Desenvolvimento de web services interoperáveis utilizando abordagem contract-first

Marcos Mele[1,2], Sandro Lopes[1], Leonardo Guerreiro Azevedo[1,2,3]

[1]Departamento de Informática Aplicada
Universidade Federal do Estado do Rio de Janeiro (UNIRIO)

[2]Programa de Pós-Graduação em Informática (PPGI)
Universidade Federal do Estado do Rio de Janeiro (UNIRIO)

[3]IBM Research

{marcos.mele, sandro.lopes, azevedo}@uniriotec.br, LGA@br.ibm.com

Abstract. *Service-Oriented Architecture (SOA) supports interoperability among heterogeneous systems. However, even using SOA, integration between systems is a challenge. In practice, usually the service contract is generated automatically from service code (code-first approach), bringing service maintenance and consumption issues in the long term. A better approach is to build the service contract first and implement the code adhering to this contract (contract-first approach). This work explores contract-first development through an example of use in order to demonstrate this approach brings a high level of interoperability, makes governance ease and reduces coupling with a particular technology.*

Resumo. *A Arquitetura Orientada a Serviços (SOA) apoia a interoperabilidade entre sistemas heterogêneos. Entretanto, mesmo usando SOA, a integração entre sistemas é um desafio. Na prática, o contrato do serviço geralmente é gerado automaticamente a partir do código (abordagem code-first), o que gera dificuldades de manutenção e consumo do serviço em longo prazo. Uma melhor abordagem é construir o contrato do serviço primeiro e, então, implementar o código aderente a este contrato. Este trabalho faz um estudo exploratório do desenvolvimento contract-first através de um exemplo de uso, a fim de demonstrar que esta abordagem traz alto nível de interoperabilidade, facilita governança e reduz acoplamento de tecnologia.*

1. Introdução

Segundo Josuttis [2007], SOA (Service-Oriented Architecture) possibilita a comunicação entre sistemas desenvolvidos em linguagens de programação distintas e sendo executados em sistemas operacionais diferentes. Esse cenário é muito comum em empresas de médio e grande porte que possuem uma infraestrutura de TI complexa.

SOA é um paradigma arquitetural que disponibiliza funcionalidades de aplicações de maneira autocontida como serviços [Erl, 2005]. Serviços são componentes de software que representam uma atividade do negócio [Hewitt 2009]. Em geral, serviços são desenvolvidos utilizando a tecnologia de *web services* e são responsáveis

pela comunicação e interoperabilidade entre sistemas. Usualmente, serviços são descritos utilizando a linguagem WSDL[1] (Web Services Description Language) e a comunicação é feita de forma padronizada utilizando troca de mensagens escritas segundo o protocolo SOAP[2] (Simple Object Access Protocol).

Como em SOA a comunicação entre provedor e consumidor é estabelecida através do contrato do serviço [Josuttis, 2007], cada serviço desenvolvido terá primeiro seu contrato estabelecido, evitando que mudanças e tendências durante o desenvolvimento afetem seus consumidores. Alguns padrões e convenções serão abordados neste trabalho, a fim de garantir interoperabilidade, redução de acoplamento, aumento de reusabilidade e melhoria em governança dos serviços.

O objetivo deste trabalho é apresentar abordagem de como se desenvolver serviços através do uso de *contract-first*. Esta abordagem propõe desenvolver os artefatos relativos ao contrato de serviço antes de sua codificação [Erl, 2008]. A utilização desse modo de desenvolvimento auxilia na criação de serviços com maior nível de independência, eficiência e baixo acoplamento à tecnologia. A proposta deste trabalho para utilização de *contract-first* é baseada no uso de padrões e boas práticas de desenvolvimento de serviços presentes na especificação do Basic Profile (BP)[3] que visam garantir interoperabilidade, redução de impacto de mudanças e facilidade de manutenção de serviços.

O foco deste trabalho está no desenvolvimento do serviço em si. Logo, foram escolhidos os resultados apresentados por Azevedo *et al.* [2013] para as fases anteriores do processo de desenvolvimento (isto é, identificação e análise de serviços).

A abordagem escolhida para implementação de serviços foi web services e a implementação da composição de serviços utilizando injeção direta no serviço composto. Os contratos dos serviços foram desenvolvidos utilizando WSDL, que é linguagem padrão de definição de web services; enquanto a API JAX-WS foi escolhida para desenvolvimento de web services devido a sua robustez e praticidade por permitir uso de *annotations* (padrão de mapeamento por anotação de classes Java).

Este trabalho está dividido da seguinte forma. A Seção 1 é a presente introdução. A Seção 2 apresenta o conceito de contrato de serviço e nossa proposta de projeto de serviços interoperáveis, apresentando o Basic Profile (BP), que corresponde a padrões de desenvolvimento de serviços com maior interoperabilidade, e traz um comparativo das abordagens *contract-first* e *code-first*. A Seção 3 trata do desenvolvimento dos serviços utilizando o framework JAX-WS 2.2 e outras especificações J2EE. Por fim, a seção 4 apresenta análises dos resultados alcançados e a conclusão do trabalho.

2. Contrato de Serviços: projetando web services interoperáveis

Um contrato de serviço determina as operações, o formato dos dados, e os detalhes técnicos de um serviço definindo o acordo entre um provedor específico para um

[1] http://www.w3.org/TR/wsdl

[2] http://www.w3.org/TR/soap/

[3] http://ws-i.org/Profiles/BasicProfile-1.2-2010-11-09.html

consumidor específico [Josuttis, 2007]. Desta forma, um contrato deve conter informações sobre os requisitos do serviço, incluindo aspectos funcionais (operações e dados de entrada e saída) e não-funcionais (como qualidade e segurança). Para serviços baseados em SOAP, o WSDL representa a parte técnica do contrato [Hewitt, 2009]. O método que constrói o WSDL primeiro e posteriormente codifica os requisitos em uma linguagem de programação (como, por exemplo, Java) a partir das definições presentes no WSDL é chamado *contract-first*. Em contrapartida, o método onde primeiro é feita a codificação do serviço e a partir da codificação o WSDL é gerado é chamado *code-first* ou *contract-last*.

2.1. Conformidade com o WS-I Basic Profile

SOA é um paradigma para a realização e manutenção de processos de negócio em um grande ambiente de sistemas distribuídos que são controlados por diferentes proprietários [Josuttis, 2007]. A integração entre sistemas que utilizam distintas tecnologias de desenvolvimento e plataformas é viável através da utilização de serviços independentes de plataforma e protocolo. O padrão de fato para desenvolvimento de serviços é web services [Erl, 2008]. Contudo, para garantir sua utilização pelo maior número de consumidores possíveis, é importante seguir boas práticas de desenvolvimento. Este trabalho considera o padrão web services utilizando o protocolo SOAP cujos principais padrões empregados são WSDL, XML, SOAP etc.

O desenvolvimento de web services requer utilização de uma série de especificações, como, por exemplo, XSD, WSDL, SOAP. O desenvolvedor possui uma variedade de possibilidades para escolhas, tais como, camadas de transporte, políticas de segurança, mecanismos de codificação de mensagens, versão de XML utilizado, entre outros. Para minimizar essas dificuldades e prover o maior nível de interoperabilidade possível, a WS-I[4] – um consórcio composto de um grande número de fornecedores com objetivo de estabelecer as melhores práticas para interoperabilidade entre web services – definiu padrões para implementação de serviços. Seu produto principal é o Basic Profile (BP), que consiste em um conjunto de padrões e boas práticas. O BP auxilia os desenvolvedores a escolherem as configurações de seus web services de forma a garantir interoperabilidade com o maior número de plataformas tecnológicas. Hewitt [2009] sumarizou os padrões e boas práticas menos óbvios presentes no BP. Boa parte da lista indicada por Hewitt foi utilizada no presente trabalho. A seguir, são apresentadas descrições do que foi empregado:

- O elemento <body> de um <SOAP envelope> deve conter exatamente zero ou um elemento filho e deve ser namespace qualified, ou seja, os elementos e/ou tipos utilizados no esquema devem estar de acordo com o namespace especificado como targetNamespace. Não deve conter instruções de processamento ou utilizar DTD (Data Type Definitions);
- Utilize literal message encoding em detrimento de SOAP encoding. Utilizando literal, os elementos e/ou tipos complexos existentes na mensagem são legíveis (literais) tanto para o consumidor quanto para o provedor, estando desacoplada

[4] http://ws-i.org/Profiles/BasicProfile-2.0-2010-11-09.html

de linguagem e codificação. Utilizando encoded, as mensagens são codificadas utilizando SOAP-Encoding, gerando um acoplamento com esta tecnologia;

- Apesar de ser permitido pela especificação SOAP 1.1, não se deve utilizar notação de ponto para redefinir o significado de um elemento <faultcode>. Essa abordagem simplifica o significado do erro. O BP indica que o detalhamento da informação contida no elemento <faultstring> deve ser mantido;
- Apesar de SOAP Action ter a pretensão de auxiliar na indicação da rota de mensagem para uma determinada operação, na prática, os serviços não permitem a utilização de SOAP Action, de forma que toda a informação pertinente é realizada no envelope SOAP e não em cabeçalhos HTTP. Assim, SOAP Action deve ser utilizado somente com uma dica;
- Todas as SOAP Actions devem ser especificadas no WSDL utilizando uma string entre aspas. Se as SOAP Actions não forem especificadas, deve-se utilizar uma string vazia entre aspas;
- Os cookies podem ser utilizados, mas sua utilização não é incentivada porque consumidores que não suportam cookies não estão aptos a consumirem o serviço. Desta forma, apesar do BP permitir sua utilização, essa tecnologia não é utilizada neste trabalho;
- Utilize XML 1.0 porque esta é a única versão que o BP oferece suporte para XML Schema e WSDL;
- Utilize codificação UTF-8 ou UTF-16;
- Tanto a especificação do WSDL quanto o BP recomendam a separação de arquivos WSDL em componentes modulares e criação de arquivos que separam a definição abstrata da definição concreta do WSDL;
- Ao importar um WSDL, deve haver coerência entre os namespaces de ambos os arquivos. Isso significa que o targetNamespace indicado no elemento <definitions> do WSDL importado deve ser o mesmo namespace utilizado no WSDL que o está importando.
- Utilize somente os recursos proporcionados pela especificação do WSDL. A utilização de extensões de WSDL pode implicar na inviabilidade de consumo de serviços por parte de alguns consumidores que não suportam tais extensões;
- O BP não permite o uso de tipos de arrays no WSDL 1.1 devido à diversidade de interpretações de como estes devem ser utilizados. Essa diversidade implica em uma série de problemas de interoperabilidade. Uma solução alternativa simples ao uso de arrays é definir o atributo maxOccurs de um tipo complexo com um valor maior que 0. É possível determinar um número indeterminado utilizando o valor unbounded para maxOccurs;
- Os estilos existentes na construção de um WSDL são Document e RPC. Quando o modelo selecionado é Document, as mensagens de entrada e saída representam documentos, ou seja, referem-se a um parts cujo tipo é element na seção body do WSDL. Quando o modelo selecionado é RPC, as mensagens correspondem aos

parâmetros de entrada e retorno, e deve-se utilizar obrigatoriamente tipos (simples ou complexos). O BP permite a utilização de ambos os estilos;

- Web services são capazes de abstrair o conteúdo de mensagens e operações fora da camada de transporte. Entretanto, apenas SOAP é suportado pelo BP em elementos <binding>.

2.2. Centralização de Esquema

Erl [2008] apresenta o padrão centralização de esquema e defende a definição de um esquema "oficial" para cada conjunto de informações. Contratos de serviços podem compartilhar esses esquemas centralizados. O objetivo é padronizar os tipos de dados em um determinado domínio, seja um processo, uma indústria (financeira, farmacêutica, petrolífera etc.) ou a organização inteira. A centralização do esquema de dados proporciona a eliminação de tipos redundantes. Entretanto, aumenta o acoplamento dos serviços com os tipos de dados comuns centralizados no esquema, o que pode significar um desafio à gestão dos tipos fundamentais.

Esta abordagem garante a interoperabilidade e eficiência dos serviços que participam do processo e diminui a possibilidade de necessidade de redefinição de uma determinada entidade apenas para um ou outro grupo de consumidores. Apesar de não eliminar a necessidade de transformação de dados, essa abordagem a minimiza e possibilita uma governança mais eficiente sobre os dados. Seguindo as melhores práticas de padrão de XML Schema, Hewitt [2009] propõe a utilização de alguns padrões de projeto para esquemas, buscando obter uma mescla de coesão, acoplando, simplicidade e exposição de tipos. São eles: Boneca Russa, Fatia de Salame, Veneziana e Jardim do Éden.

A fim de aproveitar vantagens cruciais da proposta, como uso de múltiplos arquivos, facilidade de reuso entre vários serviços e o baixo acoplamento dos tipos de dados, neste trabalho, foi utilizado o padrão Jardim do Éden a fim de aproveitar o uso de múltiplos arquivos, facilidade de reuso entre vários serviços e o baixo acoplamento dos tipos de dados, porém com adaptações para que possam ser minimizadas as desvantagens enumeradas por Hewitt [2009].

2.3. Contract-first versus code-first

Na abordagem *code-first*, primeiramente é desenvolvida a codificação e, em seguida, gera-se o contrato do serviço com auxílio de um framework. Essa abordagem é mais utilizada devido a sua praticidade [Kalin, 2009]. O desenvolvedor não precisa conhecer e desenvolver a documentação técnica do contrato de serviços (WSDL, XSD e WS-Policy) e, por isso, o desenvolvimento é mais rápido. Contudo, em projetos de médio a grande porte, onde se planeja utilizar SOA ou a mesma já está em execução, essa abordagem pode interferir em princípios básicos da arquitetura como alinhamento com o negócio, interoperabilidade, acoplamento, reuso e durabilidade, como apresentado a seguir. Em *contract-first*, primeiramente elabora-se o contrato do serviço e, em seguida, utiliza-se desse contrato para gerar a codificação correspondente. Como a codificação depende do contrato, observa-se um acoplamento positivo da lógica (codificação) para o contrato (WSDL, XSD e WS-Policy) [ERL, 2009]. A abordagem *contract-first* é menos difundida. Possíveis explicações para esse fato são: maior complexidade de

desenvolvimento devido à necessidade de domínio sobre elementos diferentes da linguagem de programação usualmente utilizada e pouca difusão das técnicas e benefícios da abordagem.

Quando se codifica primeiro, não é possível determinar completamente como o contrato será gerado. Para serviços já disponibilizados, essa abordagem traz uma dependência significativa à tecnologia utilizada para manutenção do serviço. Na prática, significa que se for identificada a necessidade de trocar a tecnologia de desenvolvimento de serviços, será necessária também a alteração do contrato, o que afeta diretamente todos os consumidores destes serviços.

Outra questão é sobre restrições de tipos de dados que podem ser expressas nas estruturas de dados que representam as entidades e algumas dessas restrições exigem desenvolvimento para serem verificadas, ou seja, parte da validação pode ocorrer no XSD e outra parte em uma instância do serviço. O desempenho de SOA torna-se crítico normalmente por causa do tempo de execução dos serviços [Josuttis, 2007]. Assim, qualquer processamento que possa ser evitado deve ser considerado.

3. Provimento de serviços interoperáveis

Para validar os conceitos abordados, foram utilizados os serviços identificados por Azevedo *et al.* [2013]. Em termos de desenvolvimento, cada agrupamento representa um projeto para implementação de um web service, enquanto que cada serviço foi implementado como uma operação do web service.

Os esquemas foram desenvolvidos utilizando o padrão XML Schema – aprovado como especificação pela W3C e incorporado pelo WS-I BP – em sua versão 1.0, utilizando a plataforma Eclipse WTP. O modelo canônico foi empregado neste trabalho considerando um esquema centralizado contendo suas entidades básicas. Além deste esquema central, deve existir um esquema distinto para cada entidade, localizado em um arquivo XSD a parte, representando seus relacionamentos e cardinalidades. Esses esquemas foram chamados de visões. Cada esquema de visão se utiliza do esquema central para manipular os tipos de dados do domínio, maximizando o reuso dos mesmos. A abordagem permite inclusive flexibilidade de que novos esquemas de visão sejam criados, de acordo com a necessidade do serviço. Adicionalmente, as especificações das estruturas das mensagens trocadas entre serviços e seus consumidores possuem apenas informações relevantes à operação, evitando o consumo de recursos desnecessariamente.

Seguindo a abordagem de centralização de esquema proposta, foi desenvolvido um esquema central, chamado ModeloCanonico.xsd, contendo todas as entidades identificadas no modelo canônico. Este esquema conteve apenas as entidades e seus tipos de dados básicos, ou seja, não incluiu relacionamentos. O objetivo é que os serviços possam reaproveitar os tipos de dados e consigam manter a mesma estrutura do modelo canônico, facilitando a manutenção da estrutura. Já o contrato do serviço se utiliza de esquemas que denominamos esquemas de visão, onde são reaproveitados os tipos complexos do esquema central e são mapeados os relacionamentos das entidades utilizadas no esquema de visão. Isto permite que apenas as estruturas necessárias sejam trafegadas, ou seja, mesmo que o modelo de dados conceitual seja todo interligado, somente os dados necessários ao consumo do serviço são trafegados.

O desenvolvimento do WSDL foi realizado manualmente e teve como base as exigências do Basic Profile da WS-I, e o conceito de *contract-first* e suas possibilidades, como modularização e a reutilização dos tipos de dados desenvolvidos. Durante o processo, alguns testes foram realizados, tais como mudanças de nomenclatura e troca de bibliotecas, para verificar que o contrato se mantinha inalterado. Após isso, a ferramenta SOAP UI foi utilizada para realizar a verificação de todas as normas apresentadas pelo WS-I BP e para cada norma de interoperabilidade, o relatório informa o resultado do teste (aprovado, reprovado, aviso, não aplicável, faltando insumo), apresentando uma mensagem detalhada das não conformidades, se estas existirem.

Para a implementação dos serviços, foram utilizadas as especificações presentes na JEE6, edição *enterprise* da linguagem Java, próprias para desenvolvimento de Web Services SOAP. O JAX-WS (*Java API for XML Web Services*) é capaz de incorporar integralmente, através do uso de anotações, todas as capacidades de um WSDL 1.1 a uma interface Java. O uso do JAX-WS se torna fundamental neste trabalho, uma vez que permite utilizar um WSDL já elaborado, tornando possível aplicar a abordagem *contract-first* para desenvolver os serviços, e obter as vantagens observadas anteriormente. Para realizar a ligação (*binding*) entre dados em XML e classes Java [Ort e Metha, 2003] foi utilizado o JAXB (*Java Architecture for XML Binding*) para mapear as classes Java através de anotações, correlacionando tipos e/ou elementos de um XML Schema e atributos de classe Java. Com o domínio do processo bem definido e centralizado, é possível manter as classes JAXB em um único projeto que pode ser utilizado por vários serviços.

Utilizando abordagem *contract-first*, foi possível verificar aderência ao WS-I BP após a confecção do contrato, i.e., os serviços implementados atenderam a todos os critérios do *checklist* do WS-I, dessa forma obtendo um alto nível de interoperabilidade do serviço. Além disso, foram realizados testes envolvendo alterações de operação, entrada de dados e mudanças de tecnologia, e o consumidor do contrato do serviço não precisou ser alterado. Comparado ao uso de *code-first* (sem o uso da nossa abordagem), mesmo mudanças no serviço sem alterar sua interface (i.e., operações e dados de entrada e saída) podem levar a geração de um WSDL e, consequentemente, impactar consumidores do serviço.

A padronização de tipos de dados por domínio (centralização de esquema), desenvolvido a partir do modelo canônico proporcionou a eliminação de tipos de dados redundantes, melhorando a governança sobre os tipos de dados, tornando desnecessária transformações de dados e, consequentemente, melhorando o desempenho e interoperabilidade dos serviços. O desempenho também foi otimizado devido à utilização do padrão Jardim do Éden, associado ao esquema de visões elaborado no projeto. Essas visões proporcionaram diminuição do tráfego de informações em mensagens SOAP, pois apenas os dados definidos no esquema de visão são trafegados.

4. Conclusões

Apesar da abordagem code-first agilizar o desenvolvimento de serviços, esta técnica traz impactos negativos à governança de uma arquitetura orientada a serviços [Ganguly e Goswami, 2011]. Neste trabalho, foi abordada a utilização do padrão de desenvolvimento *contract-first* como alternativa ao *code-first* e as implicações negativas

que a geração automática de contrato traz. Para isso, foi dada sequência ao desenvolvimento dos serviços candidatos identificados e analisados por Azevedo *et al.* [2013], cujos passos são os principais para um ciclo de vida de desenvolvimento de serviços, como o proposto por Gu e Lago [2007].

A implementação realizada da abordagem de desenvolvimento *contract-first* aderente ao WS-I BP permitiu, ao longo do trabalho, avaliar mudanças na lógica de codificação, nomenclatura de métodos e de atributos, e troca de bibliotecas da tecnologia, sem que houvesse impacto nos testes de consumo do serviço, enquanto que simulações com a geração automática do contrato gerassem mudanças impactantes, como por exemplo, no namespace e nome dos atributos. Foi possível perceber também, a possibilidade de existirem diferentes implementações para o mesmo contrato, permitindo atender diferentes características de consumo do serviço. A centralização de esquema, por sua vez, permitiu que as entidades do modelo fossem construídas para possuir maior semântica, incorporando as restrições que o domínio expunha na modelagem, tais como valores padrão, mínimos e máximos. Isto também traz benefícios ao consumidor do serviço, possibilitando validar a mensagem que será enviada ao serviço antes da mensagem chegar ao seu destino.

Com este trabalho, conseguimos demonstrar as possibilidades de uso da abordagem *contract-first* comparado ao uso de *code-first*, utilizando tecnologias modernas e utilizadas em larga escala na indústria, apresentando os benefícios ao em utilizar *contract-first.*

5. Referências

AZEVEDO, L. G., SANTORO, F. M., BAIÃO, F., DIIRR, T., SOUZA, A., SOUZA, J. F., and SOUSA, H. P. (2013) "A method for bridging the gap between business process models and services". iSys: Revista Brasileira de Sistemas de Informação, 6(1):62–98.

ERL, T. (2005) Service-Oriented Architecture: concepts, technology, and design. Prentice Hall Englewood Cliffs.

ERL, T. (2008) SOA: Principles of Service Design, Prentice Hall Upper Saddle River.

GANGULY, K., GOSWAMI, P. (2011) "Developing web services, Part 1: Developing the code and contract first approach web service with Axis2". IBM developerWorks. http://www.ibm.com/developerworks/library/ws-devaxis2part1/, acesso 27/10/2014.

GU, Q., LAGO, P., (2007) "A stakeholder-driven service life cycle model for SOA". In: Proc. of 2nd International Workshop on Service Oriented Software Engineering.

HEWITT, E. (2009) Java SOA Cookbook. O'Reilly Media inc.

JOSUTTIS, N. M. (2007) SOA in Practice: The Art of Distributed System Design, O'Reilly Media, Inc.

KALIN, M. (2009) Java Web Services: Up and Running, 2009. O'Reilly Media, Inc.

ORT, E. METHA, B. (2003) "Java Architecture for XML Binding (JAXB)". Sun Developer Network. <http://www.oracle.com/technetwork/articles/javase/index-140168.html>, acesso 03/10/2014.