UNIVERSIDADE DE SÃO PAULO
Instituto de Ciências Matemáticas e de Computação

ISSN - 0103-2585

CONSISTÊNCIA NA PUBLICAÇÃO E RECUPERAÇÃO DE INFORMAÇÕES DE WEB SERVICES NO REGISTRO UDDI

Júlio Cézar Estrella
Júlio Cesar Frigo Silva
Regina Helena Carlucci Santana
Marcos José Santana

Nº 337

RELATÓRIO TÉCNICO

São Carlos
Fev/2009

Consistência na Publicação e Recuperação de Informações de Web Services no Registro UDDI

Júlio Cézar Estrella, Julio Cesar Frigo Silva, Regina Helena Carlucci Santana, Marcos José Santana

Departamento de Sistemas de Computação
Grupo de Sistemas Distribuídos e Programação Concorrente
Universidade de São Paulo - Instituto de Ciências Matemáticas e de Computação
{jcezar, rcs, mjs}@icmc.usp.br, j.frigo@grad.icmc.usp.br
http://www.lasdpc.icmc.usp.br

Resumo Atualmente a publicação de informações sobre *Web Services* não é automatizada no contexto dos mais diversos motores de processamento de mensagens *SOAP*, tal o como Axis2. Se pensarmos em provedores e registros de serviços de forma distribuída, os provedores e os registros precisam se comunicar para garantir a consistência das informações sobre a localização do *Web Service*. Para isso, é preciso que quando ocorra a publicação de um serviço no provedor, o registro de serviços seja avisado. Também deve ser tratada a situação em que o serviço não esteja mais presente no provedor de serviços. A solução encontrada foi automatizar o processo de *publish/unpublish* de informações sobre os *Web Services*.

1 Introdução

Web Services tem sido um tópico amplamente pesquisado pela comunidade acadêmica e por empresas como IBM, Hewlett-Packard – HP e Microsoft [22]. O surgimento dos *Web Services* têm provocado uma mudança de paradigma na integração de aplicações, onde serviços podem ser compartilhados para permitir a otimização de processos de negócios numa ampla estrutura de tecnologia de informação (TI). A adoção efetiva dos *Web Services* não está tão longe de se tornar uma realidade, uma vez que as indústrias vêm considerando duas frentes: padrões e produtos [10]. Para que os *Web Services* sejam de fato consolidados, o desenvolvimento de um conjunto de padrões está em constante evolução para permitir a interoperabilidade dos produtos que estão por surgir num futuro próximo. Atualmente, os *Web Services* são capazes de integrar aplicações executando em diferentes plataformas, habilitar informações de uma base de dados de aplicações para serem disponibilizadas para outras, além de permitir que aplicações internas se tornem disponíveis na rede mundial de computadores, isto é, na Internet. Registros *UDDI* são componentes importantes em uma arquitetura orientada a serviços. São responsáveis pelo armazenamento de informações sobre os *Web Services*, de modo que clientes encontrem essas informações e façam a invocação de um serviço diretamente dos provedores de serviços. Trabalhos relacionados mencionam superficialmente a questão de desempenho de uma arquitetura orientada a serviço com a participação do *UDDI* [13], [14], [3]. A proposta deste documento é apresentar um novo mecanismo de notificação *publish/unpublish* automático que permita que as informações referentes aos web services sejam publicadas de forma consistente em um registro *UDDI*, permitindo assim que um *broker* de *Web Services* acesse tais informações sem se preocupar com a incosistência dos dados.

A seção 2 trata de uma contextualização de *Web Services*. A seção 3 detalha os principais trabalhos relacionados envolvendo *UDDI* e o processo de publicação de serviços. A seção 4 demonstra um novo mecanismo de notificação (*publish/unpublish*) automático de *Web Services*. Os resultados obtidos são, por sua vez, apresentados na seção 5. Finalmente as conclusões são apresentadas na seção 6.

2 SOA e Web Services

Um *Web Service* é definido pela W3C (World Wide Web Consortium) como uma aplicação identificada por uma *URI* (*Uniform Resource Identifier*), cujas interfaces e ligações são definidas, descritas e descobertas utilizando-se uma linguagem padrão como *XML* (*Extensible Markup Language*) [7]. *Web Services* são uma implementação de uma arquitetura orientada a serviços, referenciada na literatura de *Web Services* como *SOA* (*Service Oriented Architecture*) [5]. Pelas definições apresentadas, pode-se observar que diferentes explicações sobre *Web Sevices* estão disponíveis, desde as mais simples até as mais completas como a sugerida pela W3C. Algumas definições mencionam que as interações entre *Web Services*

ocorrem tipicamente como chamadas *SOAP* (*Simple Object Access Protocol*), protocolo de comunicação baseado em *XML* para a interação de aplicações [17]. *SOAP* é apresentado como um *backbone* para uma nova geração de aplicações de computação distribuída, independente de plataforma e de linguagens. Além disso, as descrições de interfaces dos *Web Services* são expressas usando uma linguagem denominada *WDSL* (*Web Service Description Language*) [24]. Na literatura sobre *Web Services* é apresentado também um protocolo para serviços de diretório que contém as descrições dos *Web Services*, denominado *UDDI* (*Universal Description Discovery and Integration*). Esse protocolo funciona como um registro de serviços sendo um importante componente da arquitetura orientada a serviços [6]. Uma arquitetura orientada a serviço é uma maneira lógica de construção de um sistema de software para prover serviços ou para aplicações de usuários finais ou para outros serviços distribuídos em uma rede, através de interfaces públicas (nesse contexto destaca-se a *WSDL*) disponíveis [16]. O *UDDI* habilita ainda os clientes dos *Web Services* a localizar serviços ou descobrir seus detalhes e permite que registros operacionais sejam mantidos para diferentes propósitos em diferentes contextos [12]. Um *Web Service* suporta interação direta com outros agentes de software usando mensagens baseadas em *XML*, trocadas via protocolos baseados na Internet [15]. A pilha conceitual dos *Web Services* envolve três camadas principais: camada física (*wire layer*), camada de descrição (*description layer*) e camada de descoberta (*description layer*). Estas são mostradas na figura 1.

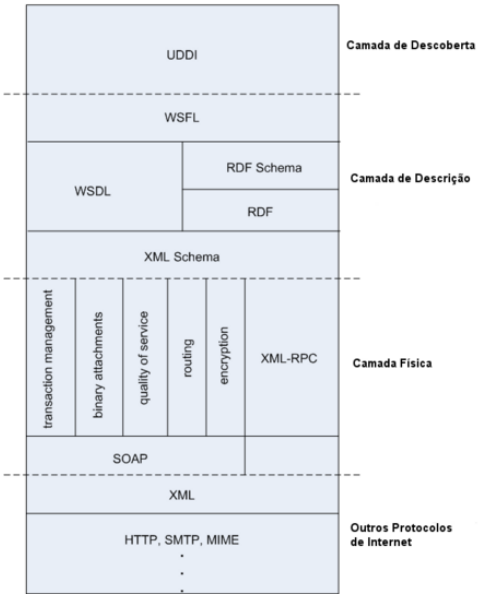

Figura 1. Pilha conceitual dos Web Services
[4]

3 Trabalhos Relacionados

Obter informações sobre *Web Services* nem sempre é uma tarefa trivial. Principalmente quando se fala em um arquitetura distribuída em que os *Web Services* estão localizados em um local e as informações sobre esses *Web Services* encontram se armazenadas em uma localidade diferente. Muitos trabalhos da literatura abordam a necessidade de um registro de serviços mais confiável e ao mesmo tempo consistente, de modo que a busca por determinado serviço seja realizada em menor tempo. Em [19] os autores apostam no uso da Web Semântica para tentar endereçar *Web Services* de forma apropriada, através de *UDDIs* confiáveis e uma infra-estrutura para publicação e descoberta de *Web Services*. O artigo discute os seguintes pontos:

- Publicação e descoberta de serviços baseada na sintaxe e semântica.
- Uso de *QoS*
- Uso de uma infraestrutura escalável que organize registros baseado em domínios e permite que usuários acessem múltiplos registros sob uma única visão.
- Descoberta de serviços por usuários finais e desenvolvedores.

Aborda como aspectos negativos da especificação UDDI:

- Descrição deficiente do serviço
- Descrição deficiente das necessidades do requisitor do serviço;
- Filtragem adicional é feita de modo manual;
- Falta de uma interface amigável ao usuário final;
- Questão de escalabilidade. Problema de centralização do *UDDI*.

Informações armazenadas no *UDDI* são semanticamente diferentes. Não há um padrão específico. A infraestrutura deveria acomodar diferentes mecanismos da publicação e descoberta. A distribuição física deveria ser transparente ao usuário (visualisar uma coleção de serviços sem saber de onde vem cada um). Apesar de o artigo demostrar o funcionamento por meio de ontologias, não há discussão alguma de como tal arquitetura funcionaria na prática, nem mesmo algum impacto em relação à avaliação de desempenho. Em [2] é abordado o problema de escalabilidade em gerenciar grandes quantidades de dados de fontes diferentes. Os autores propõem o desenvolvimento de uma arquitetura para descoberta de *Web Services* distribuídos. As limitações do *UDDI* (padrão) levam a pensar na proposta de um sistema de *UDDI* distribuído, pois a descoberta eficiente de *grids services* é essencial para o sucesso da computação em grade. Uma vez que o foco do trabalho seja a construção de *UDDIs* distribuídos, os autores não relatam como seria avaliada a eficácia do mecanismo proposto e nem mesmo quais técnicas seriam utilizadas para esse fim. Esses pontos estão em aberto e não são discutidos no artigo. O registro *UDDI* precisa prover alta vazão, baixo tempo de resposta, alta disponibilidade e acesso a dados consistentes. Um modo de satisfazer tais requerimentos é através da replicação, como mostra o trabalho apresentado em [23]. Muitos pesquisadores investigam mecanismos

para melhorar a interface de acesso do *UDDI* para a busca de informações dos *Web Services*. Poucos trabalhos, no entanto, se preocupam em avaliar o impacto do *UDDI* no desempenho da invocação de um *Web Service*, como por exemplo o trabalho caracterizado em [20].

O trabalho apresentado em [25] se preocupa com a questão da invocação de um *Web Service* a partir das informações contidas no *UDDI*. Mencionam que em um ambiente real é difícil invocar dinamicamente *Web Services* que possuem a mesma funcionalidade dentre muitos existentes em uma base muito grande de *Web Services*. Os autores propõem um framework que extende o registro *UDDI* e analizam a aplicabilidade e as características deste framework. Não se preocupam, porém, com a avaliação de desempenho. Em [11] é proposta uma abordagem generalizada para construir um registro *UDDI* baseado em *QoS*, cujo objetivo é tornar a busca de informações mais eficiente. Em [13] é proposto um registo *UDDI* orientado a domínio. Nesta proposta, uma base de dados centralizada e externa é usada para armazenar informações não funcionais de serviços e suas relações. O trabalho discutido em [3] propõe um método para avaliar modos de operação em um ambiente de computação orientada a serviço. Os autores avaliaram o desempenho de algumas implementações de *UDDI* e incorporaram os resultados de medidas de desempenho em um software de simulação. Utilizando este software de simulação, foram avaliados dois modos de operação que suportam a utilização do *UDDI* para permitir a confiabilidade de serviços em um cenário de processos de negócios. Em [18] é apresentado um sistema de controle de acesso ao *UDDI*. A idéia é limitar a busca de informações através de um controle de acesso que permita que o desempenho do *UDDI* não seja degradado. O trabalho é interessante, mas nenhuma avaliação de desempenho é abordada. Em [14] é caracterizado um exemplo de como grafos de *Web Services* podem ser suportados pelo registro *UDDI*, adicionando a ele uma estrutura de dados auxiliar. Embora a conclusão do artigo mencione algum tipo de experimento, nenhum deles está associado à avaliação de desempenho.

A falta de características adicionais que poderiam estar presentes no *UDDI* também é objeto de estudo na literatura especializada. O trabalho discutido em [21] propõe a implementação de uma extensão para o registro *UDDI*, denominada *UDDIe*. Dentre as deficiências apontadas no *UDDI* destaca-se:

- As páginas se baseiam em poucos atributos e não provêem um mecanismo de atualização do registro de serviço. Dessa forma a pesquisa por um determinado serviço fica restrita aos atributos listados e poderá haver serviços que não estão mais disponíveis ou com uma especificação desatualizada.

As soluções propostas pela implementação do *UDDIe* consistem em além das páginas brancas, amarelas e verdes presentes no *UDDI*, adicionar páginas azuis que dizem respeito as propriedades associadas a um serviço. Os serviços devem apresentar um período de tempo de permanência no registro. Dessa forma, teríamos as seguintes extensões ao *UDDI*:

- *Leasing*: serviços com tempos definidos

– Extensão da classe *businessService* com entidade *propertyBag*: pesquisa por outros atributos.
– Extensão do método de pesquisa: permite resultados por faixas numéricas/-lógicas.

Como as alterações tratam-se na verdade de extensões, o *UDDIe* pode co-existir com *UDDI*. O ponto positivo do artigo diz respeito aos propósitos do *UDDIe*, e quando ele deve ser aplicado em relação ao *UDDI*. Explica as extensões prospostas no modelo extendido, e exemplifica um serviço publicado utilizando tais extensões. No final do artigo cita brevemente *QoS*. Como ponto negativo não apresenta quaisquer métricas que justifiquem as proposições feitas, uma vez que se baseia apenas em conceitos.

O trabalho caracterizado em [20] tem como objetivo avaliar o desempenho de registros *UDDI* baseando-se em duas questões:

– Desempenho em operação normal.
– Desempenho em operações simples com processos concorrentes.

Basicamente a idéia consiste em avaliar primeiro o registro em condições normais e então avaliar a degradação do desempenho quando outras funções são executadas no mesmo registro de maneira concorrente. Para verificar tais métricas foram utilizados:

– Registros *jUDDI* (implementação *UDDI* baseada em Java)
– Uma ou mais aplicações *UDDI* foram estabelecidas para um mesmo domínio de interesse
– Grupos específicos de usuários com acesso a um conjunto limitado de registros que referenciam um grupo muito específico de serviços.
– *Tomcat 4.1 Web Server*.
– *MySQL database*.

A conclusão obtida foi que um grande número de solicitações concorrentes não apresentam um efeito significativo na performance dos registros *UDDI*. Porém publicações concorrentes fazem com que o tempo do serviço aumente 5 vezes para requisições e 10 vezes para publicações. Para tais valores, considerando uma publicação adicional por segundo, quando há 2 publicações adicionais por segundo, o registro fica travado e entra em *deadlock* em alguns casos. O ponto positivo deste artigo são a utilização de medidas para avaliação de desempenho e a exposição de métricas que justificam as conclusões obtidas no trabalho.

O desempenho de todas as interações em uma arquitetura orientada a serviços pode ser afetado se o registro *UDDI* não apresentar mecanismos rápidos de busca de informações dos *Web Services*. Para desenvolvedores, a característica mais importante em um motor de busca de informações é a facilidade de uso e a velocidade. Neste quesito, o trabalho apresentado em [9] mostra um mecanismo que utiliza Web Semântica para permitir que *Web Services* sejam descobertos baseado em funcionalidades específicas. Por outro lado, há pesquisas que tentam incluir no registro *UDDI* um mecanismo baseado em grafos, de modo que os

Web Services sejam agrupados, para facilitar por exemplo, a construção de um processo de negócio basedo em um fluxo de informações interligados [14]. Os autores afirmam que escolher um *Web Service* mais adequado a um processo de negócios, não é uma tarefa trivial, principalmente quando as questões de *QoS* são preponderantes. Otimizar a busca de informações através de algoritmos eficientes é também um assunto abordado em [8]. Apesar de todas os benefícios que o paradigma *SOA* apresenta atualmente, encontrar e validar conteúdos adequados (*Web Services*) ainda é um dos grandes desafios envolvidos na construção eficiente de um repositório de serviços [1].

4 Publicação e Recuperação de Informações de Web Services no Registro UDDI

A publicação e recuperação de informações de *Web Services* é um tópico bastante pesquisado na literatura. Em particular isto é importante pois a recuperação de informações pode melhorar ou piorar o desempenho associado à procura de um *Web Service* por parte de um determinado cliente. Frequentemente as informações sobre os *Web Services* são mapeadas de uma estrutura do registro *UDDI* para um banco de dados relacional (*MySQL*, *PostegreeSQL*, *Oracle*) e isso tem um custo em termos de desempenho. Se a procura por um *Web Service* não for otimizada, há um grande o risco de serem retornadas informações erradas e inconsistentes. Além disso, no contexto atual, os motores de processamento de mensagens *SOAP* não possuem um mecanismo de notificação automática quando o servidor *UDDI* e o provedor de *Web Services* estão configurados em localizações (máquinas) distintas. Sendo assim, toda a vez que um *Web Service* for publicado em um provedor, o operador do sistema precisa obter as informações do *Web Service* (*endpoint*, características) e enviá-las ao registro *UDDI*. Um outro problema está associado a inconsistência de informações no caso em que o *Web Service* não mais exista no provedor. Caso isso ocorra, as informações do *Web Service* presentes no registro *UDDI* também precisam ser excluídas, pois um cliente pode obtê-las de um *Web Service* que não mais existe no provedor. Na próxima seção serão mostrados detalhadamente como ocorre o processo manual de publicação e recuperação de informações de *Web Services* no registro *UDDI*. Posteriormete será apresentado um mecanismo que automatiza este processo e garante a consistência das informações armazenadas no registro *UDDI* toda a vez que é feito *deploy/undeploy* de um *Web Service* no provedor.

4.1 Notificação Manual

Deploy

Considerando o modo tradicional, o provedor de Web Services *recebia* as aplicações (*deployment*) de acordo com o esquema ilustrado na figura 2:

1. O responsável pela aplicação faz o *deploy* no diretório ***webapps/axis2/WEB-INF/services***.

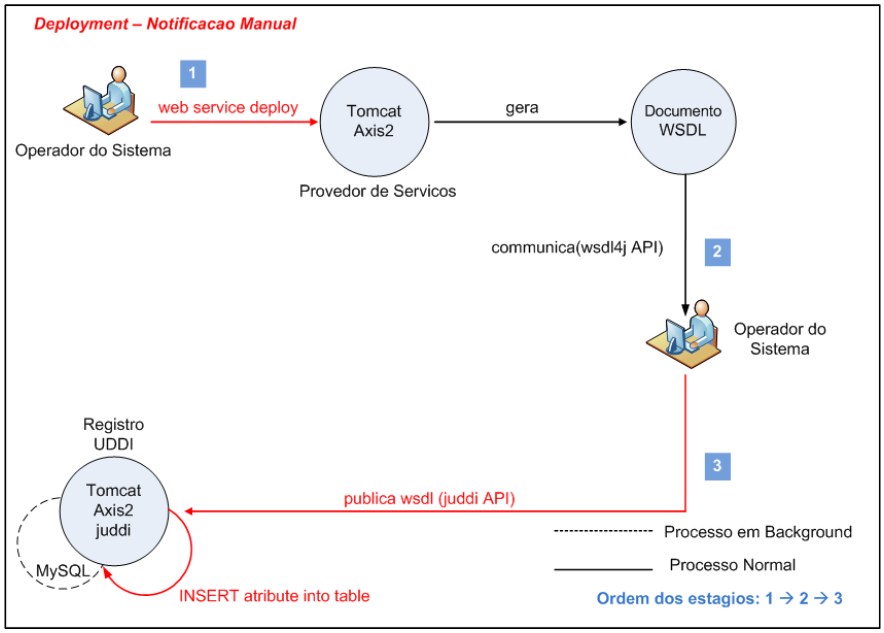

Figura 2. Notificação Manual - Deploy

2. É gerado o arquivo ***.aar*** referente à aplicação e posteriormete a *WSDL* associada a esta aplicação. Essa *WSDL* pode ser acessada segundo o domínio: ***http://<ip>:<porta>/axis2/services/<nome-do-webservices>***
3. Depois de realizado o *deploy* um agente humano se encarrega de publicar a *WSDL* do *Web Service* no registro *UDDI*, executando o código de publicação. Este código contém o endereço do documento *WSDL* de onde as informações devem ser obtidas. Como o registro *UDDI* pode ser um entidade independente do provedor de *Web Services*, começam a surgir alguns desafios que serão apresentados oportunamente.

O código de publicação é composta de quatro classes escritas em linguagem *Java*:

1. *Save*
2. *Business*
3. *ServiceInterface*
4. *ServiceImplementation*

A classe *Save* funciona como uma classe *main*, que instancia as demais e passa os parâmetros básicos para o processo de publicação, tais como o endereço do documento *WSDL*, o endereço do registro *UDDI*, nome do usuário e senha. A classe *Business* se encarrega de publicar as demais informações sobre o *Web*

Service. As duas classes restantes fazem o serviço mais importante, isto é, instanciam um objeto da biblioteca *WSDL4J* e através dele obtém as informações do documento *WSDL*, para assim publicar essas informações no registro *UDDI*.

Do modo descrito até aqui, a publicação ficaria dependente de um agente humano que deveria executar a classe *Save*, além de passar para esta o local onde buscar o documento *WSDL* para o *Web Service* que acabou de ser implantado no provedor. Caso isso não fosse realizado, o *Web Service* estaria disponível no provedor de *Web Services*, mas não seria publicado no registro *UDDI*. Essa limitação é um grave problema quando há a participação de um *broker* de *Web Services* entre o registro *UDDI* e o provedor de serviços. Se o *broker* precisar consultar um registro de *Web Services* para determinar a existência de um determinado *Web Service*, é preciso que este registro possua informações consistentes em relação à implementação e localização do *Web Service*. Caso contrário, o *broker* pode, em determinado instante, acessar informações inconsistentes sobre um *Web Service*, pois este pode não mais existir ou estar localizado em um *endpoint* diferente.

Undeploy

Os passos para o processo manual de exclusão de informações do *Web Service* são mostrados na figura 3:

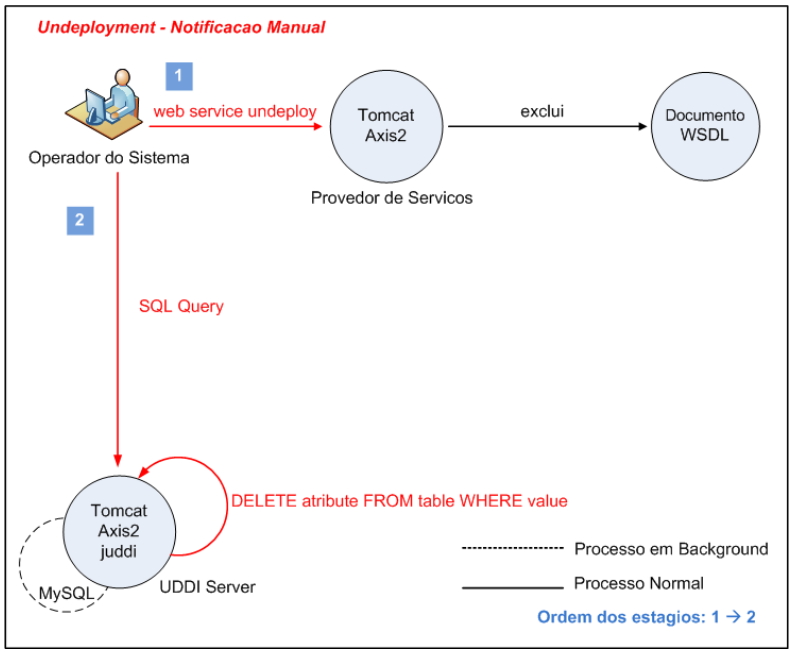

Figura 3. Notificação Manual - Undeploy

1. O operador do sistema faz o *undeploy* (retirada) do *Web Service* no provedor. Quanto o *undeploy* é realizado, a *WSDL* do *Web Service* associado é excluída automaticamente.
2. Após a exclusão da *WSDL* do provedor de *Web Services*, o operador do sistema precisa realizar uma busca no banco de dados do registro *UDDI* e excluir as informações associadas ao *Web Service* que foi excluído no provedor. Se esta operação não for efetuada, haverá inconsistência de informações entre o registro *UDDI* e o provedor de *Web Services*, pois este não mais possui o *Web Service*, enquanto o registro *UDDI* ainda mantém as informações daquele em sua base de dados. A literatura de *Web Services* discute esse problema e muitos autores argumentam que o uso do *UDDI* em larga escala esbarra no problema de insconsistência que está sendo tratado neste documento.

4.2 Notificação Automática

A solução apresentada neste documento permite que o processo de publicação de informações sobre um *Web Service* seja automático e consistente. Isto é obtido através de um *listener* que pode ser habilitado no motor de processamento de mensages *SOAP* denominado Apache Axis2. Este *listener* é uma interface *Java* que possui métodos associados a determinados eventos do Apache Axis2 e que podem ser implementados de acordo com necessidades específicas. Para a solução desenvolvida, foi implementada uma classe chamada de ***Observer*** (representando o *listener*) e nela foram implementados os eventos relativos ao ***deploy*** e ***undeploy*** de um *Web Service* qualquer em um provedor de *Web Services*. A idéia foi implementar essa interface de modo que o cliente do registro *UDDI* pudesse ser chamado no momento da publicação do *Web Service* e que este cliente utilizasse as informações geradas sobre o *Web Service* publicado no provedor para enviá-las ao registro *UDDI*.

Pré-Requisitos

Para que a solução de notificação (***publish/unpublish***) automática seja funcional, é necessário a utilização de algumas bibliotecas além das apresentadas pelo jUDDI. Também será preciso alterar algumas configurações padrão do motor de processamento Apache Axis2. Para que a engine Axis2 consiga instanciar as entidades do jUDDI (em tempo de execução) necessárias para a comunicação com o registro, todas as bibliotecas utilizadas (tanto as do jUDDI quanto as da ferramenta WSDL4J) devem ser armazenadas no no diretório ***/tomcat/lib*** e referenciadas no ***classpath*** do sistema operacional utilizado. No mesmo diretório também deve ser adicionado uma nova biblioteca denominada ***discovery-commons***, utilizada pela ferramenta WSDL4J.

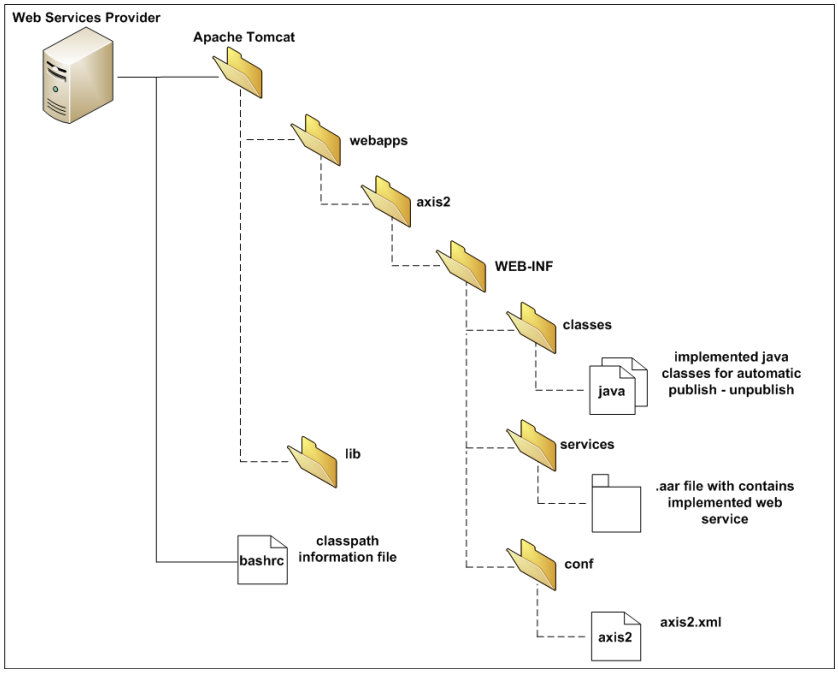

Figura 4. Estrutura de diretórios do provedor de Web Services

Para que o Axis2 dê suporte à interface (*listener* de eventos) é necessário alterar o arquivo de configuração ***axis2.xml***. Esta alteração consiste basicamente em referenciar a classe implementada como ***Listenner***. Foi necessário editar o arquivo ***axi2.xml***, armazenado no diretório ***apache-tomcat/webapps/axi2/WEB-INF/conf*** e entre os delimitadores da *tag* ***</listener>*** adicionar:

```
<listener class="Observer"/>
```

Por padrão o diretório é o ***apache-tomcat/webapps/axi2/WEB-INF/classes***, porém a classe não necessita encontrar-se neste diretório, deste que seu caminho seja corretamente referenciado. Nesta solução foi adotado o diretório padrão. É necessário implementar agora a interface do Axis2, criando a classe ***Observer.java*** e adicionar o seguinte método:

```
public void serviceUpdate(AxisEvent event, AxisService service)
{
   switch (event.getEventType())
   {
   // handle new service deployment
   case AxisEvent.SERVICE_DEPLOY :
        System.out.println("New service deployed. Service
```

```
                        name " + axisService.getName());
        break;

    // handle service removal
    case AxisEvent.SERVICE_REMOVE :
        System.out.println("Service removed from the system. Service
                        name " + axisService.getName());
        break;

    // handle service start
    case AxisEvent.SERVICE_START :
        System.out.println("Service Started. Service name " +
                        axisService.getName());
        break;

    // handle service stop
    case AxisEvent.SERVICE_STOP :
        System.out.println("Service Stopped. Service name " +
                        axisService.getName());
        break;
    }
}
```

O método `serviceUpdate` é responsável por capturar os eventos de *update* que ocorrem no *servlet*, informando qual evento ocorreu e o nome do serviço envolvido. O processo de publicação automático ocorre através de uma chamada externa para as classes que implementam a publicação no registro *UDDI* quando o evento ***SERVICE_DEPLOY*** ocorre. Estas classes foram alteradas para que tornassem o processo independente e modular.

Deploy

Para o mecanismo de notificação automática em relação ao *deploy* são apresentados os principais estágios como mostra a figura 5.

- Realizado o *deploy* da aplicação
- Provedor de serviços gera a *WSDL* a partir do *deploy* da aplicação
- *Thread* captura evento *"deploy"* e chama a ferramenta *WSDL4J*
- A ferramenta *WSDL4J* obtém a *WSDL* do serviço publicado e repassa para a *thread*
- *Thread* realiza chamada ao cliente do registro *UDDI* e passa como parâmentro a *WSDL*
- Com a *WSDL* do serviço, o cliente do registro *UDDI* faz uma requisição ao reigstro *UDDI* para publicar a *WSDL*
- Uma vez que a *WSDL* foi publicada no registro *UDDI*, uma mensagem de notificação é retornada ao cliente do registro

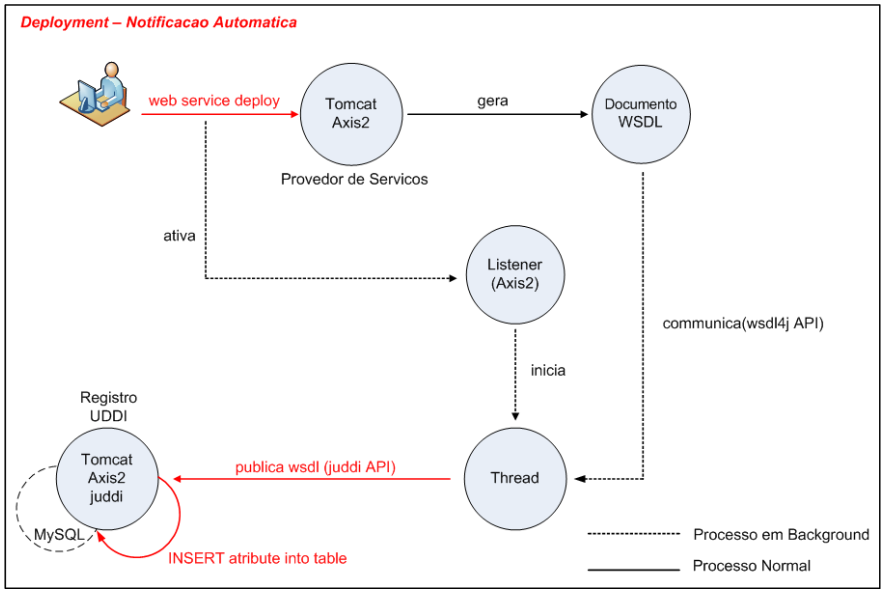

Figura 5. Notificação Automática - Deploy

Dentre os benefícios obtidos com a utilização do mecanismo de notificação (***publish***) automática, destaca-se:

- A classe ***Observer*** (*Listener*) passa a ser o ***main***, sendo executado de forma automática, diferentemente da classe ***Save*** anteriormente apresentada. Quando ocorre um evento relacionado ao *deploy* de um *Web Service*, a classe ***Observer*** chama a classe ***Save***, aproveitando assim a implementação anterior.
- A classe ***Save*** atua de maneira análoga ao método manual, chamando porém as classes ***Business***, ***ServiceImplementation*** e ***ServiceInterface***. A diferença é que as alterações foram realizadas nas duas últimas classes, de modo que estas se comportassem como *threads*. O uso de *threads* tem relação com a construção e acesso ao documento *WSDL* gerado, quando o *deploy* de um *Web Service* é realizado no provedor de *Web Services*. Antes, o acesso era realizado após o processo de *deploy* e o endereço do documento *WSDL* era informado. No mecanismo de notificação (***publish/unpublish***) automático, o acesso à *WSDL* ocorre de forma paralela ao processo de *deploy*, com o endereço da *WSDL* sendo montado aproveitando-se o nome do *Web Service* publicado no provedor.
- Para que não ocorra o risco de haver um acesso a este documento *WSDL* antes que ele tenha sido gerado, é utilizado uma variável ***mutex***. Ou seja, o acesso somente é liberado quando a *WSDL* é totalmente gerada no provedor. Este mecanismo garante portanto exclusão mútua e evita inconsistências associadas ao acesso do documento *WSDL*.

– O uso de *threads* torna também independente o processo de publicação de um *Web Service*, pois ainda podem ocorrer exceções em relação à comunicação com o registro *UDDI*.

Undeploy

O mecanismo de notificação em relação ao *undeploy* é apresentado na figura 6.

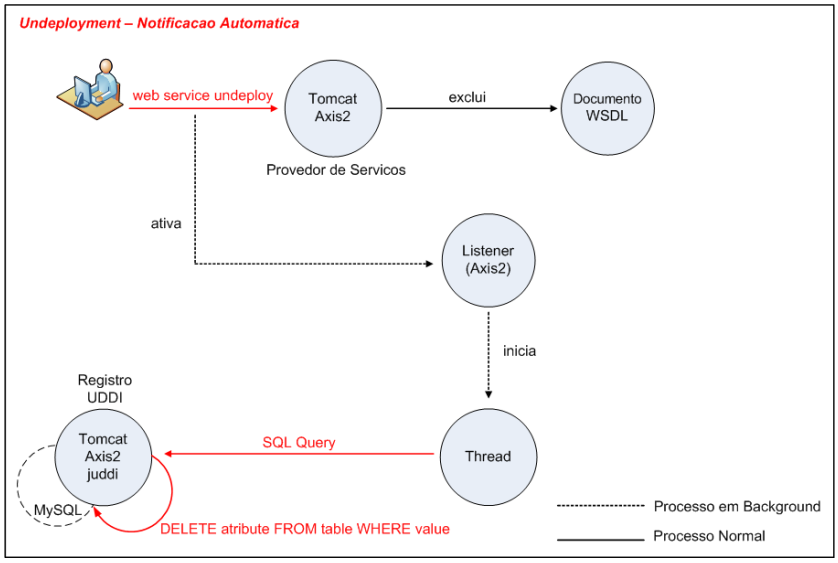

Figura 6. Notificação Automática - Undeploy

– Usuário ou sistema realiza o *undeploy* da aplicação
– Provedor de serviços captura a *WSDL* do serviço associado ao *undeploy*
– *Thread* captura evento *"undeploy"* e chama a ferramenta WSDL4J
– A ferramenta *WSDL4J* obtém a *WSDL* do serviço publicado e repassa para a *thread*
– Thread chama o cliente do registro *UDDI* e passa como parâmentro a *WSDL*
– Com a *WSDL* do serviço, o cliente do registro *UDDI* faz uma requisição ao registro para apagar a *WSDL* do serviço associado ao *undeploy*
– Uma vez excluída a *WSDL* no registro *UDDI*, uma mensagem de notificação é retornada ao cliente do *UDDI*

Os benefícios obtidos com a utilização do mecanismo de notificação (***unpublish***) automática são apresentados a seguir:

– Quando ocorre o evento *undeploy*, a classe ***Observer*** (*Listerner*) faz a chamada da classe *Unpublish* para que as informações sejam removidas do registro *jUDDI*. Atualmente, nenhum mecanismo de remoção é utilizado, o que torna a base de dados do registro *jUDDI* inconsistente, uma vez que a *API* do *jUDDI* não provê suporte à metodos de remoção, mas somente para consulta e escrita de informações. Com as informações obtidas sobre o *Web Service*, a classe *Unpublish* realiza uma consulta (*query*) na base de dados do registro *jUDDI* e faz a remoção das informações do *Web Service* que está sendo *"excluído"* do provedor de *Web Services*. Trata-se de uma classe de mais baixo nível, pois implementa funcionalidades que a *API* do *jUDDI* não suporta.

5 Conclusões

Este documento abordou um dos principais problemas da literatura de arquiteturas orientadas a serviços discutidos atualmente. A publicação e recuperação de informações de *Web Services* é um tópico bastante discutido atualmente, mas infelizmente nenhum mecanismo funcional tem sido apresentado para melhorar o processo de publicação e recuperação de informações e também garantir a consistência dessas informações em um regiistro *UDDI*. Diante da problemática aqui apresentada, foi formulada uma nova abordagem que garantisse a consistência dos dados armazenados no registro *UDDI*, juntamente com um novo mecanismo que capturasse as informações dos *Web Services* quando estes fossem implantados no provedor. A solução destacada neste documento ainda deve passar por um rigoroso teste de avaliação de desempenho, de modo a verificar a viabilidade de aplicá-la no desenvolvimento de arquiteturas orientadas a serviços que fazem uso de um repositório de informações de *Web Services*.

6 Trabalhos Futuros

A nova abordagem discutida neste documento ainda deve passar por refinamentos de modo que a solução seja melhorada. Para isso, alguns trabalhos futuros são considerados:

– Especificação dinâmica de parâmetros: O provedor informa em tempo de execução, seu nome de usuário, senha, endereço do registro *jUDDI Server*, etc. Atualmente isto é feito no código de forma estática.
– Consistência no *deploy/undeploy*: Não foi realizada qualquer verificação de informações que já tenham sido publicadas, uma vez que sempre é gerada uma nova chave primária a ser publicada no banco de dados do registro *jUDDI*. Um *rollback* de informações duplicadas, ou uma consulta prévia poderia resolver este problema antes que a informação referente ao *Web Service* fosse salva.
– Determinação do tempo de espera das *threads*: Este tempo está definido como dez segundos. O ideal seria a *thread* ser lançada assim que o documento *WSDL* fosse totalmente gerado.

– Garantir que caso ocorra uma exceção no processo de publicação, este seja retomado para que as informações sejam de fato publicadas.
– Determinar quais informações são realmente necessárias de serem publicadas, mantendo assim uma coerência com o que está sendo pesquisado posteriormente. A idéia é não publicar informações irrelevantes.
– Proposta de inclusão de métodos na *API* do *jUDDI* que dê suporte à remoção dos dados da base de dados, de modo que este processo seja de alto nível, assim como o processo de inclusão e consulta.
– Criação de uma biblioteca que dê suporte à notificação *publish/unpublish* automático.
– Criação de um interface web para o cliente interagir com o *broker/uddi*.

Referências

1. C. Atkinson, P. Bostan, O. Hummel, and D. Stoll. A practical approach to web service discovery and retrieval. In *ICWS*, 2007.
2. S. Banerjee, S. Basu, S. Garg, S. Garg, S.-J. Lee, P. Mullan, and P. Sharma. Scalable grid service discovery based on uddi. In *MGC '05: Proceedings of the 3rd international workshop on Middleware for grid computing*, pages 1–6, New York, NY, USA, 2005. ACM.
3. M. B. Blake, A. L. Sliva, M. zur Muehlen, and J. V. Nickerson. Binding now or binding later: The performance of uddi registries. In *HICSS '07: Proceedings of the 40th Annual Hawaii International Conference on System Sciences*, page 171c, Washington, DC, USA, 2007. IEEE Computer Society.
4. C. A. Brooks. An introduction to web services, 2002. Available at: `http://www.cs.usask.ca/~cab938/`. Last access: 28/01/2009.
5. A. Erradi and P. Maheshwari. A broker-based approach for improving web services reliability. In *Proceedings of IEEE International Conference on Web Services (ICWS-05)*. IEEE CS Press, 2005.
6. P. Farkas and H. Charaf. Web services planning concepts. *Journal of WSCG*, 11(1), February 2003.
7. C. Ferris and J. Farrell. What are Web services? *Communications of the ACM*, 46(6):31–31, June 2003.
8. J. Hicks, M. Govindaraju, and W. Meng. Search algorithms for discovery of web services. In *Internation Conference on Web Services – ICWS*, pages 1172–1173, 2007.
9. T. Kawamura, T. Hasegawa, A. Ohsuga, M. Paolucci, and K. Sycara. Web services lookup: A matchmaker experiment. *IT Professional*, 7(2):36–41, 2005.
10. H. Kreger. Fulfilling the web services promise. *Commun. ACM*, 46(6):29–ff, 2003.
11. A. Kumar, A. El-Geniedy, and S. Agarwal. A generalized framework for providing qos based registry in service oriented architecture. In *SCC '05: Proceedings of the 2005 IEEE International Conference on Services Computing*, pages 295–306, Washington, DC, USA, 2005. IEEE Computer Society.
12. N. Leavitt. Are web services finally ready to deliver? *Computer*, 37(11):14–18, 2004.
13. J. Liu, N. Gu, Y. Zong, Z. Ding, and Q. Zhang. Service registration and discovery in a domain-oriented uddi registry. In *CIT '05: Proceedings of the The Fifth International Conference on Computer and Information Technology*, pages 276–283, Washington, DC, USA, 2005. IEEE Computer Society.

14. J. Liu, J. Liu, and L. Chao. Design and implementation of an extended uddi registration center for web service graph. *Web Services, 2007. ICWS 2007. IEEE International Conference on*, pages 1174–1175, July 2007.
15. M. Ouzzani and A. Bouguettaya. Efficient access to web services. *IEEE Internet Computing*, 8(2):34–44, 2004.
16. M. P. Papazoglou. Service-oriented computing: Concepts, characteristics and directions. In *Proceedings of the Fourth International Conference on Web Information Systems Engineering (WISE-03)*. IEEE CS Press, 2003.
17. M. P. Papazoglou and D. Georgakopoulos. Service-oriented computing. *Communications of the ACM*, 46(10), October 2003.
18. R. Peterkin, J. Solomon, and D. Ionescu. Role based access control for uddi inquiries. In *COMSWARE*, 2007.
19. T. Pilioura, G.-D. Kapos, and A. Tsalgatidou. Pyramid-s: A scalable infrastructure for semantic web service publication and discovery. In *RIDE '04: Proceedings of the 14th International Workshop on Research Issues on Data Engineering: Web Services for E-Commerce and E-Government Applications (RIDE'04)*, pages 15–22, Washington, DC, USA, 2004. IEEE Computer Society.
20. G. Saez, A. L. Sliva, and M. B. Blake. Web services-based data management: Evaluating the performance of uddi registries. In *ICWS '04: Proceedings of the IEEE International Conference on Web Services*, page 830, Washington, DC, USA, 2004. IEEE Computer Society.
21. A. ShaikhAli, O. F. Rana, R. Al-Ali, and D. W. Walker. Uddie: An extended registry for web services. In *SAINT-W '03: Proceedings of the 2003 Symposium on Applications and the Internet Workshops (SAINT'03 Workshops)*, page 85, Washington, DC, USA, 2003. IEEE Computer Society.
22. R. Siblini and N. Mansour. Testing web services. In *AICCSA '05: Proceedings of the ACS/IEEE 2005 International Conference on Computer Systems and Applications*, pages 135–vii, Washington, DC, USA, 2005. IEEE Computer Society.
23. C. Sun, Y. Lin, and B. Kemme. Comparison of uddi registry replication strategies. In *ICWS '04: Proceedings of the IEEE International Conference on Web Services*, page 218, Washington, DC, USA, 2004. IEEE Computer Society.
24. J. P. Thomas, M. Thomas, and G. Ghinea. Modeling of web services flow. In *IEEE International Conference on E-Commerce, Newport Beach, California, USA, June 24 - 27, 2003*, pages 391–398, 2003.
25. J. Yu and G. Zhou. Dynamic web service invocation based on uddi. In *CEC-EAST '04: Proceedings of the E-Commerce Technology for Dynamic E-Business, IEEE International Conference*, pages 154–157, Washington, DC, USA, 2004. IEEE Computer Society.