeneidade e complexidade

[1] Parcialmente financiado pelo CNPq Projeto Universal 019/2004 Nro. 481523/2004-9 e Fundação Araucária FA-6651/04.

destes ambientes. Contudo, para que todo o potencial dos serviços web possa ser explorado faz-se necessário a definição de uma infra-estrutura de desenvolvimento que atenda os requisitos de confiabilidade e alta disponibilidade. Esta infra-estrutura deve ser flexível o suficiente para que todas as características dos serviços web sejam mantidas.

Ainda são poucos os trabalhos que endereçam os requisitos de tolerância a faltas para os serviços web. O principal problema encontrado para a proposição de uma infra-estrutura tolerante a faltas nestes sistemas se concentra no fato dos servidores web não manterem uma conexão ativa durante todas as requisições do cliente e, como conseqüência, serem *stateless*. Por este motivo, aplicações críticas construídas sobre os protocolos Internet utilizam técnicas simplificadas. Por exemplo, usam mecanismos básicos que detectam a falha e direcionam futuras requisições para servidores redundantes. Estes mecanismos não são capazes de tolerar falta durante o processamento de uma requisição.

Atualmente não existem especificações padrão que tratem a tolerância a faltas nos serviços web. As poucas propostas encontradas na literatura [Dialani et. al., 2002], [Aghdaie, Tamir 2002], [Deron et. al., 2003] provêm à tolerância a faltas em serviços web baseado na abordagem de replicação passiva [Budhiraja et. al., 1993] ou implementam mecanismos básicos de detecção de falha do primário e ativação de um servidor backup. Nestas propostas a replicação dos serviços esta limitada a um único domínio e, como conseqüência, são vulneráveis a falhas em roteadores, *gateways* e outros componentes que realizam interfaces com a rede.

O presente artigo apresenta uma proposta de utilização da técnica de replicação semi-ativa para alcançar a tolerância a faltas em arquiteturas orientadas a serviços. O modelo proposto é baseado em uma infra-estrutura denominada de *FTWeb* que permite que os serviços estejam replicados e distribuídos sobre a Internet. Esta infra-estrutura possui componentes responsáveis por invocar concorrentemente as réplicas do serviço, aguardar o processamento, analisar as respostas processadas, e retornar a resposta ao cliente. Estes componentes são baseados nos modelos e conceitos do padrão FT-CORBA da OMG [OMG, 2002] para o desenvolvimento de aplicações distribuídas e tolerantes a faltas. O objetivo desta abordagem é prover tolerância nas seguintes classes de faltas: parada, omissão e valor.

Este artigo está organizado da seguinte forma: a seção 2 apresenta o modelo conceitual e os padrões que compõem os serviços web. A seção 3 apresenta uma visão geral da especificação FT-CORBA da OMG, e na seção 4 é apresentado a infra-estrutura *FTWeb*. Na seção 5 são apresentados os aspectos referentes à implementação. A avaliação de desempenho do FTWeb é apresentada na seção 6. A seção 7 discute os trabalhos relacionados e, por fim, na seção 8 o artigo é concluído.

2. SERVIÇOS WEB

Os serviços web [WS-ARCH, 2004] são identificados por um URI (*Unique Resource Identifier*), e são descritos e definidos usando XML. A Figura 1 apresenta o modelo conceitual dos serviços web.

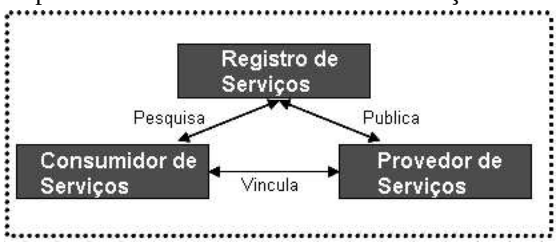

Figura 1. Modelo Conceitual dos Serviços Web.

O provedor de serviços é a entidade responsável pela publicação de um serviço web em um registro de serviços. Qualquer cliente que utilize um serviço web criado por um provedor é chamado de consumidor de serviços. Usualmente, o consumidor realiza uma pesquisa sobre o registro onde o provedor publicou a descrição do seu serviço. A partir desta descrição, o consumidor pode obter do servidor o mecanismo para vínculo (*binding*), podendo então executar o serviço web. Um registro de serviços é a localização central onde o provedor pode relacionar seus serviços web. Através do registro central, consumidores podem encontrá-los e em seguida, vinculá-los.

Para obter a comunicação entre as aplicações sem considerar detalhes de sua implementação é necessário que cada uma das operações realizadas pelas entidades sejam padronizadas. Visando obter a interoperabilidade foram criados os seguintes padrões: a *Web Service Description Language* é um padrão que utiliza XML para descrever serviços web. Basicamente, o documento WSDL define os métodos que estão presentes no serviço, os parâmetros de entrada/saída para cada um dos métodos, os tipos de dados, o protocolo de transporte e a URL do local onde o serviço web está hospedado. O padrão *Universal Description, Discover, and Integration* permite que os provedores publiquem detalhes sobre seus serviços web em um registro central. Também fornece um padrão para permitir que os

consumidores localizem os provedores e os serviços web por eles oferecidos. O *Simple Object Access Protocol* é um protocolo, baseado em XML, usado para trocar informações entre aplicações, independentemente do sistema operacional, da linguagem de programação ou do modelo do objeto.

3. A ESPECIFICAÇÃO FT-CORBA

O suporte de tolerância a faltas para aplicações desenvolvidas segundo o modelo de objetos distribuídos CORBA é especificado conforme o padrão *Fault Tolerant* CORBA (FT-CORBA [OMG, 2002]). Esta especificação define um conjunto de interfaces de serviços para a implementação de técnicas de replicação em ambientes distribuídos e heterogêneos. Os objetos de serviço que fornecem as funcionalidades básicas para a construção de aplicações distribuídas tolerantes a faltas são [OMG, 2002]: *Serviço de Gerenciamento de Replicação, Serviço de Gerenciamento de Falhas* e *Serviço de Recuperação e Logging*.

O *Serviço de Gerenciamento de Replicação* (SGR) interage diretamente com o *Serviço de Gerenciamento de Grupos de Objetos*, atuando de forma dinâmica na entrada e saída de objetos replicados. No processo de criação e remoção de réplicas utiliza o objeto *Fábrica Genérica* que interage com os objetos *Fábrica Locais* responsáveis pela criação e remoção das réplicas nas estações que compõem o sistema distribuído. O *Serviço de Gerenciamento de Propriedades* é responsável por definir as propriedades de tolerância a faltas de cada grupo de objetos. Este serviço define ao SGR como cada grupo deve ser gerenciado.

O *Serviço de Gerenciamento de Falhas* (SGF) apresenta as interfaces dos serviços para a monitoração e notificação de falhas. A detecção de falhas é realizada em três níveis: servidor, objeto e processo. Estes detectores são baseados em mecanismos de *timeout*. O *Serviço Notificador de Falhas* tem a função de notificar ao SGR as falhas registradas pelos detectores. Através desta notificação, o SGR mantém consistente a lista de membros dos grupos.

O *Serviço de Recuperação e Logging* tem como principal objetivo registrar as requisições recebidas pelo servidor, manter consistente o estado das réplicas e efetuar os procedimentos de recuperação em réplicas faltosas.

4. DESCRIÇÃO DA INFRA-ESTRUTURA FTWEB

A idéia fundamental do *FTWeb* é a utilização da técnica de replicação semi-ativa para alcançar a tolerância a faltas em arquiteturas orientadas a serviço. As réplicas de um determinado serviço são organizadas em um grupo e todas as réplicas livres de falhas recebem, executam e respondem as requisições enviadas pelos clientes. Para implementar a ordem FIFO, necessária a replicação semi-ativa, foi utilizada a abordagem do seqüenciador [Defago, Shiper 2000] por ser a mais adequada às características dos serviços web e, em termos algorítmicos, menos complexo que outras encontradas na literatura.

Através desta abordagem é possível replicar os objetos em servidores dispersos geograficamente (em diferentes domínios) e delegar a sua administração à infra-estrutura *FTWeb* através de uma interface bem definida. Esta infra-estrutura absorve as funcionalidades providas pelo *FT-CORBA* e fornece componentes que invocam objetos CORBA na forma de serviços web. O *FTWeb* é divido em 3 diferentes módulos: *WSClient Driver* que atua no domínio da aplicação, *WSDispatcher* responsável pelo gerenciamento das réplicas e execução do serviço e *WSWrapper* utilizado para integração com objetos CORBA. Estes módulos são apresentados na Figura 2.

WSClient Driver

O módulo *WSClient Driver* é responsável em detectar uma falha no componente *WSDispatcher Engine* e transferir o processamento da requisição ao *WSDispatcher Engine Backup* localizado em um servidor independente. Este componente foi definido como um interceptador na camada SOAP e fica localizado no cliente. Esta abordagem tem como objetivo prover a tolerância a faltas de forma transparente para os clientes da aplicação, evitando assim que o *WSDispatcher Engine* seja um ponto crítico de falha.

Caso o *WSDispatcher Engine* falhe após o processamento da requisição, ou seja, no momento da entrega da resposta ao cliente, o componente *WSClient Driver* irá transferir a requisição ao *WSDispatcher Engine Backup* que invocará os serviços replicados. Através de mecanismos de *log* contidos nas réplicas é possível verificar se a requisição já foi processada e então as réplicas simplesmente retornam a resposta ao *WSDispatcher Engine Backup*. Outros mecanismos como "Máquina de Estados Finitos" [Fred Schneider 1990] podem ser utilizados para evitar o re-processamento de requisições em caso de falhas dos componentes intermediários. A Figura 2 apresenta o fluxo normal e o fluxo em caso de falhas no *WSDispatcher Engine*.

Replicação Semi-Ativa

Na replicação semi-ativa todas as réplicas livres de falhas recebem e executam a requisição, entretanto apenas a réplica líder é responsável pelo retorno das mensagens ao cliente. Esta abordagem, utilizando a infra-estrutura *FTWeb*, pode ser implementada da seguinte forma: a requisição é submetida pelo cliente e o *WSDispatcher Engine* redireciona a

execução para todas as réplicas, no entanto aguarda apenas a resposta da réplica líder (Figura 2). Neste modelo não é necessário mecanismo de log no *WSDispatcher Engine*, assim como na abordagem replicação ativa. Desta forma o *WSDispatcher Engine* e o *WSDispatcher Engine Backup* podem estar localizados em redes distintas.

Em caso de falhas na réplica líder, o mesmo protocolo utilizado na replicação passiva para escolha do primário, é utilizado neste modelo para definir o novo líder. Os mecanismos de recuperação, detecção de falhas e gerenciamento de réplicas são utilizados da mesma forma que na replicação ativa. Assim como na replicação passiva o sistema de configuração deve possuir na sua interface um mecanismo para a definição da réplica líder. As funções de seqüenciador do componente *WSInvoker* são necessárias nesta abordagem a fim de garantir o determinismo entre as réplicas. O componente Response Analizer não é utilizado neste modelo.

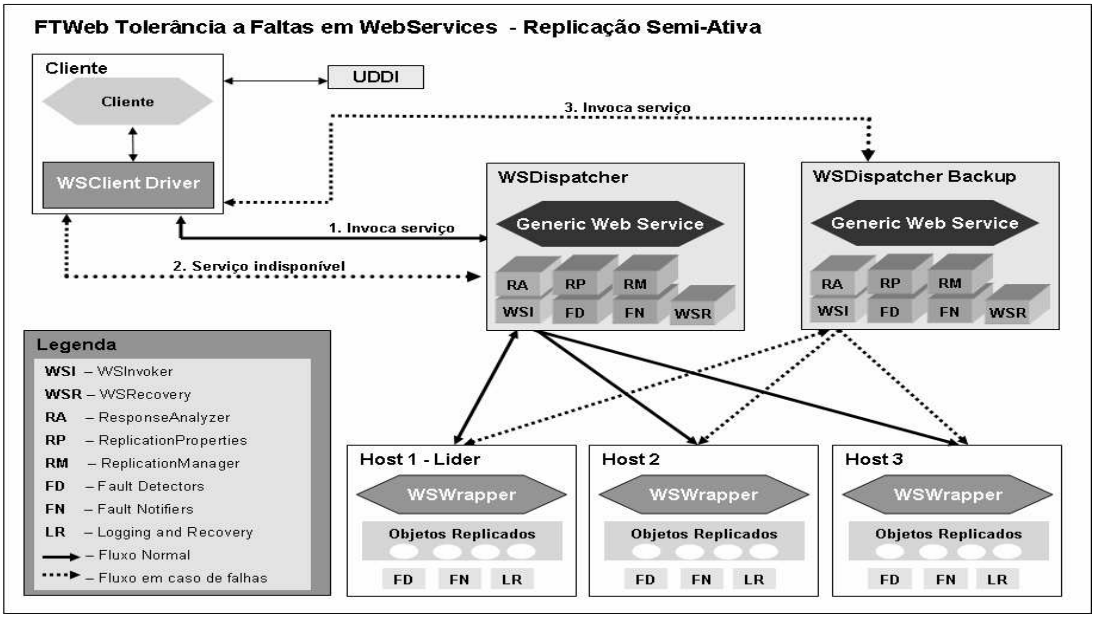

Figura 2 – Replicação semi-ativa com a infra-estrutura FTWebWSDispatcher Engine

O *WSDispatcher Engine* é o módulo central da infra-estrutura *FTWeb* e possui os mecanismos responsáveis por gerenciar réplicas, invocar concorrentemente as réplicas do serviço, analisar as respostas processadas, detectar e iniciar o processo de recuperação do estado em réplicas faltosas. O *WSDispatcher* é formado pelo conjunto de componentes descritos a seguir:

Generic Web Service

O componente *Generic Web Service* (Figura 2) é um serviço web genérico responsável em obter do cliente a referência do serviço web e os parâmetros necessários para a sua execução. Após a execução, este componente é encarregado de retornar a resposta obtida ao cliente. A sua utilização faz com que o cliente perceba como um único serviço, um conjunto de serviços replicados, independentes e dispersos geograficamente.

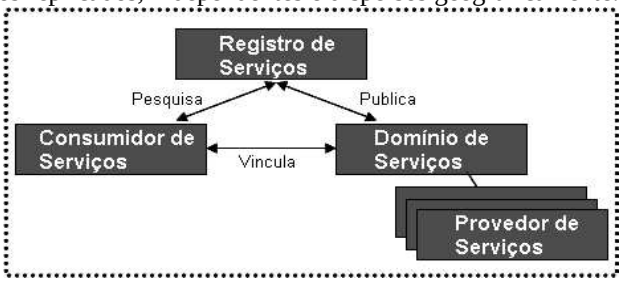

Figura 3. Domínio de Serviços.

Para compor os grupos, necessários nas abordagens de replicação, foi utilizado o conceito de domínio de serviços [Tan et. al., 2004]. Um domínio de serviços permite a agregação e o compartilhamento de múltiplas descrições de serviços web (WSDL). As informações de ligação referem-se ao grupo, permitindo que vários serviços sejam

virtualizados como um único serviço. Podem ser atribuídas regras ao domínio para administrar e controlar o comportamento dos serviços agregados. A Figura 3 exibe a diferença entre o modelo de domínio de serviços e o modelo convencional de serviços web (Figura 1).

O *WSDispatcher Engine* possui um sistema de configuração (*Configuration System*) onde o administrador do serviço realiza a criação do grupo e indica através dos documentos WSDL as réplicas que farão parte do grupo. Neste sistema também são definidas as propriedades de replicação e gerenciamento de falhas.

WSInvoker

Este componente é apresentado na Figura 4. O seu funcionamento e a integração com os demais componentes podem ser descritos através de uma seqüência de 5 passos:

1. O cliente invoca o *Generic Web Service* informando a referência do grupo do serviço, o método a ser executado e os parâmetros necessários para a sua invocação;
2. O componente *Generic Web Service* invoca o *WSInvoker* e passa as informações obtidas do cliente;
3. *WSInvoker* interage com os componentes *Replication Manager* e *Replication Properties* para obter a localização das réplicas e as propriedades de tolerância a faltas definidas para o grupo;
4. O *WSInvoker* difunde a requisição, com garantia de ordenação FIFO, nas réplicas do serviço e gerencia a sua execução;
5. Após obter as respostas de todas as réplicas o *WSInvoker* invoca o componente *Response Analizer*, que realiza uma votação entre as respostas obtidas. O *WSInvoker* retorna ao *Generic Web Service* a resposta indicada pelo *Response Analizer*.

O *WSInvoker* atua como um seqüenciador e assegura o determinismo entre as réplicas através de ordenação atômica. Isso implica em um sincronismo na execução do serviço, o mesmo serviço não pode ser executado por mais de um cliente ao mesmo tempo. Através do determinismo alcançado pelo módulo *WSDispatcher Engine* é possível a sua utilização em serviços web *statefull*, ou seja, serviços que mantêm informações de seu estado durante as requisições do cliente.

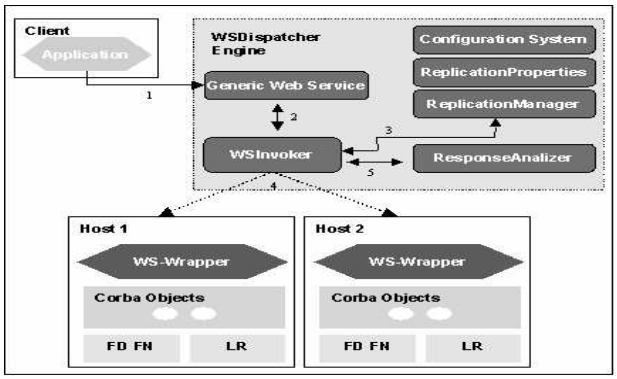

Figura 4. Funcionamento do WSInvoker.

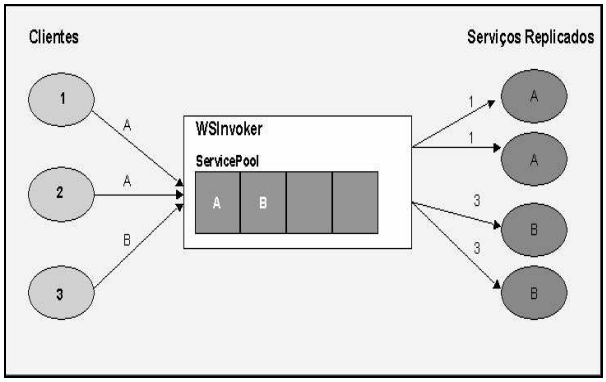

Figura 5 - Funcionamento do Seqüenciador

A Figura 5 apresenta o funcionamento do seqüenciador utilizado neste modelo. O *WSInvoker* possui uma estrutura denominada de *ServicePool*, nesta estrutura ficam registrados apenas os serviços em execução a fim de garantir que o serviço não seja acessado por dois clientes ao mesmo tempo. A Figura 5 apresenta 3 clientes requisitando a utilização de serviços. Os clientes representados nesta figura como 1 e 2 submetem requisições para o serviço A. O cliente 3 submete a requisição ao serviço B. Quando as requisições são repassadas ao *WSInvoker*, a estrutura *ServicePool* garante que apenas um cliente acessará o serviço enquanto o outro cliente aguardará a sua liberação. No exemplo representado pela Figura 5 o cliente 2 aguarda o término do processamento da requisição do cliente 1 e o cliente 3 acessa o serviço paralelamente. Caso outro cliente tente acessar o serviço B deverá obrigatoriamente aguardar o término do processamento da requisição realizada pelo cliente 3.

O repasse da requisição do cliente pelo *WSInvoker* é feito através de difusão confiável com ordem FIFO implementado através de múltiplas invocações ponto-a-ponto. Esta invocação é baseada na especificação *WS-Reliable Message* [WS-RM, 2004] na qual define uma extensão ao protocolo SOAP para que este suporte comunicações ponto-a-ponto confiável.

Se uma réplica apresenta falha no momento da sua execução ou não responde no tempo limite estabelecido nas configurações do serviço, o componente *WSInvoker* ativa os mecanismos de notificação para que o *ReplicationManager* retire a réplica faltosa do grupo do serviço. Neste caso, a réplica faltosa, fica fora do grupo até que esta tenha o seu estado restabelecido através dos mecanismos de recuperação. O *WSInvoker* não atua no controle

transacional, se todas as réplicas apresentarem falhas o cliente deve possuir a lógica necessária para realizar a compensação de transações.

Response Analizer

O componente *Response Analizer* é opcional (Figura 4) e atua como um votador. Após a execução de todas as réplicas, o componente *WSInvoker* delega ao componente *Response Analizer* a análise de todas as respostas obtidas. A resposta com mais ocorrências é assumida. Este componente pode ser utilizado para tolerar falta de valor.

Replication Manager

O componente *Replication Manager* estende as funcionalidades de gerenciamento de réplicas do FT-CORBA para os serviços web. Este componente controla dinamicamente a adição de novas réplicas e a remoção de réplicas faltosas segundo as regras definidas no *Replication Properties*.

Replication Properties

Este componente realiza o mapeamento das propriedades de tolerância a faltas definidas no *FT-CORBA* para a infra-estrutura *FTWeb*. Como mencionado anteriormente, o *WSDispatcher Engine* possui um sistema de configuração (apresentado na Figura 4, como *Configuration System*) que permite ao administrador do serviço definir as propriedades de replicação e gerenciamento de faltas. Através do sistema de configuração o componente *Replication Properties* obtém estas propriedades no formato XML. Estas propriedades são definidas como:

- *Replication Style:* define o estilo de replicação como replicação passiva fria, replicação passiva quente ou replicação semi-ativa.
- *Monitoring Style*: define o estilo de monitoração como PULL ou PUSH. No estilo PULL o detector de falhas envia periodicamente mensagens para o objeto monitorado verificando se ele está ativo. No estilo PUSH, o objeto réplica envia mensagens periodicamente ao detector de falhas indicando que está ativo;
- *Monitoring Interval And Timeout*: define o intervalo de monitoramento (*ping*) e o tempo máximo de resposta (*timeout*) do serviço monitorado para determinar se está faltoso;
- *Response Timeout*: define o tempo de resposta (*timeout*) do serviço quando invocado pelo *WSInvoker*;
- *Recovery*: indicador do processo de recuperação do serviço. O estado do serviço pode ser recuperado automaticamente através de mecanismos fornecidos pela infra-estrutura *FTWeb* ou manualmente pelo administrador.

Fault Detector e Fault Notifier

Estes componentes estendem as funcionalidades de detecção e notificação de falhas do FT-CORBA para os serviços web. Para que um serviço web seja monitorado é necessário que ele implemente a interface *PullMonitorable* que contém o método *isAlive()*. Através da invocação deste método o componente *Fault Detector* monitora as réplicas. A monitoração é realizada conforme as propriedades obtidas através do *Replication Properties* e definidas no sistema de configuração. Quando ocorre uma falha o componente *Fault Notifier* recebe uma notificação de falha do *Fault* Detector. O *Fault Notifier* notifica o *Replication Manager* que retira a réplica faltosa do grupo do serviço web. A Figura 6 apresenta o gerenciamento de faltas da infra-estrutura *FTWeb*.

WSRecovery

Este componente é responsável pela recuperação do estado de réplicas faltosas. O *WSDispatcher Engine* possui uma console de monitoração que exibe todas as réplicas que apresentaram falhas durante a requisição de uma transação ou durante o processo de monitoração. Esta console permite ao administrador do serviço, iniciar o processo de recuperação de uma ou mais réplicas. O próprio administrador pode informar o estado do serviço em caso de faltas em todas as réplicas ou iniciar o processo de recuperação apenas das réplicas faltosas, este processo é denominado de recuperação manual.

Na recuperação automática o *WSRecovery* verifica periodicamente as réplicas faltosas, obtém o estado das réplicas não faltosas e através do mecanismo de votação definido pelo componente *Response Analizer*, restabelece o estado da réplica que apresentou a falta. O funcionamento deste componente é similar ao *WSInvoker* (Figura 4) no entanto a sua funcionalidade é invocada a partir da console de monitoração ou através da notificação do componente *Fault Detector* ao detectar uma réplica faltosa.

Tanto o processo de recuperação manual quanto o automático concorrem com os clientes na execução do serviço web, ou seja, o serviço web somente é liberado para ser utilizado pelos clientes após o término do processo de recuperação. Quando a réplica que apresentou a falta tem o seu estado re-estabelecido, esta é inserida novamente pelo componente *Replication Manager* no grupo do serviço web correspondente.

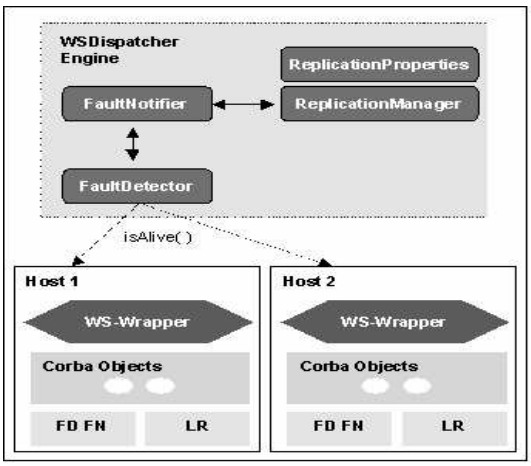

Figura 6. Gerenciamento de Faltas.

WSWrapper

Para explorar a integração entre as tecnologias CORBA e serviços web foi construído o módulo *WSWrapper* que realiza a interface entre o módulo *WSDispatcher Engine* e os objetos que processarão no provedor as requisições dos clientes. Este componente foi baseado nos modelos definidos em [Gokhale et. al 2003][Jandl et. al 2003]. Através deste componente, requisições SOAP são convertidas em invocações a objetos CORBA. O *WSWrapper* utiliza a interface de invocação dinâmica para invocar os objetos, podendo ser utilizado para executar qualquer objeto CORBA no provedor do serviço. Através desta abordagem é possível replicar os objetos em servidores dispersos geograficamente e delegar a sua administração ao *WSDispatcher Engine*.

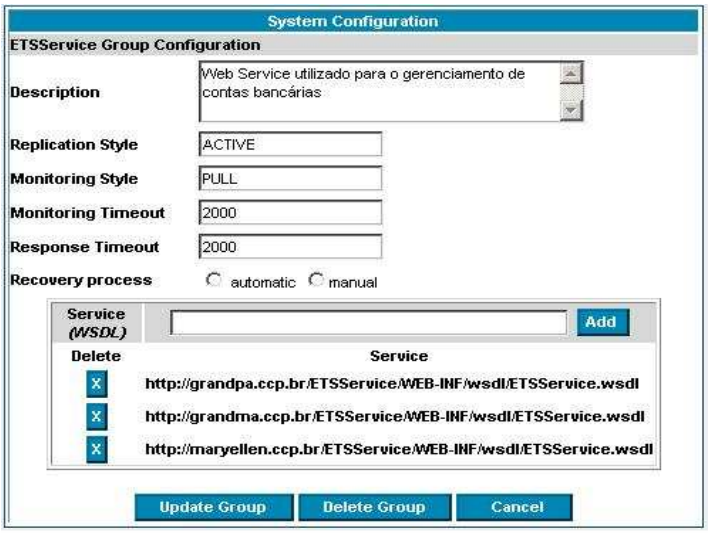

Figura 7. Sistema de Configuração.

5. IMPLEMENTAÇÃO DO FTWEB

A implementação da infra-estrutura *FTWeb* foi realizada utilizando a linguagem Java JDK 1.4.2, GroupPac 1.4 e JacORB 1.4. O servidor de aplicações utilizado foi *IBM WebSphere Application Server 5.1* e todas as APIs utilizadas neste ambiente estão em conformidade com os padrões de interoperabilidade definidos pelo *WS-I Basic Profile 1.0* [WS-I, 2004]. Os componentes que formam o módulo central *WSDispatcher Engine*, podem ser classificados segundo a forma em que são implementados como: componentes implementados na forma de uma aplicação J2EE e instalados no servidor de aplicações tais como o *Generic Web Service*, o *WSInvoker* e o *Response Analyzer*. Objetos CORBA implementados sob a especificação *FT-Corba*. *ReplicationManager*, *ReplicationProperties*, e *Fault Detector*. O diagrama de classes contendo os principais componentes do *WSDispatcher* pode ser visto na Figura 8. Em uma visão geral a relação entre esses objetos, é descrita a seguir:

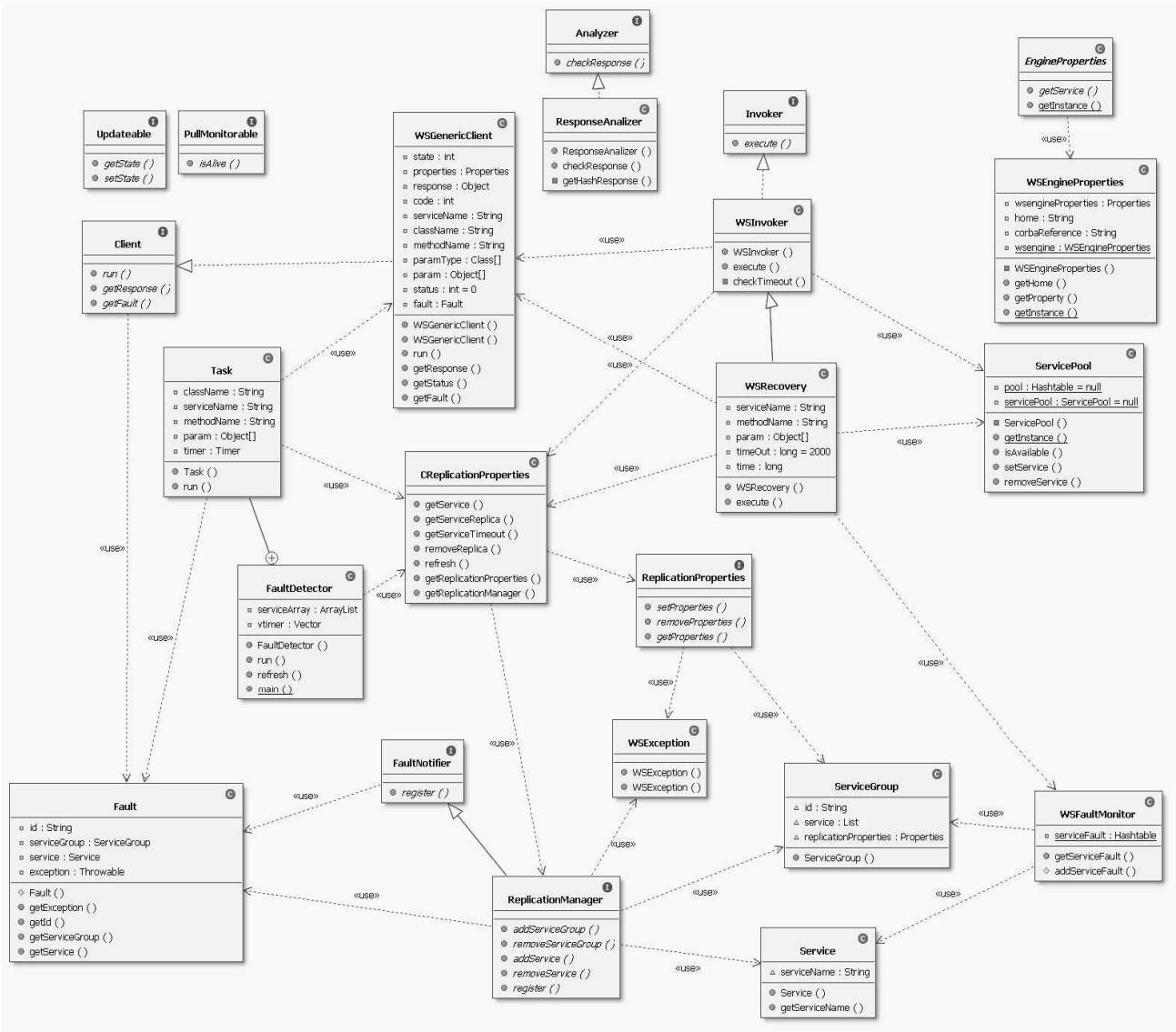

Figura 8 – Diagrama de Classes WSDispatcher Engine

- A classe WSEngineProperties define as implementações que serão utilizadas dos componentes WSGenericClient, WSInvoker e ResponseAnalizer estas classes possuem interfaces definidas permitindo que as suas funcionalidades sejam reimplementadas e adaptadas conforme os requisitos dos serviços nos provedores.
- A classe ServicePool é utilizado pelo WSInvoker para implementar as funcionalidades do seqüenciador de requisições.
- A classe CReplicationProperties é utilizada pelo WSInvoker e WSRecovery e tem a responsabilidade de obter as informações gerenciadas pelas classes ReplicationManager e ReplicationProperties. Desta forma os objetos implementados sob a tecnologia CORBA são abstraídos dos componentes que formam a aplicação J2EE.
- A classe WSRecovery estende as funcionalidades da classe WSInvoker e reimplementa o método execute a fim de realizar a recuperação do estado de réplicas faltosas. Esta classe utiliza a classe FaultMonitor para obter a relação das réplicas que apresentaram falhas.
- A classe WSException é utilizada pelas classes ReplicationProperties e ReplicationManager para o lançamento de exceções em suas operações
- A classe EngineException é utilizada pelas classes WSInvoker, ResponseAnalyser, WSGenericClient para o lançamento de exceções em suas operações.
- A classe WSGenericClient é responsável por concretizar a invoção dos serviços nos provedores e é utilizado pelo WSInvoker, FaultDetector e WSRecovery.

- As classes ServicePool e Service representam respectivamente o grupo de serviços e os serviços nos provedores.
- A classe Task é utilizada para a programação de eventos que ocorrem com periodicidade. Esta classe é utilizada pelo FaultDetector nos mecanismos de detecção de faltas.
- As interfaces PullMonitorable e Updateable devem ser implementadas pelos serviços para que seja possível realizar a monitoração e a recuperação do seu estado.

O sistema de configuração foi desenvolvidos em *Java Server Pages 1.2* e *Java Servlet 2.3* permitindo que os administradores possam configurar os serviços remotamente fazendo o uso apenas de um navegador *web*. A Figura 7 apresenta a interface do sistema de configuração.

Os componentes *WSInvoker*, *Generic Web Service*, *Response Analizer* são disponibilizados como uma aplicação J2EE no servidor de aplicações. O componente *WSInvoker* executa as réplicas em paralelo usando *threads* provido pela linguagem Java. Os componentes *ReplicationManager*, *ReplicationProperties WSRecovery*, *FaultDetector* e *FaultNotifier* são objetos implementados sob as especificações de tolerância a faltas do CORBA, as interfaces dos objetos *ReplicationManager*, *Fault Detector* e *ReplicationProperties* podem ser visualizadas na Figura 9:

```
public interface ReplicationProperties {
   void setProperties(ServiceGroup serviceGroup, Properties props);
   void removeProperties(ServiceGroup serviceGroup);
   Properties getProperties(ServiceGroup serviceGroup);
}
public interface FaultDetector {
   void registered(Service service);
   void unregistered(Service service);
}
public interface ReplicationManager {
   void addServiceGroup(ServiceGroup serviceGroup);
   void removeServiceGroup(ServiceGroup serviceGroup);
   void addService(ServiceGroup serviceGroup, Service service);
   void removeService(ServiceGroup serviceGroup, Service service);
   void registerNotify(FaultNotifier fault);
}
```

Figura 9. Interfaces de Gerenciamento de Replicação.

Para realizar o processo de monitoração e recuperação os serviços web devem implementar as interfaces *PullMonitorable* e *Updateable*. Estas interfaces podem ser visualizadas na Figura 10:

```
public interface Updateable {
   void setState(State state);
   State getState();
}
```

```
public interface PullMonitorable {
        booalen isAlive();
}
```

Figura 10. Interfaces de Monitoração e Recuperação de estado.

O *WSWrapper* foi desenvolvido para permitir a exposição de objetos implementados sob a arquitetura CORBA. Este componente é um serviço web que realiza conversão de requisições SOAP em invocações a objetos CORBA. A invocação dos objetos é realizada através da interface de invocação dinâmica (DII) provida pelo CORBA.

O *FTWeb* não afeta a operabilidade de serviços já existentes, podendo coexistir no mesmo ambiente, serviços executados sob o *FTWeb* e serviços web tradicionais. O desenvolvimento de serviços web tolerantes a faltas usando a infra-estrutura *FTWeb* é bastante simplificada, requer apenas a inserção dos métodos de monitoração e recuperação do estado da réplica fornecidos pela infra-estrutura. O *FTWeb* pode ainda ser utilizado em diferentes tipos de serviço web: *statefull*, *stateless*, síncronos e assíncronos.

6. AVALIAÇÃO DE DESEMPENHO

Visando verificar o desempenho da infra-estrutura proposta foram executados testes em uma rede local de 10Mbps compostas por computadores Intel Pentium IV 2.8 GHz com 1Gb de memória RAM e sistema operacional Microsoft 2000 Professional. O *WSDipatcher Engine* foi instalado em dois servidores, sendo um servidor *backup*. As réplicas foram distribuídas em até sete computadores todos contendo o *IBM WebSphere Application Server 5.1*.

Para avaliar o tempo de resposta adicionado pela infra-estrutura *FTWeb* considerando o tamanho das mensagens, foram realizados testes com grupos de serviços com até 4 réplicas e com variação no tamanho da mensagem de 1 a 256 Kbytes. Para avaliar o custo da tolerância a faltas provida pela infra-estrutura, foram realizados testes com o mesmo serviço sem a utilização do *FTWeb*.

É possível observar na Figura 11 que para mensagens com até 16 Kbytes a variação do número de réplicas que compõem o grupo, não afeta significativamente o tempo de resposta do serviço. Os testes apresentaram aproximadamente 25% de acréscimo em relação a um serviço executado sem o *FTWeb*. Entretanto, o acréscimo no tempo de resposta pode chegar a 60%, considerando mensagens com 256 Kbytes e 4 réplicas compondo o serviço utilizando o esquema de votação. Para os testes executados com esquema de votação, o tempo era determinado pela execução da réplica localizada no computador mais lento. Os testes executados sem votação, enviando ao cliente a primeira mensagem processada, apresentaram tempos iguais, aos testes realizados utilizando o *FTWeb* com apenas 1 serviço no grupo.

Para avaliar o tempo de resposta adicionado pelo *FTWeb*, considerando o número de usuários simultâneos acessando o serviço, foram realizados testes com grupos de serviços com até 4 réplicas e com variação de 2 a 20 usuários simultâneos. O tamanho da mensagem utilizada foi 4 Kbytes.

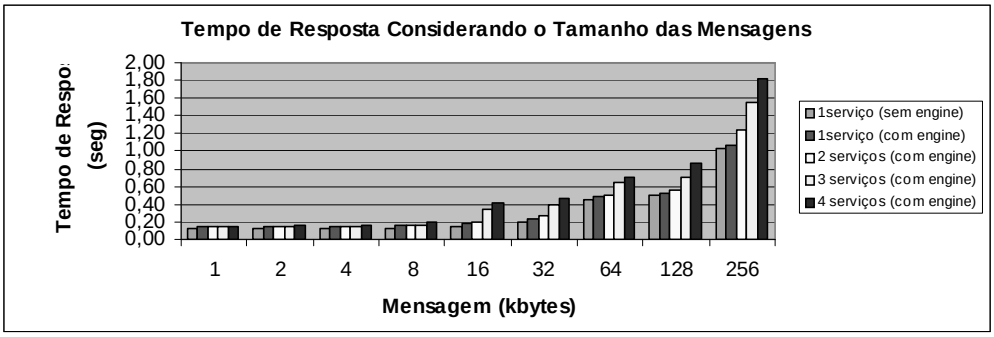

Figura 11. Tempo de Resposta Considerando o Tamanho da Mensagem.

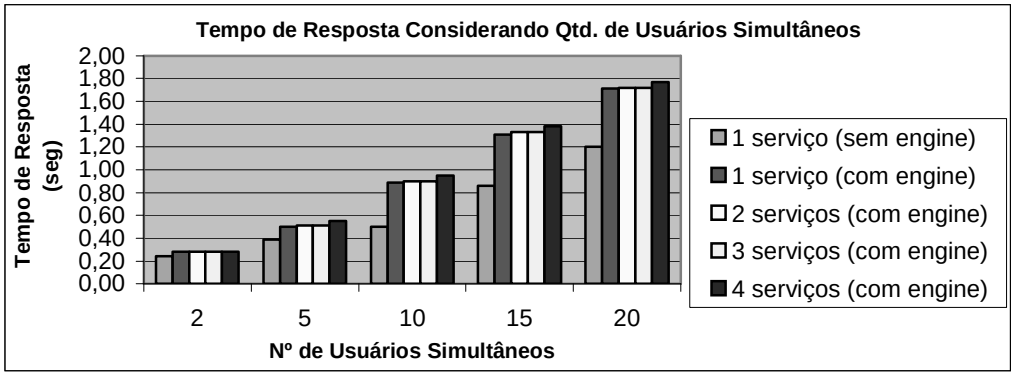

Figura 12. Tempo de Resposta Considerando a Qtd. De Usuários Simultâneos.

É possível observar na Figura 12 que a variação do número de réplicas que compõem o grupo não afeta significativamente tempo de resposta do serviço. O tempo de resposta é afetado quando incrementado o número de usuários simultâneos devido aos mecanismos que provêm o determinismo das réplicas. Quando submetido a 20 usuários simultâneos o acréscimo no tempo de resposta foi de aproximadamente 41%.

Para avaliar a disponibilidade fornecida pelo FTWeb, considerando serviços dispersos geograficamente, foram realizados testes com 2 grupos de serviços web diferentes. O primeiro grupo contém 3 serviços web públicos, localizados em diferentes países que realizam operações de conversão de temperatura. O segundo grupo contém 3 serviços web públicos localizados em diferentes países que realizam operações de conversão de moeda. É possível observar na Tabela 1 que quando os serviços são submetidos a 1000 execuções individualmente, cada serviço apresenta um percentual de falha. Entretanto quando estes serviços são agrupados e submetidos a 1000 execuções, utilizando a infra-estrutura FTWeb o percentual de falha é 0.0%.

7. TRABALHOS RELACIONADOS

Embora a disponibilidade e confiabilidade sejam requisitos cruciais em aplicações críticas, os trabalhos relacionados à proposição de modelos tolerantes a faltas em arquiteturas orientadas a serviço são recentes. O modelo abordado em [Deron et. al., 2003] propõe extensões no padrão SOAP permitindo a utilização da técnica de replicação passiva para

alcançar a tolerância a faltas. Este modelo realiza alterações no documento WSDL inserindo informações referentes à réplica primária e às réplicas *backup*. A utilização de interceptadores na camada SOAP no cliente permite o redirecionamento da requisição para as réplicas em caso de falhas do primário. No servidor, os interceptadores adicionam componentes para registro de *log*, detecção de faltas e gerenciamento das réplicas. O *FTWeb* não realiza alterações no documento WSDL, os serviços que compõem o grupo são descritos no sistema de configuração, seguindo a abordagem de domínio de serviços. Na infra-estrutura *FTWeb*, a utilização do interceptador é restrita à detecção de falhas na própria infra-estrutura, permitindo que na falha do *WSDispatcher Engine* primário as requisições possam ser encaminhadas para um *WSDispatcher Engine* backup.

Serviço: Conversão de Temperatura

FTWeb	Xmethod (USA)	WebServiceX (UK)	DeveloperDays (USA)
0,0%	0,9%	1,7%	0,5%

Serviço: Conversão de Moeda

FTWeb	Xmethod (USA)	WebServiceX (UK)	DeveloperDays (USA)
0,0%	1,2%	0,5%	0,0%

Serviços Equivalentes -1000 execuções por serviço

Tabela 1 – Teste de Disponibilidade Considerando Serviços Dispersos Geograficamente

O modelo proposto em [Aghdaie, Tamir, 2002] realiza alterações no *kernel* do sistema operacional e no servidor web provendo um mecanismo de tolerância a faltas transparente para o cliente. Neste modelo, todas as requisições recebidas pelo servidor são registradas e enviadas para um servidor *backup*. As alterações realizadas no *kernel* do sistema operacional provêem a implementação de um mecanismo de *multicast* permitindo que as requisições sejam enviadas para o servidor *backup* e para o servidor primário. As alterações realizadas no servidor web permitem a manipulação e geração de respostas para o cliente. Em comparação com este modelo, o *FTWeb* é mais portável, pois atua simplesmente como uma camada de software adicional sem a necessidade de alterações no sistema operacional ou no servidor web.

Os trabalhos abordados em [Dialani et. al., 2002], [Townend, Xu, 2004] propõem modelos tolerantes a faltas para serviços web implementados e executados sob as especificações de serviços em *grid* [OGSA, 2003]. Em [Dialani et. al., 2002] a arquitetura proposta tem como principal objetivo realizar a detecção e a recuperação em situações de falhas, este modelo não trata a tolerância a faltas através da replicação de objetos, mas através de mecanismos de *checkpoint* e *rollback*. [Townend, Xu, 2004] propõem a implementação de um mecanismo que executa um conjunto de serviços web equivalentes, entretanto implementados sob diferentes plataformas *(n-version)*. Após a execução um esquema de votação atua sobre as respostas retornando a resposta mais coincidente.

Apesar do tema serviços em *grid* não fazer parte do escopo do modelo definido pelo *FTWeb*, algumas similaridades podem ser encontradas entre os modelos. Assim como os serviços em *grid*, o *FTWeb* permite a utilização de serviços web *statefull* e garante o determinismo entre as réplicas. A diversidade de programas pode ser utilizada no *FTWeb*, que permite que serviços web implementados sobre diferentes arquiteturas componham o mesmo grupo de serviços.

8. CONCLUSÃO

Este artigo apresentou uma proposta de utilização da técnica de replicação semi-ativa para alcançar a tolerância a faltas em arquiteturas orientadas a serviços. Esta abordagem provê tolerância para as seguintes classes de faltas: parada, omissão e valor. A infra-estrutura proposta é baseada em um mecanismo denominado de *WSDispatcher Engine* que possui componentes responsáveis por: compor grupos de serviços, detectar e recuperar faltas, invocar concorrentemente as réplicas do serviço, garantir o determinismo entre as réplicas e estabelecer um esquema de votação sobre as respostas retornadas pelos serviços.

As réplicas que compõem o grupo do serviço podem estar localizadas em servidores dispersos geograficamente, evitando vulnerabilidade a falhas em roteadores, *gateways* e outros componentes que realizam interfaces com a rede A aplicação do modelo em serviços web já implementados e operacionais é bastante simplificada requer apenas a inserção de métodos de monitoração e recuperação do estado da réplica.

Os testes realizados com o protótipo mostram que os custos de desempenho são aceitáveis considerando o ganho em disponibilidade e confiabilidade propiciado pela infra-estrutura. A configuração e a monitoração das réplicas que compõem o serviço podem ser realizadas remotamente através de um sistema provido pela infra-estrutura, fazendo o uso apenas de um navegador *web*. Como objetivo para trabalhos futuros está a integração do modelo *FTWeb* com as especificações de serviços em *grid* [WS-RF, 2004].

REFERÊNCIAS BIBLIOGRÁFICAS

Aghdaie, N., Tamir, Y. (2002). Implementation and Evaluation of Transparent Fault-Tolerant Web Service with Kernel-Level Support. Proceedings of the IEEE International Conference on Computer Communications and Networks Miami, Florida, pp. 63-68

Budhiraja, N., Marzulo, K., Schneider, F. B. e Toueg, S. (1993). Distributed Systems, chapter 4. The Primary-Backup Approach. Addison Wesley. 2nd edition.

Defago, X., Schiper, A., Urban, P. (2000). Totally Ordered Broadcast and Multicast Algorithms. Technical Report DSC/2000/036, Dept. of Communication Systems, EPFL.

Deron L., Fang, C., Chen, C., Lin, F. (2003). FT-SOAP: A Fault-Tolerante Web Service. Tenth Asia-Pacific Tenth Asia-Pacific Software Engineering Conference Software Engineering Conference, Chiang Mai Thailand pp.310

Dialani, V., Miles, S., Moreau, L. De Roure, D., Luck, M. (2002). Transparent Fault Tolerance for Web Services Based Architectures. Euro-Par 2002. Parallel Processing: 8th International Euro-Par Conference Paderborn, Germany Proceedings. Volume 2400

Gokhale, A., Kumar, B., Sahuguet A. (2002). "CORBAWeb Services", Proceedings of the 11th International World Wide Web Conference, Honolulu, Hawaii.

Jandl, M., Radinger, W., Goeschka, K. M. (2003). Integration of CORBA with Directory Services and Web Services, ACM/IFIP/USENIX International Middleware Conference, Rio de Janeiro, Brasil

OGSA Open Grid Services Architecture (2003) www.globus.org/ogsa/

OMG FT-CORBA Specification (2002). Common Object Request Broker Architecture: CoreSpecification Chapter 23. www.omg.org.

OMG WSDL - SOAP to CORBA Interworking (2003) OMG Document. http://www.omg.org

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

SOAP Simple Object Access Protocol (2003) – World Wide Web Consortium http://www.w3c.org/TR/soap/

Tan, S., Vellanki, V., Xing, J., Topol B., Dudley G. Service Domains (2004). IBM System Journal VOL 43, N4.

Townend, P., Xu, J. (2004). Replication-based Fault Tolerance in a Grid Environment, Proceedings of the UK e-Science All Hands Meeting

UDDI Universal Description, Discovery and Integration (2002) – OASIS http://www.oasis-open.org/committees/uddi-spec/doc/contribs.htm#uddiv1

WSDL Web Services Description Language (2001) – World Wide Web Consortium http://www.w3c.org/TR/wsdl/

WS-ARCH Web Services Architecture (2004) W3C World Wide Web Consortium http://www.w3.org/TR/ws-arch/

WS-I Web Service Interoperability Organization Basic Profile 1.0 (2004). http://www.ws-i.org/Profiles/BasicProfile-1.0-2004-04-16.html

WS-RF OGSI - Open Grid Services Specification (2004). 1.0 http://www-128.ibm.com/developerworks/library/ws-resource/ws-wsrf.pdf

WS-RM - Web Services Reliable Messaging Specification (2004). 1.0 ftp://www6.software.ibm.com/software/developer/library/ws-reliablemessaging200502.pdf

XML Extensible Markup Language (2000) – World Web Consortium http://www.w3.org/TR/2000/REC-xml-20001006