Desenvolvendo Serviços Web Tolerante a Faltas usando BPEL

Jim Lau[1], Lau Cheuk Lung[2], Joni da Silva Fraga[1], Giuliana Santos[3]

[1]DAS – Departamento de Automação e Sistemas
UFSC – Universidade Federal de Santa Catarina
Florianópolis – Santa Catarina – Brasil

[2]PPGIA - Programa de Pós-Graduação em Informática Aplicada
PUC-PR - Pontifícia Universidade Católica do Paraná

[3]Faculdade de Ciências da Universidade de Lisboa
Bloco C6, Piso 3, Campo Grande, 1749-016
Lisboa - Portugal.

{jim,fraga}@das.ufsc.br, lau@ppgia.pucpr.br,
giuliana@lasige.di.fc.ul.pt

Abstract. *The web services technology provides an approach for developing distributed applications by using simple and well defined interfaces. Due to the flexibility of this architecture, it is possible to compose business processes integrating services from different domains. This paper presents an approach, which uses the specification of services orchestration, in order to create a fault tolerant model combining active and passive replication technique. This model supports fault of crash and value. The characteristics and the results obtained by implementing this model are described along this paper*

Resumo. *A tecnologia de serviços web provê uma abordagem para o desenvolvimento de aplicações distribuídas utilizando interfaces simples e bem definidas. Devido à flexibilidade desta arquitetura, é possível a composição de processos de negócios integrando serviços de domínios distintos. Este artigo apresenta uma proposta de utilização das especificações para orquestração de serviços, na implementação de uma arquitetura tolerante a faltas utilizando uma combinação de técnica de replicação ativa e passiva. Este modelo suporta falta de crash e valor. As características e os resultados obtidos com a definição desta arquitetura são descritos no decorrer deste artigo.*

1. Introdução

Nos últimos anos, novas tecnologias e padrões de desenvolvimento estão surgindo, permitindo uma maior integração entre os diversos aplicativos e serviços disponíveis na Internet. Os serviços web fazem parte deste cenário e propõem um modelo de serviços distribuídos que utilizam interfaces de acesso simples e bem definidas. Entre as principais vantagens na adoção deste modelo pode-se destacar:

- Menor custo no desenvolvimento: permite a reutilização de componentes de software e conseqüentemente o desenvolvimento mais rápido de sistemas.

- Integração com sistemas legados: permite a integração com sistemas estabelecidos e operacionais.
- Melhores interfaces com parceiros comerciais: através de intercâmbio eletrônico de dados de baixo custo.

Um serviço web é um componente de software que aceita requisições de outros sistemas através da infra-estrutura da Internet. As especificações dos serviços web foram projetadas com o objetivo de integrar aplicações, permitindo o suporte a transações síncronas e assíncronas. O modelo de serviço web não é a primeira proposta de integração, outras tecnologias possuem o mesmo objetivo. Entretanto, o que faz dos serviços web tão atrativos é a utilização de padrões neutros e protocolos amplamente utilizados e consolidados.

A especificação XML (*Extensible Markup Language*) é utilizada para realizar o intercâmbio de dados. O XML é superficial e extensível podendo incorporar com facilidade recursos empresarias como transações. Os serviços web são construídos com base em padrões de comunicação existentes, o que os torna independentes de protocolo de transporte, podem ser utilizados com os protocolos HTTP, FTP, SMTP e outros.

Com o objetivo de explorar todo o potencial dos serviços web como um modelo de integração, as empresas Microsoft, IBM e BEA uniram esforços na especificação de um modelo padrão para a definição e a integração de processo de negócios. Esta especificação recebeu o nome de *Business Process Execution Language For Web Services* (BPEL4WS[1]) [WS-BPEL,2005] e define um modelo e uma gramática para descrever o comportamento do processo de negócios baseada na interação entre o processo e os parceiros. Estende o modelo dos serviços web e habilita o suporte a transações. A especificação define um modelo interoperável que facilita a expansão da integração de processos automatizados dentro de uma mesma corporação e/ou entre empresas (*business-to-business*). Esta composição de serviços também recebe o nome de orquestração e inclui a lógica do negócio e a ordem das execuções definidas a partir de fluxos de controles que atravessam organizações e aplicações.

Apesar de toda a flexibilidade provida pelo BPEL e pela própria arquitetura dos serviços web para a construção de soluções distribuídas, faz-se necessário a padronização de mecanismos de suporte para serviços Web tolerante a faltas que atenda os requisitos de confiabilidade e disponibilidade, fundamentais em aplicações críticas. O conjunto de especificações padronizadas pela W3C [W3C,2005] e OASIS [OASIS,2005] não contemplam estes requisitos e tem motivado alguns grupos de pesquisa no sentido de propor extensões para adicionar mecanismos para tolerância a faltas nestas soluções.

Em [Aghdaie and Tamir,2002], [Dialani, et al.,2002],[Liang, et al.,2006], são propostos modelos tolerantes a faltas baseados na técnica de replicação passiva e implementam mecanismos simplificados que detectam a falta e direcionam futuras requisições para servidores redundantes. Em nosso trabalho anterior [Santos, et al.,2005], propomos uma infra-estrutura que utiliza a técnica de replicação ativa com componentes que transforma pedidos SOAP em invocações de objeto de CORBA. Em

[1] Ou simplesmente BPEL

[Dobson,2006] é apresentado um estudo usando o BPEL para a implementação de um serviço web tolerante a faltas, porém nesta solução o servidor onde executa a lógica do processo de negócio é um simples ponto único de falha. Este trabalho representa um esforço inicial para implementar uma orquestração de serviços web transparente tolerante a faltas.

Neste artigo, nos propomos uma arquitetura chamada FTWS-Orch onde o objetivo principal é prover uma arquitetura orientada a serviços tolerante a faltas em serviços web. O FTWS-Orch combina a utilização de técnicas de replicação ativa e passiva utilizando um middleware baseado na orquestração de serviços web.

O texto esta organizado na seguinte forma: uma visão geral da plataforma dos serviços web é apresentada na seção 2. Na seção 3 é apresentada especificação WS-BPEL. Na seção 4 descreveremos a arquitetura FTWS-Orch. Na seção 5 e 6 são apresentados a implementação e os testes juntamente com os resultados e na seção 7 são abordados os trabalhos relacionados e finalmente na seção 8 temos a conclusão do trabalho.

2. Serviços Web

Nos últimos anos, o modelo de arquitetura orientado a serviço vem se destacando como uma arquitetura de software que relaciona os componentes de um sistema em um ambiente distribuído onde são disponibilizados serviços que podem ser acessados dinamicamente através de uma rede.

Um serviço web pode ser definido, de forma simples, como um conjunto de operações disponível e acessível em escala global através de um endereço eletrônico do tipo URL. Outra, mais genérica, define um serviço web como uma interface que descreve uma coleção de operações que são acessíveis pela rede através de um mecanismo de mensagem XML.

A Figura 1 apresenta o modelo comum de Serviços Web identificando os três tipos de papeis e as operações que são executadas. Os papéis mostrados no diagrama são: provedor de serviço, consumidor de serviço e o serviço de registro.

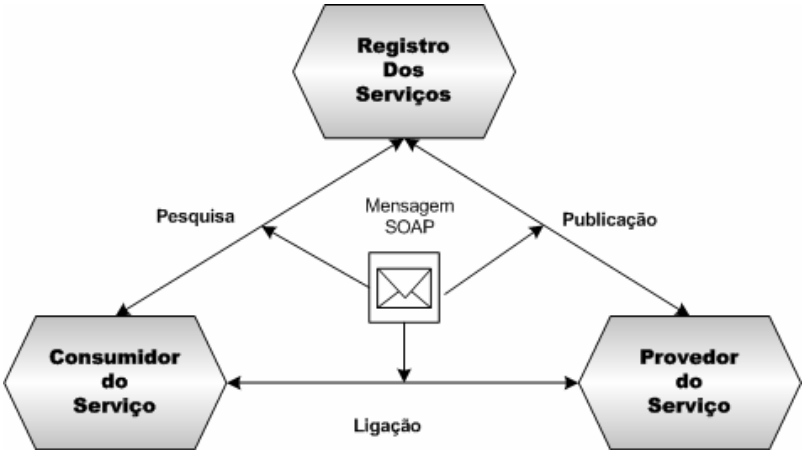

Figura 1 - Modelo conceptual serviços web

1. Provedor de serviço – responsável pela descrição e publicação de um determinado Serviço Web no registro de serviço. O provedor também é responsável por descrever as informações de ligação do serviço usadas para sua invocação. As informações estão representadas em um documento XML escrito na linguagem padrão WSDL [WSDL,2001];
2. Consumidor do serviço – responsável por descobrir um serviço, obter a sua descrição e, usá-lo para se ligar a um provedor a fim de invocar um serviço web através de um URL;
3. Registro dos serviços – mantém um diretório com informações sobre serviços, como por exemplo, nome, provedor e categoria. O padrão adotado na SOA para registro é o UDDI [Clement, et al.,2001].

A troca de mensagem entre provedores e consumidores de serviços é através do protocolo SOAP [SOAP,2003]. É um protocolo baseado em XML para descrever o serviço, que prove todos os detalhes necessários para interagir com o serviço, incluindo o formato da mensagem, o protocolo de transporte e a sua localização.

A interação entre os três elementos envolve: a publicação da informação sobre um determinado serviço, a descoberta dos serviços disponíveis e a ligação entre esses serviços. A arquitetura de serviços web apresenta algumas vantagens, dentre as quais se destacam o suporte para diferentes tipos de cliente; a elevada manutenibilidade; a facilidade de reuso; a escalabilidade com isso garantindo a interoperabilidade necessária para a arquitetura, além da capacidade de compartilhar e reutilizar serviços e recursos.

3. Processo de Negócios

A definição de processo de negócios envolve especificar o comportamento de cada participante envolvido sem revelar o seu funcionamento interno. A separação de aspectos públicos de aspectos privados permite a independência na modificação da implementação sem afetar o protocolo de negócio. Os requisitos fundamentais para a descrição de protocolos de negócio são:

- Protocolos de negócios invariavelmente são dependentes de dados comportamentais;
- Habilidade para especificar condições de exceção e suas conseqüências, incluindo a recuperação de seqüências;
- Habilidade para trabalhar com interações de negócio de longa duração, incluindo múltiplas ocorrências. Cada interação deve ser vista como uma unidade de trabalho com seus próprios requisitos. Deve possuir a habilidade de coordenar as várias unidades de trabalho e as atividades concorrentes em vários níveis de granularidade.

O BPEL4WS define uma gramática para descrever o comportamento dos processos de negócios baseada na interação entre os processos e os seus parceiros. A interação com cada parceiro ocorre através de interfaces de serviços web. O processo define como as múltiplas interações com esses parceiros serão coordenadas para realizar um determinado processo de negócio. O BPEL4WS também define mecanismos para

tratar exceções e falhas de processamento, e introduz mecanismos para definir como uma atividade individual ou um conjunto de atividades pode ser compensado em caso de exceções ou de requisições reversas do parceiro. Esta composição de serviços também recebe o nome de orquestração (*orchestration*).

A orquestração descreve a interação entre serviço, no nível de troca de mensagem, incluindo uma lógica de negócio e a ordem de execução. Uma orquestração é um processo de negócio executável, controlado por um dos participantes do processo.

A Figura 2 apresenta o fluxo de execução de um *engine* BPEL. No lado esquerdo o fluxo de processo de negócio e no lado direito o fluxo de execução controlado pelo *engine* que faz o papel de "maestro". Enquanto recebe uma requisição, o *engine* passa a compor a orquestração de acordo com o fluxo de processo, ordenando a execução de cada componente de forma seqüencial ou paralela. O fluxo de execução é a forma pela qual o *engine* executa cada interação em sua lógica de processo.

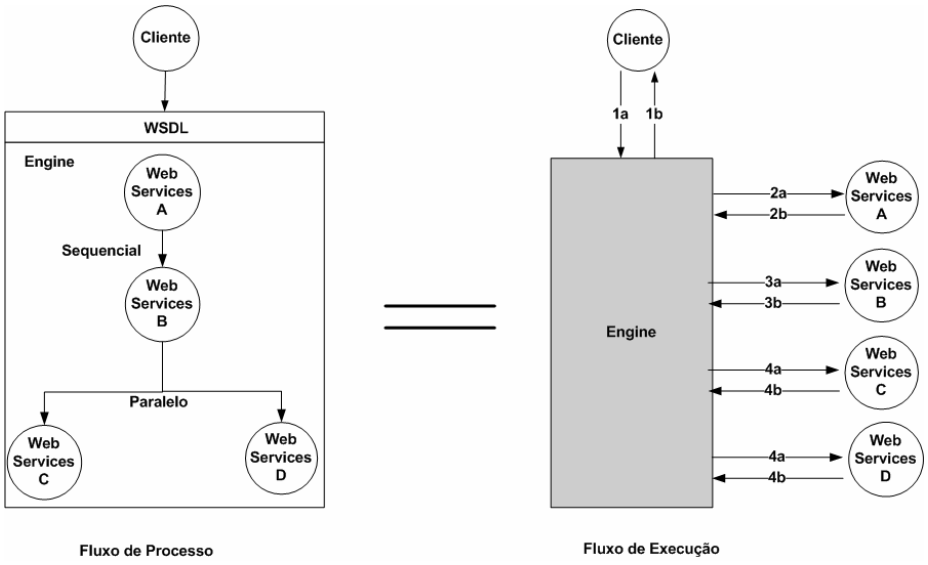

Figura 2 – Fluxo de processo e de execução

O fluxo de execução inicia com a requisição do cliente ao *engine* (1a). Neste momento, o primeiro nodo ou serviço a receber a requisição do cliente é o web services A. O *engine* faz uma invocação a esse serviço (2a) e espera a resposta da requisição (2b). O próximo passo do fluxo de execução seqüencial é a invocação de uma requisição ao web services B, então uma invocação (3a) é feita a esse web services e espera o retorno da resposta do serviço (3b). Em seguida, o *engine* em um processo simultâneo faz uma invocação em paralelo (4a) para dois web services C e D e as respostas das invocações retornam para o *engine* (4b). Depois de executar o fluxo de processo dentro do *engine*, uma resposta, como resultado da execução do fluxo de processo, é enviada para o cliente (1b).

Todas as invocações nos web services são feitas a partir de um documento WSDL de cada serviço. Fazendo com que o *engine* consiga invocar ou receber requisições desses serviços. O BPEL oferece uma linguagem padrão para definir:

- O envio de mensagem XML para um serviço remoto;
- Uma estrutura para manipular os dados XML;

- O recebimento das mensagens XML assíncronas para serviço remoto;
- O tratamento de eventos e as exceções;
- Define também a execução de seqüência de eventos em paralelo e o cancelamento de eventos de um processo quando ocorre uma exceção.

Estes são os principais itens para compor um conjunto de serviços dentro de um processo de negócio colaborativo e transacional. O BPEL é baseado em XML Schema, SOAP e WSDL.

4. Descrição da Arquitetura FTWS-Orch

A idéia principal do modelo FTWS-Orch é o uso das especificações de composição de serviços para definir uma arquitetura para o desenvolvimento de aplicações tolerante a faltas utilizando uma combinação de técnica de replicação ativa e passiva.

As réplicas de um determinado serviço são agrupadas através da definição de um processo negócio. Todas as réplicas implementam o mesmo serviço, recebendo, executando e respondendo as requisições enviadas pelos clientes. Este modelo permite a utilização de serviços web síncronos e assíncronos e serviços implementados em plataformas heterogêneas.

Através do modelo especificado neste trabalho é possível compor processos negociais utilizando serviços tolerantes a faltas. Conforme a sua funcionalidade, esta solução pode ser divida em três módulos principais: composição e invocação de serviços, detecção de falhas, tolerância a faltas transparente para o cliente.

4.1 Composição e Invocação dos serviços

Para criar os grupos necessários para a abordagem de replicação, uma ferramenta para o gerenciamento de processo de negócio é usada. O administrador do sistema define o fluxo de execução informando os documentos WSDL que farão parte da composição de serviço. Esses documentos referem-se às réplicas de um determinado serviço. O fluxo define que estas réplicas serão executadas concorrentemente conforme a técnica de replicação ativa, isto é, todas as réplicas livres de falhas do grupo são ativas: recebem, processam de forma paralela e produzem a mesma saída.

O cliente observa esta composição de serviços como um único serviço. Entretanto, estes serviços são replicados, independentes e podem estar localizados em diferentes domínios.

Na Figura 3 é apresentada à infra-estrutura FTWS-Orch. Nela estão contidas os *Engine Primário, Backup e Monitor*. As funções desses engine são de gerenciar as execuções, redirecionar as requisições no caso de exceções e a interface entre os clientes. Neste processo o *engine Primário* é definido conforme o uso da técnica de replicação passiva juntamente com o *engine Backup*, enquanto dentro do *engine* tanto *Primário* como *Backup* são definidos o uso de técnica de replicação ativa entre os serviços web.

Após a definição do processo de negócios, o administrador realiza a publicação em um registro UDDI como um serviço web tradicional ou passa a descrição do serviço web, através do documento WSDL aos clientes.

O cliente realiza pesquisa no registro de serviços e obtém o documento WSDL do processo de negócio e executa a invocação de serviço. As operações do cliente de pesquisa e invocação podem ser observadas no passo 1 da Figura 3.

O *engine* é responsável pela interação entre o cliente e os serviços web replicados (passo 2 e 3 da Figura 3). O *engine* primário obtém do cliente a referência da composição de serviços web, os parâmetros necessários para a sua execução, e gerenciam a execução do serviço em todas as réplicas e retorna a resposta ao cliente. Neste modelo uma resposta é retornada para o cliente quando há pelo menos uma réplica livre de falha.

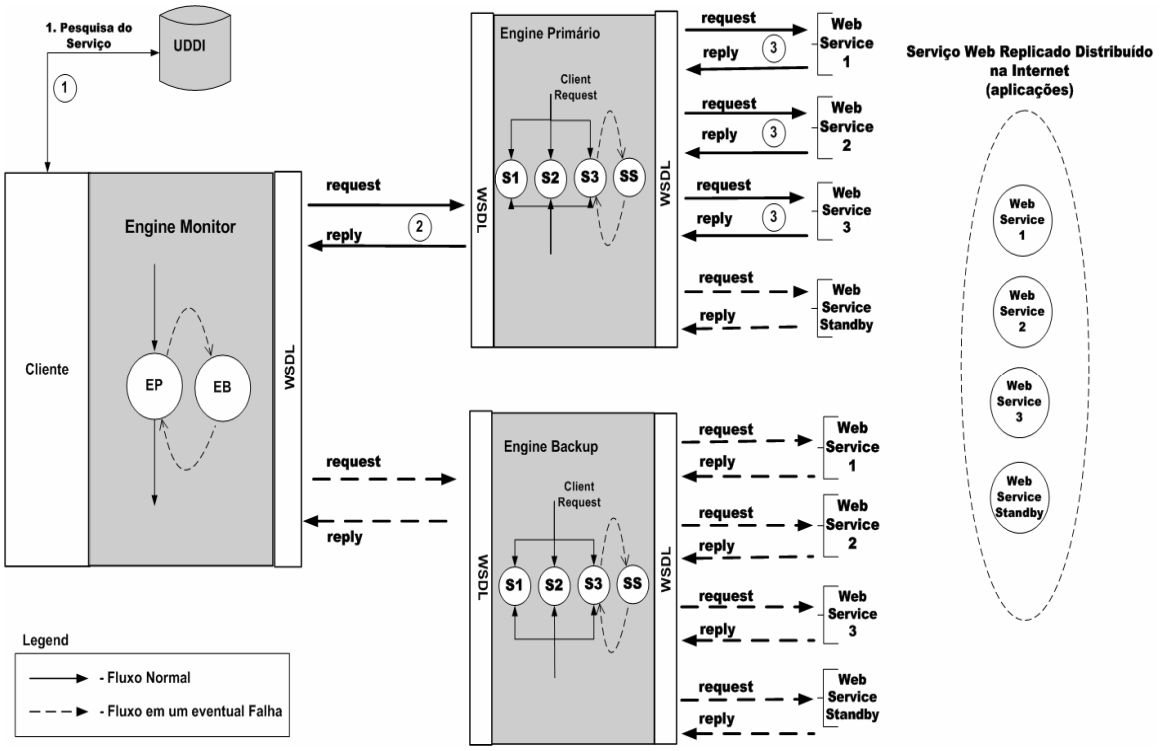

Figura 3 – Infra-estrutura FTWS-Orch.

4.2 Detecção de Falha para Replicação de Serviços Web

Por ser um processo que interfere no desempenho da solução, neste modelo não há componentes que realizem a monitoração de réplicas para descobrir réplicas faltosas. Por outro lado, quando uma réplica apresenta falha, o FTWS-Orch redireciona os pedidos do cliente dinamicamente a uma réplica *standby*, como uma técnica de replicação passiva.

Nesta proposta inicial somente trabalhamos com serviços web *stateless*, por isso, não são implementados mecanismos de recuperação do estado das réplicas ou mecanismo para garantir o determinismo no serviço. Em um trabalho futuro serão adicionados mecanismos para realizar a atualização do estado das réplicas, a mesma abordagem usada em [Santos, et al.,2005].

A flexibilidade apresentada pela especificação no processo de negócio permite definir para cada réplica, uma réplica *standby* especifica (SS na Figura 3) ou definir

para todas as réplicas aponte para a mesma réplica *standby*. Isto é, as réplicas S1, S2, S3 aponta a uma réplica *standby* especifica ou essas réplicas aponte para uma mesma réplica de serviço em *standby*. Conforme a Figura 3 dentro de cada *engine* (*Primário* e *Backup*) tem uma composição de serviço com réplicas *standby* definidas.

4.3 Tolerância a Faltas Transparente para o Cliente

As facilidades dos processos de negócios são usadas também no lado do cliente para evitar que o *Engine Primário* se torne um ponto único de falha. A invocação do *Engine Primário* pode ser realizada como um processo de negócio, na ocorrência de uma exceção no *Engine Primário* um redirecionamento dinâmico ocorre para o *Engine Backup* usando a técnica de replicação passiva. Isto é, o *Engine Backup* o passa a substituir este *engine* faltoso passando a responder as requisições dos clientes. Toda essa operação de redirecionamento é através do *Engine Monitor* (Figura 3). É através desse *engine* que a ocorrência de falha no *Engine Primário* se torna transparente para o cliente.

A Figura 3 apresenta um fluxo normal (linhas continuas) e o fluxo em uma eventual falha (linhas tracejadas) no *Engine Primário*. Através de um mecanismo de *logs* contidos nas réplicas do serviço é possível verificar se a requisição já foi processada pelo serviço, caso positivo, as réplicas apenas retornam a resposta da requisição ao *Engine Backup* – a requisição não é re-executada.

Tolerância a Faltas de Valor

Para o suporte de faltas de valores é possível adicionar na composição um serviço web que atue como um votador e que obtém as respostas de todas as réplicas e retorna o valor mais coincidente para o cliente tolerando a falta de valor. Quando este componente é usado é necessário ter pelo menos 2f + 1 réplicas e enviar a resposta para o cliente, onde f é o número de réplicas faltosas, caso contrário uma mensagem de erro é retornada.

5. Implementação

Para validar a proposta, um protótipo deste modelo foi implementado utilizando a ferramenta *BEA Weblogic Workshop 8.1.* Entretanto, este protótipo pode ser aplicado em outras ferramentas, por exemplo, *IBM's BPWS4J*[2] e *Collaxa Orchestration Server*[3] . Foi definido um processo negócio variando o número de serviços utilizados. Estes serviços foram instalados em diferentes computadores em uma rede local. Este implementa o acesso ao sistema de arquivos no servidor onde estão hospedados.

Na Figura 4 é apresentada à interface da ferramenta de composição de processo de negócio. O grupo de serviço é representado pelo Web Services A, Web Services B e Web Services C, que são executados concorrentemente. Diferentemente de outras abordagens em nosso trabalho não é necessária a troca de serviços web, já implementados para incluir métodos para monitorar serviços.

---

[2] http://www.alphaworks.ibm.com/tech/bpws4j

[3] http://www.javaskyline.com/20030311_collaxa.html

Quando o cliente faz uma requisição ao grupo de serviço, esta operação é realizada através de uma transação, em que a requisição é enviada aos três serviços. As exceções neste serviço são detectadas e o pedido é redirecionado para o Standby_Service de acordo com o especificado no fluxo OnExecption.

Através dessa abordagem é possível implementar um processo de negócio tolerante a faltas que usa uma composição como parte de uma outra composição. O modelo proposto permite que o administrador modifique facilmente o grupo de serviço. O administrador executa a modificação em um grupo de serviço obtendo o documento WSDL de cada serviço e modifica o processo de negócio para trabalhar com um novo serviço.

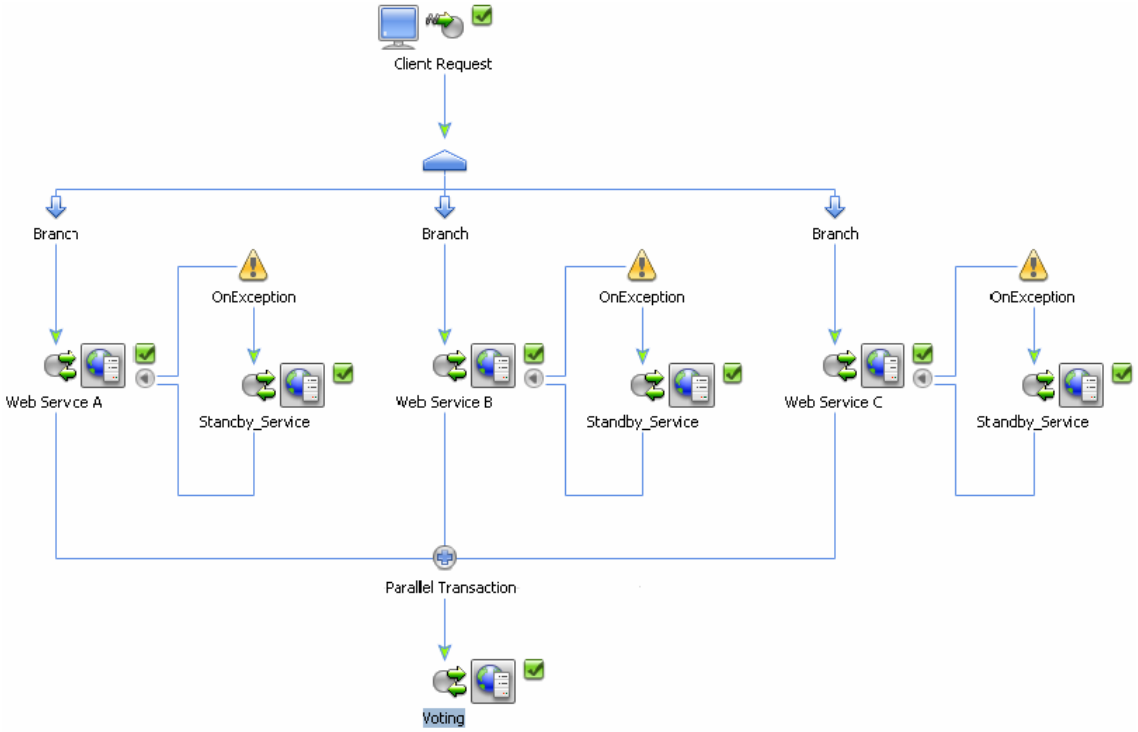

Figura 4 - Interface de composição do processo de negócio.

Depois de definido o processo de negócio é obtido um documento XML que será usado pelo *engine* BPEL para o fluxo de execução. A Figura 5 apresenta as principais partes de um arquivo BPEL para definir o modelo FTWS-Orch. As réplicas são executadas concorrentemente usando o elemento *flow*. A detecção de faltas na réplica primária e o redirecionamento para réplica *standby* é executada pelo elemento *faultHandlers*.

```
<flow name="Parallel-Transaction" jpd:name="Parallel Transaction">
  <sequence name="Branch">
    <faultHandlers jpd:name="OnException">
      <catchAll>
        <scope name="Standby_Service">
            ...
<invoke
name="Standby_Service"
```

```
     partnerLink="Standby_Service"
portType="ctrl:FTWS_Orch.Standby_ServicePT"
operation="nome"
inputVariable="input"
outputVariable="output"/>
<assign></assign>...
</catchAll>
     </faultHandlers>
<scope name="Web_Service_1">
...
</scope></sequence>
  <sequence name="Branch">
    <faultHandlers jpd:name="OnException">
      <catchAll>
        <scope name="Standby_Service">
            ...
<invoke name="Standby_Service"
partnerLink="Standby_Service"
portType="ctrl:FTWS_Orch.Standby_ServicePT"
operation="nome"
inputVariable="input"
outputVariable="output"/>
<assign></assign>...
</catchAll>
     </faultHandlers>
     <scope name="Web_Service_2">
...
</scope>
</sequence>
</flow>
```

Figura 5 – FTWS-Orch BPEL

6. Avaliação de Desempenho

Visando verificar o desempenho desta arquitetura proposta foram executados testes em uma rede local de 100Mbps compostas por computadores Intel Pentium 4 2.0GHz 1Gb de RAM e sistema operacional Windows XP. O protótipo foi criado na ferramenta *BEA Weblogic Workshop*. O *engine* FTWS-Orch foi instalado em dois servidores , sendo um servidor backup. As réplicas foram distribuídas em até quatro computadores todos contendo *Web Apache Tomcat* versão 5.5.12 integrado com o servidor de *Web Service* (*Axis*).

Para avaliar o tempo de resposta adicionado pelo FTWS-Orch considerando o tamanho das mensagens, foram realizados testes com variação no tamanho das mensagens de 1 a 32 Kbytes. Há uma limitação no tamanho da mensagem devido a plataforma utilizada para a construção do processo de negócio que não permite que mensagens maiores que 32 Kbytes sejam utilizadas nos serviços web.

É possível observar na Figura 6 para mensagem com até 4 Kbytes a variação do número réplicas que compreendem o grupo, não afeta significativamente o tempo de resposta do serviço. Os testes apresentam um acréscimo de 23% aproximadamente no tempo em relação a um serviço executado sem o FTWS-Orch.

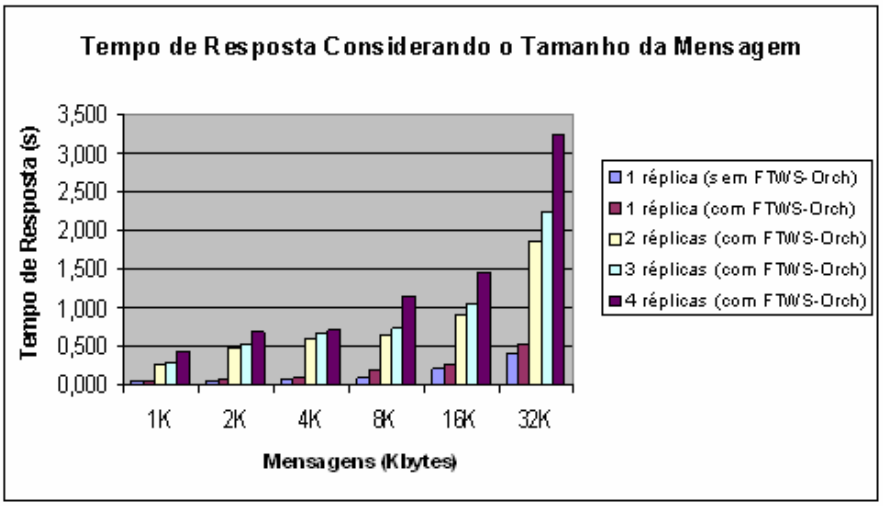

Figura 6 – Tempo de resposta considerando o tamanho da mensagem.

Entretanto, o tempo de resposta adicionando que utiliza o esquema de votador pode alcançar até 90% considerando mensagens com 32 Kbytes e 4 réplicas que compões o grupo de serviço. O tempo de resposta usando o mecanismo de votação é determinado pelo tempo de resposta da réplica mais lenta.

Para avaliar o tempo de resposta considerando o número de usuários simultâneos foram realizados testes com até 4 réplicas e a variação no número de usuários entre 2 a 20. Na Figura 7 é possível observar o tempo de resposta pode alcançar 14 segundos com 20 usuários simultâneos.

Foram realizados testes a fim de verificar a diferença no tempo total de resposta de serviço quando uma réplica apresenta falta e a sua requisição é redirecionada para uma réplica *standby*. O tempo médio observado foi de 1,2 segundos levando em consideração um grupo de serviço com quatro réplicas e uma réplica faltosa. O tempo de resposta adicionado para o redirecionamento foi de 40% em relação ao tempo observado com todas as réplicas operacionais.

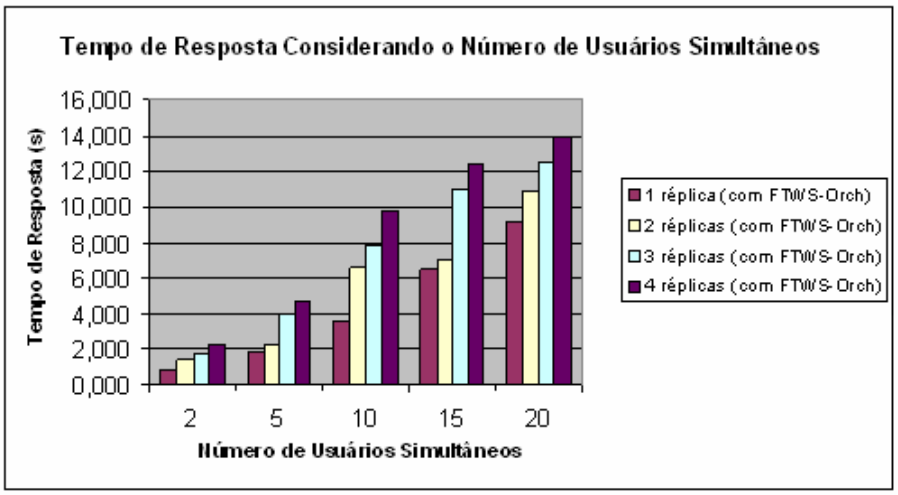

Figura 7 – Tempo de resposta considerando o número de usuários simultâneos.

O tempo de resposta apresentado acima estão acima do esperado, principalmente se for comparado com o nosso trabalho anterior [Santos, et al.,2005]. Entretanto em um trabalho futuro pretendemos otimizar o *engine* BPEL para conseguir melhores resultados.

7. Trabalhos Relacionados

Embora a disponibilidade e a confiabilidade sejam requisitos vitais em aplicações criticas os trabalhos relacionados para proposição de um modelo tolerante a faltas na arquitetura orientada a serviço ainda são bastante recentes.

O modelo abordado em [Liang, et al.] propõe extensões no padrão SOAP permitindo o desenvolvimento 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 réplicas em caso de falhas do primário. No servidor, os interceptadores adicionam componentes para registro de *log*, detecção de falhas e gerenciamento das réplicas. O FTWS-Orch não executa alterações no documento WSDL e trabalha com elementos definidos em BPEL. Os grupos de serviços são criados seguindo a abordagem de composição de serviços.

O modelo proposto [Aghdaie and Tamir,2002] realiza alterações no *kernel* do sistema operacional e no servidor de web provendo um mecanismo de tolerância de faltas transparente para o cliente. Neste modelo, cada pedido recebido pelo servidor é registrado e enviado a um servidor *backup*. As alterações realizadas no kernel do sistema operacional prover a implementação de um mecanismo *multicast* que permite pedidos sejam enviadas a um servidor *backup* e o servidor primário. As alterações realizadas no servidor web permitem a manipulação e a geração de respostas para os clientes. Em comparação com este modelo, FTWS-Orch é mais portátil, desde que atue simplesmente como outra camada de software que não necessite modificações no sistema operacional ou o servidor de web.

Os trabalhos abordados em [Dialani, et al.] [Zhang, et al.] [Townend and Xu], propõem modelos tolerantes a faltas para serviços web implementados e executados sob as especificações de serviços em *grid* [OGSA]. Em [Dialani, et al.] o objetivo principal da arquitetura proposta é a detenção e a recuperação em situações de faltas, este modelo não trata a tolerância de falta através da replicação de objetos, mas através de mecanismos de checkpoint e rollback. Em [Zhang, et al.] é utilizado a técnica de réplica passiva através de mecanismos de notificação providos pela infra-estrutura de *grid*. Em [Townend and Xu] propõe a implementação de um mecanismo que executa um conjunto de serviços de web equivalentes, mas implementado sob diferentes plataformas (*n-version*). Após a execução um esquema de votação, o modelo atua sobres as respostas retornando a resposta mais coincidente. Apesar do tema, o serviço em *grid*, não fazer parte do escopo do FTWS-Orch, algumas semelhança podem ser encontrados entre os modelos. A diversidade do programas pode ser usada, permitindo serviços web implementados sob diferentes arquiteturas compreender o mesmo grupo de serviço.

O trabalho abordado em [Dobson] explora o uso da especificação WS-BPEL para alcançar a tolerância a falta em arquiteturas orientadas a serviço. Em nossa opinião, a principal fraqueza desta abordagem não é mostrar a falta no servidor de aplicação, onde o processo de negócios é executado, mas a abordagem apresentada para o *engine* BPEL por ser um ponto único de falha. Este trabalho menciona os serviços web como *stateful*, mas não apresenta como obter o determinismo necessário na técnica de replicação ativa.

Em nosso trabalho a flexibilidade do padrão BPEL é extensível para implementar um processo de negócios no lado do cliente permitindo em caso de falhas nas requisições no *engine* primários possa ser entregue ao *Engine Backup*. Em nosso trabalho anterior a infra-estrutura FTWeb [Santos, et al.] executa réplicas concorrentemente usam o modelo de *threads* e mecanismos para detectar réplicas faltosas. O FTWS-Orch gerencia a execução de réplicas usando o *engine* BPEL e combinando o uso da replicação ativa e passiva para detectar e redirecionar as réplicas faltosas a uma réplica *standby*.

O WS-FTM (*Web Service-Fault Tolerance Mechanism*) [Looker and Munro] é baseado em [Dialani, et al.] e aplica a técnica de *N-version* ao domínio de Serviços de Web para aumentar a confiabilidade do sistema. Diferentemente da nossa abordagem todos os componentes nesta infra-estrutura são localizados no lado de cliente.

8. Conclusão

Este artigo explora a flexibilidade provida pela especificação de processo de negócios e juntamente com o uso da técnica de replicação ativa e passiva para alcançar tolerância a falta em arquitetura orientada a serviço. Esta abordagem no trabalho prover o suporte a tolerância a falta de valor e *crash* com o uso da especificação BPEL. Neste trabalho não implementamos um sistema de monitoramento, quando uma réplica faltosa é detectada os pedidos automaticamente entregam a uma réplicas *standby*.

Com esta abordagem o administrador pode escolher quais serviços são mais críticos e definir para esse serviço uma réplica *standby* especifica ou escolher uma

mesma réplica *standby* para todo o grupo, com essa estratégia permite reduzir o custo de monitoramento em varias réplicas *standby*.

Os testes realizados no protótipo, sem o mecanismo de votação, mostrou que o desempenho é considerado aceitável, se levar em consideração a disponibilidade e a confiabilidade permitido por este modelo. Contudo este trabalho representa uma etapa inicial e em um trabalho futuro a proposta é desenvolver um modelo completo que inclua o uso de serviços web *statefull*.

Referências

Aghdaie, N. and Tamir, Y. (2002). Implementation and Evaluation of Transparent Fault-Tolerant Web Service with Kernel-Level Support. Proc. IEEE Intl. Conf. on Computer Communications and Networks.

Clement, L., Hately, A., Riegen, C. v. and Rogers, T. (2001). UDDI - Universal Description, Discovery and Integration, http://www.uddi.org/specification.html.

Dialani, V., Miles, S., Moreau, L., Roure, D. D. and Luck, M. (2002). Transparent Fault Tolerance for Web Services based Architectures Eighth International Europar Conference (EURO-PAR'02).

Dobson, G. (2006). Using WS-BPEL to Implement Software Fault Tolerance for Web Services. Proceedings of the 32nd EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO-SEAA'06).

Liang, D., Fang, C.-L., Chen, C. and Lin, F. (2006). "*Fault tolerant web service*." Jornal of Systems Architecture.

Looker, N. and Munro, M. (2005). "*WS-FTM: A Fault Tolerance Mechanism for Web Services*."

OASIS (2005). Organization fot the Advancement of Structured Information Standards, www.oasis.open.org.

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

Santos, G. T., Lung, L. C. and Montez, C. (2005). FTWeb: A Fault Tolerant Infrastructure for Web Services. Ninth IEEE International EDOC Enterprise Computing Conference (EDOC'05).

SOAP (2003). Simple Object Access Protocol. W3C World Wide Web Consortium., www.w3c.org/TR/soap.

Townend, P. and Xu, J. (2005). Fault Tolerance within a Grid Environment. Engineering and Physical Sciences Research Council (EPSRC´05).

W3C (2005). World Wide Web Consortiom, www.w3c.org.

WS-BPEL (2005). Business Process Execution Language, www.software.ibm.com/software/developer/library/ws-bpel.pdf.

WSDL (2001). http://www.w3.org/TR/wsdl

Zhang, X., Zagorodnov, D. and Hiltunen, M. (2004). Fault-Tolerant Grid Services Using Primary-Backup: Feasibility and Performance. Cluster, San Diego, California.