Composição dinâmica de *Web-Services*

Sérgio Crespo C. S. Pinto, Rogério Samuel M Martins, Jorge L. V. Barbosa, João Gluz

Programa Interdisciplinar em Computação Aplicada - PIPCA – UNISINOS
Caixa Postal 15.064 – 91.501-970. São Leopoldo – RS – Brasil

crespo@unisinos.br, matrix.sharp@gmail.com, jbarbosa@unisinos.br, jcgluz@unisinos.br

Abstract. *The invocations of available services in the internet are static, always referring the same web service and to the same web method. When the service presents low readiness the performance of the application is reduced. To avoid this problem it is necessary that the application has the ability to identify the best available service and then it can invoke it.*

The inclusion of new protocols and new functionalities in web servicesarchitecture can allow that applications find available services in the internet and, besides, they can measure the quality of the service and redirect the call for the best service.

Resumo*. As invocações a serviços disponíveis na internet são construídas de forma estática, sempre referenciando o mesmo web service e o mesmo web method. Quando este serviço apresentar baixa disponibilidade o desempenho da aplicação será reduzido. Para evitar este problema é necessário que a aplicação tenha a habilidade de identificar o melhor serviço disponibilizado e então possa invocá-lo.*

A inserção de novos protocolos e novas funcionalidades na arquitetura de web services pode permitir que as aplicações encontrem serviços disponíveis na internet e, além disso, possam medir a qualidade do serviço e assim direcionar sua chamada para o melhor serviço.

1. Introdução

Com o surgimento das linguagens de script na Web, um novo potencial de serviços despertou, passando os navegadores de simples ferramentas de exibição de textos, imagens e sons, para processamento de informação e interatividade na Web. Os diversos scripts existentes tornaram-se serviços que foram utilizados como componentes por vários *sites*. Em meados de 2000 surgem os *Web-Services* como uma forma de melhorar o reuso e a padronização de utilização de serviços já existentes [Crespo 2000].

O temo *Web-Services* define um componente acessível via Internet que é autocontido, autodescrito, universalmente interoperável, configurável em tempo de execução e publicado e localizado por registros que são eles próprios web services. Um cliente ao usar um web service deve escolher aquele que melhor otimiza sua execução, mas é impossível garantir que um serviço será sempre ótimo por causa da mudança constante, dinâmica e imprevisível da Internet. A composição de serviços ainda enfrenta sérios desafios [Foster 02]. Quando é criada uma composição entre serviços, estabelecem-se critérios para que os serviços possam interagir visando sempre um ótimo desempenho. Isto nem sempre é garantido, pois o serviço solicitado não oferece garantia de qualidade.

As três mudanças na rede que podem afetar a composição do serviço são [Cronin 01]:

- Ruptura da conexão: não é possível garantir que um provedor de serviços permanecerá conectado com o cliente do início ao fim da execução do serviço;
- Mudança na taxa de transferência: existem duas situações que podem fazer com que a composição deixe de ser ótima. A primeira é quando a taxa de transferência da composição cai abaixo de outra possível composição. A segunda é quando a taxa de outra possível composição sobe acima da taxa da composição atual. Isto acontece onde não existe a garantia de qualidade de serviço em pelo menos um elo da rede;
- Entrada de novos componentes: uma entrada de um novo provedor de serviço pode levar a segunda situação da taxa de transferência e invalidar a composição atual.

O principal objetivo do artigo é extender a arquitetura de *Web-Services* para que a aplicação possa suportar composição dinâmica com web services, permitindo que a composição seja feita em tempo de execução pela aplicação.
O artigo, na seção 2, apresenta uma revisão das tecnologias associadas à *Web-Services*, na seção 3, são apresentados três trabalhos relacionados ao tema de composição dinâmica de *Web-Services*. Na seção 4, apresenta-se a arquitetura para composição dinâmica de *Web-Services*, na seção 5, são apresentados dois experimentos reais utilizando a arquitetura proposta. Na seção 6, as conclusões são apresentadas e na última seção as refrências bibliográficas.

2. Uma visão das tecnologias associadas ao uso de Web-Services

Web-Services são componentes de software, ou uma unidade lógica de aplicação, que se comunica através de tecnologias padrões de Internet. Ele provê dados e serviços para outras aplicações ou serviços. Diferentemente dos web sites tradicionais, projetados para as pessoas interagirem com informação, os *Web-Services* conectam aplicações diretamente com outras aplicações. A idéia básica é que essa conexão se dê sem que seja necessário efetuar grandes customizações nas próprias aplicações. Além disso, uma das premissas fundamentais é que o padrão usado pelas conexões seja aberto e independente de plataforma tecnológica ou linguagens de programação [Graham 02].

A arquitetura dos *Web-Services* é formada por três participantes: o solicitante de serviços, aqui denominado cliente, o provedor de serviços que é quem provê o *Web-Services* e o registro de serviços que é o diretório. Os participantes são ilustrados pela Figura 1 mostrando as colaborações entre eles. O provedor de serviços é responsável por disponibilizar o serviço e armazenar sua descrição em WSDL [Ferris 03] contendo detalhes de interfaces, operações e mensagens de entrada e saída. O solicitante do serviço é uma aplicação que invoca uma interação com o serviço, podendo ser um navegador web ou outra aplicação qualquer como um outro *Web-Service*. O registro de serviços é o local onde os provedores publicam seus serviços e onde os solicitantes fazem a procura [Souza 03]. Com base na análise da descrição dos serviços retornada pelo diretório, o solicitante pode decidir qual é o mais adequado a ser usado.

Após, o cliente inicia uma interação com o serviço fazendo uma requisição, onde o serviço retorna o resultado de sua execução em um pacote SOAP.

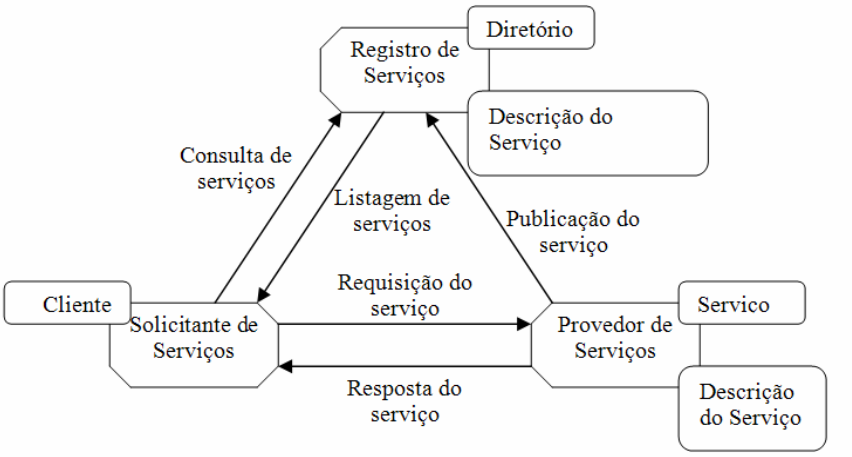

Figura 1 - Arquitetura de web services.

Para que seja possível uma fácil integração entre os componentes da arquitetura de *Web-Services*, a colaboração entre eles está baseada no uso de protocolos padronizados Estes protocolos transportam informações em XML, destacando-se entre eles como os mais utilizados o *Hypertext Tranfer Protocol* (HTTP), o *Simple Object Access Protocol* (SOAP), o *Web Service Description Language* (WSDL) e o *Universal Description, Discovery and Integration* (UDDI) [Souza 03].

3. Trabalhos relacionados

Esta seção descreve alguns trabalhos relevantes que tratam de propostas de composição dinâmica de *Web-Services*. O primeiro trabalho é de Li Gang, [Gang 04] que implementa um modelo de conector adaptável, o segundo é o trabalho de Hausmann [Hausmann 03] que utiliza ontologias para seleção automática de serviços.

Nesta mesma linha aparece o trabalho de Ruoyan [Zhang 07] que constrói composições automáticas de serviços utilizando ontologias. No fim da seção é feito uma análise dos trabalhos relacionados.

3.1. Modelo de conector de serviço adaptável

Neste trabalho, [Gang 04], apresenta um modelo de conector de serviço adaptável como solução para minimizar os problemas da volatilidade dos ambientes de rede. Conexões de serviço são tratadas como componentes individuais chamados conectores de serviço. Desta forma, é criado um modelo de conector de serviço adaptável que adota um mecanismo baseado em papel para ajustar as conexões entre serviços. Um papel é uma abstração de serviços com funcionalidades em comums. Esta abstração oferece uma estrutura de conector que habilita a reconfiguração da interação dos serviços. Com este mecanismo, um papel oferece estrutura de conexão flexível, habilitando adaptação de conexão de serviço por reconfiguração, como pode ser visto na Figura 2.

As características do papel apresentam um conjunto de funções que este oferece, sendo estas implementadas pelos serviços, invisíveis ao cliente. Quando uma mudança inesperada causa uma modificação na interação do papel com o serviço, o conector pode se adaptar às mudanças de interação de serviço, reconfigurando as características do papel. Quando um serviço envolvido em uma interação está indisponível, outro com a mesma característica pode substituí-lo, aumentando a flexibilidade da conexão.

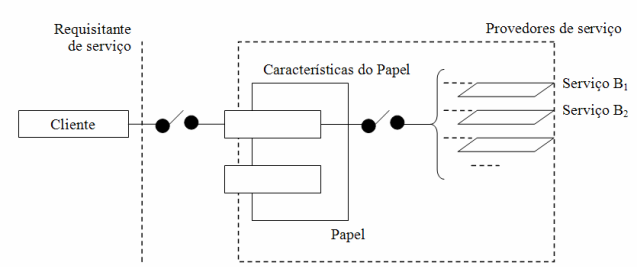

Figura 2 - Modelo do conector de serviço baseado em papel

3.2 Seleção automática de web services usando regras de transformação de grafo

A idéia do trabalho de [Hausmann 03] é permitir que um cliente possa descobrir um serviço em tempo de execução. Como a descoberta do serviço envolve a compreensão da semântica do serviço, ela é executada manualmente em tempo de desenvolvimento. Assim, o autor propõe o uso de regras de transformação de grafos para descrever a semântica de *Web-Services*, permitindo uma especificação precisa da semântica necessária para a descoberta automática de serviços.

Para a construção de uma aplicação que utilize *Web-Services*, o desenvolvedor consulta um diretório e analisa os serviços disponíveis através da sua interface, deduzindo assim sua semântica. Isto permite que ele escolha um ou um grupo de serviços que realizam a tarefa desejada. No exemplo de uma compra de livros, temos os seguintes serviços:

- placeOrder(ISBN : Integer, Address : String, CCData : Integer);
- placeOrder(ISBN : Integer, Address : String, BankAcc : Integer);

Observando as interfaces dos serviços pode-se deduzir que eles recebem o numero do livro que se deseja comprar, o endereço onde a compra deverá ser entregue e ou o número do cartão de crédito ou o número da conta para transferência do dinheiro.

Para um sistema de descoberta automática, a simples analise da interface não é suficiente, pois não é possível deduzir a semântica do serviço com precisão. Para que isto seja possível, Hausmann propõe o uso de ontologias, assim o cliente pode verificar de forma automática se o serviço realiza a tarefa desejada.

Um *Web-Service* pode usar uma ontologia para especificar sua semântica. A descrição semântica de um *Web-Service* estabelece um contrato especificando o seu significado e o seu propósito [Champion 02]. De acordo com [Fensel 03] um contrato consiste de uma pré e uma pós-condição. Para decidir automaticamente qual serviço usar, um cliente precisa usar um algoritmo que compare sua pré e pós-condição com a pré e pós-condição dos serviços oferecidos. Se a pré-condição do serviço é um sub-grafo da pré-condição do cliente então o cliente possui toda a informação necessária para executar o serviço; se a pós-condição do cliente é um sub-grafo da pós-condição do serviço então o serviço gera todos os efeitos esperados pelo cliente com alguns resultados a mais.

3.3 Composição automática de web services semânticos

Como em [Hausmann 03], [Zhang 07] propõe o uso de ontologias para fazer a descoberta automática dos serviços que satisfazem a necessidade do cliente. A diferença está na forma do uso da ontologia. Em vez de ser usada para identificar as entradas do serviço e seus efeitos sobre o sistema, é usada para identificar o serviço em si. E o algoritmo não apenas escolhe um serviço, mas constrói uma composição de serviços, com o menor tempo de execução e o melhor fluxo de dados, para a realização da necessidade do cliente. Para a especificação da ontologia do serviço, é construída uma

estrutura hierárquica que parte do serviço mais genérico até o serviço mais especializado, como mostra a Figura 3. Cada nodo da hierarquia irá especificar um serviço abstrato, definindo o nome e o tipo dos parâmetros de entrada e saída.

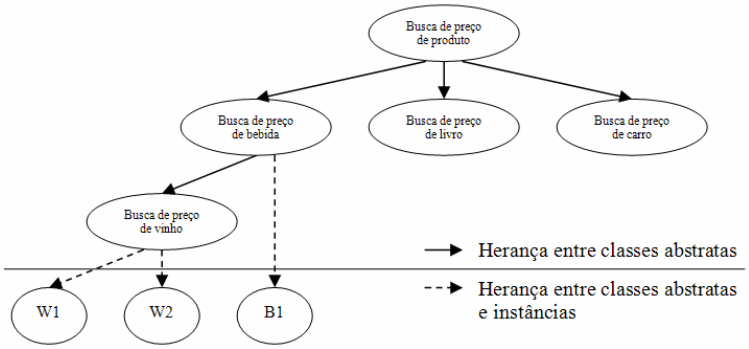

Figura 3 - Ontologia do domínio de procura de preço e instancias de serviços [Zhang 07]

No exemplo da Figura 3 observamos que o serviço W1 e o serviço W2 pertencem a mesma ontologia, assim podemos deduzir que eles têm a mesma interface e mesma semântica. Já o serviço B1 tem uma semântica mais genérica, significando que os parâmetros de entrada e de saída recebem e retornam uma categoria mais genérica. Uma necessidade do cliente é expressa através de uma *composite service query* de uma maneira muito similar a uma descrição de serviço em DAML-S[1]. Esta *query* inclui a descrição e a interface do serviço composto, definindo as entradas, as saídas e as restrições da composição. O usuário pode especificar parcialmente como a composição do serviço deve trabalhar e o tipo dos serviços individuais esperados [Ankolenkar 02].

4. Composição dinâmica de *Web Services*

Esta seção descreve uma arquitetura baseada em Padrões de Projeto de forma a permitir composição dinâmica de *Web-Services*. Tendo em vista o problema de se compor serviços na Web de forma dinâmica, faz-se necessário repensar a sua arquitetura clássica. Podemos assim destacar duas características que precisam estar presentes em uma composição de serviços ótima:

- Receber notificação de novos serviços publicados no diretório de serviços;
- Poder escolher um dentre vários serviços disponíveis para composição.

4.1 Arquitetura baseada em padrões de projeto

Usar padrões de projeto na modelagem da arquitetura de *Web-Services* torna mais legível o diagrama e explicita o papel de cada componente dentro da estrutura. As duas características presentes na composição dinâmica de serviços podem ser capturadas por padrões de projeto. A primeira característica é capturada pelo padrão *Observer* [Gamma 95] e a segunda é capturada pelo padrão *Strategy* [Gamma 95], os quais são compreendidos pelo padrão MCV [Buschmann 96] que define uma dependência um-para-muitos entre objetos, de maneira que quando um objeto muda seu estado todos os seus dependentes são notificados e atualizados automaticamente; e também para variar o comando a ser executado, dependendo do contexto.

Para que a primeira característica da composição dinâmica seja suportada pelos *Web-Services* é necessário mudar o fluxo de controle de construção. Na composição estática o diretório lista apenas uma vez para a aplicação os serviços disponíveis, muitas vezes

[1] http://www.daml.org/

em tempo de compilação. A partir daí a aplicação interage somente com o serviço escolhido e mais nenhum outro.

A primeira característica requer um modelo onde o diretório e as aplicações estejam interagindo constantemente para que a aplicação seja notificada sobre a publicação de novos serviços. A Figura 4 mostra o fluxo de controle de *Web-Services* baseado no padrão MVC. O modelo do padrão é formado pelos *Web-Services* que são do interesse dos clientes. Os clientes são como as *view* do MVC, precisam refletir o modelo para que possam garantir que a composição atual seja ótima. Os diretórios são colocados no lugar do controlador, pois a responsabilidade de monitorar o modelo (publicação de serviços) é retirada dos clientes.

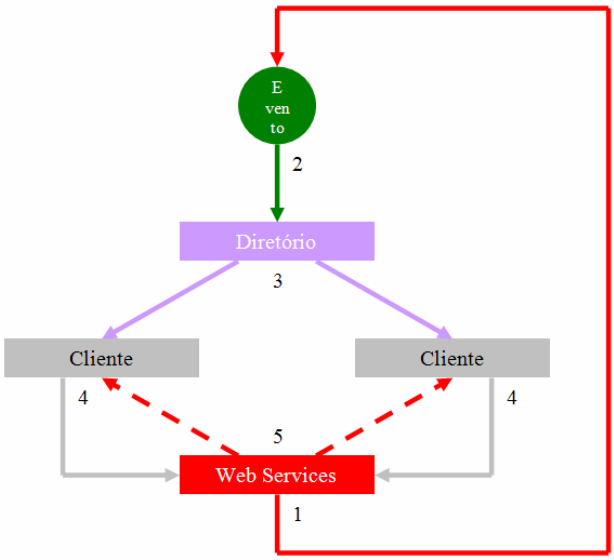

Figura 4 - Fluxo de controle de composição baseado no modelo MVC

Quando uma publicação de um novo serviço ocorre, é gerado um evento (1). Esta publicação é capturada pelo diretório de serviços (2), por estar recebendo o seu registro. Assim o diretório pode notificar os clientes (3) informando que um novo serviço está disponível. Com a notificação os clientes atualizam a lista de serviços disponíveis e passam a levantar estatísticas sobre os novos serviços. Quando existir uma composição mais otimizada do que a composição atual, os clientes desfazem a composição e criam uma nova composição (4 e 5). Este fluxo de controle permite que quando novo um serviço for publicado, um cliente possa ser notificado e então possa refazer sua composição se for preciso.

4.2 Estrutura

Para que o novo fluxo de controle seja viabilizado e para que a arquitetura apresente as duas características pertinentes à composição dinâmica, anteriormente citadas, é necessário modificar a arquitetura clássica para que o diretório de serviços possa notificar a aplicação sobre o registro de um novo serviço e também para que a aplicação possa refazer sua composição quando necessário.

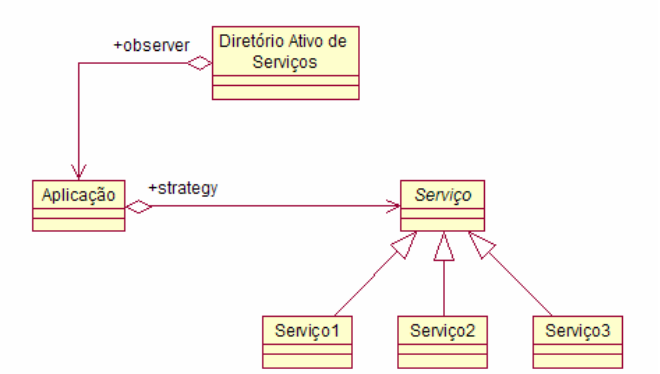

Figura 5 – Arquitetura abstrata de *Web-Services* com suporte a composição dinâmica baseada em padrões de projetos

Cada uma das características está relacionada com um padrão de projeto, como mostra a Figura 5. A primeira característica está relacionada com o padrão *Observer*, pois a aplicação deve se registrar no diretório como observadora e a cada nova publicação de serviços o diretório deve notificar a aplicação, que deve então tomar as medidas necessárias para sua atualização. A segunda característica está relacionada ao padrão *Strategy*, que deve escolher o serviço mais adequado de acordo com o contexto que a aplicação se encontra. O contexto deve ser baseado na disponibilidade dos serviços para a aplicação.

O diretório de serviços cumpre o papel do sujeito, que contém as informações (que é a lista de serviços publicados) relevantes à aplicação. Os serviços são as estratégias que a aplicação deve executar.

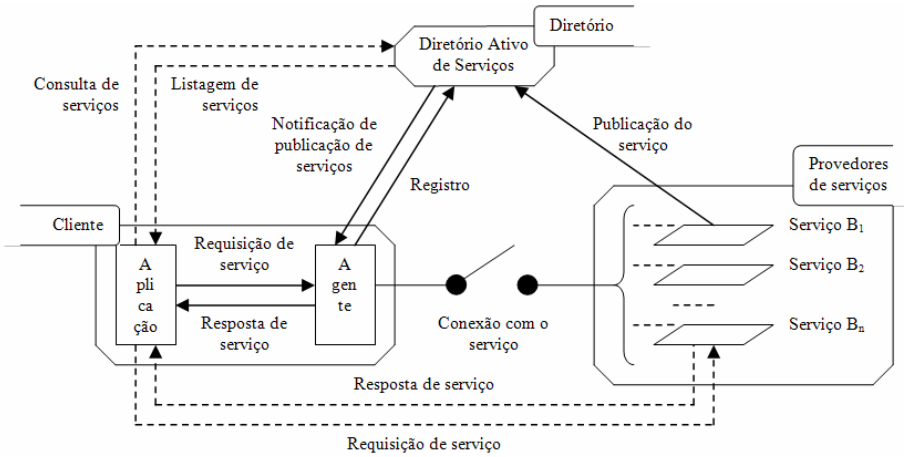

Figura 6 – Arquitetura concreta de web services com suporte a composição dinâmica

A Figura 6 mostra a arquitetura concreta utilizando como molde a arquitetura abstrata baseada em padrões de projeto, apresentada na Figura 5. Para que as colaborações entre os componentes sejam realizadas, estes necessitam ser alterados e novos protocolos de comunicação precisam ser inseridos (Tabela 1). O diretório de serviços passa a ser chamado de diretório ativo de serviços por notificar a aplicação cada vez que um novo serviço é publicado. Os protocolos de consulta e listagem de serviços passam a ser desnecessários, sendo substituídos pelo registro e notificação de publicação de serviços. O registro do cliente no diretório ativo é uma chamada *Web-Service* que passa sua localização (URL) e qual o tipo de serviço esperado. Utilizando estes parâmetros, o diretório de serviços faz uma chamada web service para a aplicação, invocando a

notificação de publicação de serviços, passando como parâmetros a localização, o nome do método e os parâmetros do serviço publicado.

O cliente precisa ser dividido em duas partes: uma que é a aplicação propriamente dita e a outra que é o agente responsável pela composição. Nesta nova estrutura, a aplicação não deve fazer a chamada diretamente ao *Web-Service*, mas sim fazer uma chamada de método a um agente que redireciona a chamada ao *Web-Service* com maior disponibilidade.

Descrevendo mais detalhadamente a arquitetura temos os seguintes participantes:

- Diretório Ativo de Serviços: contém a lista de serviços publicados;
- Cliente: composto pela aplicação e pelo agente;
    - Aplicação: faz uma requisição ao serviço através do agente;
    - Agente: estabelece qual é o serviço com maior disponibilidade para a composição e faz a interface entre a requisição do cliente e o serviço;
- Serviço: executa a ação requisitada pelo cliente.
- Publicação do serviço: inclui o serviço na lista de serviços disponíveis do Diretório Ativo de Serviços;
- Registro: registra a aplicação para receber notificações sobre a inclusão de novos serviços;
- Notificação de publicação de serviço: notifica a aplicação sobre a publicação de um novo serviço;
- Requisição de serviço: solicitação da aplicação para o agente fazer uma requisição a um serviço;
- Resposta de serviço: resposta do agente devolvendo à aplicação a resposta dada pelo serviço;
- Conexão com o serviço: composição entre o agente e o serviço.

Tabela 1 - Ponto de implementação dos protocolos da arquitetura de web services usando composição dinâmica

Protocolo	**Entidade invocadora**	**Entidade**	***Web method***
Publicação do serviço	*Web service*	Diretório Ativo de Serviços	*Publish*
Registro	Agente	Diretório Ativo de Serviços	*Register*
Notificação de publicação de serviço	Diretório Ativo de Serviços	Agente	*Update*
Conexão com o serviço	Agente	*Web service*	Não definido

Um agente de software avalia os requisitos sobre a qualidade do serviço. O agente poder escolher qual o melhor serviço para uma requisição. A principal unidade de medida é o tempo de resposta do serviço. Como a internet é muito volátil e o tempo de resposta varia constantemente, é preciso utilizar um histórico das chamadas realizadas ao serviço. Esse histórico deve ser ponderado, considerando a última chamada com maior peso. Matematicamente a equação do tempo de resposta ponderado pode ser dada por [simplificado de Larson 04]:

$$T_r = \frac{\frac{T_{r-1}}{\alpha t} + T_{uc}}{2}$$

Onde T_{uc} é o tempo de resposta da última chamada, T_{r-1} é a média ponderada do tempo de resposta anterior, t é o intervalo de tempo entre a medição de T_{r-1} e T_r e α é um fator de ajuste do tempo, quanto menor t e α, maior será o peso de T_{r-1}. O serviço que tiver a

menor média, será o escolhido para composição. O agente é uma entidade ativa e esta periodicamente coletando informações sobre os serviços. Para isto, ele deve fazer uma chamada com todos os parâmetros nulos aos *Web-Services*, medindo o tempo de resposta e atualizando suas estatísticas. Necessariamente os serviços não deverão executar nenhuma ação semântica sobre qualquer sistema e sobre si próprios, apenas deverão suportar uma execução simulada. Outras informações também podem ser coletadas pelos agentes, dependendo do que se pretende medir.

A identificação semântica do serviço é dada pela marcação ontológica hierárquica como descrito em [Zhang 07]. Esta marcação é uma etiqueta do tipo texto contendo todos os níveis hierárquicos do serviço separando cada nível por uma barra dupla. Como exemplo, os serviços W1 e W2 da Figura 3 receberiam como marca a etiqueta: "Busca de preço de produto//Busca de preço de bebida//Busca de preço de vinho" e o serviço B1 receberia a etiqueta: "Busca de preço de produto//Busca de preço de bebida". Como o serviço B1 é mais genérico que os serviços W1 e W2, seus parâmetros de entrada e de saída são um subconjunto dos parâmetros de W1 e W2, já W1 e W2 possuem os mesmos parâmetros, não diferindo na ordem e na quantidade. Serviços em outros ramos na hierarquia podem ter parâmetros em comum, mas não precisam respeitar nenhuma estruturação.

4.3 Diretório Ativo de Serviços implementado

O diretório ativo de serviços, Figura 7, é um *Web-Service* com dois *web methods*. Um *web method* é responsável pela publicação do serviço e deve ser invocado quando o serviço se torna disponível pela primeira vez. Ele recebe como parâmetros a localização do serviço e a ontologia do serviço.

```
[...]
public class DAS : System.Web.Services.WebService
{

    ...

    [WebMethod]
    public void Publish(string address, string ontologies)
    {
        Services.Add(new Register(address, ontologies));
        foreach (Register client in Client)
        {
            if (ontologies.Substring(0, client.Ontologies.Length) ==
client.Ontologies)
            {
                notify(client.Address, address);
            }
        }
    }

    [WebMethod]
    public void Register(string address, string ontologies)
    {
        Client.Add(new Register(address, ontologies));
        foreach (Register service in Services)
        {
            if (service.Ontologies.Substring(0, ontologies.Length) == ontologies)
            {
                notify(address, service.Address);
            }
        }
    }

    private void notify(string client_addreess, string service_address)
    {
        AbstractAgent agent = new AbstractAgent();
        agent.Url = client_address;
        agent.Update(service_address);
    }

}
```

Recebe a publicação de um serviço notificando todos os clientes registrados.

Recebe o registro de um cliente, notificando-o com todos os serviços já publicados.

Invoca o *web service* do cliente para notificação.

Figura 7 - Esqueleto de código do diretório ativo de serviços

A descrição UDDI foi removida para simplificar a implementação e por ser substituída pela marca ontológica, que permite deduzir todas as informações relacionadas ao serviço. O outro *web method* é o registro do cliente no diretório ativo. Ele recebe como parâmetros o endereço do cliente e a ontologia que o serviço a ser invocado deve ter.

O diretório faz uso do framework do agente, descrito na Figura 8, para fazer a notificação aos clientes, chamando o *web method Update*, que é implemento pelo *Web-Service* do agente, passando como parâmetros o endereço do novo serviço publicado. Ao ser publicado, todos os clientes que esperam aquele tipo de serviço são notificados recebendo como parâmetro o endereço do serviço, para então coletarem estatísticas. O mesmo acontece quando um cliente se registra no diretório. Ele recebe de imediato a notificação de todos os serviços que já estão cadastrados e que se enquadram na ontologia especificada, como mostra o esqueleto de código da Figura 8.

4.4 Cliente

O cliente é dividido em dois módulos, um deles é a aplicação propriamente dita e o outro é o agente. A aplicação contém a lógica de negócio e deve ser especializada para cada aplicação. O que os clientes têm em comum é o código que está por traz do agente e por isso pode ser colocado em um framework para ser compartilhado, como mostra a Figura 8. Este código contém o fluxo de execução para a tomada de decisão sobre qual *Web-Service* o cliente deve chamar.

```
[...]
public abstract class AbstractAgent : System.Web.Services.WebService
{

    [...]

    public AbstractAgent()
    {
        DAS das = new DAS();
        das.Register(GetHostName(), ontologies);
        Collector = new Thread(new ThreadStart(this.Collect));
    }

    [...]

    [WebMethod]
    public void Update(string address)
    {
        Services.Add(new Measure(address, Alfa));
    }

    private void Collect()
    {
        while (!Exit)
        {
            Thread.Sleep(Interval);
            foreach (Measure service in Services)
            {
                [...]
            }
        }
    }

    public abstract Object CallWebService(string Address);

    public string BestServiceAddress
    {
        get
        {
            [...]
        }
    }

    public object CallBestWebService()
    {
        return CallWebService(BestServiceAddress);
    }

}
```

Registra o cliente no diretório ativo de serviços. E inicia o coletor de estatísticas

Recebe a notificação do diretório ativo de serviços.

De tempos em tempos, coleta estatísticas de todos os serviços.

Invoca o melhor serviço.

Figura 8 - Esqueleto de código do Framework do Agente

Ao ser instanciado, o agente deve primeiramente se registrar no diretório ativo de serviços passando como parâmetros a ontologia e seu endereço e logo após iniciar o coletor de estatísticas. Assim o agente assume dois papéis: de uma *thread* e ao mesmo

tempo de um *Web-Service*. Como *Web-Service*, o agente possui o *web method Update* chamado pelo diretório ativo de serviços. Quando o diretório invoca esse método, é passado como argumento o endereço do serviço publicado, permitindo ao agente adicionar este serviço em sua lista de coleta de estatísticas.

Como o agente é uma entidade ativa, ele cria uma *thread*, a qual é a responsável pela coleta de informações sobre os serviços. Em certos intervalos de tempos a *thread* executa um *looping* sobre todos os serviços cadastrados e então mede o tempo de resposta de cada um atualizando as estatísticas sobre os serviços. Baseado nas estatísticas o agente pode definir qual é o *Web-Service* com maior disponibilidade e através do método *CallBestWebService*, chamado pela aplicação, este serviço é requisitado.

4.5 Análise

A Tabela 2 faz a comparação entre os adaptadores de conexão de serviços incluindo o conector de composição dinâmica, proposto neste artigo. Como pode ser visto, nos outros modelos é necessária a interferência de usuários ou programadores. Na composição dinâmica a própria aplicação é responsável por decidir qual é o serviço que deve ser usado. Quando um serviço que estava sendo usado torna-se indisponível, nenhuma adaptação é necessária, já que a aplicação está preparada, através do agente, para executar esta adaptação, refazendo a composição com outro serviço que possui a mesma semântica, através das marcações ontológicas.

Na conexão de serviço baseado em fluxo de controle e nos adaptadores de serviço, quando isto acontece, é necessária a modificação do código fonte. Neste momento fica explícito o problema causado pelo desuso do padrão *Strategy*, que tenta resolver justamente este tipo de problema. No conector de serviço baseado em papel é necessário uma reconfiguração manual da lista de serviços que podem ser usados quando nenhum serviço disponível está listado, valendo-se de uma aplicação parcial do padrão *Strategy*.

Tabela 2 - Propriedades dos adaptadores de conexão de serviço.

Característica	**Conexão de serviço baseada em fluxo de controle**	**Adaptadores de serviço**	**Conector de serviço baseado em papel**	**Composição dinâmica**
Executor da adaptação	Programadores	Programadores	Usuários	Aplicação
Modo de adaptação	Por modificação do código fonte	Por modificação do código fonte	Por reconfiguração	Desnecessária
Grau de autonomia	Não automático	Não automático	Semi-automático	Automático
Momento de adaptação	Em tempo de construção	Em tempo de construção	Ambos os tempos de execução e de construção	Em tempo de execução
Mudanças incrementais	Não	Não	Sim	Desnecessário
Efeitos semânticos	Sim	Sim	Sim	Não
Controle de impactos de mudança	Não	Não	Sim	Desnecessário

Em resumo, composição dinâmica de serviços é baseada na identificação semântica dos serviços e na sua disponibilidade, permitindo assim ser refeita toda e qualquer configuração necessária que tenha impacto sobre a execução da aplicação sem a necessidade de alteração na aplicação cliente.

5. Experimentos

Para a execução dos testes de avaliação da nova arquitetura foi construído um ambiente de execução para os estudos de casos. A estação de trabalho onde é executado o cliente fica completamente separada dos *Web-Services*, que executam cada um em servidores distintos, na internet. A localização do diretório ativo de serviços não é importante por apresentar baixa iteração com o cliente e com os serviços, apenas necessitando ter disponibilidade para ambos, assim optou-se por colocá-lo na mesma rede local do cliente.

O cliente é executado em uma máquina desktop com processador Pentium 4 de 3 GHz e com memória de 512 MB. O diretório ativo de serviços também é executado numa máquina desktop com processador AMD de 64 bits e clock de 3.2 GHz. Estas duas máquinas estão na mesma rede local conectadas por um *hub Fastethernet* de 10/100Mbps. O sistema operacional instalado nas máquinas é o Windows XP da Microsoft com o IES como servidor web. Tanto o cliente como o diretório foram implementados na plataforma Asp.Net usando como linguagem de programação o C#.

O conhecimento da configuração dos servidores dos serviços é irrelevante para o cliente já que suas disponibilidades serão medidas pelo agente e assim a configuração das máquinas e velocidades de conexões poderão ser desprezadas e substituídas pelas estatísticas de disponibilidade dos serviços.

Os testes de avaliação da arquitetura se basearam em dois estudos de casos. O primeiro experimento aconteceu em cima de um sistema de análise de crédito e o outro em cima de um sistema de compra de livros. Os sistemas foram construídos usando web services reais e disponíveis na internet.

5.1 Experimento 1: Análise de crédito de pessoa física

Quando uma pessoa vai fazer empréstimos em financeiras ou instituições bancárias, sua proposta normalmente é submetida a uma análise automática de crédito. A análise ocorre através da aplicação de regras contidas na proposta e a principal regra a ser considerada é a consulta do CPF do cliente em serviços de proteção ao crédito. Os serviços mais usados são o SPC e o SERASA. Estes serviços disponibilizam meios de conexão com seus bancos de dados para que os sistemas automáticos de análise possam realizar as consultas necessárias. E uma das formas de conexão é através de *Web-Services*. O sistema construído para o estudo de caso, Figura 9, recebe uma proposta contendo um CPF de um cliente, então em posse deste CPF realiza uma consulta a um serviço de proteção ao crédito usando *Web-Services*. Dois serviços foram cadastrados no diretório ativo de serviços, o SPC e o SERASA. Assim o agente mede a disponibilidade dos serviços e então quando a consulta é solicitada ele chama o melhor serviço.

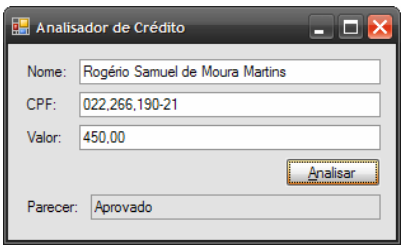

Figura 9 -Tela do sistema de análise de crédito usado no estudo de caso

A Tabela 3 e Tabela 4 mostram as estatísticas levantadas pelo agente durante um dia de execução do sistema, que podem ser visualizadas nos gráficos na Figura 10 e Figura 11 Os tempos de resposta foram gerados em milisegundos e convertidos para minutos. O alfa foi parametrizado em 1000 e o intervalo de medição foi estipulado em 1 hora.

Tabela 3 - Tempo de resposta do SPC e do SERASA nas doze primeiras horas

WS	01h	02h	03h	04h	05h	06h	07h	08h	09h	10h	11h	12h
SPC	4:18	4:50	4:17	4:46	4:35	4:27	4:53	4:53	4:59	4:29	4:04	4:48
T_r^{SPC}	129	145	128	143	137	133	146	146	149	134	122	144
SRS	2:55	3:02	3:11	2:53	3:08	3:02	5:40	5:30	5:45	5:51	5:41	5:43
T_r^{SRS}	87	91	95	86	94	91	170	165	172	175	170	171

Tabela 4 - Tempo de resposta do SPC e do SERASA nas doze últimas horas

WS	13h	14h	15h	16h	17h	18h	19h	20h	21h	22h	23h	24h
SPC	4:33	5:07	4:32	4:48	4:41	4:46	4:41	4:42	4:15	4:10	4:46	4:35
T_r^{SPC}	136	153	136	144	140	143	140	141	127	140	143	137
SRS	5:34	5:55	5:48	6:01	6:03	5:45	3:14	3:05	3:47	3:43	4:00	3:54
T_r^{SRS}	167	177	174	180	181	172	127	122	118	111	125	122

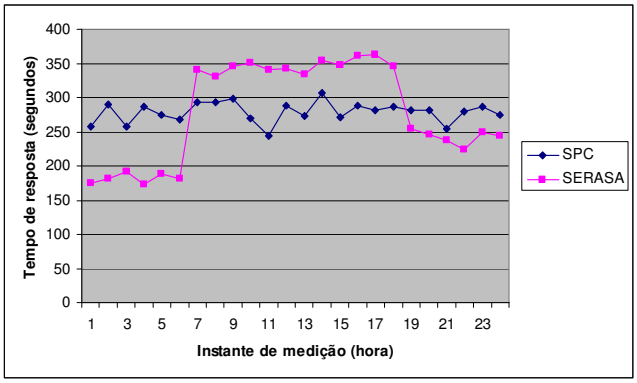

Figura 10 - Tempo de resposta do SPC e SERASA

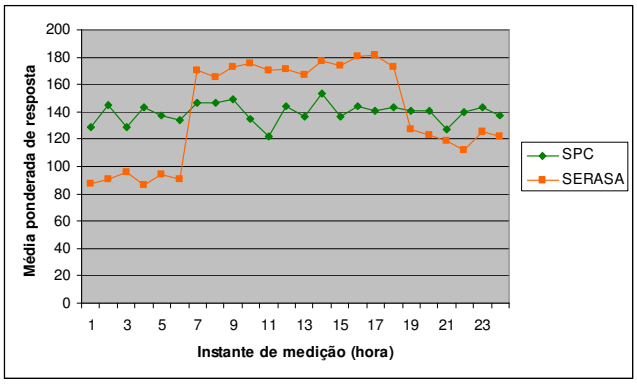

Figura 11 - Média ponderada de resposta do SPC e SERASA

Como mostra a Figura 11, o *Web-Service* do SPC apresentou tempo de resposta mais rápido do que o do SERASA entre as 7h e 19h, nos outros momentos o serviço do SERASA apresentou maior disponibilidade. Assim, a Figura 12 mostra o momento de invocação de cada *Web-Service*, quando este apresenta menor tempo de resposta.

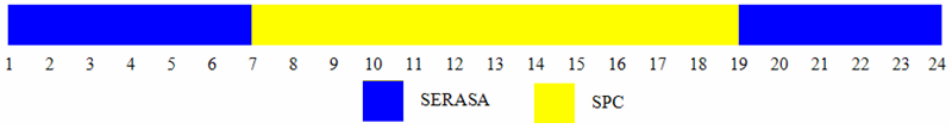

Figura 12 - Instante de uso de cada *Web-Services*

5.2 Experimento 2: Consulta a títulos de livros na Web

A grande maioria dos sites de venda de livros passou a disponibilizar *Web-Services* para que seus clientes possam consultar seus bancos de dados através de uma aplicação. Isto permite integrar o serviço de vendas de livros dos sites com outras ferramentas quaisquer. Assim é possível desenvolver aplicações que mostrem informações dos livros de uma livraria virtual como ilustra a Figura 13.

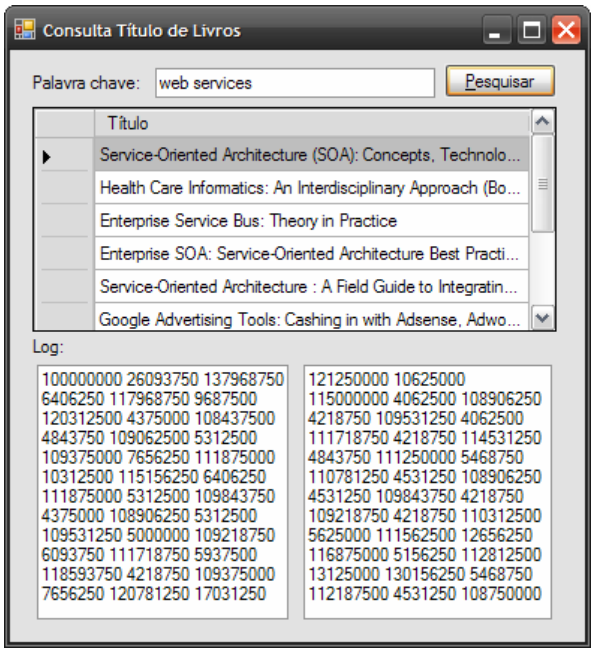

Figura 13 - Tela do sistema de consulta de títulos de livros na Amazon.

O sistema permite que o usuário entre com uma palavra-chave e a partir dela são listados todos os títulos dos livros cadastrados na livraria virtual.

A livraria virtual escolhida para este estudo de caso foi a Amazon (http://www.amazon.com) por ter sites espalhados pelo mundo. Exatamente como se pretende neste trabalho, a livraria permite escolher o *Web-Service* com maior disponibilidade e invocá-lo, pois todos os serviços espalhados pela rede mundial possuem a mesma semântica, e mais, todos são iguais, possuindo banco de dados e regras de negócios idênticas. Dois *Web-Services* foram cadastrados no diretório ativo de serviços onde o agente do cliente se registrou. Um deles foi o serviço disponibilizado no site internacional (http://soap.amazon.com/schemas2/AmazonWebServices.wsdl) e o outro foi o serviço disponibilizado no site do Reino Unido (http://soap.amazon.co.uk/schemas2/AmazonWebServices.wsdl). Para a execução do agente o tempo do intervalo de medição foi configurado em 10 segundos e o alfa em 7,5. O gráfico da Figura 14 mostra a média ponderada do tempo de resposta dos web services.

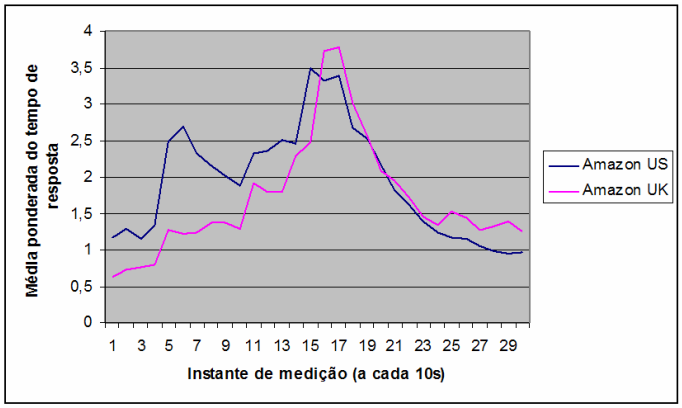

Figura 14 - Média ponderada do tempo de resposta dos *Web-Services* da Amazon

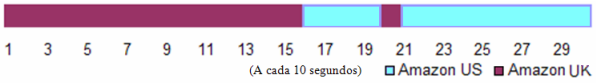

Figura 15 - Instante de uso de cada *Web-Services*

Como o esperado, a disponibilidade entre os *Web-Services* variou, tornando em alguns momentos um melhor do que o outro, sendo assim, o agente alternou as invocações sempre para o serviço com maior disponibilidade, como mostra a Figura 15.

6. Conclusões

As medições do tempo de resposta dos *Web-Services* nos experimentos refletem a estrutura dinâmica da internet. Os dados coletados revelam a constante variação no tempo de resposta de cada serviço oferecido, mostrando que não existe um padrão no tempo de resposta, o que torna imprevisível a disponibilidade de um serviço no momento de sua execução.

Aplicações, como o sistema de análise de crédito apresentado, que tem como principal requisito não funcional o desempenho e o tempo de resposta ao usuário, necessitam usar em qualquer momento o serviço que irá oferecer o melhor tempo de resposta. Para isto surge a necessidade de estar constantemente monitorando todos os serviços para medir sua disponibilidade e assim invocar o melhor serviço.

Fica evidente a necessidade da aplicação de poder medir a performance dos serviços e também poder alternar sua chamada ao serviço que melhor satisfazer algum critério estabelecido.

A arquitetura de composição dinâmica de *Web-Services* resolve este problema através de dois padrões de projeto MVC.

Um trabalho futuro de extrema importância é o suporte a transações nas invocações. Neste trabalho a invocação simplesmente é direcionada para qualquer serviço com maior disponibilidade. Mas muitas vezes surge à necessidade de realizar uma seqüência de invocações todas para o mesmo serviço, e isto não é tratado pela arquitetura e implementação atual.

Referências Bibliográfias

Ankolenkar, M. Burstein, J. R. Hobbs, O. Lassila, et. Al. WS Description for the semantic web. The First Intl Semantic Web Conference, 2002.

Crespo, Sérgio C. S. Pinto, (2000). Composição em WebFrameworks. Rio de Janeiro, 150 f. Tese (Doutorado) – Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática.

Buschmann, Frank; Meunier, Regine; Rohnert, Hans; Sommerlad, Peter; Stal, Machael. (1996), "Pattern-oriented software architecture a system of patterns". John Wiley and Sons ltd., England.

Champion, M.; Ferris, C.; Newcomer, E; Orchard, D. (2002) " Web service architecture". W3C Working Draft.

Cronin, Gareth. (2001), Web services: distributed computing for the new millennium? In: University of Auckland. http://www.croninsolutions.com/writing/WebServices.pdf, Maio de 2001.

D. Fensel, C. Bussler. "The web service modeling framework". (2003).Source: Electronic Commerce Research and Applications, Volume 1, Number 2, Summer , pp. 113-137(25), Publisher: Elsevier

Ferris, Christopher; Farrell, Joel.(2003) "What are web services?", Communications of ACM. vol46#6.

Foster, Ian; Kesselman, Carl; Nick, Jeffrey M.; Tuecke, Steven.(2002) "Grid services for distributed system integration", Computer, vol35#6.

Gamma, Erich; Helm, Richard; Johnson, Ralph; Vlissides, John. (1995). "Design patterns: elements of reusable object-oriented software". Reading Addison-Wesley, 1995. 395 p.

Gang, Li; Han, Yanbo; Zhao, Zhuofeng; Wang, Jianwu; Roland, M. Wagner. (2004) "An Adaptable Service Connector Model". In: Chinese Academy of Science #20026180-22.

Graham, Steve; Simeonov, Simeon ; Toufic Boubez; Glen Daniels; Doug Davis; Yuichi Nakamura; Ryo Neyama (2002) "Building web services with Java." Sams.

Hausmann, J. H., R. Heckel and M. Lohmann (2003), Towards automatic selection of web services using graph transformation rules, in: R. Tolksdorf and R. Eckstein, editors, Berliner XML Tage.

Larson, Ron; Farber, Betsy (2004). Estatística Aplicada. Prentice Hall Brasil.

Souza, Vinicius Costa de; Pinto, Segio Crespo Coelho da Silva (2003). "Sign WebMessage: uma ferramenta para comunicação via web através da Língua Brasileira de Sinais – Libras". In: Simpósio Brasileiro de Informática e Educação, Vol1, pg. 421-430, Rio de Janeiro.

Zhang, Ruoyan; Arpinar, I. Budak; Meza, Boanerges Aleman (2007). Automatic Composition of Semantic Web Services using Process and Data Mediation. In: Technical Report, LSDIS lab, University of Georgia, February 28.