Implementação de um Repositório de Versões de Serviços Web usando OrientDB

Lucas J. K. Alves, Karin Becker

Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Porto Alegre, Brasil

lucaskalves@gmail.com.br, karin.becker@inf.ufrgs.br

Resumo. *Versionamento é uma técnica muito usada para gerenciar mudanças em* web services, *visando minimizar o impacto em aplicações cliente. Este artigo apresenta uma implementação eficiente de um repositório de versões de serviços usando OrientDB, um sistema de gestão de banco de dados orientado a grafos.*

1. Introdução

Arquiteturas orientadas a serviço e *web services* tornaram-se padrão para o desenvolvimento de aplicações de baixo acoplamento. Uma descrição usando uma linguagem padronizada (*e.g.* Web Service Definition Language (WSDL)), fornece às aplicações cliente os aspectos externamente relevantes de um serviço. Para evitar impactar clientes devido a mudanças na interface de um serviço, muitas vezes o provedor cria versões deste, possibilitando que o cliente utilize uma versão anterior até que possa ajustar-se às mudanças. O cliente deve identificar as porções do serviço modificadas e se elas lhe causam impacto negativo (*i.e.* incompatíveis). Esta tarefa é difícil, principalmente em caso de descrições extensas e modificações frequentes (*e.g.* serviço Trading[1] do eBay, com 270.000 linhas e modificado a cada duas semanas).

Yamashita *et al.* (2012)(a) propuseram um modelo de versionamento de serviços mais granular que facilita a gestão de versões, junto com um algoritmo de versionamento que converte uma descrição WSDL em versões neste modelo. Assim, é possível versionar apenas as partes da descrição que foram alteradas, e relacioná-las com versões pré-existentes das porções inalteradas do serviço. O presente artigo apresenta uma implementação eficiente de um repositório de versões para este modelo, utilizando um sistema de gestão de banco de dados (SGBD) orientado a grafos.

No restante deste artigo, a Seção 2 apresenta o modelo de versionamento e a sua implementação original. A Seção 3 descreve a implementação baseada em grafos e a Seção 4 analisa seu desempenho. A Seção 5 apresenta conclusões e direções futuras.

2. Modelo de Versões Orientado a *Features* e Implementação Original

Para um maior controle sobre as partes específicas da interface de um serviço que são alteradas, Yamashita *et al.* (2012)(a) propuseram um modelo de versionamento orientado a características (*features)*, descrito no diagrama da Figura 1. Uma *feature* relaciona-se a um trecho textual de uma descrição WSDL, correspondendo a um aspecto de um serviço, operação ou tipo de dado. *Features* são versionadas, e uma interface de

[1] http://developer.ebay.com/DevZone/XML/docs/Reference/eBay/

serviço é então descrita por uma coleção de versões interconectadas de *features*, formando um grafo.

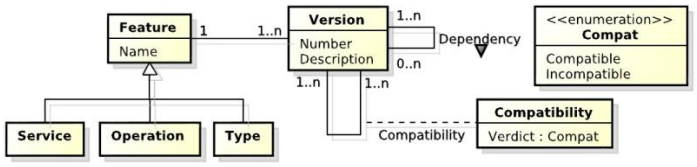

Figura 1. Modelo de Versões Orientado a *Features*

Um *framework* de evolução de serviços (Figura 2) recebe uma nova descrição WSDL, converte esta descrição textual no modelo de versões orientada a *features*, detectando as partes alteradas em relação a versões pré-existentes. Assim, cria novas versões quando detecta mudanças, ou estabelece relações de dependência com versões existentes de *features*. O resultado é armazenado em um repositório de versões.

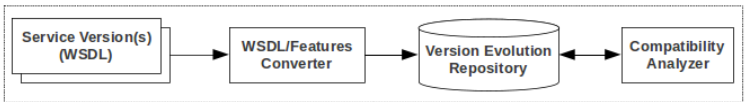

Figura 2. *Framework* de Evolução de Serviços

O processo de criação de versões está representado na Figura 3. Primeiramente, converte-se cada elemento da definição WSDL em uma *feature*. Depois, cria-se um grafo de dependências, onde os vértices representam *features* e as arestas, a relação de dependência entre elas. Uma dependência existe se a definição de uma *feature* é baseada em outra (*e.g.*uma operação que utiliza como parâmetro um tipo descrito no mesmo WSDL). Após, compara-se cada *feature* desse grafo com as *features* equivalentes presentes no repositório de versões, versionando a *feature* se uma mudança é detectada. O caminhamento no grafo é *bottom up*, isto é, comparação começa pelas *features* sem dependentes. As mudanças podem ocorrer devido a alterações na descrição da *feature* ou devido a alterações em alguma dependência dela (propagação). Sempre que uma *feature* não é modificada, uma versão equivalente dela existente no repositório é reaproveitada, evitando redundâncias.

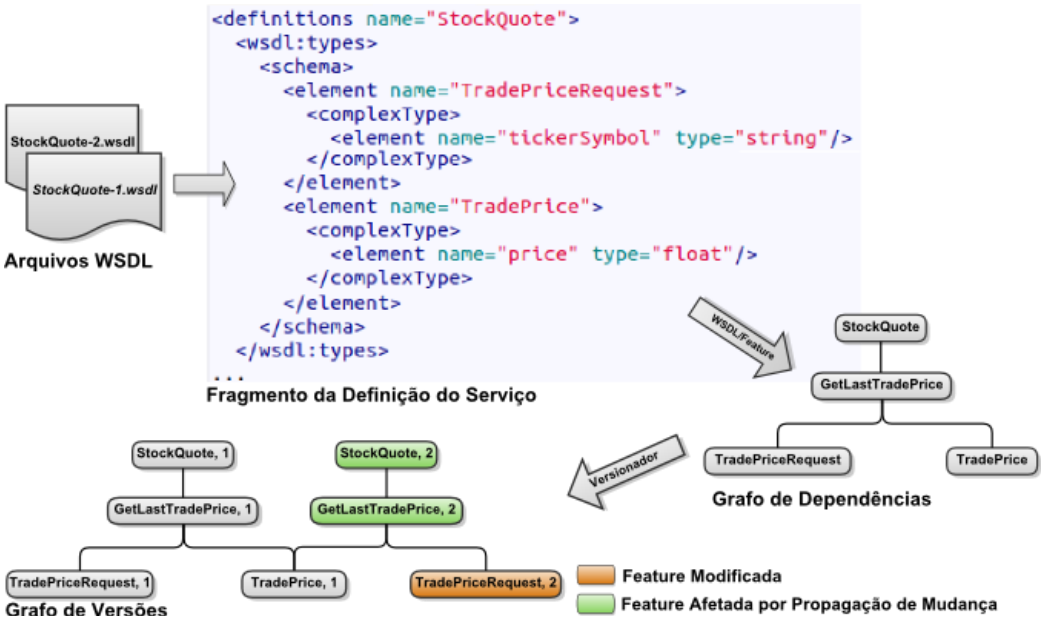

Figura 3. Diagrama de Versionamento

A implementação original do sistema de versionamento não apresentava bom desempenho devido à forma como as informações eram persistidas, bem como ao processo de comparação de versões. Os dados eram persistidos em arquivos XML puros (sem nenhum SGBD), o que dificultava a centralização e manipulação do repositório de versões. Além disso, considerando as bibliotecas de manipulação de arquivos XML utilizadas (SAX e JDOM), o processo de comparação de versões exigia o caminhamento entre os subgrafos de cada *feature* do repositório para detectar mudanças nas suas dependências. Isso impactava significativamente seu desempenho.

3. Implementação Baseada em Grafos

Para resolver esses problemas, foram tomadas duas ações: a) utilizar SGBD orientado a grafos para persistir os dados das versões; b) melhorar o processo de comparação de *features* com versões do repositório para aumentar seu desempenho. Foi escolhido utilizar um SGBD orientado a grafos em vez de, por exemplo, um SGBD XML, pois o tipo de estrutura grafo representa bem o modelo de versionamento utilizado. Utilizou-se o OrientDB[2], o qual é *Open-Source*, permite diversos níveis de controle de integridade (*ACID*, *Multiversion Concurrency Control* e *Basically Available, Soft State, Eventual Consistency*), tem licença permissiva e disponibiliza diversas maneiras de acesso aos dados (SQL, API Nativa e Gremlin[3]). OrientDB permite representar dados como vértices e arestas, cujos atributos são descritos através de classes associadas a estes.

Considerando o modelo da Figura 1, a classe *Version* foi mapeada para o conceito de vértice do OrientDB, e a associação *dependency* entre versões, no conceito de aresta. Como existem versões de serviços, operações e tipos, foram definidas as classes *ServiceVersion*, *TypeVersion* e *OperationVersion,* e associadas aos vértices. Definiu-se uma superclasse *FeatureVersion* que define os atributos comuns *name*, *description*, *number* e *signature*, este último, detalhado abaixo. A relação *dependency* pode ser consultada utilizando as maneiras de acesso disponíveis pelo OrientDB.

Na implementação original, o processo de comparação envolvia o caminhamento pela nova definição WSDL e compará-la com o repositório de versões XML utilizando as API citadas. Esse processo de comparação de versões foi alterado para evitar a necessidade de comparação recursiva de subgrafos no repositório, visando detectar mudanças nas dependências. O atributo *signature* (assinatura) de uma *feature* é um código *hash* criado a partir da descrição textual da *feature*, combinado com as assinaturas de suas dependências, recursivamente. A partir da definição WSDL, é possível recriar a assinatura e compará-la com as assinaturas das versões presentes no repositório. Assim, é possível detectar mudanças nas *features* apenas comparando as suas assinaturas diretamente. Primeiramente, procuramos por *features* no repositório com o mesmo nome da *feature* sendo analisada. Se existir e tiver uma versão com a mesma assinatura, essa versão é reaproveitada. Se existir, mas nenhuma tiver a mesma assinatura, trata-se de uma nova versão. Se não existir *feature* com o mesmo nome, é uma nova *feature*.

---

[2] http://www.orientdb.org/

[3] http://gremlin.tinkerpop.com/

4. Experimento

Para comparar a eficiência da implementação original com a atual, foram versionadas vinte descrições do serviço Trading do provedor eBay. Esse serviço é bastante complexo, com muitas operações e tipos, logo, o modelo de versionamento é testado em situações compatíveis com aplicações reais. Foram comparados os tempos de criação de uma nova versão utilizando a implementação original XML e a implementação OrientDB, sempre no mesmo ambiente computacional. A melhora de desempenho obtida através da nova implementação pode ser vista no gráfico da Figura 4.

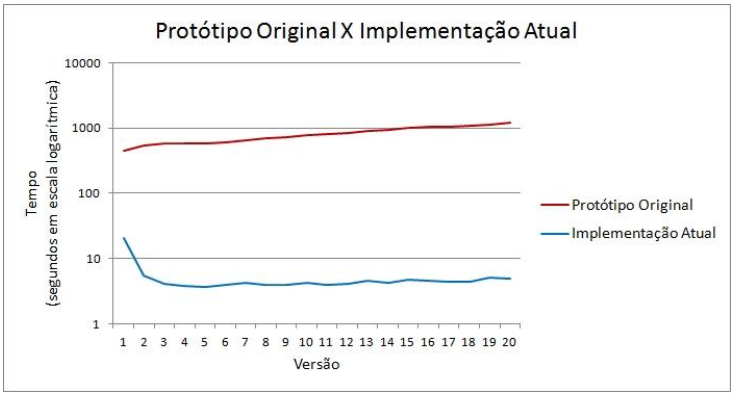

Figura 4. Comparação de Desempenho entre as Implementações

5. Considerações Finais e Trabalhos Futuros

As mudanças propostas na implementação do sistema de versionamento aumentaram significativamente seu desempenho. O uso de um SGBD, comparado com o repositório em arquivo, facilitou a centralização das informações. O uso de um SGBD orientado a grafos contribuiu para a melhoria do desempenho, por alinhar-se com as relações de dependência entre versões. A mudança no processo de comparação diminuiu a quantidade de acessos ao repositório. O OrientDB facilitou também as consultas na estrutura de grafo, muito importante para outras aplicações do *framework*, tais como detecção de compatibilidade (Yamashita *et al.* 2012 (b)).

Entre as vantagens percebidas na utilização do OrientDB estão seu bom desempenho, sua facilidade de integração com aplicações Java, interface gráfica para manuseio de seus dados e existência de uma ótima comunidade de desenvolvimento. Como principal dificuldade cita-se a escassez de guias e livros a seu respeito.

Futuramente, integraremos o sistema de gestão de versões com os outros módulos do *framework*, tais como o módulo de gerencia de perfis de uso e de análise de uso (Yamashita *et al.* 2012 (b)).

Referências

Yamashita, M., Becker, K. e Galante, R. (2012). "A *Feature*-based Versioning Approach for Assessing Service Compatibility". *JIDM* 3(2): 120-131.

Yamashita M., Vollino B., Becker K. e Galante R. (2012). "Measuring Change Impact based on Usage Profiles". Em *Proceedings of the ICWS*, 2012. p. 226-233.

W3C. "WSDL – Web Service Description Language". Capturado em http://www.w3c.org/TR/wsdl (Agosto 2012).