Smart Proxies para Invocação de Serviços Web Replicados

José Geraldo R. Junior
PUC Minas
Belo Horizonte, Brasil
jgrjunior@gmail.com

Glauber Tadeu S. Carmo
PUC Minas
Belo Horizonte, Brasil
glaubertadeu@gmail.com

Marco Tulio O. Valente
PUC Minas
Belo Horizonte, Brasil
mtov@pucminas.br

ABSTRACT

Smart proxies are objects often used to adapt and customize distributed object-oriented systems in a non-invasive way. For example, smart proxies are common used to provide support to quality of service attributes. In this paper, we investigate the use of smart proxies in order to provide replication transparency in middleware systems that support the implementation of Web services. The system presented in the paper, called SmartWS, supports the prime replicated server selection policies proposed in the literature. The paper also provides guidelines that help Web service users to choose the policy more suitable to a given application. A new server selection policy that combines several advantages of the already proposed policies is also described. The paper also presents results obtained from experiments performed with a prototype implementation of SmartWS. Such results reinforce the policy selection guidelines presented in the paper.

RESUMO

Smart proxies são objetos usados para adaptar e customizar de forma não-invasiva aplicações distribuídas, notadamente para adicionar nas mesmas parâmetros de qualidade de serviço. Neste artigo, investiga-se o uso de smart proxies para acrescentar transparência de replicação em sistemas de middleware destinados ao desenvolvimento de clientes de serviços Web. O sistema proposto, chamado SmartWS, inclui suporte às principais políticas propostas na literatura para seleção de serviços replicados. Ao longo do artigo, fornecem-se diretrizes que auxiliam um usuário de serviços Web a optar pela política mais adequada a sua aplicação. Adicionalmente, propõe-se uma nova política de seleção de réplicas que congrega pontos positivos das políticas existentes. Por fim, com o intuito de fornecer resultados quantitativos sobre a implementação do protótipo de SmartWS, são apresentados experimentos realizados com clientes reais de serviços Web replicados.

1. INTRODUÇÃO

Computação orientada por serviços é um novo paradigma de computação distribuída que acredita-se possuir potencial para alterar de forma radical o modo com que sistemas Web são atualmente projetados, implementados, publicados e localizados [12, 20]. O objetivo final deste paradigma é viabilizar o desenvolvimento de aplicações distribuídas fracamente acopladas que se beneficiem da infra-estrutura global de comunicação disponibilizada pela Web para integrar e automatizar processos de negócios envolvendo diferentes organizações. Atualmente, serviços Web constituem o conjunto de tecnologias com maior potencial para transformar em realidade as idéias e princípios propostos pelo paradigma de computação orientada por serviços [1, 10, 7]. Essencialmente, serviços Web são aplicações distribuídas que se valem de linguagens e protocolos baseados em XML para definição de interfaces remotas (WSDL [29]), para publicação e localização de serviços (UDDI [25]) e para troca de mensagens com processos remotos (SOAP [24]).

No entanto, diversos desafios precisam ser ainda vencidos a fim de que o paradigma de computação orientada por serviços e a tecnologia subjacente de serviços Web sejam largamente utilizados no desenvolvimento de sistemas. Dentre estes desafios, merece especial destaque a incorporação em serviços Web de mecanismos que garantam o atendimento a atributos de qualidade de serviço [17, 31]. Particularmente, dois destes atributos – desempenho e disponibilidade – são críticos em aplicações destinadas a uma rede com as características da Internet.

Por outro lado, três décadas de pesquisa e prática em sistemas distribuídos demonstram que replicação é um conceito chave quando objetiva-se incrementar o desempenho e a disponibilidade de aplicações distribuídas [11]. Assim, no caso prático de emprego de serviços Web, vislumbra-se a existência de diversos servidores, funcionalmente semelhantes e distribuídos ao longo da Internet. No entanto, não é razoável assumir qualquer nível de comunicação e coordenação entre estes servidores, visto que os mesmos são – e continuarão a ser – desenvolvidos de forma independente e autônoma por programadores localizados em qualquer ponto do planeta. Em tais cenários, cabe essencialmente aos clientes de serviços replicados a realização das seguintes tarefas: (a) escolha do servidor que melhor atenda aos seus requisitos de qualidade de serviço; (b) compatibilização da interface provida por este servidor com a interface requerida pelo cliente. Mais uma vez a teoria e a prática em

sistemas distribuídos recomendam que estas tarefas sejam encapsuladas por um sistema de *middleware* de apoio ao desenvolvimento de clientes de serviços Web, de forma a evitar um entrelaçamento entre o código funcional destes clientes e o código responsável pela implementação das referidas tarefas [27]. Em resumo, o sistema de *middleware* utilizado deve prover *transparência de replicação*.

No entanto, sendo serviços Web uma tecnologia recente, as plataformas de *middleware* mais utilizadas atualmente para desenvolvimento deste tipo de sistema ainda não disponibilizam suporte a replicação. Sendo assim, propõe-se neste artigo o uso de *smart proxies* para estender um sistema de *middleware* com suporte a transparência de replicação, isto é, a extensão proposta inclui a especificação e geração de *proxies* especiais, destinados a encapsular as tarefas relacionadas com a seleção da "melhor réplica" que atenda aos requisitos da aplicação cliente que está sendo desenvolvida. Smart proxies são um dos principais mecanismos de meta-programação usados para extensão e customização de sistemas de *middleware* [14, 28]. Por exemplo, suporte a *smart proxies* já foi proposto para outras plataformas de *middleware*, como TAO [28] e Java RMI [8, 23], sempre com o objetivo de adicionar parâmetros de qualidade de serviço a uma aplicação de forma modular e não-invasiva.

A extensão proposta, chamada SmartWS, disponibiliza recursos para: (a) definição das interfaces e da localização dos servidores que disponibilizam um determinado serviço Web; (b) definição das políticas que serão usadas pelos clientes para seleção dos servidores especificados no item anterior; (c) definição de objetos adaptadores, responsáveis pela conversão das interfaces providas por um servidor para a interface requerida pelo cliente; (d) definição de um protocolo de invocação, isto é, um conjunto de regras que determinem as seqüências válidas de chamadas de métodos definidos na interface de um serviço Web. Particularmente, SmartWS inclui suporte às seguintes políticas já propostas na literatura para seleção de serviços replicados [9, 18]: estática, randômica, paralela e melhor mediana. Avalia-se também no artigo os cenários em que cada uma destas políticas deve ser aplicada e os principais benefícios proporcionados pelas mesmas. O objetivo é fornecer diretrizes que permitam a um programador de clientes de serviços Web optar pela política mais adequada a sua aplicação. Adicionalmente, propõe-se no artigo uma nova política de seleção de réplicas, chamada PBM (*Parallel Best Median*), que congrega pontos positivos das políticas anteriores. Por fim, com o intuito de fornecer resultados quantitativos sobre os ganhos de desempenho e disponibilidade das políticas propostas, são apresentados alguns experimentos realizados com clientes reais de serviços Web replicados.

O restante deste artigo está organizado como descrito a seguir. Na Seção 2, são apresentadas as principais funcionalidades providas pelo sistema SmartWS. A Seção 3 descreve a interface de programação do sistema e a Seção 4 discute os resultados de alguns experimentos práticos realizados com a implementação de SmartWS. A Seção 5 discute trabalhos relacionados e a Seção 6 conclui o presente artigo.

2. SMARTWS: FUNÇÕES BÁSICAS

2.1 Políticas para Seleção de Servidores

2.1.1 Seleção Estática

Nesta política, cabe ao programador definir estaticamente, em tempo de implantação da aplicação, a ordem segundo a qual as réplicas de um determinado serviço serão invocadas. O primeiro serviço nesta ordem é invocado; se ele falhar, invoca-se o próximo e assim sucessivamente. Este processo se repete até que uma réplica responda com sucesso ou então até que todas as réplicas falhem, quando propaga-se uma exceção para o cliente.

Aplicações: Esta política é adequada para cenários onde se conhece *a priori* as características e o tempo médio de resposta de cada réplica de um serviço. Além disso, estas características não devem sofrer variações significativas ao longo do tempo. Com isso, é plenamente possível determinar e fixar em tempo de implantação do sistema a ordem com que as réplicas de um serviço devem ser invocadas. Evidentemente, deve-se invocar primeiro o servidor que reconhecidamente possui o menor tempo de resposta e assim sucessivamente. Com isso, otimiza-se o desempenho do cliente e simultaneamente acrescenta-se ao mesmo um mecanismo de tolerância a falhas (isto é, no caso do servidor mais rápido falhar, transparentemente invoca-se o próximo e assim sucessivamente). A política de seleção estática permite ainda alocar estaticamente clientes a suas réplicas. Por exemplo, pode-se definir que clientes de um determinado país acessarão preferencialmente os servidores deste país; em caso de eventual indisponibilidade deste servidor, pode-se definir que deve ser acessado, por exemplo, o servidor de um país vizinho. A principal desvantagem desta política é o fato de que cenários estáveis e bem conhecidos são mais comuns em redes corporativas do que em redes abertas e dinâmicas, como a Internet.

2.1.2 Seleção Randômica

Esta política escolhe aleatoriamente, dentre um conjunto de servidores Web, aquele que será invocado pelo cliente. Caso esta invocação seja bem sucedida, o resultado da mesma é retornado à aplicação cliente. Caso esta invocação falhe, faz-se uma nova escolha dentre as réplicas restantes. Este processo se repete até que uma réplica responda com sucesso ou então até que todas as réplicas falhem, quando propaga-se uma exceção para a aplicação cliente.

Aplicações: Esta política é recomendada quando se sabe *a priori* que *todas* as réplicas de um determinado serviço possuem características e tempos de resposta semelhantes. Ou seja, nestes casos, não faz sentido definir qualquer ordem no acesso às mesmas. Além disso, como esta política distribui uniformemente as requisições geradas por um cliente entre as réplicas disponíveis, ela contribui para balancear a carga de processamento de tais réplicas. Por outro lado, assim como no caso da seleção estática, a principal desvantagem desta política é que cenários constituídos por servidores com capacidades e cargas de processamento semelhantes nem sempre são comuns no contexto de serviços Web.

2.1.3 Seleção Paralela

Esta política invoca concorrentemente todas as réplicas conhecidas de um determinado serviço Web. A primeira resposta obtida é retornada à aplicação cliente e as demais são descartadas. A política falha quando todas as invocações disparadas concorrentemente falharem.

Aplicações: Seleção paralela é recomendada quando redução no tempo de resposta de invocações é um dos principais requisitos de um sistema. Por outro lado, esta política pode introduzir uma carga de processamento considerável tanto no cliente (que deve disparar múltiplas *threads* para realizar as invocações) como nos servidores disponíveis (pois cabe a todos eles processar a requisição do cliente). Além disso, esta política gera um maior tráfego na rede.

2.1.4 Seleção via Melhor Mediana

Para cada réplica de um serviço Web, esta política calcula a mediana dentre as k-últimas invocações deste serviço. O serviço efetivamente invocado é aquele que possui a menor mediana. Se este serviço falhar, prossegue-se invocando o serviço com a segunda menor mediana e assim sucessivamente. A política falha quando falharem todas as invocações disparadas segundo a ordem das medianas.

Aplicações: Em geral, em amostras sujeitas a diferenças consideráveis entre o menor e o maior elemento, como é freqüente no caso do tempo de resposta de servidores Web [9], a mediana representa de forma mais adequada uma amostra do que a média. Ao contrário das políticas estática e randômica, a seleção via melhor mediana propicia ainda uma adaptação a cenários dinâmicos, onde o tempo de resposta dos servidores consultados apresenta variações ao longo do tempo. Porém, caso tais cenários também sejam sujeitos a falhas, esta capacidade é prejudicada. O motivo é que se o melhor servidor falhar, nas próximas invocações, sua mediana ainda poderá ser a menor e, portanto, ele voltará a ser selecionado. O resultado será uma degradação do tempo de resposta do serviço (já que deve-se esperar a falha do primeiro servidor para só então redirecionar a chamada para o segundo). Uma alternativa, adotada em SmartWS, consiste em aumentar de forma arbitrária a mediana de um servidor que falhou, de forma a forçar sua retirada da primeira posição da fila de servidores a serem invocados. Esta alternativa, no entanto, tem a desvantagem de prejudicar este servidor caso o mesmo se recupere e volte a apresentar tempos de resposta que o qualifique como possuindo a menor mediana.

2.1.5 Seleção PBM

A política PBM (*Parallel Best Median*) combina aspectos positivos das políticas de seleção paralela e via melhor mediana. Suponha um sistema com r réplicas de um serviço Web. A política PBM invoca de forma concorrente o serviço com a menor mediana m, juntamente com os serviços cujas medianas são menores ou iguais a $m * k$, limitados a um máximo de p servidores ($k > 0$, $p \leq r$). Assim, k define o grau de paralelismo da solução e p define um limite superior para este paralelismo. Se todas as invocações disparadas em paralelo falharem, dispara-se de forma seqüencial invocações aos demais servidores, segundo a ordem de suas medianas, de forma a aumentar o grau de tolerância a falhas da política. Assim, uma invocação via política PBM falha quando todas as réplicas falharem.

No caso de falha de um servidor, na seleção PBM registra-se que seu tempo de resposta é igual ao maior tempo de resposta dos demais servidores mais uma unidade de tempo. O objetivo é incrementar sua mediana e, com isso, reduzir a chance deste servidor ser novamente selecionado em invocações futuras. No entanto, a cada n invocações de um serviço, a política PBM se comporta, nas próximas t invocações, de forma idêntica à seleção paralela. O objetivo é atualizar o tempo de resposta de todos os servidores, inclusive servidores que falharam em alguma invocação e tiveram seu tempo de resposta arbitrariamente incrementado. A idéia é, ao contrário do que ocorre na seleção via melhor mediana, permitir que tais servidores voltem a ser selecionados caso após uma seqüência de falhas, os mesmos se recuperem e forneçam tempos de respostas atrativos.

Cabe ao usuário da política PBM definir os valores dos parâmetros k, p, n e t mencionados acima, em função dos requisitos de sua aplicação e das características do ambiente de rede e dos servidores consultados. Particularmente, se $p = 1$, a política PBM se comporta de forma similar à política da melhor mediana (se diferenciando pelo fato de a cada n invocações, disparar as próximas t invocações em paralelo). Por outro lado, se $p = r$ e $k = \infty$, então a política PBM tem comportamento idêntico à política paralela.

Aplicações: Como afirmado, a seleção PBM conjuga em uma só política as vantagens das políticas de seleção paralela e via melhor mediana. Em linhas gerais, esta política deve ser usada quando se deseja conciliar tempo de resposta, balanceamento de carga e adaptação a cenários dinâmicos e sujeitos a falhas.

2.2 Adaptadores de Interfaces

A natureza aberta e global da Web favorece a disponibilização de servidores que implementam serviços funcionalmente semelhantes. Por outro lado, não é razoável supor que todas as réplicas de um determinado serviço utilizam a mesma interface remota. Em geral, estas interfaces podem diferir em diversos detalhes, incluindo nome dos métodos, número, tipo e ordem dos parâmetros de entrada, etc.

A fim de compatibilizar as interfaces remotas de um conjunto de servidores replicados, o sistema proposto se vale dos conceitos de interfaces abstratas e adaptadores de interfaces. Clientes de serviços replicados utilizam sempre interfaces abstratas, as quais têm a função de padronizar as diversas interfaces remotas providas pelas réplicas utilizadas. Objetos adaptadores são então usados para compatibilizar a interface abstrata de um cliente com as interfaces concretas das réplicas que este cliente consulta. Basicamente, um objeto adaptador deve implementar a interface abstrata e possuir uma referência para o serviço que está sendo adaptado. A implementação dos métodos da interface abstrata em um objeto adaptador deve se responsabilizar por adaptar e converter tais métodos à sintaxe esperada pelo método da interface remota da réplica. Em SmartWS, adaptadores devem ser definidos manualmente

pelos programadores de clientes de serviços Web, já que estes possuem conhecimento tanto da interface abstrata de um serviço como da interface remota das réplicas utilizadas no sistema.

Aplicações: Conforme afirmado, adaptadores devem ser usados quando as interfaces remotas providas pelas réplicas de um serviço são funcionalmente semelhantes, mas seus tipos não são compatíveis, devido a diferenças sintáticas na assinatura dos métodos providos.

Exemplo: A fim de apresentar o uso de adaptadores de interfaces, serão utilizados dois servidores Web públicos, originalmente desenvolvidos com o objetivo de ilustrar o uso da tecnologia de serviços Web. Ambos servidores disponibilizam um serviço de conversão de temperatura (de graus Celsius para Fahrenheit e vice-versa). O primeiro servidor, hospedado em um repositório de serviços Web conhecido como WebserviceX[1], disponibiliza a seguinte interface remota:

```
interface ConvertTemperatureSoap {
  int convertTemp(temp, int fromUnit, int toUnit);
}
```

O segundo servidor, hospedado na empresa de treinamento DeveloperDays[2], disponibiliza a seguinte interface remota:

```
interface ITempConverter {
   int ctoF(int t);
   int ftoC(int t);
}
```

Suponha uma aplicação cliente que utilize a seguinte interface abstrata para acessar estes dois servidores:

```
interface TemperatureConverter {
  int toFahrenheit(int tc);
  int toCelsius(int tf);
}
```

A fim de assegurar transparência de replicação, deve-se definir os seguintes objetos adaptadores para compatibilizar a interface abstrata do cliente com as interfaces remotas dos dois servidores. O primeiro adaptador converte a interface abstrata para a interface `ConvertTemperatureSoap` requerida pelo servidor `WebserviceX`, conforme mostrado a seguir:

```
class WebserviceXAdapter
        implements TemperatureConverter {
   private ConvertTemperatureSoap server;
   WebserviceXAdapter (ConvertTemperatureSoap s){
     this.server= s;
   }
   public int toFahrenheit(int tc) {
     return server.convertTemp (tc,
           TemperatureUnit.degreeCelsius,
           TemperatureUnit.degreeFahrenheit);
   }
   public int toCelsius(int tf) {
     return server.convertTemp (tf,
          TemperatureUnit.degreeFahrenheit,
          TemperatureUnit.degreeCelsius);
   }
}
```

O segundo adaptador converte a interface abstrata `TemperatureConverter` para a interface remota `ITempConverter`, conforme mostrado a seguir:

```
class DeveloperDaysAdapter
        implements TemperatureConverter {
   private ITempConverter server;
   DeveloperDaysAdapter (ITempConverter server) {
      this.server= server;
   }
   public int toFahrenheit(int tc) {
      return server.ctoF(tc);
   }
   public int toCelsius(int tf) {
      return server.ftoC(tf);
   }
}
```

2.3 Sessão

No caso de servidores replicados, freqüentemente é necessário desabilitar o uso de uma política de seleção dinâmica de réplicas após a execução de determinadas operações. Basicamente, em algumas situações, se uma determinada requisição op_{start} é destinada a uma réplica r de um serviço, deve-se destinar todas as outras requisições subseqüentes deste cliente para a mesma réplica r, até que uma outra operação op_{end} seja executada. Nestes casos, diz-se que as operações op_{start} e op_{end} delimitam uma sessão de uso de uma réplica, isto é, as mesmas respectivamente desabilitam e voltam a habilitar o emprego de uma política de seleção de réplicas.

Aplicações: Em geral, sessões de uso de réplicas devem ser estabelecidas quando uma operação altera o estado de uma réplica e exige que operações seguintes sejam processadas sobre o estado alterado. Por exemplo, em um sistema de comércio eletrônico, operações como `pesquisaProduto` podem ser processadas por qualquer servidor. Por outro lado, as operações `login` e `logout` delimitam uma sessão, fixando o uso de uma réplica.

2.4 Protocolo de Invocação

Interfaces são normalmente usadas para definir as assinaturas dos métodos constituintes de um serviço e com isso viabilizar verificação estática de tipos em clientes de tais métodos. No entanto, quase sempre existe um conjunto de regras que restringe a ordem com que as operações descritas em uma interface podem ser invocadas. Por exemplo, em um sistema de comércio eletrônico, a operação `logout` somente pode ser executada se antes tiver sido invocada a operação

[1] `www.Webservicex.net`
[2] `www.developerdays.com`

`login`. Em sistemas distribuídos, protocolos de invocação são usados para definir as seqüências válidas de mensagens que podem ser enviadas para servidores remotos [5, 30, 6].

Assim, o sistema SmartWS possibilita associar a interfaces abstratas a definição de um protocolo que define as seqüências legais de invocação de seus métodos. Este protocolo é descrito por meio de um autômato finito determinístico associado a cada cliente dos serviços providos por uma interface remota. As transições deste autômato representam chamadas aos métodos da interface abstrata à qual o protocolo está associado. Ao se invocar um método remoto, caso o estado atual do autômato não possua uma transição correspondente ao método, então a referida chamada é inválida, isto é, a mesma não deveria ter sido solicitada no estado atual do sistema. Para indicar este fato, o sistema propaga uma exceção para a aplicação cliente.

Aplicações: Em sistemas distribuídos tradicionais, para redes locais ou corporativas, o uso explícito de um protocolo de invocação pode não representar uma vantagem considerável, já que nestes casos clientes são desenvolvidos em geral pela mesma equipe de programadores encarregada do desenvolvimento de provedores de serviços. Como esta equipe conhece em detalhes o protocolo de invocação de cada interface remota, as chances de ocorrência de seqüências inválidas de chamadas são menores. Por outro lado, esta característica não se repete no caso de aplicações baseadas em serviços Web. Nestes casos, um determinado serviço pode ter milhares ou milhões de clientes, desenvolvidos por programadores diferentes. Com isso, as chances de ocorrência de seqüências inválidas de chamadas são maiores e, portanto, a incorporação explícita de código nos clientes para detectar tais chamadas evita que elas sejam transmitidas pela Web e processadas em servidores remotos. Existe ainda uma tendência que interfaces de serviços Web tenham "maior granularidade" – isto é, tenham mais métodos complexos – que interfaces de aplicações tradicionais baseadas em objetos distribuídos [4]. Esta tendência também motivou a incorporação de protocolos de invocação em SmartWS.

Exemplo: A Figura 1 descreve um autômato que modela o protocolo de invocação de uma aplicação de comércio eletrônico. Este autômato expressa uma regra de uso do sistema segundo a qual determinadas operações, tais como adicionar e remover produtos do carrinho e efetuar compras, somente podem ser solicitadas caso o usuário esteja "logado" no sistema.

Antes de finalizar esta subseção, é importante ressaltar que o objetivo de um protocolo de invocação é tão somente capturar seqüências de chamadas que são inválidas por construção, isto é, sem que seja necessário analisar requisitos semânticos do sistema alvo da invocação. Em outras palavras, existem seqüências de invocação que violam requisitos funcionais de uma aplicação e que não são detectadas por meio de um protocolo de invocação. Por exemplo, se um usuário tentar efetuar uma compra sem ter adicionado nenhum produto em seu carrinho de compras. Certamente, não é razoável dotar um protocolo de invocação de poder de expressão suficiente para detectar tal categoria de chamadas inválidas, pois isso demandaria especificar no cliente parte significativa da lógica da aplicação servidora.

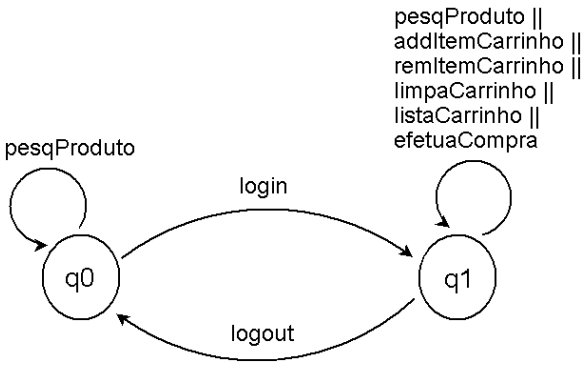

Figura 1: Protocolo de invocação de um sistema de comércio eletrônico

3. INTERFACE DE PROGRAMAÇÃO

Em SmartWS, *proxies* especiais, chamados *smart proxies*, são encarregados de implementar as funcionalidades básicas descritas na Seção 2. Basicamente, *smart proxies* são objetos que situam-se entre o cliente de um serviço Web e o *stub* disponibilizado pelo sistema de *middleware* para prover acesso remoto a este serviço. Assim, toda chamada remota de método realizada via SmartWS é capturada pelo *smart proxy* gerado pelo sistema. Cabe a este *proxy* selecionar uma réplica do serviço remoto que está sendo requisitado pelo cliente, possivelmente invocar um adaptador para compatibilizar a interface abstrata deste cliente com a interface remota do servidor selecionado e então destinar a chamada ao *stub* do sistema de *middleware* subjacente.

Em um *middleware* para desenvolvimento de aplicações baseadas na tecnologia de serviços Web, *stubs* são gerados automaticamente, a partir de uma especificação em WSDL. Da mesma forma, em SmartWS o código de *smart proxies* também é criado de forma automática (exceto o código dos adaptadores e da interface abstrata), a partir de uma especificação realizada pelo programador da aplicação cliente de um serviço Web. Esta especificação consiste nas seguintes cláusulas: `interface`, `webserver`, `policy`, `session` e `protocol`. A sintaxe de cada um destas cláusulas é descrita nos parágrafos seguintes[3].

Por meio da cláusula `interface` define-se o nome da interface abstrata que padroniza a comunicação entre a aplicação cliente e o smart proxy, usando uma sintaxe semelhante à do seguinte exemplo:

```
interface
   name= TemperatureConverter
```

[3]Como a sintaxe da linguagem usada em SmartWS para especificação de *smart proxies* é relativamente simples, optou-se neste artigo por omitir a definição de sua gramática. Porém, acredita-se que os exemplos mostrados são suficientes para que o leitor possa intuir boa parte da sintaxe proposta pelo sistema.

Neste exemplo, `name` denota o nome da interface Java contendo a definição de uma interface abstrata.

A cláusula `webserver` é usada para descrever os servidores que são acessados pela aplicação cliente. Para cada servidor, deve-se informar os seguintes valores: um *alias* para o servidor, a URI do arquivo contendo a especificação WSDL deste servidor, e opcionalmente o nome da classe adaptadora da interface abstrata para a interface implementada pelo servidor. O exemplo seguinte ilustra a sintaxe utilizada nesta parte da especificação:

```
webserver
   alias= Webx
   uri= www.Webservicex.net/ConvertTemp.asmx?WSDL
   adapter= WebserviceXAdapter
```

Normalmente, um arquivo de especificação de *smart proxies* possui mais de um elemento `webserver`.

A cláusula `policy` especifica a política de seleção de réplicas que deve ser implementada pelo smart proxy a ser gerado, usando-se para isso as seguintes palavras-chave: `static`, `random`, `parallel`, `median` ou `pbm`. A política escolhida é aplicada sobre todos os servidores especificados nas cláusulas `webserver` definidas no arquivo de especificação. No caso da política estática, os servidores são invocados segundo a ordem com que são declaradas as cláusulas `webserver` neste arquivo. O exemplo seguinte ilustra a sintaxe utilizada para selecionar a política PBM:

```
policy
  name= pbm
  k= 1.2;  p= 2; n= 10;  t= 3
```

A cláusula `session` especifica os métodos da interface abstrata que delimitam uma sessão de uso dos servidores especificados no arquivo, conforme mostra o exemplo abaixo:

```
session
  begin= login
  end= logout
```

Neste exemplo, os métodos `login` e `logout`, respectivamente, iniciam e finalizam uma sessão de uso de uma réplica.

Por fim, a cláusula `protocol` especifica o protocolo de invocação que determina a ordem com que devem ser chamados os métodos da interface abstrata. O exemplo a seguir mostra a especificação do protocolo de invocação descrito na Figura 1.

```
protocol
  states= q0, q1
  transitions=
     q0, pesqProduto, q0;
     q0, login, q1;
     q1, pesqProduto || addItemCarrinho ||
          remItemCarrinho || limpaCarrinho ||
          listaCarrinho || efetuaCompra, q1;
     q1, logout, q0;
```

4. RESULTADOS EXPERIMENTAIS

Nesta seção são descritos alguns experimentos realizados com o intuito de validar o projeto de SmartWS e as diretrizes propostas na Seção 2.1 para escolha da política de seleção de réplicas mais adequada a uma determinada aplicação cliente de serviços Web. Nestes experimentos utilizou-se como exemplo o serviço de conversão de temperatura descrito na Seção 2.2. Foi implementado um cliente que solicita a conversão de 100 temperaturas de grau Celsius para Fahrenheit. Este cliente foi executado em uma máquina Pentium IV 3 Ghz, com 1 GB RAM, Microsoft Windows 2000, JDK 5.0 e interface de rede FastEthernet 100 Mbps.

A seguir, descrevem-se os quatro experimentos realizados com este cliente, cada um deles correspondendo a uma configuração diferente de servidores replicados. Nos experimentos 3 e 4, a política de seleção estática não foi avaliada, uma vez que como o tempo de resposta dos servidores variava constantemente, não foi possível definir estaticamente uma seqüência de acesso aos mesmos.

Experimento 1: Neste primeiro experimento foram utilizados três servidores: WebserviceX e DeveloperDays (descritos na Seção 2.2) e um servidor local instalado no laboratório de Computação Distribuída da PUC Minas, possuindo a seguinte configuração: AMD Athlon 2600+, 320 MB RAM, Microsoft Windows 2000, JDK 5.0, Apache Axis 1.3 e Apache Tomcat 5.5.9. Neste cenário, conhecia-se de antemão que o tempo de resposta do servidor local era sempre inferior aos tempos dos servidores remotos.

O gráfico da Figura 2 apresenta os tempos médios de resposta obtidos pelas políticas de seleção de réplicas propostas por SmartWS. Como esperado, a política de seleção estática – com o servidor local possuindo prioridade de invocação – apresentou um tempo médio de resposta muito bom (praticamente idêntico ao da mediana). A política PBM apresentou um tempo de resposta ligeiramente maior[4]; já a resposta da política paralela foi cerca de 36% superior ao menor tempo (certamente devido ao *overhead* que a paralela gera na aplicação cliente devido à criação de múltiplas *threads*). Já o tempo de resposta da política randômica foi cerca de 18 vezes superior ao menor tempo.

Assim, este experimento reforça a recomendação da Seção 2.1.1, onde foi afirmado que a política estática é adequada para cenários onde se conhece *a priori* as características e o tempo médio de resposta de cada réplica de um serviço.

Experimento 2: No segundo experimento foram utilizados três servidores locais idênticos ao servidor local descrito no experimento 1, instalados no laboratório de Computação Distribuída da PUC Minas. Estes servidores foram iniciados sem nenhuma carga extra de processamento, além daquela gerada pelo experimento.

O gráfico da Figura 3 apresenta os tempos médios de resposta obtidos por cada uma das políticas de seleção de réplicas. Como mostra o gráfico, os tempos médios de res-

[4]Neste experimento e nos seguintes foram os seguintes os parâmetros de configuração da política de seleção PBM: k=1.2, p=2, n=10 e t=3.

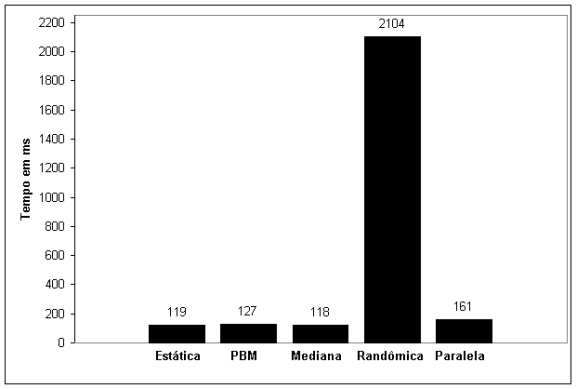

Figura 2: Tempos de resposta do primeiro experimento

posta das políticas randômica, mediana e estática foram os menores. Mais uma vez, a política PBM apresentou um tempo ligeiramente maior e a resposta da política paralela foi cerca de 42% pior que a melhor política neste cenário. O gráfico da Figura 4 apresenta o número de chamadas que foram endereçadas a cada um dos três servidores do experimento. Como se pode observar, as políticas randômica e PBM foram as que apresentaram a melhor distribuição de servidores.

Assim, combinando os resultados apresentados nos dois gráficos, pode-se afirmar que a política randômica foi a que apresentou o melhor comportamento neste segundo experimento. Esta conclusão reforça a diretriz definida na Seção 2.1.2, onde afirmou-se que a política randômica é recomendada quando se sabe *a priori* que *todas* as réplicas de um determinado serviço possuem características e tempos de resposta semelhantes.

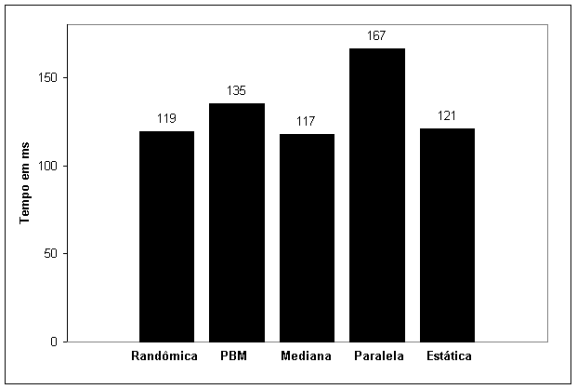

Figura 3: Tempos de resposta do segundo experimento

Experimento 3: No terceiro experimento, foram utilizados três servidores remotos: WebserviceX e DeveloperDays (descritos na Seção 2.2) e um servidor instalado no CEFET-MG. Sendo estes servidores remotos, durante o período de desenvolvimento e condução do experimento, o tempo de resposta dos mesmos variava de forma considerável, dependendo do dia da semana e do horário de acesso aos mesmos. Porém, a

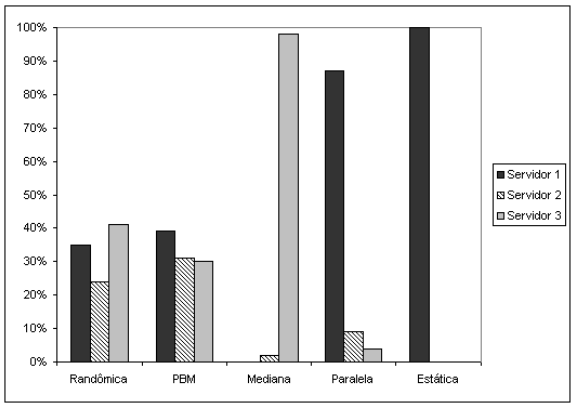

Figura 4: Distribuição de requisições do segundo experimento

experiência prévia de uso dos três servidores permitia afirmar que os mesmos apresentavam sempre uma baixa taxa de falhas.

O gráfico da Figura 5 apresenta os tempos médios de resposta obtidos por cada uma das políticas de seleção de réplicas. Como pode ser verificado no gráfico, o tempo médio de resposta da política baseada na melhor mediana foi o menor; o tempo das políticas PBM e paralela foram cerca de 7% superiores ao da mediana.

Este experimento reforça a recomendação da Seção 2.1.4, onde foi afirmado que a melhor mediana é uma política que propicia uma adaptação a cenários dinâmicos, onde o tempo de resposta dos servidores consultados apresenta variações ao longo do tempo, sem, no entanto, falharem.

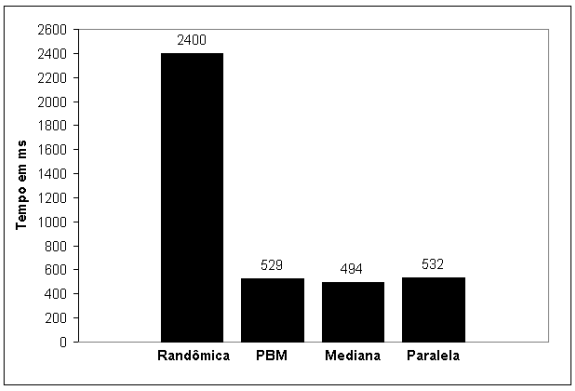

Figura 5: Tempos de resposta do terceiro experimento

Experimento 4: No quarto experimento, foram utilizados dois servidores que ficaram disponíveis durante todo o teste (os servidores WebserviceX e DeveloperDays) e um servidor local que ficou indisponível na primeira metade do teste (isto é, este servidor local falhou nas primeiras cinqüenta invocações do experimento e respondeu com sucesso às cinqüenta invocações finais). Este servidor local tinha a mesma configuração de *hardware* e *software* dos ser-

vidores descritos no experimento 1.

O gráfico da Figura 6 apresenta os tempos médios de resposta obtidos por cada uma das políticas de seleção de réplicas. Como pode ser verificado no gráfico, a política PBM apresentou o menor tempo de resposta; já política paralela apresentou tempo de resposta 6% superior à PBM. O gráfico da Figura 7 mostra o número de chamadas que foram endereçadas ao servidor local na segunda metade do experimento (quando este servidor passou a estar ativo). A partir deste momento, pode-se ver que as políticas PBM e paralela sempre passaram a utilizar o servidor local. Por outro lado, a mediana não foi capaz de detectar o restabelecimento do servidor local, pelos motivos descritos na Seção 2.1.4.

Concluindo, este experimento reforça a recomendação da Seção 2.1.5, onde afirma-se que a política PBM é mais adequada para cenários dinâmicos, sujeitos a variações imprevisíveis no tempo de resposta dos servidores envolvidos, os quais inclusive podem falhar ou permanecer indisponíveis com grande freqüência.

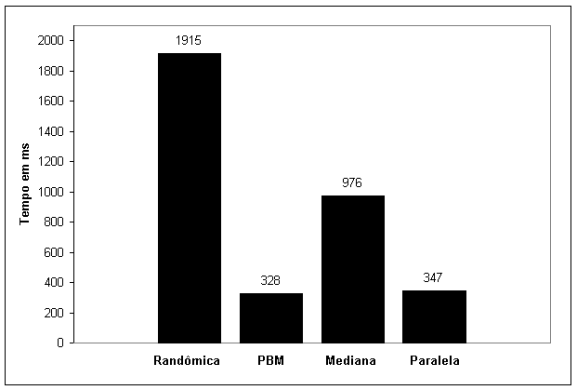

Figura 6: Tempos de resposta do quarto experimento

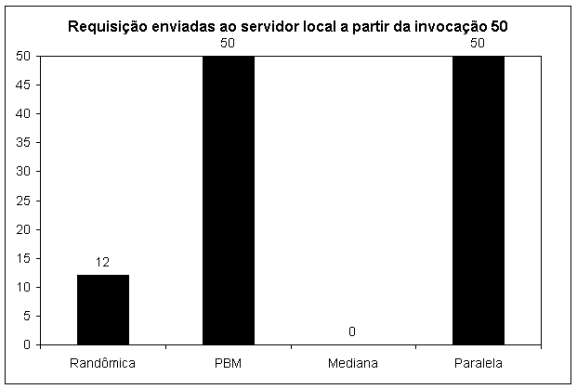

Figura 7: Distribuição de requisições do quarto experimento

5. TRABALHOS RELACIONADOS

O uso de políticas para seleção transparente de serviços Web replicados, implementadas por clientes, já foi objeto de outros estudos. Mendonça e Silva apresentam o *framework* RWS [18], o qual serviu de inspiração para o projeto e implementação do sistema descrito neste artigo. RWS implementa cinco políticas para seleção de servidores Web replicados: randômica, paralela, HTTPing, melhor última e melhor mediana. O artigo apresenta ainda uma avaliação empírica das políticas implementadas, realizada utilizando dois clientes e quatro servidores localizados em três continentes. Os resultados desta avaliação motivaram a ausência em SmartWS das políticas HTTPing e melhor última.

SmartWS possui as seguintes contribuições em relação a RWS: (a) proposta da política PBM para seleção de servidores; (b) suporte a adaptadores de interfaces; (c) suporte ao estabelecimento de sessões de uso de um servidor; (d) suporte a protocolos de invocação. É importante mencionar que no sistema proposto, *smart proxies* são gerados automaticamente, a partir de uma linguagem de especificação de alto nível. Já em RWS, cabe à aplicação cliente comandar a definição da política de seleção de réplicas. Por fim, acredita-se que no presente artigo são fornecidas diretrizes mais claras e objetivas sobre os ambientes de rede adequados a cada uma das políticas de seleção de réplicas propostas. Estas diretrizes foram validadas por meio de experimentos envolvendo o acesso a servidores reais.

Ledru [15] propõe a incorporação de *smart proxies* em Jini com o objetivo de redirecionar chamadas remotas em caso de falhas no servidor primário encarregado do processamento das mesmas. *Smart proxies* já foram propostos também para Java RMI [8, 23], CORBA [28, 14] e LuaOrb [16], sempre como um mecanismo de meta-programação bastante útil para estender e customizar de forma não-invasiva os serviços pré-definidos providos pela camada de *middleware*.

O problema de adaptação de interfaces em sistemas de objetos distribuídos já foi investigado em outros trabalhos. Vayssiere sugere o uso de um repositório de adaptadores para sistemas baseados em Jini [26]. Na referida proposta, um cliente Jini que esteja interessado em iniciar uma conversação com um serviço do tipo A deve consultar o serviço de nomes de Jini. Caso este possua um *proxy* registrado do tipo A, o mesmo é retornado para o cliente. Caso contrário, o próprio serviço de nomes consulta o repositório de adaptadores, procurando por um adaptador de A para um outro tipo A'. Caso exista tal adaptador, o mesmo é retornado para o cliente. Em comum com SmartWS, esta proposta também requer que adaptadores sejam gerados manualmente. Antonellis e colegas descrevem uma metodologia para determinação do grau de similaridade de duas interfaces remotas [2]. O objetivo é viabilizar a substituição de um servidor por outro que implemente uma interface similar. No entanto, a metodologia proposta não chegou a ser validada com servidores Web reais, implementados e disponibilizados por terceiros na Internet. Ponnekanti [21] descreve uma taxonomia dos principais tipos de incompatibilidades que podem ocorrer em uma interface remota quando a mesma evolui de forma independente de seus clientes.

Van den Bos e Laffra [5] definem um sistema denominado PROCOL que e usa expressões regulares para definir a sequência de métodos que um objeto pode acessar. Enquanto SmartWS propõe capturar somente seqüências de chamadas inválidas por construção, sem analisar os requisi-

tos semânticos, PROCOL permite restringir o acesso a alguns métodos dependendo do estado do objeto. Esta restrição é definida através do uso de *guardas*, que aumentam o poder das expressões regulares. No entanto, acreditamos não ser razoável incorporar ao protocolo de invocação a capacidade de detectar tal categoria de chamadas inválidas, pois para isso, há a necessidade de especificar no cliente parte significativa da lógica da aplicação servidora. De forma similar a SmartWS, Yellin e Strom [30] utilizam um autômato finito determinístico, especificando um conjunto de estados e transições, para expressar as condições de invocação e bloqueio do protocolo. Porém, assim como Canal [6], Yellin e Strom não apresentam suporte a serviços Web.

WS-Replication [22] é um sistema que utiliza comunicação em grupo para suportar replicação ativa em serviços Web. Na replicação ativa, todos as requisições para um determinado serviço replicado são processadas na mesma ordem em todas as suas réplicas. Entretanto, WS-Replication requer que as réplicas acessadas sejam configuradas com os componentes providos pelo sistema para suportar uma semântica de comunicação em grupo baseada em SOAP. Desta maneira, WS-Replication não disponibiliza transparência de replicação quando acessa provedores de serviços Web padrões, como os servidores usados nos experimentos descritos na Seção 4.

6. CONCLUSÕES

Apresentou-se neste artigo o sistema SmartWS, o qual utiliza o conceito de *smart proxies* para adicionar transparência de replicação em aplicações clientes de serviços Web. SmartWS provê suporte às principais políticas de seleção de servidores replicados propostas na literatura, incluindo as políticas de seleção estática, randômica, melhor mediana e paralela. O sistema suporta ainda uma nova política de seleção, chamada PBM, que combina aspectos positivos da política baseada na melhor mediana e da política de seleção paralela. Além de políticas de seleção de servidores replicados, SmartWS oferece recursos para adaptação de interfaces remotas, estabelecimento de sessões de uso de réplicas e para definição de protocolos de invocação.

No sistema proposto, *smart proxies* são gerados automaticamente, a partir de uma linguagem de especificação de alto nível, por meio da qual configura-se a política de acesso a servidores replicados de uma aplicação cliente. Um protótipo do sistema foi implementado em Java, tendo sido utilizado nos experimentos descritos na Seção 4. A versão atual do sistema suporta a geração de *smart proxies* compatíveis apenas com a plataforma Apache Axis [3], largamente utilizada no desenvolvimento de aplicações distribuídas baseadas na tecnologia de serviços Web. Futuramente, pretende-se prover suporte a outras plataformas de *middleware*, como JAX-WS [13] e Microsoft .NET [19].

Além de descrever o projeto e a implementação de SmartWS, foram propostas no artigo algumas diretrizes para ajudar o programador de clientes de serviços Web a escolher a política de seleção de réplicas mais adequada a um determinado ambiente computacional. Estas diretrizes, sumarizadas na Tabela 1, foram comprovadas em um um experimento do qual participaram servidores Web reais, implementados e disponibilizados na Internet por terceiros.

Ambiente Computacional	Política
Servidores com tempos de resposta conhecidos *a priori*, estáveis e diferentes entre si	Estática
Servidores com tempos de resposta conhecidos *a priori*, estáveis e semelhantes entre si	Randômica
Servidores com tempos de resposta desconhecidos ou que variam ao longo do tempo, mas que não apresentam altas taxas de falha	Mediana
Servidores com tempos de resposta desconhecidos ou que variam ao longo do tempo e que falham com freqüência considerável	PBM

Tabela 1: Diretrizes para escolha de políticas de seleção de réplicas

É importante mencionar ainda que os experimentos realizados mostraram que a política paralela não é recomendada na maioria dos cenários práticos, já que a mesma pode ser perfeitamente substituída pela política PBM. A principal desvantagem da política paralela é a carga considerável de processamento que a mesma impõe em todos os componentes de uma aplicação distribuída, incluindo clientes, rede e servidores.

Como trabalho futuro, pretende-se investigar o desenvolvimento de ferramentas capazes de escolher automaticamente a melhor política para um determinado ambiente de rede. Pretende-se também realizar mais experimentos práticos, envolvendo outros serviços Web disponibilizados na Internet.

Agradecimentos: Este trabalho foi desenvolvido como parte de um projeto de pesquisa financiado pela FAPEMIG (processo CEX-817/05 - Edital Universal 2005).

7. REFERÊNCIAS

[1] G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web Services: Concepts, Architecture and Applications*. Springer Verlag, 2004.

[2] V. D. Antonellis, M. Melchiori, L. D. Santis, M. Mecella, E. Mussi, B. Pernici, and P. Plebani. A layered architecture for flexible web service invocation. *Software Practice and Experience*, 36(2):191–223, 2006.

[3] Apache Axis. http://ws.apache.org/axis/.

[4] S. Baker and S. Dobson. Comparing service-oriented and distributed object architectures. In *OTM Conferences*, volume 3760 of *Lecture Notes in Computer Science*, pages 631–645. Springer, 2005.

[5] J. V. D. Bos and C. Laffra. Procol: a parallel object language with protocols. In *Object-oriented programming systems, languages and applications*, pages 95–102. ACM Press, 1989.

[6] C. Canal, L. Fuentes, E. Pimentel, J. M. Troya, and A. Vallecillo. Adding roles to CORBA objects. *IEEE Transactions Software Engineering*, 29(3):242–260, 2003.

[7] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and S. Weerawarana. Unraveling the web services web: An introduction to SOAP, WSDL, and UDDI. *IEEE Internet Computing*, 6(2):86–93, 2002.

[8] M. T. de Oliveira Valente, J. P. Santos, and C. F. de Moura Couto. Interceptação de Métodos Remotos em Java RMI. In *VII Simpósio Brasileiro de Linguagens de Programação*, pages 50–63, May 2003.

[9] S. G. Dykes, K. A. Robbins, and C. L. Jeffery. An empirical evaluation of client-side server selection algorithms. In *IEEE INFOCOM*, pages 1361–1370, 2000.

[10] C. Ferris and J. A. Farrell. What are web services? *Commun. ACM*, 46(6):31, 2003.

[11] T. K. George Coulouris, Jean Dollimore. *Distributed systems Concepts and Design*. Addison-Wesley, 2005.

[12] M. N. Huhns and M. P. Singh. Service-oriented computing: Key concepts and principles. *IEEE Internet Computing*, 9(1):75–81, 2005.

[13] JAX-WS. http://java.sun.com/webservices/.

[14] R. Koster and T. Kramp. Structuring qos-supporting services with smart proxies. In *IFIP/ACM Middleware Conference*, volume 1795 of *Lecture Notes in Computer Science*, pages 273–288. Springer, 2000.

[15] P. Ledru. Smart proxies for jini services. *SIGPLAN Notices*, 37(4):57–61, 2002.

[16] H. Mello and N. Rodriguez. Smart proxies in LuaOrb automatic adaptation and monitoring. In *23o Simpósio Brasileiro de Redes de Computadores*, 2005.

[17] D. A. Menasce. QoS issues in web services. *IEEE Internet Computing*, 6(6):72–75, 2002.

[18] N. C. Mendonca and J. A. F. Silva. An empirical evaluation of client-side server selection policies for accessing replicated web services. In *ACM symposium on Applied computing*, pages 1704–1708. ACM Press, 2005.

[19] Microsoft .NET Web Services Technology. http://msdn.microsoft.com/webservices/.

[20] M. P. Papazoglou and D. Georgakopoulos. Service oriented computing. *Communications of the ACM*, 46(10):24–28, 2003.

[21] S. Ponnekanti and A. Fox. Interoperability among independently evolving web services. In *ACM/IFIP/USENIX International Middleware Conference*, volume 3231 of *Lecture Notes in Computer Science*, pages 331–351. Springer, 2004.

[22] J. Salas, F. Perez-Sorrosal, M. Patiño-Martínez, and R. Jiménez-Peris. WS-Replication: a framework for highly available web services. In *15th World Wide Web Conference*, pages 357–366. ACM, 2006.

[23] N. Santos, P. Marques, and L. Silva. A framework for smart proxies and interceptors in RMI. In *15th International Conference on Parallel and Distributed Computing Systems*, 2002.

[24] Simple Object Access Protocol (SOAP). http://www.w3c.org/TR/soap.

[25] Universal Description, Discovery and Integration (UDDI). http://www.uddi.org.

[26] J. Vayssière. Transparent dissemination of adapters in jini. In *DOA*, pages 95–104, 2001.

[27] B. Verheecke, W. Vanderperren, and V. Jonckers. Unraveling crosscutting concerns in web services middleware. *IEEE Software*, 23(1):42–50, 2006.

[28] N. Wang, K. Parameswaran, and D. C. Schmidt. The Design and Performance of Meta-Programming Mechanisms for Object Request Broker Middleware. In *6th USENIX Conference on Object-Oriented Technologies & Systems (COOTS)*, pages 219–232, 2001.

[29] Web Services Description Language (WSDL). http://www.w3c.org/TR/wsdl.

[30] D. M. Yellin and R. E. Strom. Protocol specifications and component adaptors. *ACM Transactions Programing Languages Systems*, 19(2):292–333, 1997.

[31] L. Zeng, B. Benatallah, A. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang. QoS-Aware middleware for web services composition. *IEEE Transactions on Software Engineering*, 30(5):311–327, 2004.