UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

CONTRATOS REST ROBUSTOS E LEVES: UMA ABORDAGEM EM DESIGN-BY-CONTRACT COM NEOIDL

LUCAS FERREIRA DE LIMA

ORIENTADOR: RODRIGO BONIFÁCIO DE ALMEIDA

DISSERTAÇÃO DE MESTRADO EM
ENGENHARIA ELÉTRICA

PUBLICAÇÃO: MTARH.DM - 017 A/99

BRASÍLIA/DF: JULHO - 2016.

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

CONTRATOS REST ROBUSTOS E LEVES: UMA ABORDAGEM EM DESIGN-BY-CONTRACT COM NEOIDL

LUCAS FERREIRA DE LIMA

DISSERTAÇÃO DE MESTRADO SUBMETIDA AO DEPARTAMENTO DE ENGENHARIA ELÉTRICA DA FACULDADE DE TECNOLOGIA DA UNIVERSIDADE DE BRASÍLIA, COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM ENGENHARIA ELÉTRICA.

APROVADA POR:

Prof. Rodrigo Bonifácio de Almeida, DSc. (ENE-UnB)
(Orientador)

Prof. XXX
(Examinador Interno)

YYY
(Examinador Externo)

BRASÍLIA/DF, 01 DE JULHO DE 2016.

DEDICATÓRIA

Este trabalho é dedicado a ...
continuação .....

AGRADECIMENTOS

(Página opcional) Agradeço .....
continuação .....

RESUMO

CONTRATOS REST ROBUSTOS E LEVES: UMA ABORDAGEM EM DESIGN-BY-CONTRACT COM NEOIDL

Autor: Lucas Ferreira de Lima
Orientador: Rodigo Bonifácio de Almeida
Programa de Pós-Graduação em Engenharia Elétrica
Brasília, julho de 2016

A adoção do paradigma arquiteturação baseado em serviços... O presente trabalho foi desenvolvido, ..., tendo como objetivo geral ..... Como objetivos específicos, o trabalho procurou .....

A proposta partiu...

Durante a realização...

Os resultados sugerem...

ABSTRACT

CONTRATOS REST ROBUSTOS E LEVES: UMA ABORDAGEM EM DESIGN-BY-CONTRACT COM NEOIDL

Autor: Lucas Ferreira de Lima
Orientador: Rodigo Bonifácio de Almeida
Programa de Pós-Graduação em Engenharia Elétrica
Brasília, julho de 2016

..

..

..

..

SUMÁRIO

LISTA DE TABELAS

LISTA DE FIGURAS

LISTA DE SÍMBOLOS, NOMENCLATURA E ABREVIAÇÕES

SOC: Software Oriented Computing, modelo arquitetural baseado em serviços.

DbC: Design by Contract, mecanismos de garantias com condições na chamadas a métodos, funções, serviços, etc.

1 INTRODUÇÃO

A computação orientada a serviços (*Service-oriented computing, SOC)* tem se mostrado uma solução de *design* de *software* que favorece o alinhamento às mudanças constantes e urgentes nas instituições [2]. Nessa abordagem, os recursos de software são empacotados como serviços, os quais são módulos bem definidos e auto-contidos, provêem funcionalidades negociais e com estado e contexto independente [9].

Os benefícios de SOC estão diretamente relacionados ao baixo acoplamento dos serviços que compõem a solução, de forma que as partes (nesse caso serviços) possam ser substituídas e evoluídas facilmente, ou ainda rearranjadas em novas composições. Contudo, para que isso seja possível, é necessário que os serviços possuam contratos bem definidos e independentes da implementação.

A relação entre quem provê e quem consome o serviço se dá por meio de um contrato. O contrato de serviço é o documento que descreve os propósitos e as funcionalidades do serviço, como ocorre a troca de mensagens, condições sobre como as operações são realizadas e informações sobre as operações [4].

Nesse contexto, a qualidade da especificação do contrato é fundamental para o projeto de software baseado em SOC. Este trabalho de pesquisa aborda um aspecto importante para a melhoria da robustez de contratos de serviços: a construção de garantias mútuas por meio da especificação formal de contratos, agregando o conceito de Desing-by-Contract.

1.1 PROBLEMA DE PESQUISA

As linguagens de especificação de contratos para SOC apresentam algumas limitações. Por exemplo, a linguagem WSDL (*Web-services description language*) [**?**] é considerada uma solução verbosa que desestimula a abordagem *Contract First*. Por essa razão, especificações WSDL são usualmente derivadas a partir de anotações em código fonte *Code First*. Além disso, os conceitos descritos em contratos na linguagem WSDL não são diretamente mapeados aos elementos que compõem as interfaces do estilo

arquitetural REST (*Representational State Transfer*). Outras alternativas para REST, como Swagger e RAML[1], usam linguagens de propósito geral (em particular JSON e YAML) adaptadas para especificação de contratos. Ainda que façam uso de contratos mais sucintos que WSDL, essas linguagens não se beneficiam da clareza típica das linguagens específicas para esse fim (como IDLs CORBA) e não oferecem mecanismos semânticos de extensibilidade e modularidade.

Com o objetivo de mitigar esses problemas, a linguagem NeoIDLfoi proposta para simplificar a especificação de serviços REST com mecanismos de modularização, suporte a anotações, herança em tipos de dados definidos pelo desenvolvedor, e uma sintaxe simples e concisa semelhante às *Interface Description Languages* – IDLs – presentes em *Apache Thrift*™e CORBA™. Por outro lado, a NeoIDL, da mesma forma que WSDL, Swagger e RAML não oferece construções para especificação de contratos formais de comportamento como os presentes em linguagens que suportam DBC (*Design by Contract*) [8], como JML, Spec# e Eiffel. Em outras palavras, a NeoIDLadmite apenas contratos fracos (*weak contracts*), sem suporte a construções como pré e pós condições.

1.2 OBJETIVO GERAL

O objetivo geral de trabalho é investigar o uso de construções de Design by Contract no contexto de computação orientada a serviços, verificando a viabilidade e utilidade de sua adoção na especificação de contratos e implentação de serviços REST.

1.2.1 Objetivos específicos

1. Realizar análise empírica de expressividade e reuso da especificação de contratos em NeoIDLem comparação com *Swagger*, a partir de contratos reais do Exército Brasileiro.

2. Extender a sintaxe da NeoIDLpara admitir construções de Design by Contract, com pré e pós condições para operações de serviços REST.

3. Implementar um estudo de caso de geração de código em *Python Twisted* com suporte a Design by Contracta partir de contratos especificados em NeoIDL

[1]http://raml.org/spec.html

4. Coletar a percepção de desenvolvedores sobre a aceitação da especificação de contratos REST com Design by Contractna NeoIDL

1.2.2 JUSTIFICATIVA E RELEVÂNCIA

- aumento do uso de SOC - potencial do SOC está no baixo acomplamento dos serviços. - aumento do uso de REST - contratos fracos prejudicam a qualidade/aumentam os erros - baixa qualidade prejudica o reuso -

O uso do padrão REST para construção de *Web Service*é crescente no contexto do desenvolvimento de soluções basedas em serviço e não se dipõe de uma linguagem padrão para especificação de contratos fortes. A linguagem específica de domínio NeoIDL- foi desenvolvida para ser uma alternativa para especificação REST, caracterizada pela coesão e simplicidade em se compreender, mas que não dispunha de suporte a pré e pós condições.

1.2.3 ESTRUTURA

2 REFERENCIAL TEÓRICO

2.1 COMPUTAÇÃO ORIENTADA A SERVIÇO

As empresas precisam estar preparadas para responder rápida e eficientemente a mudanças impostas por novas regulações, por aumento de competição ou ainda para usufruir de novas oportunidades. No contexto atual, em que as informações fluem de modo extremamente veloz, o tempo disperdiçado pelas organizações para se adaptar a um novo cenário tem um preço elevado, gerando expressiva perda de receita e, em um determinados casos, podendo causar a falência.

No campo das instituições governamentais, a eficiência na condução das ações do Estado impõem que a estrutura de troca de informações entre os mais variados entes seja continuamente adaptável, mutuamente integrada. Pode-se tomar como exemplo a edição de nova lei que implique alteração no cálculo do tempo de serviço para aposentadoria. A nova fórmula deve se propagar para ser aplicada em várias intituições que compõem a máquina pública.

Nessas situações, os sistemas de informação das organizações devem possibilitar que a dinâmica de adaptação ocorra sem demora, sob pena de, em vez de serem ferramental para apoiar continuamente os processos de negócio, se tornem entrave para a ágil incorporação dos novos processos. Por outro lado, a nova configuração deve se manter integra e funcional com o já complexo cenário de TI.

A eficiência na integração entre as soluções de TI é determinante para que se consiga alterar uma parte sem comprometer todo o ecossistema. A integração possibilita a combinação de eficiência e flexibilidade de recursos para otimizar a operação através e além dos limites de uma organização e a habilita para inteoperar facilmente [10].

A computação orientada a serviços – SOC – endereça essas necessidades em uma plataforma que aumenta a flexibilidade e melhora o alinhamento com o negócio, a fim de reagir rapidamente a mudanças nos requisitos de negócio. Para obter esses benefícios, os serviços devem cumprir com determinados quesitos, que incluem alta autonomia ou baixo acoplamento [3]. Assim, o paradigma de SOC está voltado para o projeto de

soluções preparadas para constantes mudanças, substituindo-se pequenas peças – os serviços – por outras.

Portando, o objetivo da SOC é conceber um estilo de projeto, tecnologia e processos que permitam às empresas desenvolver, interconectar e manter suas aplicações e serviços corporativos com eficiência e baixo custo. Embora esses objetivos não sejam novos, SOC procura superar os esforços prévios como programação modular, reuso de código e técnicas de desenvolvimento orientadas a objetos [11].

As vertentes mais visionárias – não ainda concretizada e utópica para muitos pesquisadores – da computação orientada a serviços prevêem uma coordenação de serviços cooperantes por todo o mundo, onde os componentes possam ser conectados facilmente em uma rede de serviços pouquíssimo acoplados e, assim, criar processos de negócio dinâmicos e aplicações ágeis entre organizações e plataformas de computação [6].

2.1.1 Terminologia

Computação orientada a serviço é um termo *guarda-chuva* para descrever uma nova geração de computação distribuída. Desse modo, é um conceito que engloba várias coisas, como paradigmas e princípios de projeto, catálogo de padrões de projeto, padronização de linguagem, modelo arquitetural específico, e conceitos correlacionados, tecnologias e plataformas. A computação orientada a serviços é baseada em modelos anteriores de computação distribuída e os extendem com novas camadas de projeto, aspectos de governança, e uma grande gama de tecnologias de implementações especializadas, em grande parte baseadas em *Web Service* [4].

Orientação a serviço é um paradígma de projeto cuja intenção é a criação de unidades lógicas moldadas individualmente para podem serem utilizadas conjutamente e repetidamente para se atender a objetivos e funções específicos associados com SOA e computação orientada a serviço.

A lógica concebida de acordo com orientação a serviço pode ser designada de **orientada a serviço**, e as unidades da lógica orientada a serviço são referenciadas como **serviços**. Como um paradigma de computação distribuída, a orientação a serviço pode ser comparada a orientação a objetos, de onde advém várias de suas raízes, além da influência de EAI, BMP e *Web Service*[4].

A orientação a serviços é composta principalmente de oito princípios de projeto (descritos na seção 2.1.3).

Arquitetura orientada a serviço - SOA representa um modelo arquitetural cujo objetivo é elevar a agilidade e a redução de custos e ao mesmo tempo reduzir o peso da TI para a organização. Isso é feito colocando o serviço no como elemento central da representação lógica da solução [4].

Como uma arquitetura tecnológica, uma implementação SOA consiste da combinação de tecnologias, produtos, APIs, extensões da infraestrutura, etc. A implantação concreta de uma arquitetura orientada a serviço é única para cada organização, entretanto é caracterizada pela introdução de tecnologias e plataformas que suportam a criação, execução e evolução de soluções orientadas a serviços. O resultado é a formação de um ambiente projetado para produzir soluções alinhadas aos princípios de projeto de orientação a serviço.

Segundo Thomas Erl [4], o termo arquitetura orientada a serviço – SOA – vem sendo amplamente utilizado na mídia e nos produtos de divugação de fabricantes que tem se tornado quase que sinônimo de computação orientada a serviço – SOC.

Serviço é a unidade da solução no qual foi aplicada a orientação a serviço. É a aplicação da orientação dos princípios de projeto de orientação a serviço que distigue uma unidade de lógica como um serviço comporada a outras unidades de serviços que podem existir isoladamente como um objeto ou componente [4].

Após a modelagem conceitual do serviço, os estágios de projeto e desenvolvimento produzem um serviço que é programa de *software* independente com características específicas para suportar a realização dos objetivos associados a computação orientada a serviço.

Cada serviço possui um contexto funcional distinto e é composto de uma lista de capacidades relacionadas a esse contexto. Então um serviço pode ser considerado um conjunto de capacidades descritas em seu contrato.

Contrato de serviço é o conjunto de documentos que expressam as meta-informações do serviço, sendo a parte fundamental a que descreve a sua interface técnica. Eles compõem o contrato técnico do serviço, cuja essência é estabelecer uma API com as funcionalidades providas pelo serviço por meio de suas capacidades [4].

Os serviços implementados como *Web Service*SOAP normalmente são descritos em seu WSDL [1], *XML schemas* and políticas (*WS-policy*). Já os serviços im-

[1] *Web Service Description Language*

plementados como *Web Service* REST não possuem uma linguagem padrão para especificação de contratos. Já foram propostas algumas alternativas como WADL [5], Swagger [1], e NeoIDL [7].

O contrato de serviço também pode ser composto de documentos de leitura humana, como os que descrevem níveis de serviços (*SLA*), comportamentos e limitações. Muitas dessas características também podem ser descritas em linguagens formais (para processamento computacional).

No contexto de orientação a serviço, o projeto do contrato do serviço é de suma importância de tal forma que o princípio de projeto contrato de serviço padronizado é dedicado exclusivamente para se padronizar a criação dos contratos de serviços [4].

2.1.2 Modelo arquitetural

...

In contrast to conventional software architectures pri- marily delineating the organization of a system in its (sub)systems and their interrelationships, the SOA cap- tures a logical way of designing a software system to provide services to either end-user applications or other services distributed in a network through published and discoverable interfaces. SOA is focused on creating a design style, technology, and process framework that will allow enterprises to develop, interconnect, and maintain enterprise applications and services efficiently and cost- effectively. [11]

Service orientation provides the underlying implementation that can make an on demand IT operating environment a reality by supporting the functions of both integration and infra- structure management [Lymann 2005a]. Service-Oriented Computing (SOC) utilizes services as the constructs to support the devel- opment of rapid, low-cost and easy composition of distributed applications. Services are autonomous, platform-independent computational entities that can be used in a platform independent way. Services can be described, published, discovered, and dynamically as- sembled for developing massively distributed, interoperable, evolvable systems. Services perform functions that can range from answering simple requests to executing sophisticated business processes requiring peer-to-peer relationships between possibly multiple layers of service consumers and providers. Any piece of code and any application component deployed on a system can be reused and transformed into a network-available service.

Serv- ices reflect a "service-oriented"approach to programming, based on the idea of composing applications by discovering and invoking network-available services rather than building new applications or by invoking available applications to accomplish some task [Papazoglou 2003]. Services are most often built in a way that is independent of the context in which they are used. This means that the service provider and the consumers are loosely coupled. This "service-oriented"approach is independent of specific programming languages or oper- ating systems.

2.1.3 Princípios SOA

O paradigma de orientação a serviço é estruturada em oito princípios fundamentais [4]. São eles que caraterizam a abordagem SOA e a sua aplicação fazem com que um serviço se diferenciem de um componente ou módulo. Os contratos de serviços permeiam a maior parte destes princípios:

Contrato padronizado fornecem um meio padrão para os serviços sejam registrados em um inventário. Os contratos de serviços são elementos fundamentais pois é por meio deles que os serviços interagem uns com os outros e com potenciais consumidores.

Baixo acomplamento

Abstração

Reusabilidade

Autonomia

Ausência de estado

Descoberta de serviço

Composição

Service-orientation represents a design approach comprised of eight specific design principles. Service contracts tie into most but not all of these principles. Let's first introduce their official definitions: • Standardized Service Contract – "Services within the same service inventory are in compliance with the same contract design standards." • Service Loose Coupling – "Service contracts impose low consumer coupling requirements and are themselves decoupled from their surrounding environment." • Service

Abstraction – "Service contracts only contain essential information and information about services is limited to what is published in service contracts." • Service Reusability – "Services contain and express agnostic logic and can be positioned as reusable enterprise resources." • Service Autonomy – "Services exercise a high level of control over their underlying runtime execution environment." • Service Statelessness – "Services minimize resource consumption by deferring the management of state information when necessary." • Service Discoverability – "Services are supplemented with communicative meta data by which they can be effectively discovered and interpreted." • Service Composability – "Services are effective composition participants, regardless of the size and complexity of the composition."

Standardized Service Contract Given its name, it's quite evident that this design principle is only about service contracts and the requirement for them to be consistently standardized within the boundary of a service inventory. This design principle essentially advocates "contract first" design for services.

Service Loose Coupling This principle also relates to the service contract. Its design and how it is architecturally positioned within the service architecture are regulated with a strong emphasis on ensuring that only the right type of content makes its way into the contract in order to avoid the negative coupling types. The following sections briefly describe common types of coupling. All are considered negative coupling types, except for the last. Contract-to-Functional Coupling, Contract-to-Implementation Coupling, Contract-to-Logic Coupling, Contract-to-Technology Coupling, Logic-to-Contract Coupling (Each of the previously described forms of coupling are considered negative because they can shorten the lifespan of a Web service contract, thereby leading to increased governance burden as a result of having to manage service contract versions.)

Service Abstraction By turning services into black boxes, the contracts are all that is officially made available to consumer designers who want to use the services. While much of this principle is about the controlled hiding of information by service owners, it also advocates the streamlining of contract content to ensure that only essential content is made available. The related use of the Validation Abstraction pattern further can affect aspects of contract design, especially related to the constraint granularity of service capabilities.

Service Reusability While this design principle is certainly focused on ensuring that service logic is designed to be robust and generic and much like a commercial product, these qualities also carry over into contract design. When viewing the service as a product and its contract as a generic API to which potentially many consumer programs will need to interface, the requirement emerges to ensure that the service's functional context, the definition of its capabilities, and the level at which each of its design granularities are set are appropriate for it to be positioned as a reusable enterprise resource.

Service autonomy

Autonomia de serviços é um princípio que visa fornecer serviços com independência de seu ambiente de execução. Isso resulta em maior confiabilidade, já que os serviços podem operar com menos dependência de recursos sobre os quais há pouco ou nenhum controle. Confiabilidade é crítico para garantir a longevidade do serviço.

Service statelessness

Interação entre softwares ou requisitos de negócio muitas vezes exigem o esforço de manter e gerenciar o controle do estado das informações entre as operações. Isso se torna mais importante em arquiteturas distribuídas onde o cliente e o servidor não estão fisicamente na mesma máquina. Numa composição de serviço, um serviço pode armazenar dados específicos da atividade em memória enquanto ele espera um outro serviço completar seu processamento. Gestão eficiente da atividade de serviço relacionado aos dados se torna mais importante como um serviço que visa a reutilização do mesmo. As diretrizes de arquitetura SOA é construir serviços stateless, deslocando a sobrecarga de gerenciamento de estado dos serviços para um outro componente/middleware externo. Isso ajuda ainda mais na escalabilidade global da solução.

Service Discoverability Because the service contracts usually represent all that is made available about a service, they are what this principle is primarily focused on when attempting to make each service as discoverable and interpretable as possible by a range of project team members. Note that although Web service contracts need to be designed to be discoverable, this book does not discuss discovery processes or registry-based architectures.

Service Composability This regulatory design principle is very concerned with ensuring that service contracts are designed to represent and enable services to be effective composition participants. The contracts must therefore adhere to the requirements of the previously listed design principles and also take multiple and complex service composition requirements into account.

2.1.3.1 Padrões de projeto

...

2.2 Web Services

...

2.2.1 SOAP (W3C)

...

2.2.1.1 Especificação de contratos

...

2.2.2 REST (Fielding)

A tecnologia REST é caracterizada principalmente pela convenção da adoção dos métodos do protocolo HTTP para definição das operações, transferência de estado nas requisições. - Ausência de padrão para especificação de contratos REST. - Fragilidade de garantias por quem consome o serviço; - Risco de a ausência de informações prejudicarem os consumidores.

2.2.2.1 Especificação de contratos

...

2.2.2.2 Outros padrões

...

2.3 Design-by-Contract

...

2.3.1 Origem

- Bertrand Meyer (eifel)

2.3.2 Implementações de DbC

- Eifel - JML - Spec#

3 A LINGUAGEM PARA ESPECIFICAÇÃO DE CONTRATOS NEOIDL

3.1 NeoIDL

3.1.1 Objetivos

3.1.2 Histórico

3.1.3 Arquitetura do framework

3.1.4 Avaliações

3.1.4.1 Expressividade

Nesta subseção falar da avaliação feita nos contratos do exército.

3.1.4.2 Potencial de reuso

Complementar a avaliação de expressividade com o potencial de reuso.

3.2 Proposta: Serviços com Desing-by-Contract

Os benefícios esperados pela adoção da arquitetura orientada a serviços somente serão auferidos com a concepção adequada de cada serviço. Por essa razão, é necessário planejar o projeto dos serviços criteriosamente antes de lançar mão do desenvolvimento, com preocupação especial em garantir um nível aceitável de estabilidade aos consumidores de cada serviço. Nessa etapa do projeto de desenho da solução, a especificação do contrato do serviço (Web API) exerce uma função fundamental.

Na sociedade civil, contratos são meios de se formalizar acordo entre partes a fim de definir os direitos e deveres de cada parte e buscar atingir o objetivo esperado dentro de determinadas regras. Cada parte espera que as outras cumpram com suas

obrigações. Por outro lado, sabe-se que o descumprimeto das obrigações costuma implicar de penalizações até o desfazimento do contrato.

Contratos entre serviços Web seguem em uma linha análoga. O desenho das capacidades (operações) e dos dados das mensagens correspondem aos termos do contrato no sentido do que o consumidor deve esperar do serviço provedor. Porém identificou-se, após ampla pesquisa realizada sobre o tema, que as linguagens disponíves para especificação de contratos atingem apenas esse nível de garantias. No contexto de web-services em REST, conforme descrito na seção 2.2.2, há ainda a ausência de padrão para especificação contratos, tal como ocorre com o WSDL adotado em SOAP.

A proposta deste trabalho é extender os níveis de garantias, de modo a promover um patamar adicional com obrigações mutuas entre os serviços (consumidor e provedor). Isso se dá para adoção do conceito de Design-by-Contract (debatido na seção 2.3) em que a execução da capacidade do serviço garantirá a execução, desde que satisfeitas as condições prévias. O detalhamento do processo é exposto nas seções que se seguem.

3.2.1 Modelo de operação

As garantias para execução dos serviços são estabelecidas em duas etapas: pré- e pós-condições. Nas pré-condições o provedor do serviço estabelece os requisitos para que o serviço possa ser executado. A etapa de pós-condições tem o papel de validar se a mensagem de retorno do serviço possui resultados válidos.

O diagrama da Figura 3.1 descreve como ocorre a operação das pré- e pós-condições. O processo se inicia com a chamada à capacidade do serviço e a identificação da existência de uma pré-condição. Caso tenham sido estabelecidas pré-condições, essas são avaliadas. Caso alguma delas não tenham sido satisfeitas, o serviço principal não é processado e o provedor do serviço retornar o código de falha definido no contrato correspondente.

Caso tenham sido definidas pós-condições, essas são acionadas após o processamento da capacidade, porém antes do retorno ao consumidor do serviço. Assim, conforme Figura 3.1, visando não entregar ao cliente uma mensagem ou situação incoerente, as pós-condições são validadas. Caso todas as pós-condições tenham sido satisfeitas, a mensagem de retorno é encaminhada ao cliente. Caso contrário, será retornado o código de falha.

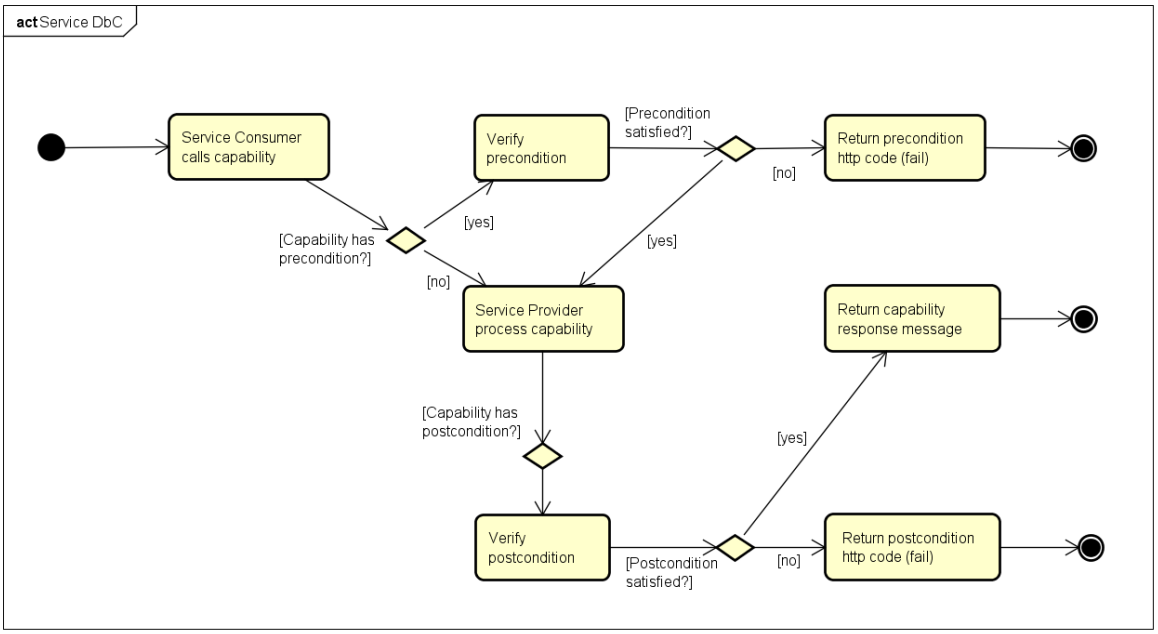

Figura 3.1: Digrama de atividades com verificação de pré e pós condições

3.2.1.1 Verificação das pré-condições

As pré-condições podem ser do tipo baseado nos parâmetros da requisição ou do tipo baseado na chamada a outro serviço. Denominamos, para o contexto desta dissertação, de básica a pré-condição baseada apenas nos parâmetros da requisição (atributos da chamada ao serviço). Nessa validação é direta, comparando os valores passados com os valores admitidos.

No caso das pré-condições baseadas em serviços, é realizada chamada a outro serviço para verificar se uma determinada condição é satisfeita. Este modo de funcionamento, que se assemelha a uma composição de serviço, é mais versátil, pois permite validações de condições complexas sem que a lógica associada seja conhecida pelo cliente. Assim, os contratos que estabelecem esse tipo de pré-condição se mantem simples.

A Figura 3.2 detalha as etapas de verificação de cada pré-condição. Nota-se que a saída para as situações de desatendimento às pré-condições, independentemente do tipo, é o mesmo. O objetivo desta abordagem é simplificar o tratametno de exceção no consumidor.

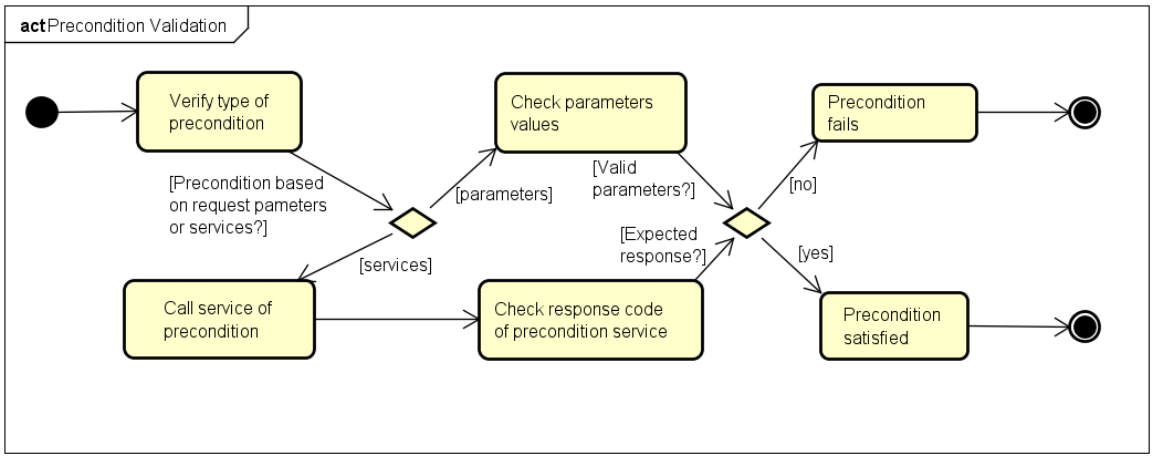

Figura 3.2: Diagrama de atividades do processamento da pré-condição

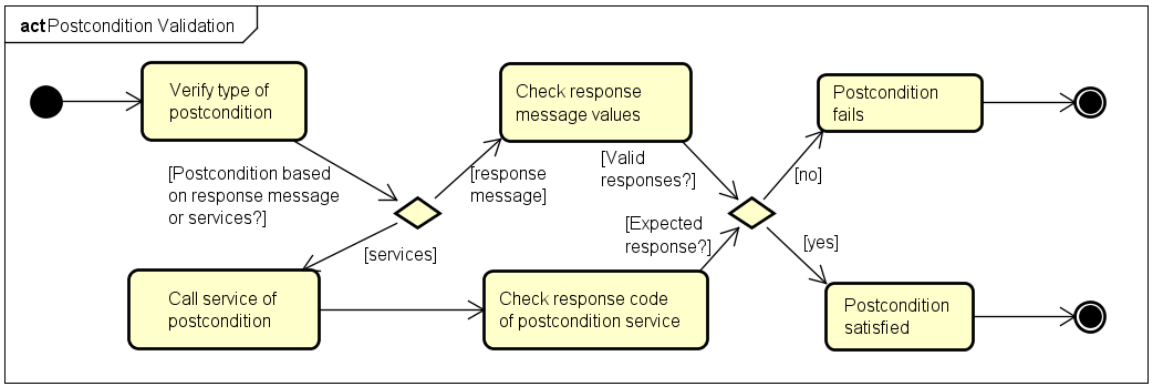

Figura 3.3: Diagrama de atividades do processamento da pós-condição

3.2.1.2 Verificação das pós-condições

A verificação das pós-condições acontece de modo muito similar a das pré-condições. Há também os dois tipos, baseado em valores e em chamadas a outros serviços. O diferencial está em que a validação dos valores passa a ocorrer a partir dos valores contidos na mensagem de retorno. A Figura 3.3 descreve as etapas necessárias para validação de cada pré-condição.

[language=NeoIDL,firstnumber=1]DBCsimple.neo

Figura 3.4: Exemplo da notação DBC básica na NeoIDL

3.3 Implementação de Design-by-Contract na NeoIDL

3.3.1 Pré-condição básica

3.3.2 Pós-condição básica

...

3.3.3 Pré-condição com chamada a serviço

...

3.3.4 Pós-condição com chamada a serviço

...

3.4 Estudo de caso

...

3.4.1 Pluggin Twisted

4 AVALIAÇÃO SUBJETIVA

4.1 Utilidade da NeoIDL com DbC

4.1.1 Método

4.1.2 GQM

4.1.3 Questionário

4.1.4 Análise dos Resultados

* Questionário montado para avaliar a utilidade de DbC com NeoIDL

* Inicialmente motivado pelo estudo do Alessandro Garcia

* GQM e Avaliação TAM

* Montagem do questionário

1. Perfil técnico-profissional do respondente 1.1 Para qual órgão ou empresa você presta serviços atualmente? 1.2 A quanto tempo você trabalha com desenvolvimento Web 1.3 A quanto tempo você desenvolve com uso de APIs Web (Web Service) 1.4 Qual o seu nível de experiência com especificação de API REST 1.5 Qual o seu nível de experiência com especificação de contratos com Swagger

3 Questões sobre especificação e implementação de APIs Web 3.1 A especificação do contrato formalmente, seja em Swagger ou NeoIDL, em relação a descrição textual, aumentará meu nível de acerto na implementação (efetividade). 3.2 Identificar e compreender as operações e atributos na especificação Swagger é simples para mim. 3.3 Identificar e compreender as operações e atributos na especificação NeoIDL é simples para mim.

4. DbC 4.1 Conhecer previamente e explicitamente as pré-condições será útil para mim. (Useful) 4.2 Aprender a identificar as pré-condições na NeoIDL parece ser simples pra mim (Easy to learn) 4.3 Parece ser fácil para mim declarar uma pré-condição na NeoIDL

(Clear and understandable) 4.4 Me lembrar da sintaxe da pré-condição na NeoIDL é fácil (Remember)

5. Geração de código 5.1 É claro e compreensível para mim o efeito da pré-condição sobre o código gerado (Controllable) 5.2 A geração do código de pré e pós-condições aumentará minha produtividade na implementação do serviço (Job performance) 5.3 Assumindo ter a disposição a NeoIDL no meu trabalho, para especificação de contratos e geração de código, eu presumo que a utilizarei regularmente no futuro. 5.4 Nesse mesmo contexto, eu vou preferir utilizar contratos escritos em NeoIDL do que descritos de outra forma

* Distribuição do questionário

* Avaliação dos resultados

* Ameaças - não foi fornecido nenhum material sobre a NeoIDL, apresentando somente o uma descrição de serviço - O questionário foi aplicado uma única vez, sem melhorias a partir do primeiro conjunto de respostas

* Questionários futuros

A principal questão de pesquisa a ser avaliada com o uso do questionário é a utilidade em se agregar ao design das especificações de serviços REST as garantias de pré e pós-condições. Em segundo momento, pressupondo a utilidade, avaliar se a NeoIDL cumpre satisfatoriamente com este propósito, agregando à sintaxe da linguagem a possibilidade de se expressar pré e pós-condições.

* Separar os respondentes em faixas de experiência. Verificar se as respostas dos menos experientes precisam ser descartadas pela pouca capacidade crítica. Separar a análise entre os respondentes que conhecem e os que não conhecem Swagger.

* Perspectivas de comparação a) Experiência com desenvolvimento com uso de REST b) Experiência com Swagger c) Utilidade da especificação formal de contratos d) Percepção da NeoIDL sem DbC

5 CONCLUSÕES E TRABALHOS RELACIONADOS

5.1 CONCLUSÕES GERAIS

A boa definição dos contratos ...

5.2 TRABALHOS RELACIONADOS E PESQUISAS FUTURAS

...

- Trabalhos relacionados com hipermedia

REFERÊNCIAS BIBLIOGRÁFICAS

[1] Swagger - the world's most popular framework for apis. http://swagger.io/. Swagger project oficial site.

[2] Chen, H.-M. Towards service engineering: service orientation and business-it alignment. In *Hawaii International Conference on System Sciences, Proceedings of the 41st Annual*, pages 114–114. IEEE, 2008.

[3] Erl, T. *Soa: principles of service design*, volume 1. Prentice Hall Upper Saddle River, 2008.

[4] Erl, T., Karmarkar, A., Walmsley, P., Haas, H., Yalcinalp, L. U., Liu, K., Orchard, D., Tost, A., e Pasley, J. *Web service contract design and versioning for SOA*. Prentice Hall, 2009.

[5] Hadley, M. J. Web application description language (wadl). 2006.

[6] Leymann, F. Combining web services and the grid: Towards adaptive enterprise applications. In *CAiSE Workshops (2)*, pages 9–21, 2005.

[7] Lima, L., Bonifácio, R., Canedo, E., Castro, T. M.de , Fernandes, R., Palmeira, A., e Kulesza, U. Neoidl: A domain specific language for specifying rest contracts detailed design and extended evaluation. *International Journal of Software Engineering and Knowledge Engineering*, 25(09n10):1653–1675, 2015.

[8] Meyer, B. Applying'design by contract'. *Computer*, 25(10):40–51, 1992.

[9] Papazoglou, M. P., Traverso, P., Dustdar, S., e Leymann, F. Service-oriented computing: State of the art and research challenges. *Computer*, (11):38–45, 2007.

[10] Papazoglou, M. P., Traverso, P., Dustdar, S., e Leymann, F. Service-oriented computing: a research roadmap. *International Journal of Cooperative Information Systems*, 17(02):223–255, 2008.

[11] Papazoglou, M. P. e Van Den Heuvel, W.-J. Service oriented architectures: approaches, technologies and research issues. *The VLDB journal*, 16(3):389–415, 2007.

APÊNDICES

A CONTRATO NEOIDL COM DBC

...