PONTIFÍCIA UNIVERSIDADE CATÓLICA DE SÃO PAULO

TIAGO LEANDRO MARTINS

MODELAGEM DE CONVERSAÇÕES REST: UMA ESTRATÉGIA DE AUTOMAÇÃO BASEADA EM FATORES DE QUALIDADE

São Paulo – SP

2019

TIAGO LEANDRO MARTINS

MODELAGEM DE CONVERSAÇÕES REST: UMA ESTRATÉGIA DE AUTOMAÇÃO BASEADA EM FATORES DE QUALIDADE

Monografia apresentada ao Curso de Engenharia de Software da Pontifícia Universidade Católica de São Paulo como um dos pré-requisitos para obtenção do título de Especialista em Engenharia de Software, sob orientação do Prof. Dr. Daniel Couto Gatti.

São Paulo – SP

2019

AGRADECIMENTOS

A Deus, pela vida.

Ao meu orientador Prof. Dr. Daniel Couto Gatti pelas preciosas instruções e valiosos ensinamentos.

À minha esposa, pelo amor e parceria.

A meus pais e minha irmã, por desde cedo fomentarem a alegria no constante aprendizado.

RESUMO

Aplicações REST tem sido amplamente utilizadas para troca de informações entre sistemas de software. Encontra-se na bibliografia recente diversos esforços para automatizar o desenvolvimento destas aplicações, utilizando-se de princípios da disciplina de desenvolvimento dirigido por modelo. Ao mesmo tempo nota-se que diversos serviços publicados na rede denominados como RESTFul de fato não o são, devido a má aplicação das restrições do estilo arquitetural. Uma vez que as possibilidades e orquestração das conversações são múltiplas, torna-se um fator importante durante a transformação dos meta-modelos observar as restrições que o estilo arquitetural REST exige. Mesmo assim, em determinados padrões de conversação onde se aplica as restrições REST, mais de uma possibilidade de modelagem torna-se possível. Nestes casos, decisões de design são tomadas, para resolução de conflitos na transformação entre os meta-modelos até a obtenção do código fonte. A proposta apresentada é que fatores de qualidade não funcionais sejam os direcionadores de escolha dentre os possíveis padrões de conversação REST em dado domínio. Para tanto é desenvolvida uma estratégia que consiste em um meta-modelo de qualidade para direcionar as transformações entre meta-modelos REST. Neste meta-modelo de qualidade são indicados os fatores de qualidade que exercem influência no domínio que origina a modelagem, bem como o impacto mutuo que estes fatores exercem sobre si, tornando possível que a decisão de design seja automatizada e embasada nos fatores de qualidade que se deseja prestigiar em detrimento de outros que não são determinantes para o bom funcionamento da aplicação.

Palavras-chave: Aplicações REST. Modelagem de software. Desenvolvimento dirigido por modelo. Fatores de Qualidade.

Lista de Ilustrações

Sumário

1 INTRODUÇÃO

Nesse capítulo são apresentados os elementos principais que definem essa pesquisa: Motivação, Objetivos, Delimitação, Contribuição, Método de Trabalho, Organização do Texto e Cronograma.

1.1 Motivação

Nos anos recentes, serviços ReST (Representional State Transfer) tem sido amplamente utilizados para construir sistemas distribuídos baseados em Web (COSTA et al., 2016). Este estilo de arquitetura é definido formalmente por uma série de restrições (FIELDING; TAYLOR, 2002) e para que uma API possa ser classificada como compatível com as definições ReST, estas restrições devem ser atentamente observadas.

Abordagens de desenvolvimento dirigido por modelo (MDD) são propostas para melhorar a implementação de serviços complexos (VOLTER et al., 2013). Abordagens desta natureza têm sido utilizadas também para desenho e implementação de aplicações ReSTful, tendo o foco em como atingir o objetivo primário de que a aplicação criada seja ao máximo compatível com as restrições ReST, apresentando maior escalabilidade e evolutividade a longo prazo (HAUPT et al., 2014).

Uma abordagem deste tipo propõe, entre outros pontos, a definição de meta-modelos – representados por diagramas de classe UML - para o desenho e implementação dos serviços ReSTful. Na camada mais alta dos meta-modelos está o modelo de domínio, amplamente utilizado em levantamento de requisitos e neste caso especialmente para definir a interface da aplicação independente da arquitetura. O meta-modelo subsequente é o modelo de recurso, sendo este um modelo introduzido pelo paradigma da orientação a recurso. Neste último modelo inicia-se a aplicação das restrições ReST, como por exemplo, a restrição da “interface uniforme”, onde apenas verbos HTTP podem ser utilizados, por exemplo: GET, PUT, POST, DELETE.

A transformação do modelo de domínio para o modelo de recurso pode ocasionar uma “incompatibilidade de impedâncias”, pois o mapeamento de um para o outro é não-trivial e não-completo pelo fato de que os modelos não são mutualmente compatíveis, podendo incluir diversas negociações e impurezas (HAUPT et al., 2014).

Para tornar esta tarefa mais assertiva, durante o desenvolvimento de APIs ReSTful dirigido por modelo, identificou-se certos padrões de conversação por meio da observação de comunicações que acontecem na prática.

Mesmo com a definição destes padrões, estas conversações – que são compostas por interações entre recursos – podem ser construídas de diversas formas, utilizando diferentes combinações de interações e seus elementos constituintes (representações, verbos HTTP, etc.). Funcionalmente, a adoção de uma estratégia não adequada pode até culminar num resultado funcional aceitável em uma primeira avaliação, entretanto, pode apresentar problemas de qualidade ao longo do tempo. Daí surge a necessidade de se definir pré-condições e características que direcionem a transformação entre os meta-modelos em um contexto de desenvolvimento dirigido por modelo.

Caso não se defina as regras de transformação por meio de uma estratégia abstrata, baseada em padrões de conversação (HAUPT et al., 2015) e que atenda os objetivos e atributos de qualidade definidos como imprescindíveis no contexto em que as integrações estão inseridas, o consumo dos serviços pode, em primeira instância, ser classificado como não compatível com a arquitetura ReST, e em um nível de criticidade maior, apresentar problemas de interoperabilidade (COSTA et al., 2014).

1.2 Objetivo

O principal objetivo deste trabalho é elaborar uma estratégia de transformação (também chamada de expansão) entre os meta-modelos de uma API ReSTful hipotética e genérica, baseada em padrões de conversação, que tenha foco nos atributos de qualidade não funcionais e que seja relevante no contexto de desenvolvimento dirigido por modelos.

Esta estratégia visa manter o serviço ou API o mais compatível possível com a arquitetura ReST, bem como resolver possíveis problemas derivados de negociações e impurezas provenientes da transformação dos meta-modelos.

A pesquisa tem a proposição, de forma mais específica, que os requisitos de qualidade não funcionais sejam os direcionadores principais na escolha das transformações entre os modelos de recurso.

1.3 Delimitação

O objeto de estudo é, considerando-se uma abordagem de desenvolvimento de APIs ReSTful dirigida por modelos, o mecanismo que realiza a transformação do meta-modelo de domínio em recurso.

O foco mais específico da pesquisa é o processo de escolha da expansão utilizada entre os dois meta-modelos que compõem o meta-modelo de recurso: centrado em conversações e centrado em interações, conforme proposto por HAUPT (2015), em casos onde houver mais de uma possibilidade de transformação.

O foco do experimento estará no desenho e arquitetura dos modelos que orquestram as expansões, por meio de diagramas de classe da UML. A implementação da transformação, ou seja, o código fonte que efetua de fato a expansão, não terá o foco principal.

1.4 Contribuições

Com este trabalho se busca refinar e agregar valor à solução publicada por HAUPT (2015) no que diz respeito a definição de pré-requisitos e características utilizados para se escolher a expansão entre meta-modelos mais adequada em relação ao domínio, privilegiando atributos de qualidade que favoreçam sua eficácia e eficiência.

1.5 Método de Trabalho

Para atingir os objetivos apresentados, as seguintes macro atividades foram executadas:

- Pesquisa bibliográfica: Exploração dos fundamentos teóricos que permeiam o cenário objeto de estudo como os conceitos de arquitetura ReST, conceitos de desenvolvimento dirigido por modelo, levantamento de requisitos de qualidade de software e métodos de avaliação. Também se estuda a abordagem de desenvolvimento dirigido por modelo aplicada em APIs ReSTful, o conceito dos meta-

modelos e detalhamento de cada um deles, bem como os padrões de transformação aplicáveis e ferramentas de implementação.

- Definição de hipóteses: Com base nos fundamentos pesquisados, as hipóteses de melhoria e refinamento serão definidas e o meta-modelo será proposto.

- Elaboração da estratégia: Consiste na amálgama entre as principais bases teóricas do trabalho, enquadrando os requisitos de qualidade importantes para arquitetura ReST nos padrões de expansão dos meta-modelos de conversação e interação.

- Execução do experimento: A partir das hipóteses levantadas, o redesenho dos padrões de expansão dos meta-modelos poderá ser realizado, utilizando-se das ferramentas adequadas.

- Avaliação dos resultados: Após execução do redesenho proposto, uma análise deverá ser realizada para que se tire as conclusões no que tange a solidificação da estratégia proposta, ou seja, deverá se verificar se a proposta de refinamento dos padrões de expansão dos meta-modelos atinge o objetivo de agregar valor ao modelo já existente.

- Redação da monografia: Esta tarefa consiste na digitação e normalização do trabalho e será executada em momentos específicos, durante toda a extensão do trabalho.

1.6 Organização do Texto

No capítulo 2, “ReST e desenvolvimento dirigido por modelo”, serão apresentados trabalhos recentes sobre Arquitetura ReST, conceitos de desenvolvimento dirigido por modelo e transformação entre modelos por meio de gramática de grafos.

No capítulo 3, “Requisitos de qualidade para conversações ReST”, serão elencados quais são os principais atributos de qualidade aplicáveis no contexto das conversações ReSTful e sua influência nos padrões de conversação.

No capítulo 4, “Estratégia de transformação dos meta-modelos”, os requisitos de qualidade serão incorporados a estratégia de transformação já existente, agregando valor ao processo de transformação proposto em trabalhos relacionados.

No capítulo 5, “Conclusão”, poderão ser avaliados os impactos da estratégia proposta, os resultados alcançados e as dificuldades que foram encontradas no decorrer da pesquisa.

No capítulo 6, “Bibliografia”, as fontes utilizadas como base para o trabalho serão devidamente expostas.

1.7 Cronograma

As macro-atividades deverão ser executadas seguindo a seguinte proposta de cronograma:

Atividade	jan/18	fev/18	mar/18	abr/18	mai/18	jun/18	jul/18	ago/18	set/18	out/18	nov/18	dez/18	jan/19	fev/19	mar/19	abr/19
Pesquisa bibliográfica																
Definição de hipóteses																
Elaboração da estratégia																
Execução de experimento																
Avaliação de resultados																
Redação da monografia																

Figura 01: Cronograma atividades do trabalho
Fonte: Autor

2 REST E DESENVOLVIMENTO DIRIGIDO POR MODELO

Neste capítulo serão apresentados conceitos e fundamentos sobre os quais o assunto do presente trabalho se baseia: estilo arquitetural ReST e desenvolvimento dirigido por modelo.

2.1 Estilo arquitetural ReST

Na tese em que o estilo arquitetural ReST foi introduzido, Fielding (2000) apresentou uma clara definição sobre arquitetura de software, tratando-a como uma abstração de elementos de um sistema de software em tempo de execução, considerando determinada fase de sua operação.

A partir desta ideia, pressupôs ainda que um estilo arquitetônico ou arquitetural é um conjunto de limitações que restringe características e papeis de determinados elementos, permitindo relações entre estes elementos dentro de qualquer arquitetura, desde que em conformidade com este estilo.

Em consonância com esta definição, para Clements (2010) um estilo arquitetural é uma especialização de elemento e tipos de relação, juntamente com um conjunto de restrições, sobre como podem ser utilizados.

Ainda, segundo Sommerville (2011) um desenho arquitetural é um processo criativo no qual os envolvidos devem tomar uma série de decisões que afetarão profundamente o sistema de software e, frequentemente, impactarão atributos de qualidade. Lidar com *trade-offs* entre atributos que competem entre si e os impactos resultantes devem ser a base para as decisões tomadas.

Nas definições citadas é possível verificar que a necessidade de se tomar decisões arquiteturais está intrinsecamente ligada a própria definição de arquitetura e estilo arquitetural. Já aqui é possível verificar a necessidade de balancear custos e benefícios das decisões e restrições arquiteturais.

Tendo como estas afirmações como norte, nesta seção se pretende apresentar o estilo ReST, seus fundamentos bem como as características que o definem.

2.1.1 Histórico e panorama

O estilo arquitetural ReST foi um produto da tese escrita por Roy Fielding (2000) onde se apresentou os fundamentos sobre os quais as arquiteturas baseadas em rede, entre elas, a própria world wide web, foram construídas.

Para Richardson e Ruby (2007) o fato de ReST ter sido apresentado por Fielding não como uma arquitetura, mas como uma forma de avaliar arquiteturas, pode ter deixado questões em aberto sobre sua utilização na prática.

Para melhor o entendimento, compara-se o termo “ReSTFul” com “orientado a objeto”. Por exemplo, por mais que um determinado aplicativo tenha sido desenvolvido em uma linguagem “orientada a objetos” é possível escrever programas, nesta mesma linguagem, que não sejam verdadeiramente e essencialmente orientados a objeto.

Ainda segundo Richardson e Ruby (2007), de forma abstrata, o protocolo HTTP (que Fielding colaborou como co-escritor) se encaixa muito bem nos critérios definidos pelo estilo ReST. Entretanto, websites, aplicações e serviços web reais frequentemente “traem” estes princípios, que em última instancia, poderiam fazer com que os recursos da Web fossem utilizados de uma forma mais inteligente e eficiente.

Um dos estilos que surgiram em contraste e se difundiram no mercado no decorrer da década de 2000 foi o estilo RPC (Remote Procedure Call), onde algoritmos internos de determinada aplicação são expostos por meio de uma complexa interface que é diferente para cada serviço. Um dos protocolos mais populares baseados em RPC é SOAP (Simple Object Access Protocol), amplamente utilizado até os dias atuais.

Como demonstrado por Richardson e Ruby (2007) este estilo arquitetural de serviço não se mostra tão eficiente quanto um serviço ReSTFul de fato por, em última instância, fazer mal-uso das restrições propostas pelo estilo arquitetural. Como um simples exemplo pode se citar que a maioria das chamadas HTTP realizadas por um

serviço SOAP são realizadas por meio do método POST, mesmo que estritamente para leitura, por exemplo.

Apesar de se utilizar do estilo RPC, SOAP trafega (como toda comunicação web) sobre HTTP, que possui, entre os métodos passíveis de utilização, o método READ, ou GET, mais indicados no caso citado.

Sendo assim, o estilo ReST busca ser empregado durante o desenho de serviços para que eles estendam a comunicação que já acontece na web de fato, utilizando os recursos disponíveis de forma mais inteligente e eficiente.

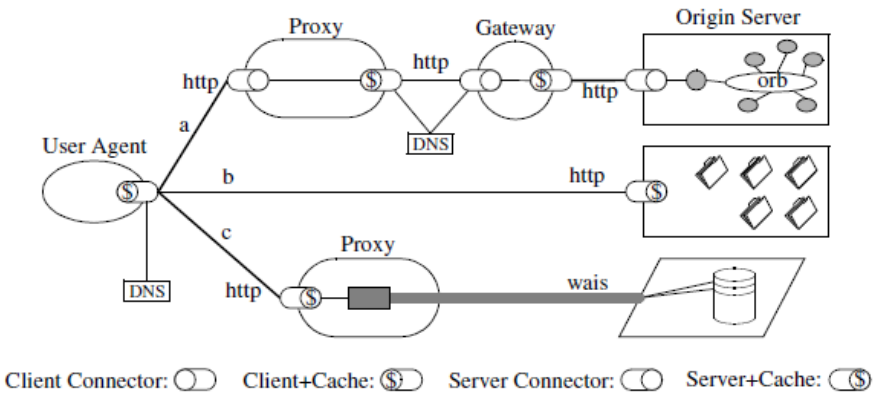
2.1.2 Características do estilo ReST

Sabendo que ReST é um estilo híbrido derivado de vários estilos arquiteturais baseados em rede e considerando a visão de uma arquitetura ReSTful conforme a figura 2, tem-se as seguintes características:

Client-Server: Este estilo hierárquico descreve que a comunicação é realizada entre cliente e servidor, sendo que o primeiro é um processo que dispara a comunicação e o segundo é um processo reativo.

Stateless: Descreve que cada requisição feita por clientes para o servidor deve conter toda a informação necessária para que seja entendida e processada, sem se valer de nenhum contexto armazenado no servidor. Por si só, favorece os atributos de confiabilidade e escalabilidade, mas desfavorece o desempenho.

Figura 02: Visão de processo de uma arquitetura ReSTful



Fonte: Fielding (2000)

Cache: Requer que dados dentro de uma resposta sejam implicitamente ou explicitamente caracterizados como passíveis ou não passíveis de cache no client-side. Favorece desempenho e escalabilidade, mas desfavorece confiabilidade.

Uniform Interface: Aplica à interface o princípio da generalidade, simplificando a arquitetura como um todo e melhorando a visibilidade das operações. Favorece interoperabilidade, mas desfavorece desempenho.

Layered System: Permite a formação de camadas hierárquicas, limitando a visualização e nível de acesso dos componentes. Favorece escalabilidade mas desfavorece desempenho.

Code on Demand: Permite estender funcionalidade efetuando download e executando código em forma de applets ou scripts. Reduz a visibilidade, mas melhora a extensibilidade.

Para que a arquitetura seja considerada como ReSTful, todas as características mencionadas devem ser observadas.

Como já citado, interações baseadas em HTTP puro (desconsiderando possíveis envelopes, como no protocolo SOAP), como as que acontecem nos browsers utilizados para navegação na World Wide Web, são por si só ReSTful.

Entretanto, com o passar do tempo, as interações ficaram mais próximas de conversações, pelo fato de os servidores passarem a prover serviços que são consumidos e processados por algoritmos no client-side.

Isto insere variáveis que podem deteriorar as características e os atributos de qualidade que deveriam ser preservados de antemão pelas características do estilo arquitetural ReST. Este problema é um dos motivadores para padronização de conversações ReST conforme será exposto no decorrer do trabalho.

2.2 Desenvolvimento dirigido por modelo

É sabido que modelagem é uma ferramenta chave em engenharia, uma vez que modelos são abstrações de um sistema e seu ambiente que permitem o endereçamento de preocupações em tempo de projeto de forma efetiva.

Partindo de uma definição mais elementar, segundo Mens (2006) define-se modelo como uma representação simplificada – ou uma descrição abstrata – de uma parte do mundo nomeada de sistema. Este modelo é útil para se obter um melhor entendimento do sistema. Em engenharia, é útil para tomada de decisões necessárias para alcançar e manter os objetivos do sistema.

O paradigma de desenvolvimento de software dirigido por modelo coloca os modelos em um mesmo patamar de importância que o próprio código-fonte, que não deixa de ser considerado um modelo por se tratar de uma abstração simplificada da linguagem de máquina, em detrimento de ser considerado como “mera” documentação.

De acordo com Stahl (2006) uma abordagem de desenvolvimento dirigida por modelo cria um grande potencial para automação da produção de software, que pode levar a um grande aumento de produtividade, que demonstra a grande aplicabilidade e relevância do paradigma.

A extensão do paradigma baseado em modelos também é muito abrangente. De acordo com OMG (2017) uma abordagem dirigida por modelo representa e suporta todo o range que envolve implementações de tecnologia, desde requisitos até modelos de negócio.

Nesta seção são apresentados conceitos de desenvolvimento dirigido por modelo (MDD) que serão utilizados em porções posteriores do trabalho, mesclando-

os com sua aplicabilidade no âmbito da interoperabilidade entre serviços e integração entre sistemas.

2.2.1 Definições de metamodelagem

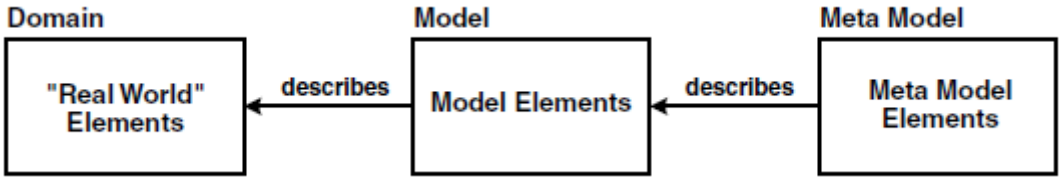
De acordo com Stahl (2006) metamodelagem é um dos principais aspectos ao tratar de desenvolvimento de software dirigido por modelo. Está inserida na disciplina de arquitetura do domínio, que é um dos primeiros passos para modelagem de sistemas de software. Seu conhecimento é necessário para lidar principalmente com os seguintes desafios:

- Construção de linguagens de modelagem específicas do domínio (DSL)
- Validação do modelo
- Transformação de modelo para modelo
- Geração de código
- Integração de ferramentas

Ainda de acordo com Stahl (2006), meta-modelos são modelos que fazem afirmações sobre modelagem, ou descrevem uma possível estrutura de modelos. Um meta-modelo define a sintaxe abstrata e semântica estática de uma linguagem de modelagem.

Entre modelos e meta-modelos existe uma relação classe-instância onde cada modelo é uma instância (instance-of) de um meta-modelo, e esta representação é possível visualizar na figura 3.

Figura 03: Relacionamento entre o mundo real, modelo e meta-modelo



.Fonte: Stahl (2006)

A princípio modelos podem ser descritos em uma linguagem de modelagem arbitrária, normalmente escolhida levando em conta a existência de ferramentas disponíveis, o que explica o fato de UML ser utilizada para modelagem em muitos casos.

A OMG (2017) define uma hierarquia com quatro meta-níveis de abstração para a metamodelagem, conforme se visualiza na figura 4. Nesta hierarquia aparece a definição de MOF (Meta Object Facility), uma linguagem de especificação de meta-modelos que provê uma plataforma aberta e independente para viabilizar o desenvolvimento e a interoperabilidade de sistemas dirigidos por modelo.

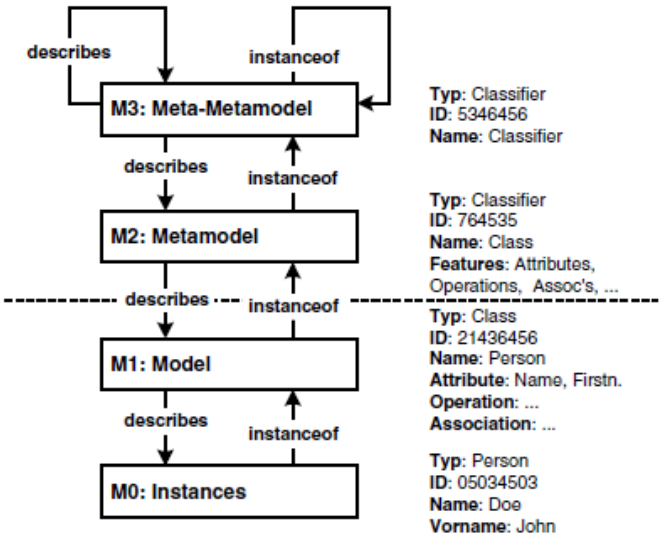
Figura 04: Hierarquia de modelos

M3 MOF	Defines a language for specifying a metamodel Example: MOF
M2 UML	Defines a language for specifying models Example: UML
M1 User Models	Defines a language that describe semantic domains Example: model of a problem domain
M0 Instance Models	Contains run-time instances of the model elements defined in a model

Fonte: OMG (2017)

Na figura 5 se visualiza um exemplo de meta-modelo criado por Stahl (2006) tendo como base a hierarquia citada na figura 4.

Figura 05: Quatro meta-níveis para modelos



Fonte: Stahl (2006)

No exemplo da figura 5, após a linha tracejada constam os níveis de abstração os quais desenvolvedores estão mais familiarizados.

Em M1 é definida uma classe. A esta classe se deu um nome, “Pessoa”, e um número de atributos, no caso, “Sobrenome” e “PrimeiroNome”.

As instâncias desta classe são criadas no nível M0, geralmente em tempo de execução. Na figura 5, a pessoa com ID interno 5034503 possui o atributo “Sobrenome” com valor “Doe” e atributo “PrimeiroNome” com valor “John”. Neste nível é possível perceber que uma classe pode possuir mais de uma instância.

No nível M2 são definidos os elementos construtivos utilizados no nível imediatamente inferior, M1, que por sua vez são instâncias dos elementos do meta-modelo definido no nível M2. Como são utilizadas classes no modelo em M1, o elemento construtor “Classe” deve ser definido em M2.

No nível mais superior M3 se define o meta-meta-modelo da OMG chamado de MOF (meta object facility) que serve para modelar linguagens no nível M2, como por exemplo, UML. A ideia é que UML não seja a única linguagem utilizada, mas que novas linguagens baseadas no MOF também possam ser utilizadas para modelagem. Interessante notar que como não há nível superior ao M3, o MOF se auto define.

Este exemplo, visualizado na figura 5, é interessante pois demonstra a ideia sobre a qual se desenvolveu o mecanismo que se pretende trabalhar nas seções posteriores do trabalho.

2.2.2 Transformação entre modelos

De acordo com Kleppe et al. (2003), transformação entre modelos é a geração automática de um modelo alvo a partir de um modelo fonte, de acordo com uma definição de transformação.

Uma definição de transformação é: um conjunto de regras de transformação que juntas descrevem como um modelo na linguagem fonte pode se transformar em um modelo na linguagem alvo.

Uma regra de transformação é a descrição de como uma ou mais construções na linguagem fonte podem se transformar em uma ou mais construções na linguagem alvo.

Kleppe et al. (2003) prossegue, distinguindo transformações feitas entre modelos na mesma linguagem chamadas de transformações exógenas ou traduções, e as transformações feitas entre modelos de linguagem diferentes, chamadas de transformações endógenas ou refraseamento.

No trabalho de Haupt (2015) são utilizadas ambas técnicas: de tradução, na transformação entre meta-modelos baseados em UML, e refraseamento, ao partir de um modelo de aplicação baseado em UML chegando ao código fonte, no caso em questão, baseado em JAVA.

Ambas as técnicas citadas podem ser classificadas como verticais, uma vez que os modelos fonte e alvo se encontram em camadas de abstração diferentes. No caso da tradução, se utiliza a técnica de geração de código fonte. Já no refraseamento, se utiliza técnica de refinamento formal.

Outros exemplos podem ser obtidos no guia MDA da OMG (2014), que frequentemente cita transformações entre diferentes modelos em diferentes níveis de abstração. Um cenário típico seria a transformação de um modelo independente de

plataforma (PIM) para um modelo específico para uma plataforma (PSM) antes da geração do código.

Sobre os mecanismos de transformação, Mens et al. (2006) efetua distinção entre declarativos e operacionais. Interpreta-se aqui o termo “mecanismos” como técnicas, linguagens, métodos para se especificar e aplicar e uma transformação. Neste caso recursos dos principais paradigmas de programação podem ser utilizados, como linguagens procedurais, orientadas a objeto, paradigmas lógicos ou funcionais, ou até mesmo uma abordagem híbrida combinando múltiplas opções.

Mecanismos declarativos focam no aspecto “o que”, no caso, “o que” precisa ser transformado em “o que”, definindo uma relação entre os modelos de origem e destino.

Mecanismos operacionais focam no aspecto “como”, no caso, “como” a transformação será realizada, especificando os passos que são requeridos para derivar os modelos alvo a partir dos modelos fonte.

Do ponto de vista de Mens et al. (2006) abordagens declarativas podem ser mais atrativas por oferecer serviços já agregados, como gerenciamento de rastreabilidade e bidirecionalidade automática. Também pode ser mais simples de ser compreendida por engenheiros de software.

Entretanto, modelos operacionais, como é o caso da gramática de grafos, podem ser necessários para implementar transformações em que abordagem declarativa falha em prover os serviços citados, especialmente para ordenar explicitamente a aplicação de um conjunto de transformações, graças a noção sequencial já embutida em sua construção.

2.2.3 MDA e Transformação de modelos no contexto de serviços ReST

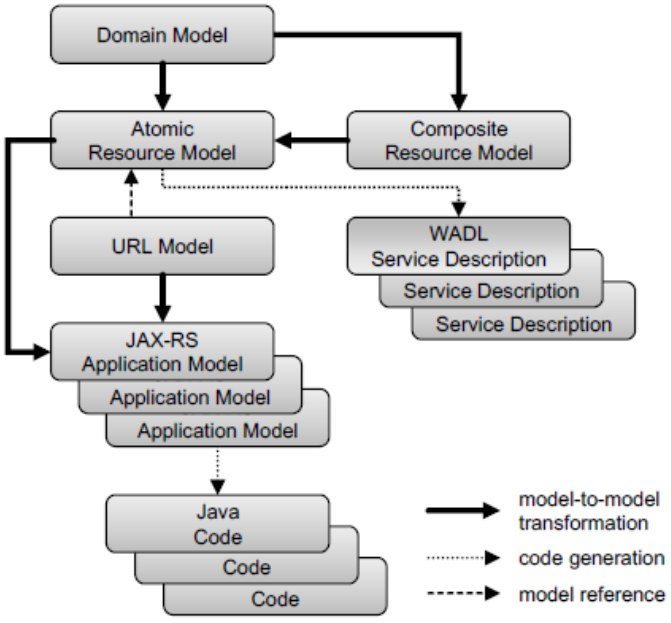
No trabalho de Haupt (2015), uma das fundações para esta monografia, a transformação entre modelos é utilizada como meio para atribuição de níveis de abstração e granularidade aos mesmos.

Para demonstrar a necessidade de transformações entre modelos, é importante introduzir, primeiramente, a ideia inicial da metamodelagem de serviços ReSTful.

Conforme descrito por Haupt (2015) modelar APIs ReST é uma tarefa desafiadora, uma vez que diversos APIs disponíveis no mercado demonstraram não-conformidades ao se avaliar se as restrições definidas pela arquitetura REST são observadas. O desrespeito as restrições geralmente implica na perda da qualidade e performance dos serviços.

Desta forma, para a abordagem sugerida, se definiu como ponto de partida os modelos de domínio. Este modelo é amplamente utilizado em engenharia de software e é a representação mais próxima da realidade, sendo também independente das restrições ReST. Este modelo é transformado então no modelo de recurso, onde algumas restrições já são aplicadas, como atribuição dos verbos HTTP para os métodos, por exemplo. As transformações seguem, passando pelos modelos de URL, WADL, Aplicação até culminar o código fonte, propriamente dito. O meta-modelo completo está demonstrado na figura 6.

Figura 06: Meta-modelo para desenho e criação de APIs ReST baseado em MDSD



Fonte: Haupt (2015)

Assim, as transformações se tornam parte essencial da arquitetura tanto no que concerne transformação entre modelos como também a transformação entre meta-modelos. Por exemplo, conforme diagrama, existe o meta-modelo centrado em interação e o meta-modelo centrado em conversação, descritos como Composite Resource Model e Atomic Resource Model.

Com este recurso é possível garantir a granularidade que se pretende, uma vez que uma conversação pode ser baseada em diversas interações. Desta forma, novas camadas de abstração que possibilitam maior controle e tomada de decisões podem ser adicionadas antes da geração do código em si.

De forma prática, no trabalho supracitado, são utilizadas técnicas baseadas em gramática de grafos para realizar a transformação entre meta-modelos, que é uma técnica bastante difundida baseada na definição de regras como especificações da transformação de estados, conforme descrito por Heckel (2006).

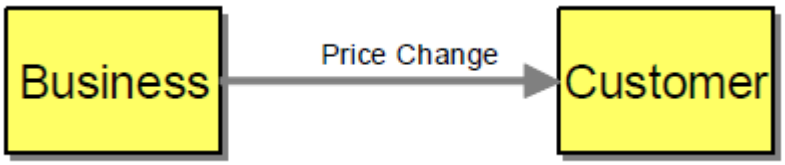
Em suma, o modelo de domínio é o ponto de partida para as transformações, e após as primeiras transformações é possível chegar em modelos intermediários que darão forma a API ReST, e assim por diante até chegar ao código fonte da API bem como à descrição do serviço.

3 REQUISITOS DE QUALIDADE PARA CONVERSAÇÕES REST

Conforme descrito na seção 2 do presente trabalho, as requisições e respostas em um cenário composto por troca de mensagens entre serviços foi, no decorrer do tempo, se tornando orientado a interações mais complexas, apresentando um padrão mais próximo do que se pode chamar uma conversação entre interlocutores.

De acordo com Hohpe (2008), conversação é a troca de mensagens individuais, mas relacionadas, ao longo do tempo. Conforme exemplificado na figura 7, uma conversação deve ocorrer entre dois ou mais participantes, que devem se comunicar por meio da troca de mensagens e que podem se associar ou relacionar com a conversação em si.

Figura 07: Simples interação entre cliente e servidor



Fonte: Hohpe (2008)

Conversações podem apresentar características diversas, entretanto, para que sejam eficientes e eficazes no contexto do estilo arquitetural ReST, atributos de qualidade devem ser observados. Nesta seção se pretende apresentar os atributos de qualidade, relacionando-os com as conversações ReST mais usuais, o que será a base para a estratégia que se busca construir.

3.1 Atributos de qualidade em desenvolvimento de software

Existe uma grande quantidade de definições sobre qualidade em desenvolvimento de software, expostos na literatura ao longo do tempo.

Para Crosby (1979) se trata de conformidade aos requisitos, enquanto para Humphrey (1989) é sobre atingir excelentes níveis de adequação ao uso.

Uma definição mais recente é dada pela ISO/IEC 25010 (2011) que define qualidade como a capacidade do produto de software de satisfazer necessidades explicitas e implícitas sob condições especificadas.

O IEEE (2014) determina qualidade como sendo a definição dos graus em que o produto de software preenche requisitos estabelecidos, dependendo diretamente de quanto estes requisitos representam de fato as necessidades, desejos e expectativas dos interessados.

Estas necessidades podem se enquadrar nas seguintes categorias conforme indicado por Bass et al. (2012):

a. Requisitos funcionais: definem o que o produto de software deve fazer, e como deve se portar ou reagir a determinados estímulos durante tempo de execução.

b. Requisitos de atributos de qualidade (ou requisitos não-funcionais): indicam os graus que devem ser apresentados para diversas propriedades alheias ao principal propósito do produto de software, mas vitais para seu bom funcionamento.

c. Restrições de design: decisões impostas ao time de desenvolvimento, como a utilização de determinado framework ou linguagem ou de programação.

Conforme descrito na norma ISO/IEC 42010 (2011), os requisitos de sistemas são definidos pelos interessados, que podem ser indivíduos, times, organizações que possuem algum interesse no sistema.

Para exemplificar a diversidade de papéis e interesses, de acordo com Bass et al. (2012) pode-se ter: o usuário do sistema, responsável pelas definições dos requisitos funcionais, em primeira instância, interessado na implementação destas funcionalidades; o desenvolvedor, responsável pelo desenvolvimento do sistema e interessado nas limitações e possibilidades de exploração decorrentes da tarefa de

desenvolvimento; o arquiteto, responsável pela arquitetura e interessado nas relações custo-benefício entre os atributos de qualidade concorrentes e abordagens de design.

Atributos de qualidade são definidos pelo IEEE (2014) como características que que afetam um item de qualidade. Conforme prescrito na literatura, estes atributos podem ser quantitativamente medidos ou quantitativamente analisados. Por exemplo, confiabilidade (Franco et al., 2012), performance (Koziolek, 2010) e manutenibilidade (Chidamber et al., 1994) podem ser quantitativamente medidos. Outros atributos que não podem ser objetivamente medidos, podem ser analisados por meio de guidelines ou checklists conforme indicado por Costa et al. (2016).

Já foram descritos inúmeros atributos de qualidade ao longo do tempo, neste trabalho o foco estará nos atributos descritos por Costa et al. (2016) como relevantes para serviços baseados no estilo de arquitetura ReST.

3.2 Padrões de conversação ReST

De acordo com Pautasso et al. (2016) a simplicidade e padronização do protocolo HTTP permitiu a adoção de conversações triviais, que podem se limitar a interações únicas.

Entretanto, na implementação de serviços ReSTful, notou-se a necessidade de se combinar diversas interações como parte de uma conversação, a fim de endereçar importantes requisitos não-funcionais.

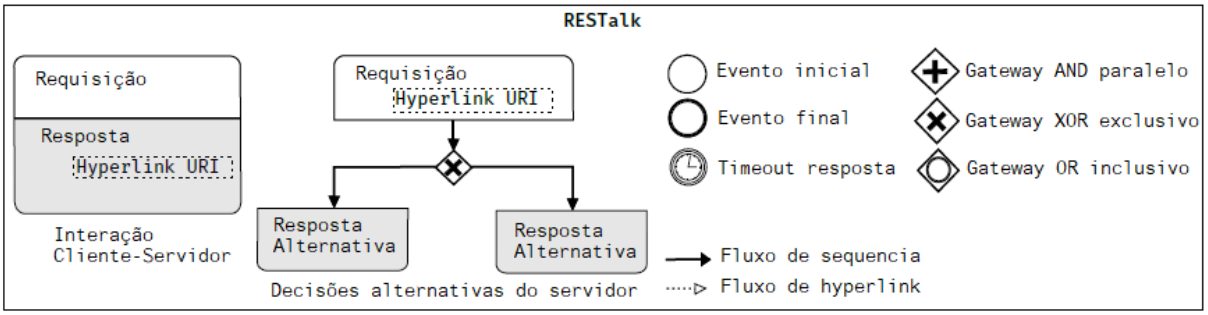
No decorrer do tempo, foi possível identificar padrões de conversação e classifica-los, com a finalidade de permitir o reuso e abstrair a complexidade de conversações que podem se tornar longas.

Sendo assim, Pautasso et al. (2016) classificou as conversações ReST em grupos, considerando o ciclo de vida dos recursos, a saber: criação, navegação, edição e proteção, em tradução livre.

Estes padrões de conversação, conforme Pautasso et al. (2016), foram apresentados por meio de uma linguagem visual de modelagem, que será também

utilizada no presente trabalho para que os padrões sejam expostos de uma forma mais clara. A figura 8 expõe os elementos construtivos desta linguagem.

Figura 08: Elementos construtivos ReSTalk.



Fonte: Pautasso (2016).

A importância de conhecer estes padrões reside em extrair quais atributos de qualidade são privilegiados e quais são degradados em cada variante. Isto será o elemento direcionador para as transformações de meta-modelos adiante expostas.

3.2.1 Padrões para criação de recursos

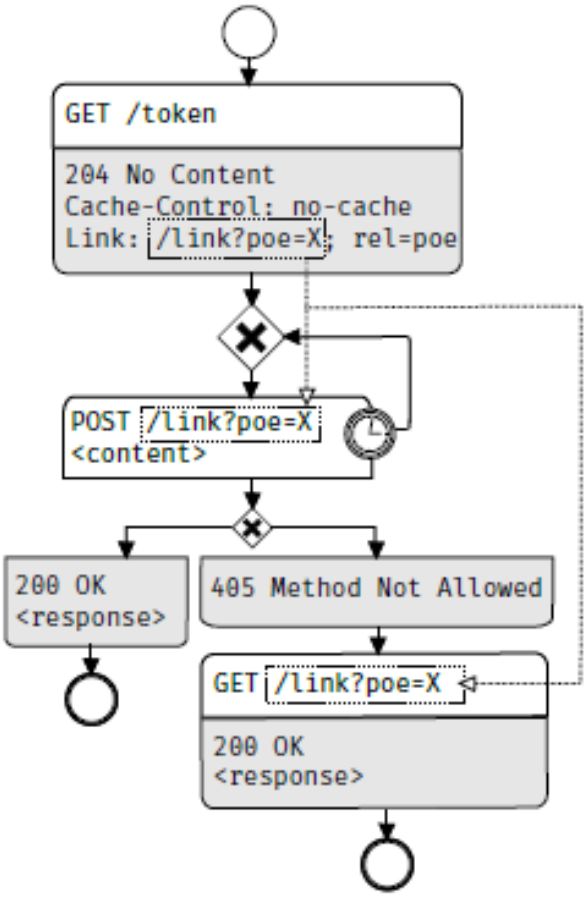
Enquanto o protocolo HTTP oferece os métodos POST e PUT para criação de recursos em uma única interação de requisição-resposta, os padrões de conversação para criação apresentados por Pautasso et al. (2016) lidam com a criação de recursos sob algumas limitações ou em cenários passíveis de falhas.

3.2.1.1 POST único

O padrão POST único pode ser utilizado para evitar o processamento de requisições duplicadas pelo servidor. Isto porque o cliente, ao solicitar a criação de um recurso, pode não receber uma resposta do servidor, e pode também não saber se o servidor recebeu sua requisição.

Isto pode ocorrer devido à falta de confiabilidade da rede. Caso o cliente se utilizasse de um método idempotente como GET, PUT ou DELETE, este problema não ocorreria. Porém um recurso pode e é muito comumente criado a partir do método POST, onde o URI é definido pelo servidor. Neste caso é necessário evitar a criação repetida de mesmo recurso sem necessidade.

Figura 09: POST único



Fonte: Pautasso (2016)

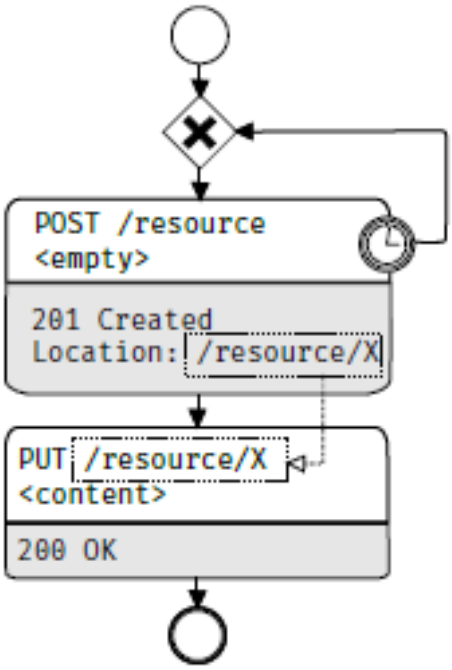
O fluxo indicado na figura 09 indica a utilização de um token, por exemplo, um único URI alvo, obtido a partir de um recurso disponibilizado pelo servidor. Como este URI único é utilizado ao efetuar a requisição POST, o servidor pode checar se o recurso correspondente já existe. O recurso só poderá ser criado se este URI não tiver sido utilizado previamente em outra requisição, caso contrário o servidor deverá responder que a ação requisitada não é permitida.

Como benefício, se evita a criação de recursos duplicados, aumentando a confiabilidade na conversação.

3.2.1.2 Criação POST-PUT

Nesta variante, conforme consta na figura 10, o problema a ser endereçado é o mesmo que o indicado no tópico anterior. Para utilização desta variante deve-se distinguir entre a criação técnica de do identificador de um recurso e a execução do comportamento de criação no domínio da aplicação.

Figura 10: Criação POST-PUT



Fonte: Pautasso (2016)

O cliente envia a primeira requisição POST vazia que resulta na criação de um recurso vazio ocasionando nenhum efeito colateral relevante no domínio da aplicação. A resposta do servidor contém um link para o URI do recurso vazio criado ao qual o cliente poderá adicionar conteúdo relevante para o domínio da aplicação usando uma requisição PUT. Esta primeira requisição PUT irá então disparar consequências no

domínio. Uma vez que o comando PUT é idempotente, repetir a requisição não acarretará impactos.

A utilização deste padrão resulta em uma simplificada destruição de recursos não utilizados e inicialização idempotente, além de um aumento na confiabilidade.

3.2.1.3 Operação de execução longa com consulta

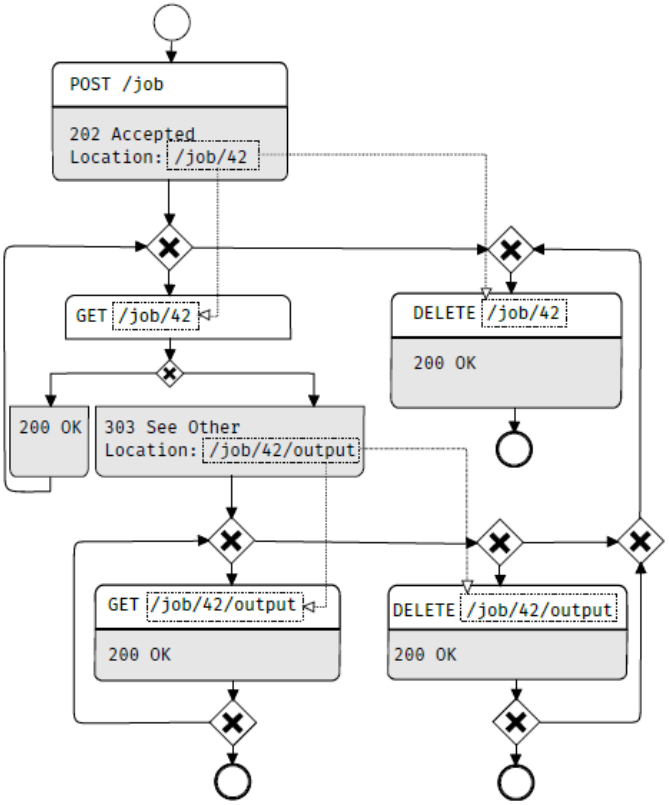
Este tipo de operação se utiliza do recurso de “polling” com a finalidade de evitar timeouts como resultado durante a espera de operações de execução longa que pode ocorrer em casos de intensivo processamento de dados ou em operações com dados complexos.

Devido à falta de confiabilidade das redes, o cliente pode perder a conexão antes de que o servidor tenha completado o processamento do resultado. As chances de que o cliente esteja indisponível no momento da resposta também aumentam com a demora no processamento pelo servidor. Caso o cliente reenvie a requisição por falta de resposta, o tempo de processamento pode aumentar ainda mais. Isto leva a conclusão que a execução de computação e entrega de seus resultados são duas preocupações que trazem demandas completamente diferentes na infraestrutura do servidor.

Para solucionar estes problemas, a própria requisição para uma operação de longa duração cria um recurso, obtendo uma resposta imediata informando onde o cliente pode encontrar os resultados. Este recurso fica disponível até que a operação solicitada, que está executando em segundo plano, tenha sido finalizada.

O cliente pode então utilizar a operação GET em conjunto com o recurso criado, a fim de ser informado sobre o progresso de sua solicitação e eventualmente ser redirecionado para outro recurso que representará o resultado, quando a operação de longa duração for finalizada. Esta operação pode ser repetida múltiplas vezes, uma vez que o recurso que controla a operação de longa duração possui sua própria URI. A operação também pode ser cancelada ao se enviar a operação DELETE para a URI de controle, implicando em parar a operação que está sendo executada em segundo plano no servidor, conforme possível visualizar na figura 11.

Figura 11: Criação com operação de longa duração



Fonte: Pautasso (2016)

Esta variante favorece a escalabilidade, uma vez que o cliente não precisa manter a conexão ativa com o servidor por toda a duração da requisição. Isto pode favorecer a quantidade simultâneas de clientes concorrentes que fazem requisições ao servidor.

Nesta situação os resultados também podem são compartilhados entre diferentes clientes, sem a necessidade de interação com o servidor, favorecendo a descoberta. A requisição também pode ser cancelada em qualquer momento o que evita processamento desnecessário, favorecendo a performance.

Por outro lado, se faz necessário que o cliente efetue consultas periódicas, chamadas de “polling”, que podem onerar a rede se implementadas de forma irresponsável. Isto pode ser evitado se o servidor informar o progresso do processamento ao cliente, em cada consulta realizada.

A segurança pode ser comprometida uma vez que o resultado das requisições pode ser compartilhado entre diversos clientes. Neste caso, ainda de acordo com Pautasso (2016) se aconselha a utilização do padrão de conversação “Autenticação básica de recurso” (seção 3.2.4.1).

3.2.2 Padrões para navegação entre recursos

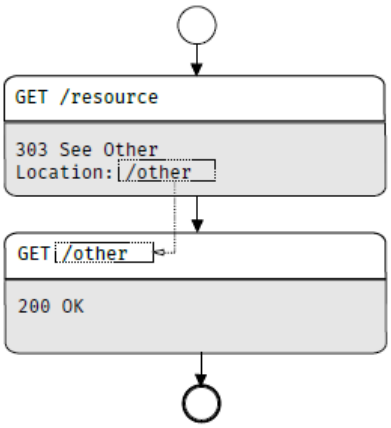
Conforme explica Pautasso (2016), a restrição ReST HATEOAS (Hypermedia As The Engine Of Application State), favorece o design de APIs a partir de um único URI como ponto de entrada e a descoberta dinâmica de recursos a partir de hypermedia. Porém este ponto de entrada pode não levar diretamente ao recurso necessário pelo cliente devido a diversos motivos como: falta de acesso, mudança de localidade do recurso ou pelo fato de um recurso fazer parte de uma coleção.

Os padrões de conversação descritos por Pautasso (2016) constantes nesta seção podem ajudar no processo de descoberta de recursos.

3.2.2.1 Redirecionamento server-side com códigos de status

Clientes podem sofrer uma vez que os serviços evoluem ao longo do tempo e a localidade dos recursos pode mudar. Para que o cliente não tenha sua requisição rejeitada, o servidor deve possibilitar o acesso ao recurso com o URI origina mesmo em situações onde houve evolução do serviço. Isto deve ser feito tendo em conta as restrições ReST que fazem com que o servidor não rastreie seus clientes, nem inicie conversação com os mesmos. Sendo assim, em casos de mudança, é necessário informar ao cliente sobre a nova localidade do recurso no momento em que a requisição está sendo enviada.

Figura 12: Redirecionamento server side com códigos de status



Fonte: Pautasso (2016)

Caso cliente faça uma requisição para um recurso com um URI desatualizado, o servidor responde com um código de status de redirecionamento 3xx, geralmente combinado com um cabeçalho “Location” guiando o cliente para o novo URI. O cliente se torna responsável pela utilização desta URI dependendo da especificação do código de retorno, conforme fluxo da figura 12.

Isto favorece o desacoplamento pois os servidores podem evoluir seus serviços independentemente dos clientes, que por sua vez não tem suas requisições invalidadas por motivo de mudança de URI. A performance também pode ser favorecida se os clientes forem redirecionados para servidores mais próximos, diminuindo a latência.

Por outro lado, a utilização desta variante pode implicar em maior complexidade no lado do cliente, uma vez que se torna necessário entender e reagir propriamente aos códigos de retorno de redirecionamento. Também necessário observar que pode haver um aumento no número de requisições para alcançar determinado recurso, em detrimento a efetuar uma única requisição para acessar diretamente um recurso já conhecido.

3.2.2.2 Navegação client-side seguindo hyperlinks

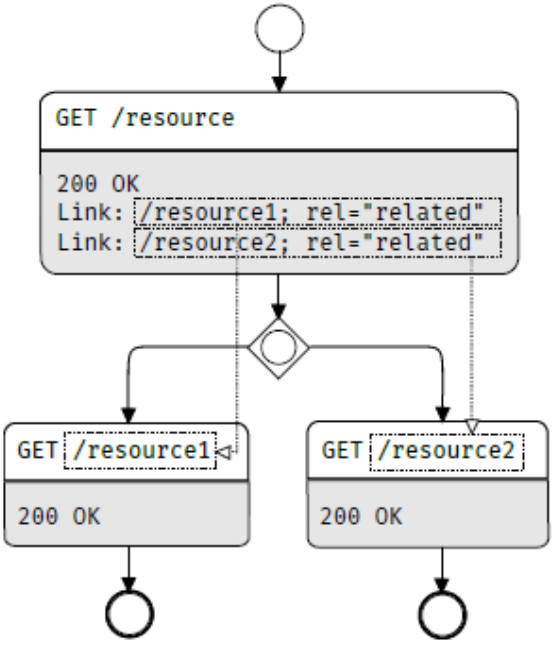
Esta variante pode ser utilizada em casos onde as requisições de cliente se relacionam com múltiplos diferentes recursos e se faz necessário conhecer as opções disponíveis ou informações relacionadas, sem se utilizar de “hard-code” para os padrões de URI dos serviços, que violaria a restrição ReST HATEOAS.

Isto é relevante devido a necessidade do cliente de descobrir recursos relacionados para mudar o estado da aplicação durante a navegação, situação em que o acoplamento entre cliente e servidor se torna mais fraco se o cliente for guiado pelo servidor em sua navegação.

Neste caso o servidor decide quais alternativas são aplicáveis para mudança do estado da aplicação, para cada requisição feita pelo cliente, bem como que outros recursos podem ser relevantes no contexto da requisição.

Para tanto, o servidor provê todos os hiperlinks relacionados com o recurso requisitado de forma que o cliente possa decidir seguir um ou mais, conforme figura 13.

Figura 13: Navegação client-side seguindo hyperlinks



Fonte: Pautasso (2016)

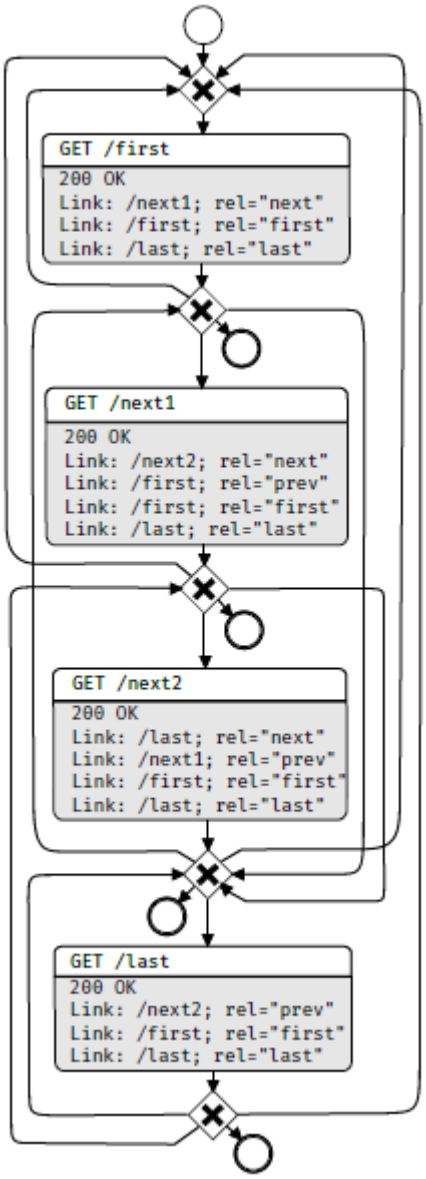
O desacoplamento é favorecido, porém a complexidade no lado do cliente pode aumentar, bem como pode ocorrer depreciação da performance, semelhantemente a variante apresentada na seção 3.2.2.1,

3.2.2.3 Coleção transversal incremental

É comum, para descobrimento de recursos, agrupa-los em coleções que, ao acessadas pelos clientes, podem dar acesso a todos os seus itens. Porém estas listas de recursos podem ser muito grandes e o cliente pode ter a necessidade de encontrar um recurso específico, conhecendo apenas o URI da coleção.

Para endereçar esta questão, quando o cliente requisita o primeiro item de uma coleção, o servidor provê links para o próximo item e para o último item da coleção. Em cada resposta para requisições GET subsequentes, torna possível o cliente escolher se quer seguir o primeiro, o anterior, o próximo ou o último item. Isto permite que a coleção seja gradualmente descoberta ao seguir os links enviados pelo servidor, conforme fluxo da figura 14.

Figura 14: Coleção transversal incremental



Fonte: Pautasso (2016)

A utilização desta variante favorece a otimização na utilização de banda, caso seja necessário consumir determinados recursos de uma coleção numerosa, evitando o download de grandes respostas. Porém pode também onerar caso seja necessário se obter a listagem de recursos completa.

Uma outra preocupação pode ser a possibilidade de mudanças concorrentes em uma coleção, que pode não se refletir nas respostas obtidas. Para endereçar esta questão, pode se pensar na adoção de algum tipo cache.

3.2.3 Padrões para edição de recursos

A Web “apenas leitura” já não é mais realidade e a edição de recursos se tornou uma operação comum. A seguir padrões especificados por Pautasso (2016) para executar este tipo de operação.

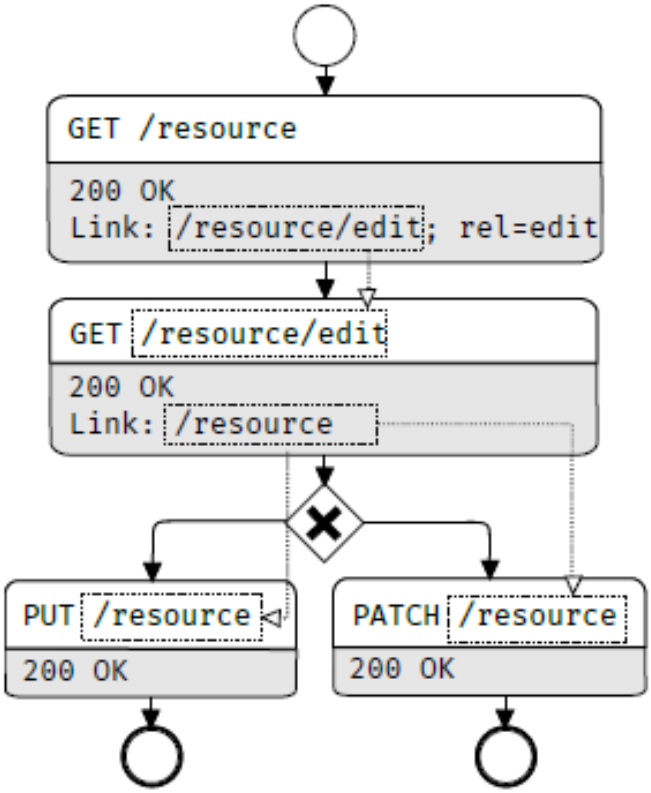
3.2.3.1 Edição de recurso (parcial)

Os clientes podem precisar modificar o estado de um recurso específico, mas não tem conhecimento de como organizar a informação na mensagem de atualização. Em alguns casos, a operação PUT pode ser utilizada de forma eficaz, porém em outros casos o cliente tenha a necessidade de saber quais elementos de um recurso são editáveis ou quais valores são válidos para um elemento de dados específico.

Para que o cliente saiba quais recursos e elementos são editáveis, pode necessitar de uma representação, por exemplo, um formulário HTML contendo recursos e seus atributos. Em certos casos operação pode se tornar muito custosa, ao se percorrer todos os elementos de todos os recursos, quando apenas uma pequena parte precisa ser atualizada.

Para endereçar esta necessidade, o servidor pode prover um hyperlink para uma página com um formulário representando todo o conteúdo editável, quando responder a uma requisição GET para um determinado recurso. Assim o cliente pode decidir editar determinado conteúdo utilizando uma requisição PUT, conforme fluxo da figura 15.

Figura 15: Edição parcial de recurso



Fonte: Pautasso (2016)

3.2.3.2 Atualização condicional para grandes recursos

Em casos onde a atualização de recursos requer uma grande quantidade de dados, que pode ser muito grande para que o servidor processe de uma só vez, algumas informações passadas no header de uma requisição preliminar podem disparar um fluxo de controle, onde o cliente é informado sobre a capacidade de processamento da requisição pretendida.

O envio de arquivos muito grandes pode ser custoso do ponto de vista da largura de banda bem como tempo de resposta, o que pode levar a um erro de rede ou conexão.

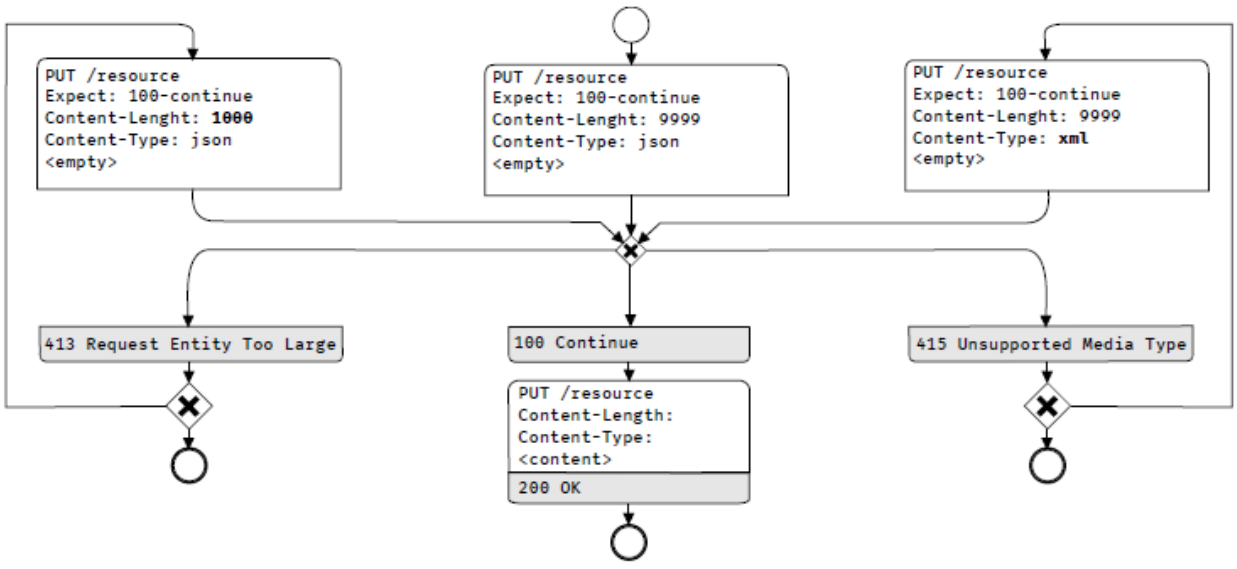
Se o cliente não está certo sobre a capacidade imposta pelo servidor, sobre os tipos de mídia apropriados, ou até mesmo sobre requisitos de autorização, enviar uma requisição muito grande que eventualmente será rejeitada representa desperdício de recursos.

Para solucionar o problema, antes de enviar as informações para atualização de forma definitiva, o cliente deve enviar uma requisição vazia com os parâmetros de cabeçalho “Expect” e “Content-Length”, ou “Content-Type” ou “Accept”, que serão utilizados pelo servidor como controle para que se indique quão apropriada é a requisição.

Se for apropriada, o servidor responde com código 100 e então o cliente pode repetir a requisição com os mesmos parâmetros, com exceção do “Expect”, e adicionando o conteúdo real.

Caso contrário, o servidor irá responder com o código 400, fazendo com que o cliente elabore uma nova estratégia de envio ou encerre a conversação, conforme figura 16.

Figura 16: Atualização condicional para grandes recursos



Fonte: Pautasso (2016)

A utilização deste padrão de conversação aumenta a eficiência, uma vez que evita o desperdício de recursos em requisições que serão rejeitadas. Por outro lado, aumenta tempo de resposta pois adiciona uma rodada antes da requisição de atualização em si.

3.2.4 Padrões para segurança de recursos

Dependendo do recurso, pode ser necessário restringir algumas ou todas ações a determinados grupos de clientes, por exemplo, ações do tipo CRUD (create, read, update, delete). Seguem opções para obtenção de permissões em recursos, segundo Pautasso (2016).

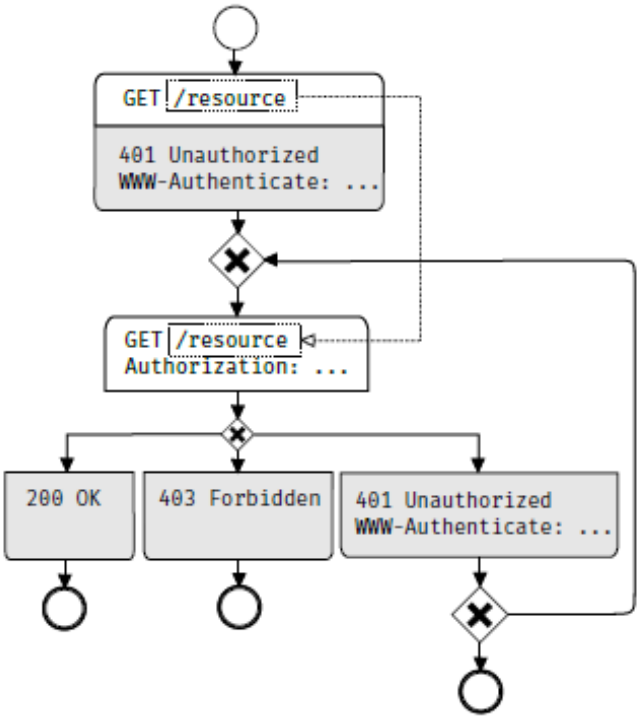
3.2.4.1 Autenticação básica de recurso

Para que acesso a recursos sejam fornecidos apenas a clientes autenticados, se faz necessária a implementação de autenticação HTTP básica. Desta forma o servidor precisa informar o cliente se determinado recurso necessita ou não de autenticação.

Assim, quando o cliente efetua uma requisição de um determinado recurso para o servidor sem as credenciais necessárias, o servidor deve retornar um erro 401 informando o cliente que o recurso necessita de autenticação.

Na próxima requisição o cliente deve informar as credenciais válidas para ter o acesso garantido, sendo este o caso, retorna código 200. Caso informe credenciais inválidas, o servidor responde com outro erro 401. Se o recurso não mais existir por ter sido eliminado por outro cliente, por exemplo, o servidor responde com erro 403, conforme figura 17. Este padrão de conversação pode ser aplicado a qualquer verbo HTTP.

Figura 17: Autenticação básica de recurso



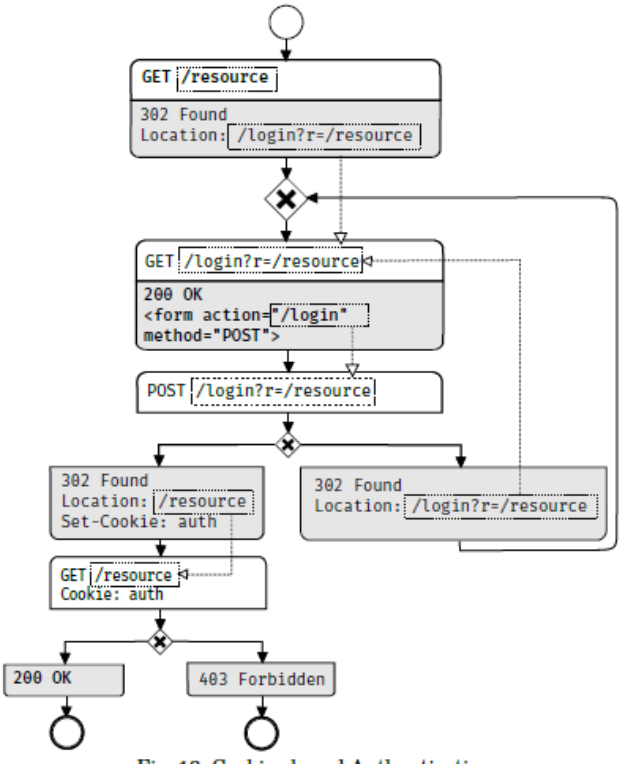
Fonte: Pautasso (2016)

Se tem como benefício a simplicidade de aplicação do padrão. Por outro lado, a segurança pode ser um ponto de fragilidade porque as informações de autenticação são codificadas, não criptografadas. Neste caso se recomenda a utilização do padrão apenas sobre HTTPS. Além disso, neste padrão não existe um mecanismo explicito para log out.

3.2.4.2 Autenticação baseada em cookies

A necessidade especificada na seção 3.2.4.1, também é a base para o padrão em questão. Nesta variante, ao efetuar uma requisição por um recurso, o cliente é redirecionado para uma página de login com o formulário que deve ser preenchido pelo cliente. Se as informações forem válidas, o cliente é redirecionado para o recurso inicialmente solicitado e um *cookie* é validado para ser utilizado em futuras requisições, caso contrário é redirecionado de volta para a página de login, conforme figura 18.

Figura 18: Autenticação baseada em cookies



Fonte: Pautasso (2016)

Nesta variante é possível implementar um tempo de expiração para a autenticação, apenas indicando o cookie como inválido.

Também neste caso é indicada a utilização apena sobre HTTPS. A eficiência pode ficar prejudicada em solicitações diferentes de GET, DELETE ou POST vazio, pois o corpo da requisição se perde após o primeiro redirecionamento.

3.3 Atributos de qualidade em conversações ReST

No trabalho de Costa et al. (2016) foram listados atributos de qualidade que se mostram relevantes no contexto das conversações ReSTful.

A seguir consta a lista dos principais atributos de qualidade que se mostram relevantes para o assunto, bem como as questões de design dos serviços derivadas de possíveis cenários onde os atributos são colocados em prova.

- Interoperabilidade:

Quais tipos de consumidores de serviços irão interagir com os serviços ReST?

O serviço ReST utiliza código por demanda?

As representações dos recursos são padronizadas dentro da aplicação?

Qual formato é utilizado para representar os recursos?

Existe um vocabulário padrão definido para os recursos?

- Confiabilidade:

De quais componentes externos depende o serviço ReST?

Como os serviços são empacotados e entregues?

Qual a abordagem para versionamento dos recursos?

Qual o modelo de domínio da aplicação?

Quais dados serão expostos como recursos?

Como são definidos os URIs dos recursos?

- Segurança

Alguma informação confidencial é exposta como recurso?

O recurso pode se comunicar com Open APIs?

O serviço ReST utiliza código por demanda?

Se considerou a classificação das informações na fase de design?

Quais tipos de consumidores de serviços irão interagir com os serviços ReST?

Quais são os mecanismos de segurança para os consumidores executarem ações nos recursos?

- Testabilidade

    Como os consumidores do serviço podem executar testes nos serviços?

- Performance

    As representações de recursos são padronizadas?

    De quais componentes externos depende o serviço ReST?

    Paginação em recursos é necessária?

    É possível incluir o subconjunto de um atributo na URI?

    O serviço consumidor utiliza um mecanismo cache?

    Existe uma replicação do serviço ReST em tempo de execução?

- Disponibilidade

    De quais componentes externos depende o serviço ReST?

    Como proteger o servidor web de sobrecarga de requisições?

- Manutenibilidade

    As representações de recursos são padronizadas?

    Como os serviços são empacotados e entregues?

    Como são definidos os URIs dos recursos?

    Qual a abordagem para versionamento dos recursos?

- Segurança

    Como são mapeadas as operações em recursos para verbos HTTP?

- Descobertabilidade

Como os recursos são documentados no registro?

Como são definidos os URIs dos recursos?

Como são mapeadas as operações em recursos para verbos HTTP?

Existe um vocabulário padrão definido para os recursos?

- Funcionalidade

A resposta para a criação de um recurso contém a localização do mesmo?

A paginação em recurso é necessária?

É possível incluir o subconjunto de um atributo na URI?

Como são definidos os URIs dos recursos?

Como são mapeadas as operações em recursos para verbos HTTP?

O que são os códigos de status HTTP nas respostas?

O recurso pode se comunicar com Open APIs?

Já foi possível ter contato com alguns destes atributos no decorrer das outras seções, principalmente as que apresentam os padrões de conversação. Também já se destacou que cada padrão previamente apresentado favorece certos atributos de qualidade em detrimento de outros.

No trabalho de Costa et al. (2016) a lista foi utilizada como parte de um mecanismo para avaliar serviços que pretendem se utilizar do estilo ReST.

Aqui, a intenção de se listar os atributos e apresentar as questões de design derivadas é demonstrar a complexidade para se manter compatível ao estilo arquitetural e que os atributos de qualidade podem se tornar importantes direcionadores para a transformação de meta-modelos, aplicável em tipos ou padrões de conversação onde mais de uma transformação é possível, como se discutirá na próxima seção.

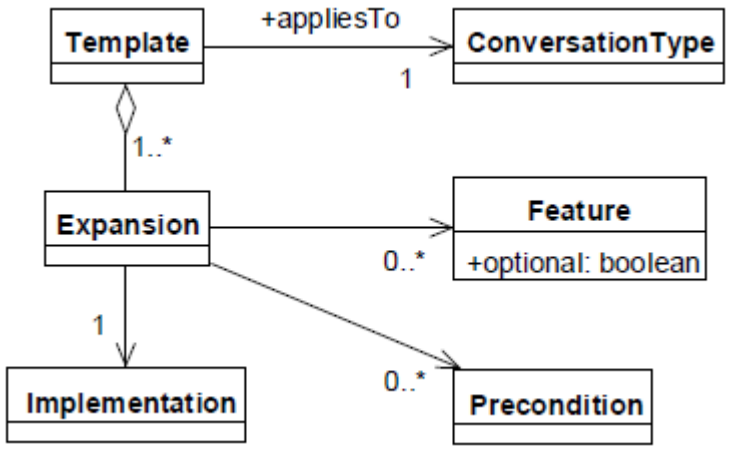
4 ESTRATÉGIA DE DIRECIONADOR BASEADO EM QUALIDADE

No trabalho de Haupt et al. (2015) onde apresenta-se a ideia de meta-modelo centrado em conversações, ressalta-se que ao transformar este meta-modelo em um centrado em interações, um importante aspecto é a observação de que tipicamente múltiplas transformações são possíveis para a mesma conversação.

Para tanto, se introduziu o conceito de *templates*, como meio de descrever a *expansão*, termo utilizado para transformação do meta-modelo centrado em interações para o meta-modelo centrado em interações.

O *template*, conforme figura 19, é um container para um conjunto de expansões que se aplicam ao mesmo tipo ou padrão de conversação. Um padrão contém tudo o que é necessário para transformar um tipo de conversação modelado em alto nível em um conjunto de recursos e interações que realizam esta conversação.

Figura 19: Meta-Modelo de template de expansão



Fonte: Haupt (2015)

A expansão é realizada por uma implementação, que pode ser um pedaço de código, um stylesheet XSLT, regras de gramática de grafos ou qualquer outro artefato que implementa a transformação de modelos.

A expansão também anexa informações adicionais à uma implementação, nomeadas de características e pré-condições. Estas características e pré-condições são fatores essenciais para escolha de determinada extensão.

Entretanto, neste meta-modelo, dois pontos não ficam evidenciados: o objetivo que o tipo de conversação busca implementar e qual o peso que atributos de qualidade não funcionais possuem no contexto do tipo de conversação em questão.

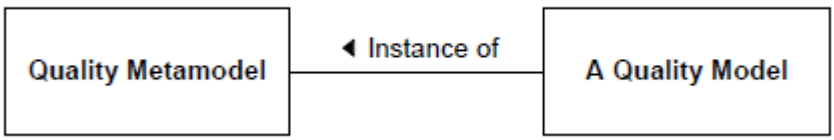
Do ponto de vista de modelagem no contexto de desenvolvimento dirigido por modelo, estes dois pontos são cruciais para que a automatização na transformação de modelos cumpra o papel de forma completa e eficiente. Nas seções seguintes a se pretende minuciar a estratégia de expansão dos meta-modelos até então apresentados.

4.1 Meta-modelo de qualidade

Uma vez que o presente trabalho visa endereçar a questão da qualidade no contexto REST dirigido por modelo, propõe-se então a agregação de um modelo de qualidade aos elementos já existentes. O modelo de qualidade proposto por Wagner et al. (2012) mostra-se relevante para esta finalidade no contexto indicado.

Sendo assim, Wagner et al. (2012), define modelo de qualidade como algo que especifica o significado de qualidade para um produto de software, conforme figura 20. O meta-modelo de qualidade (chamado de Quamoco, no trabalho citado) define as regras que o modelo de qualidade precisa seguir.

Figura 20: Meta-modelo de qualidade Quamoco



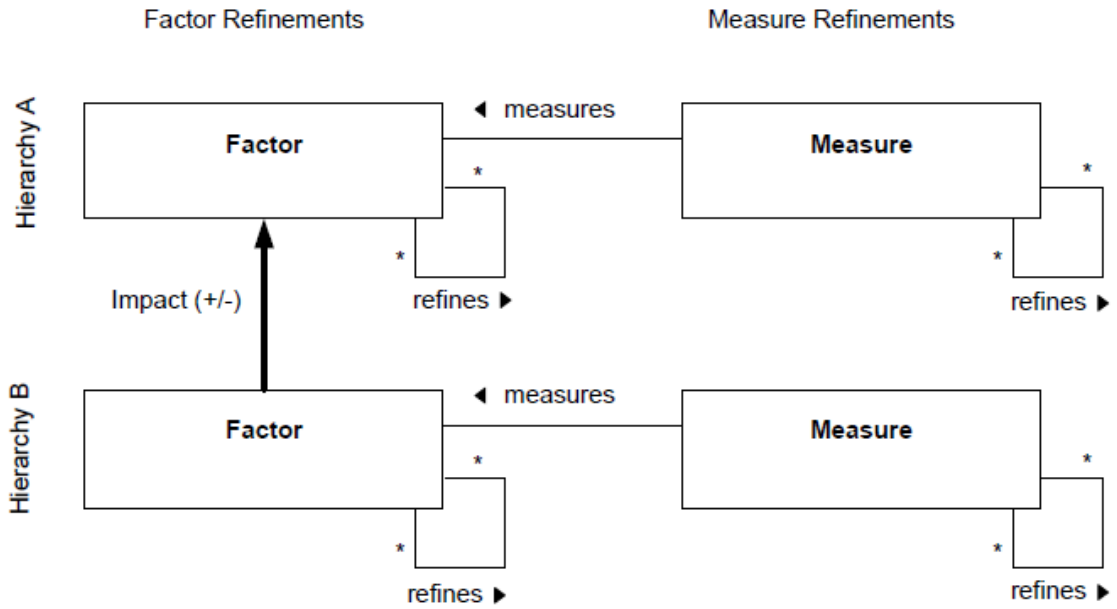
Fonte: Wagner (2012)

O meta-modelo de qualidade é definido em 2 níveis: o nível de definição e o nível de aplicação. O primeiro trata dos conceitos que definem a qualidade do produto e como acessá-los, o segundo especifica como armazenar os dados e parâmetros relacionados com os atributos de qualidade. No presente trabalho iremos nos ater ao nível de definição, por ser o nível que agrega os elementos necessários no presente contexto.

4.1.1 Nível de definição do meta-modelo de qualidade

De acordo com Wagner et al. (2012) a figura 21 descreve conceitos e relacionamentos providos pelo modelo de qualidade e seu ponto de partida são os Fatores de Qualidade. Estes fatores podem ser especificados por outros fatores e, de forma hierárquica, é possível definir o impacto de determinado fator em outro. Uma Medição também pode ser atrelada ao fator, a fim de que se atribua valores ao mesmo.

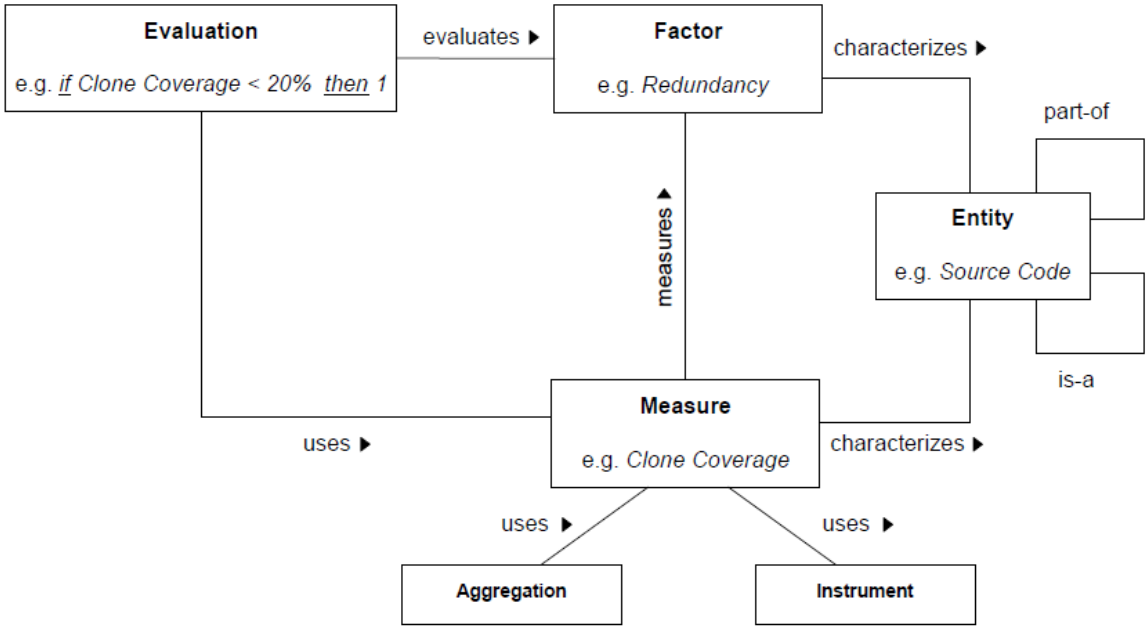
Figura 21: Nível de definição do meta-modelo de qualidade



Fonte: Wagner (2012)

Em uma visão adicional (figura 22), é possível verificar que outros elementos são agregados ao meta-modelo. Os fatores e suas medidas caracterizam uma Entidade, que pode ser um produto de software ou partes do mesmo. O elemento Avaliação utiliza medidas para disponibilizar informações quantitativas sobre os fatores.

Figura 22: Nível de definição do meta-modelo de qualidade II



Fonte: Wagner (2012)

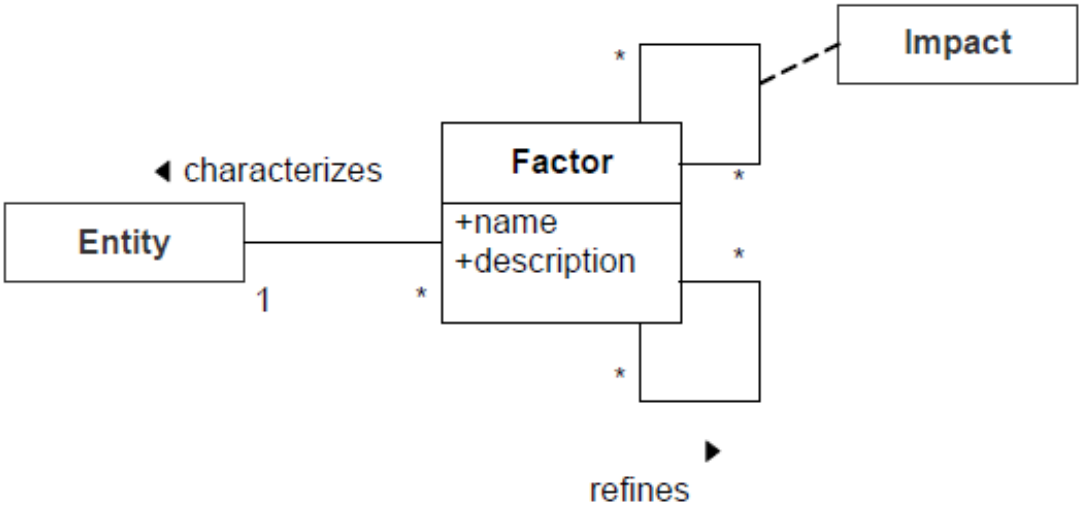
Uma vez que os modelos de qualidade podem focar em diferentes pontos de vista do que se define como qualidade de um produto, muitas hierarquias são possíveis, como as definidas pela norma ISO 2500, por exemplo.

4.1.1.1 Fatores de qualidade

De acordo com Wagner et al. (2012), fatores de qualidade são os blocos básicos de construção dos modelos de qualidade. Conforme já citado, os fatores

constituem propriedades do produto de software que se relacionam com sua qualidade. O meta-modelo é apresentado na figura 23.

Figura 23: Meta-modelo para fatores de qualidade



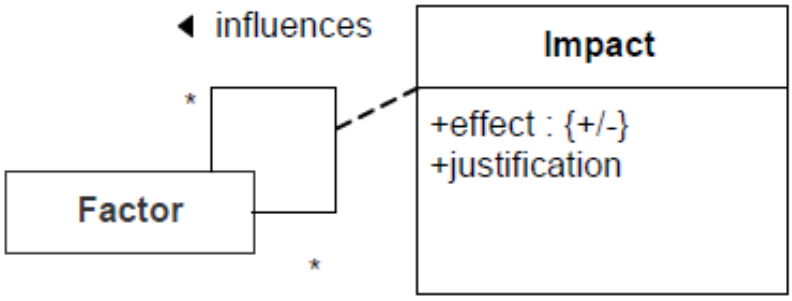
Fonte: Wagner (2012)

Um fator é sempre definido de forma que se possa determinar o grau em que está presente no produto, e pode ser decomposto em outros subfatores por meio da relação “Refina”. Sua utilização resulta em estruturas em árvore.

4.1.1.2 Impactos

Um ponto importante é a influência que um fator de qualidade pode exercer sobre outro fator. Esta influência pode ser descrita por meio da relação “Impacta”, que pode ser tanto negativa quanto positiva. Por exemplo, “Redundância” pode impactar negativamente “Manutenibilidade”. É possível visualizar esta representação no meta-modelo pela figura 24.

Figura 24: Impacto em fatores de qualidade



Fonte: Wagner (2012)

Se o impacto possui efeito positivo, o grau em que o produto possui o fator alvo é melhorado se o produto possui o fator fonte. Se o impacto for negativo, o grau em que o fator alvo é possuído pelo produto diminui se o produto possui o fator fonte.

4.2 Fatores de qualidade como direcionadores de transformações

Propõe-se a criação de um novo meta-modelo para que se introduza os atributos de qualidade não-funcionais como direcionadores de escolha da expansão entre os meta-modelos centrados em conversação. Este meta-modelo é apresentado na figura 24.

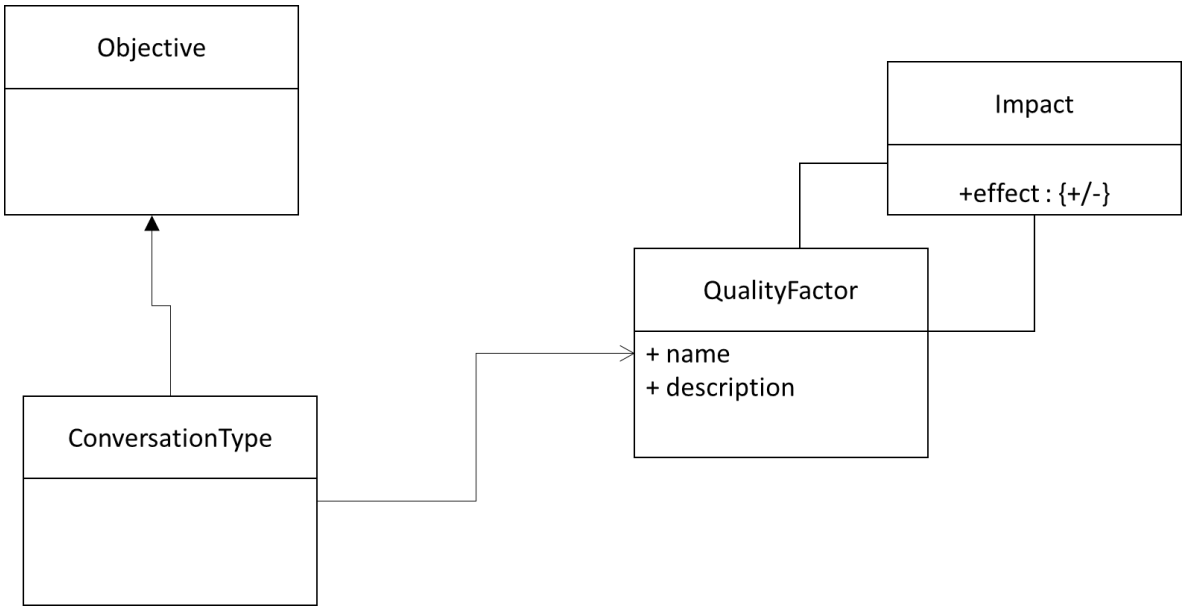
Utiliza-se como base o meta-modelo de qualidade Quamoco apresentado na seção anterior, de acordo com o trabalho de Wagner et al. (2012).

Para tanto o elemento QualityFactor é introduzido como item chave no meta-modelo para templates centrado em atributos de qualidade não funcionais. Este elemento possui como atributos o nome do fator de qualidade e sua descrição.

Neste caso se adiciona também à modelagem um elemento que representa o objetivo do tipo de conversação, conforme classificações utilizadas na seção 3.2. Enquanto o tipo de conversação é o elemento que liga o novo meta-modelo ao meta-

modelo para templates, o objetivo é o elemento que o liga ao meta-modelo centrado em interações.

Figura 25: Meta-modelo centrado em atributos de qualidade não funcionais

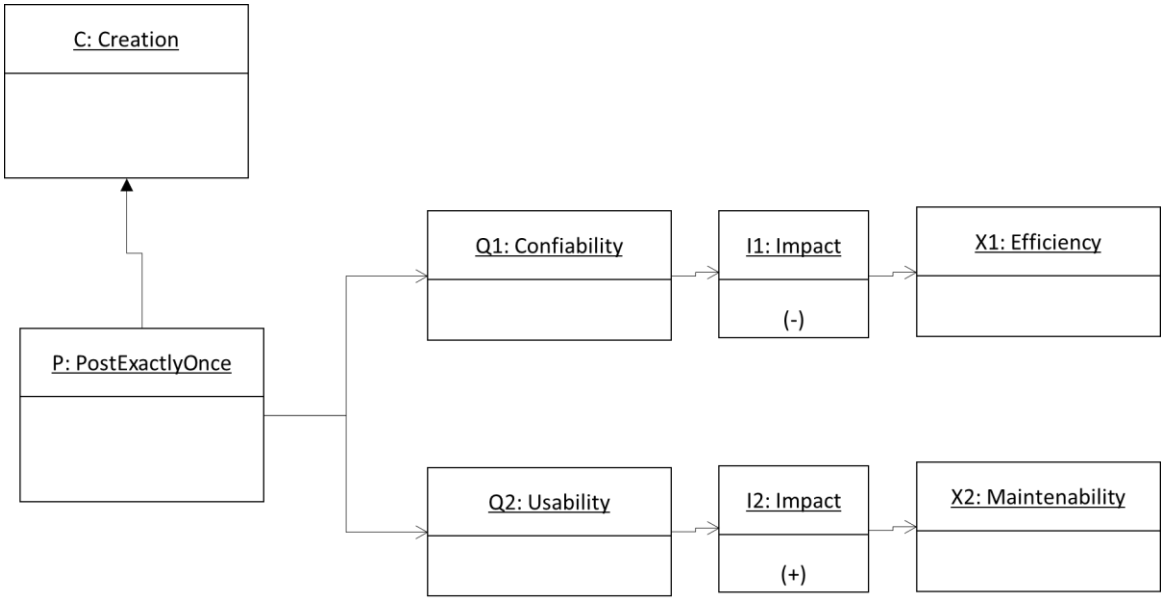


Fonte: Autor

Na figura 26 consta um exemplo de um modelo para o tipo de conversação postagem única, que é um tipo de conversação cujo objetivo é a criação de recursos, com a correspondente pontuação de seus atributos de qualidade. Este modelo é derivado do meta-modelo de qualidade previamente apresentado.

Em uma rápida análise do modelo, já é possível verificar qual seriam os atributos de qualidade afetados positiva e negativamente nesta variante de conversação criacional.

Figura 26: Exemplo de modelo centrado em atributos de qualidade não funcionais

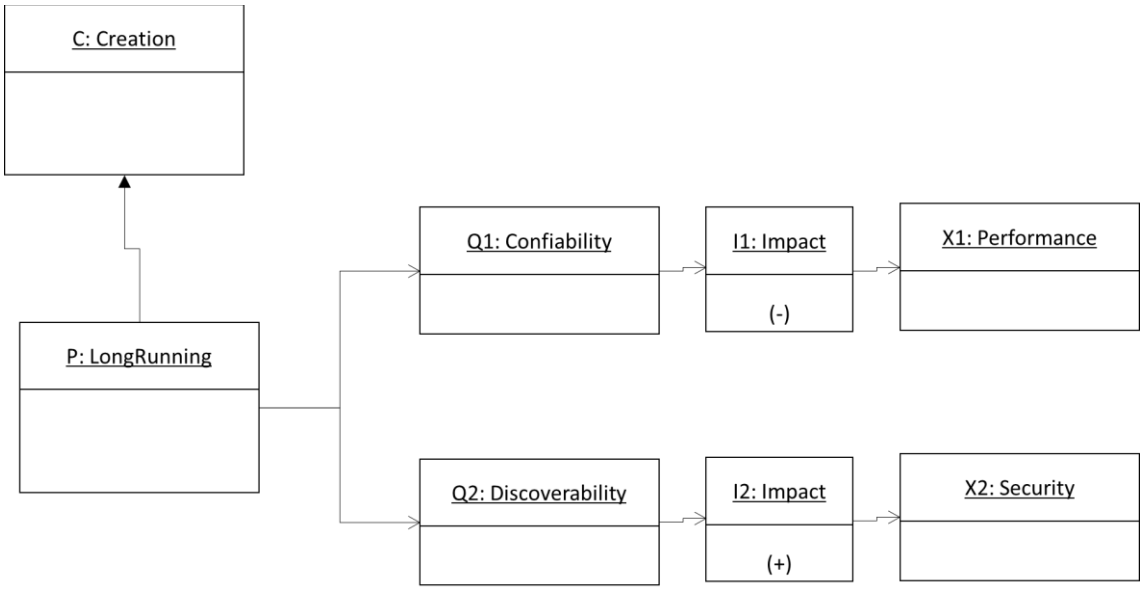


Fonte: Autor

Outro possível exemplo neste momento seria o padrão de criação com operações de longa duração, que favorece fatores de qualidade descobertabilidade e confiabilidade em detrimento de outros como performance e segurança, como visto nas seções anteriores.

Uma possível modelagem para esta transformação é o que se tem na figura 27. Assim, se evidencia no meta-modelo importantes aspectos utilizados no domínio do desenvolvimento dirigido por modelos. No exemplo indicado os valores atribuídos aos atributos de qualidade são ilustrativos e podem variar de acordo com o modelo de domínio que originou as expansões.

Figura 27: Exemplo de modelo centrado em atributos de qualidade não funcionais



Fonte: Autor

É importante ressaltar que o motor que realiza a expansão dos meta-modelos deve ser ajustado para que se considere este novo modelo como um novo direcionador na escolha das expansões possíveis, além das características e pré-condições já estabelecidas no modelo de templates existente, que no caso do trabalho de Haupt et al. (2015) é feito por gramática de grafos.

Caso o motor de transformação identifique que determinado tipo de conversação desfavorece certo atributo considerado crítico em tempo de projeto, deve ser preterido em favor de outro tipo de conversação, em que os atributos de qualidade estejam alinhados com a expectativa do stakeholder.

5 CONCLUSÃO

Durante o trabalho a proposição foi contribuir principalmente com as iniciativas de Haupt (2014 e 2015), que introduziu o conceito de conversações entre APIs REST por meio de uma abordagem dirigida por modelo, e Pautasso (2016), que apresentou uma linguagem para padrões de conversação RESTful, conforme citado previamente na delimitação dos objetivos da presente monografia.

De fato, a contribuição se deu, então, na adição dos fatores de qualidade como direcionadores das decisões automatizadas de transformação entre os meta-modelos envolvidos na abordagem citada para, principalmente, ser uma alternativa aos problemas de casamento de impedâncias conforme citado por Haupt et. al na seção III-A de seu trabalho "*A model-driven approach for REST compliant services*" de 2014.

Desta forma, o pesquisador propôs, embasado da pesquisa bibliográfica realizada, ajustes na abordagem já existente de forma a acrescentar o cuidado com os atributos de qualidade como determinante fator na modelagem. O objetivo, em última instância, foi alcançado por meio da apresentação dos meta-modelos de expansão com os direcionadores de qualidade, a fim de permitir a escolha do modelo de conversação mais adequado ao domínio em que a modelagem se dá.

Durante a confecção e elaboração do trabalho foi possível rever extensa bibliografia e tomar contato com iniciativas nas mais diversas frentes em engenharia de software, como metamodelagem, desenvolvimento dirigido por modelo, fatores de qualidade, entre outros. Estes assuntos foram imprescindível substrato para que se pudesse avançar no sentido do objetivo proposto inicialmente.

Aqui é justo destacar que muitos destes assuntos foram tema central em aulas da Especialização em Engenharia de Software que demandou a presente monografia, sendo que a abordagem realizada pelos professores em muito instigou o pesquisador e teve uma grande participação dentre os fatores que motivaram a pesquisa e o aprofundamento teórico no tema. No geral os assuntos foram apresentados de forma separada nas aulas, com a finalização do trabalho foi possível compreendê-los de forma integrada e, dado o tema, como se complementam em um campo interessantemente multidisciplinar.

Foi possível notar durante a elaboração do trabalho, que especificamente o assunto metamodelagem de APIs REST é objeto de estudo mais difundido em universidades estrangeiras, este nicho, especialmente, em universidades da Alemanha. Pode-se notar este fato ao verificar a bibliografia selecionada e os respectivos pesquisadores que a produziram. Sendo assim, foi primordial dispender um considerável número de horas para pesquisa e garimpagem de artigos que pudessem ser utilizados em consonância com o tema proposto.

Como citado, o ponto de partida para toda a pesquisa foi o trabalho de Florian Haupt et al. (2015), onde se apresenta a abordagem de modelos para conversação em APIs REST. Neste trabalho Haupt teve como colaborador Cesare Pautasso, e este participou de uma empreitada posterior (2016), gerando um artigo que se tornou uma segunda base para o presente trabalho, onde se apresenta os padrões para as conversações modeladas anteriormente. A estes trabalhos somaram-se algumas pesquisas consagradas, como a própria tese de doutorado de Roy T. Fieldings (2000), trabalho clássico que remonta às origens da internet como a conhecemos atualmente.

Outro pesquisador que forneceu subsídio para a monografia e que possui considerável produção na área de integração é Gregor Hohpe. Sua obra “Enterprise Integration Patterns” (2003), por exemplo, é bastante conhecida.

Há de se citar que houve certa dificuldade para encontrar materiais que propusessem um meta-modelo de qualidade aderente ao contexto da pesquisa. Não por acaso, após um considerável investimento de tempo, encontrou-se o trabalho de origem germânica “The Quamoco Quality Meta-Model” proposto por Wagner et al. (2012), que se tornara peça importantíssima no texto elaborado.

Destaca-se positivamente como ressalva à vasta produção estrangeira, o trabalho de Costa et al. (2016) que de forma admirável representa a academia brasileira, mais especificamente a Universidade Federal do Rio de Janeiro, com um trabalho muito detalhado em que se apresenta abordagens para avaliações de arquiteturas RESTful, e que em muito contribuiu para a presente monografia.

Notou-se durante a elaboração dos meta-modelos centrados em qualidade, que o contrapeso entre fatores de qualidade pode ser, em determinada situação, distinto do que ocorreria em outras situações. Ou seja, fatores de qualidade que se impactam mutuamente em determinada situação não necessariamente seguiriam a mesma

regra em situação diferente. Além disso, o peso com que se impactam os fatores também pode variar ou até mesmo se tornar irrelevante em diferentes situações. Importante citar que o substantivo “situação” aqui refere-se diretamente ao domínio que está sendo representado, o que significa, em suma, que em domínios distintos as relações entre os fatores de qualidade não necessariamente serão uma constante, e que também algumas relações existentes podem não ser até mesmo relevantes no âmbito em que se realiza a modelagem.

Além disso, depende também da modelagem e da intenção empregada ao fazê-la, a ordem em que os fatores de qualidade se impactam. Notou-se que a tendência da escolha de um fator em detrimento de outro é primariamente afetada pela importância que este fator possui no domínio a se modelar, sendo que se define, quase que imperceptivelmente, uma hierarquia de fatores de qualidade no domínio em questão. Este é um dos aspectos subjetivos a se lidar quando se trata de design de software, e que foi possível se deparar durante a confecção do trabalho.

Devido a duração da pesquisa, a extensão da revisão bibliográfica e a quantidade de disciplinas envolvidas, o objetivo do trabalho foi delimitado na citada proposição de melhoria da abordagem de modelos existente, exclusivamente no âmbito da modelagem. Então, durante a elaboração do cronograma, devido a esta projeção, desde o início considerou-se que uma abordagem de cunho mais prático não seria possível, ao menos no presente trabalho. Isto foi objetivamente constatado, principalmente devido ao investimento de tempo que de fato foi dispendido na pesquisa bibliográfica e elaboração da estratégia centrada em qualidade.

Desta forma, para futuros trabalhos, propõe-se a avaliação e aplicação prática dos direcionadores de qualidade durante a modelagem de conversações entre APIs RESTful, a fim de que, primariamente, os impactos gerados pela melhoria da automatização possam ser dimensionados.

Um ponto possivelmente importante a se dissecar futuramente é a integração do modelo proposto no processo de transformações de grafos e suas implicações práticas, que nos trabalhos de Haupt foram implementados utilizando ferramentas do IDE (ambiente de desenvolvimento integrado) Eclipse. A partir de conclusões tiradas deste tipo de estudo poderão ser propostas melhorias e refinamento da estratégia e dos modelos propostos no trabalho presente.

Outro ponto interessante a se aprofundar é a hierarquia implícita de fatores de qualidade gerada ao se realizar o modelo centrado em qualidade. Este ponto aqui foi tratado como um aspecto subjetivo derivado e intrínseco às atividades relacionadas com design de software. Entretanto, seria interessante verificar se a citada estrutura pode ser tratada de forma objetiva e aferir de que relações são possivelmente dependentes.

6 BIBLIOGRAFIA

Fielding,T.F., **Architectural Styles and the Design of Network-based Software Architectures**, Dissertação – University of California, 2000.

Fielding, R.T.,Taylor, R.N., *Principled design of the modern Web architecture*, Digital Library, 2002. Disponível em <https://www.ics.uci.edu/~taylor/documents/2002-REST-TOIT.pdf>. Acesso em: 05 de abril de 2017.

Hohpe,G.,Woolfe,B., *Conversation Patterns,* 2006. Disponível em <http://drops.dagstuhl.de/opus/volltexte/2006/828/>. Acesso em 08 de Agosto de 2018.

Völter,M.,Stahl,T.,Bettin,J.,Haase,A.,Helsen,S., **Model-driven software development: technology, engineering, management**, John Wiley & Sons, ISBN 978-1-118-72576-4, 2013.

Clements,P.,Bachmann,F.,Bass,L.,Garlan,D.,Ivers,J.,Little,R.,et al., **Documenting Software Architectures: Views and Beyond**, 2ªed. Addison-Wesley, ISBN 978-0-321-55268-6, 2010.

Sommerville,I., **Engenharia De Software**, 9ªed. Pearson, ISBN 978-8-579-36108-1, 2011.

Richardson,L., Ruby,S., **RESTful Web Services**, 1ªed. O'Reilly, ISBN 978-0-596-52926-0, 2007.

Stahl,T.,Völter, M., **Model-Driven Software Development**, 1ªed. John Wiley & Sons, ISBN 978-0-470-02570-3, 2006.

Kleppe, A., Warmer, J., Bast., W., **MDA Explained, The Model-Driven Architecture: Practice and Promise**, 1ªed. Addison Wesley, ISBN 978-0-321-19442-8, 2003.

Crosby, Philip B.. **Quality is Free**: The art of making quality certain. 1ª ed. Mcgraw-hill, 309 p. ISBN 978-0-070-14512-2, 1979.

Humphrey, Watts S., **Managing the Software Process**. 1ª ed. Addison-Wesley Professional, 512 p. ISBN 978-0-201-18095-4, 1989.

Bass, L., Clements, P., Kazman, R., **Software Architecture in Pratice**, 3ª ed. Addison-Wesley Professional, 640 p. ISBN 978-0-321-81573-6, 2012.

Haupt,F.,Karastoyanova,D.,Leymann,F.,Schroth,B., *A model-driven approach for REST compliant services* – Proc. of the IEEE International Conference on Web Services (ICWS), p.129-136, 2014.

Haupt,F.,Leymann,F.,Pautasso,C, *A conversation based approach for modeling REST APIs* – Proc. of 12th Working IEEE / IFIP Conference on Software Architecture (WICSA), 2015.

Franco,J.M., Barbosa,R., Zenha-Rela,M., *Automated reliability prediction from formal architectural descriptions* – Proc. of Joint Working IEEE / IFIP Conference on Software Architecture (WICSA) and European Conference on Software Architecture, pp.302–309. doi:10.1109/WICSA-ECSA.212.50, 2012.

Costa,B.,Pires,P.,Delicato,F.,Merson,P., *Evaluating REST architectures - Approach, tooling and guidelines*, The Journal of Systems and Software, N.112, p.156-180, 2016.

Pautasso,C., Ivanchikj,A., Schreier,S., *A Pattern Language for RESTful Conversations*. EuroPLoP'16, 2016.

Wagner,S.,Lochmann,K.,Winter,S.,Deissenboeck,F.,Juergens,E.,Herrmannsdoerfer, M., Heinemann,L., *The Quamoco Quality Meta-Model*, Proc. Of International Conference on Software Engineering, 2012.

Mens,T.,Van Gorp, P., *A Taxonomy of Model Transformation*, Theoretical Computer Science N.152, p.125-142, 2006.

Heckel,R., *Graph Transformation in a Nutshell*, Theoretical Computer Science N.148, p.187-198, 2006.

Koziolek,H., *Performance evaluation of component-based software systems: A survey. Perform*. *Eval*. p. 634-658, 2010.

Chidamber,S.R., Kemerer,C.F., *A metrics suíte for object oriented design*. IEEE Trans. Softw. Eng. 20(6) p. 476–493, 1994.

OBJECT MANAGEMENT GROUP. **MOF**: Meta-Modeling and the MOF. 2017. Disponível em: <https://www.omg.org/ocup-2/documents/Meta-ModelingAndtheMOF.pdf>. Acesso em: 28 abr. 2019.

OBJECT MANAGEMENT GROUP. **MDA GUIDE REV. 2.0**: Model Driven Architecture. 2014. Disponível em: <http://www.omg.org/mda/>. Acesso em: 28 abr. 2019.

ISO/IEC. **ISO 25010**: Software Product Quality Model. 2011. Disponível em: <https://iso25000.com/index.php/en/iso-25000-standards/iso-25010?limit=3&limitstart=0>. Acesso em: 28 abr. 2019.

ISO/IEC/IEEE. **ISO 42010**: Systems and software engineering - Architecture description. 2011. Disponível em: <https://www.iso.org/standard/50508.html>. Acesso em: 28 abr. 2019.