Web Services REST

Conceitos, análise e implementação

M.F. Ribeiro[1] e R.E.Francisco[1]
[1]Instituto Federal Goiano – Campus Morrinhos
michelfribeiro@outlook.com – rodrigoefr@gmail.com

Resumo

O presente artigo promoveu a análise e conceituação do estilo arquitetural REST (Representational State Transfer - Transferência de Estado Representacional), proposto por Roy Thomas Fielding, em sua tese de doutorado no ano 2000, e desenvolveu um protótipo de implementação com objetivo de mensurar qualitativamente a eficácia, a viabilidade e o grau de dificuldade na adoção de REST para provimento de interfuncionalidade entre sistemas distribuídos por meio de Web services.

O trabalho se fundamentou em pesquisas bibliográficas e exploratórias, buscando inicialmente construir uma base conceitual bibliográfica, para então implementar um protótipo e mensurar qualitativamente os aspectos de viabilidade, eficácia e dificuldade de implementação.

Cumpridas todas as etapas a que se propôs, o estudo resultou na confirmação da eficácia e viabilidade da adoção de REST no provimento de serviços em um universo de sistemas distribuídos. O grau de dificuldade para implementação foi considerado baixo em termos práticos, mas demandando a necessidade de domínio dos conceitos teóricos envolvidos.

Palavras-chave: REST, Web service, Serviço Web, Sistemas distribuídos.

Abstract

This article promoted the analysis and conceptualization of architectural style REST (Representational State Transfer - Representational State Transfer), proposed by Roy Thomas Fielding in his doctoral thesis in 2000, and developed a prototype implementation in order qualitatively measure the effectiveness, feasibility and the degree of difficulty in the adoption of REST to providing interoperability with between distributed systems.

The work was basing on literature and exploratory research, initially looking at building a conceptual basis to then implement a prototype and measure the qualitative aspects of feasibility, effectiveness, and difficulty of implementation.

Fulfilled all the steps it has set, the study resulted in the confirmation of the effectiveness and feasibility of the adoption of REST in providing services in a world of distributed systems. The degree of difficulty for implementation was considering low in practical terms, but the need for demanding field of theoretical concepts involved.

KEY-WORDS: REST, Web service, Distributed systems.

Introdução

As avançadas funcionalidades providas pelos sistemas computacionais modernos revolucionaram o modo como vivemos, trabalhamos e consumimos produtos, serviços e informações. Muitas destas funcionalidades dependem da adequada comunicação entre diversos sistemas oferecidos por múltiplas organizações e alocados em servidores geograficamente dispersos. O sucesso destas operações exige elevados níveis de interoperabilidade onde, por questões de segurança, escalabilidade e interfuncionalidade, a adoção tecnologias eficientes para comunicação entre sistemas distribuídos constitui pilar fundamental.

Operações cotidianas, como a compra de um livro em um *e-commerce* (loja virtual), empregam serviços de várias empresas - a loja proprietária do site, a empresa de cartão de créditos, a transportadora, etc. - e estes precisam de execução transparente ao usuário final.

Neste contexto, a componentização de sistemas necessita transpor barreiras geográficas e tecnológicas, vez que, em ambiente *Web*, diversas são as aplicações e múltiplas as arquiteturas e tecnologias heterogêneas adotadas na construção dos mais diversos aplicativos, tanto clientes quanto servidores.

Para atender esta demanda, foram desenvolvidos os *Web services*, componentes de software que, por meio de padrões previamente estabelecidos, fornecem serviços específicos e promovem trocas de informações entre sistemas, independentemente das arquiteturas, tecnologias ou linguagens de programação utilizadas na construção das aplicações envolvidas. (DEITEL H.; DEITEL P., 2010).

> Um *Web service* é um componente de negócio que fornece uma funcionalidade reutilizável para clientes, ou consumidores, e pode ser pensado como um componente com acessibilidade verdadeiramente global – se houver direitos de acesso apropriados (SHARP, 2011, P.716).

A realização do presente estudo propõe uma análise do estilo arquitetural REST em termos conceituais, seguido da implementação destes conceitos no desenvolvimento de uma aplicação REST – composta por um protótipo de *Web service* REST e um aplicativo cliente REST que utiliza os serviços oferecidos por este *Web service*. Por fim, aspectos de viabilidade e eficácia são avaliados, assim como o grau de dificuldade identificado para implementação deste estilo arquitetural.

METODOLOGIA

Alicerçados em pesquisas bibliográficas e exploratórias, o presente artigo buscou inicialmente construir uma base conceitual para então implementar um protótipo e mensurar qualitativamente os aspectos de viabilidade, eficácia e dificuldade de implementação do estilo arquitetural REST em Web services.

Web services – Histórico e Demandas

Nas décadas de 1960 e 1970, softwares eram vistos como ferramentas de apoio para rotinas específicas. Não havia acoplamento entre sistemas, de modo que cada software não projetava uma visão ampla da organização. Assim, haviam diversos sistemas para tarefas isoladas o que, pela adoção do paradigma da arquitetura estruturada, acarretava elevados custos de manutenção (SAUDATE, 2013).

Ainda que nas últimas décadas as arquiteturas de sistemas tenham evoluído para o paradigma orientado a objetos, a aproximação das necessidades de negócios nas organizações se mostrava insatisfatória, vez que cada setor da organização demandava um sistema diferente.

> Dentro desta infraestrutura de TI, cada sistema possui um conjunto de dados e informações próprio. Devido ao fato de os processos de negócio envolverem diversas áreas da organização, eles passam a demandar funções de mais de um sistema e tornam necessárias trocas de informações entre estes sistemas (FUGITA; HIRAMA, 2012, p.1).

Essa necessidade gerencial de integração constitui pilar no competitivo e acelerado mercado globalizado atual, onde "[...] raros sistemas operam isoladamente. Uma organização realiza operações comerciais e, para isso, costuma ter aplicativos com suporte para essa funcionalidade empresarial" (SHARP, 2011, p.715).

Neste universo de sistemas distribuídos, o termo serviço pode ser conceituado como a "execução de um trabalho ou realização de uma função de um prestador para um requisitante" (FUGITA; HIRAMA, 2012, p.2). A comunicação entre estas duas partes (requisitante e servidor) se propaga superando obstáculos geográficos e heterogeneidades tecnológicas, trafegando por um meio de comunicação onipresente e disponível para todas as partes envolvidas: a Web. Seria portanto, um *Web service*.

Para que a comunicação entre os sistemas envolvidos ocorra, a tecnologia dos Web services deve "permitir a interação entre aplicações desenvolvidas em diferentes linguagens de programação, plataformas, middleware e sistemas operacionais" (FUGITA; HIRAMA, 2012, p.2), permitindo "que aplicações de outros fornecedores construam serviços com valor agregado sobre aqueles inicialmente fornecidos pela empresa" (COULOURIS; DOLLIMORE; KINDBERG,2007, p.675) o que acarreta em novos negócios e produtos finais inovadores em termos tecnológicos e conceituais.

WEB SERVICES – ARQUITETURAS E APLICABILIDADE

Ao empreender pesquisas bibliográficas por *Web services* na literatura de TI (Tecnologia da Informação), obtêm-se duas principais arquiteturas: SOAP (*Simple Object Access Protocol* - Protocolo Simples de Acesso a Objetos) e REST.

Ambas possuem suporte nas principais IDEs (*Integrated Development Environment* – Ambiente de Desenvolvimento Integrado) disponíveis. Ambas são amplamente utilizadas, oferecem soluções em *Web services* atendendo uma ampla gama de necessidades, utilizam das estruturas de rede já existentes mas utilizam estas estruturas de maneiras diferentes.

Enquanto SOAP é um protocolo construído sobre o HTTP (*Hypertext Transfer Protocol* – Protocolo de Transferência de Hipertexto) e o corpo de sua mensagem é um documento XML (*eXtensible Markup Language* – Linguagem de Marcação Extensível), em que a comunicação com o servidor ocorre por meio de *tags* descritas através de WSDL (*Web service Description Language* – Linguagem para Descrição de *Web service*) (SHARP, 2011), REST disponibiliza seus serviços por meio de uma URL (*Uniform Resource Location* – Localização Uniforme de Recursos) inicial, à partir da qual a navegação é guiada conforme a lógica de negócios proposta, provendo orientações dinâmicas quanto à forma de construção e endereçamento das requisições, adotando um formato de comunicação maximizado que atenda às necessidades de cada aplicação. Em termos REST este formato é nomeado representação.

No que tange à implementação comercial, *Web services* possuem duas principais vertentes de uso: componentes de aplicações reutilizáveis e comunicação entre softwares existentes. A primeira aborda o desenvolvimento de sistemas distribuídos que oferecem serviços para acoplamento em aplicações clientes. A segunda atua como uma interface para provimento de compatibilidade entre aplicações já em operação. Ambas são amplamente utilizadas num contexto de interoperabilidade e colaboração entre sistemas distribuídos. Objetivando liberdade integral para implementação pratico-conceitual, o protótipo desenvolvido neste trabalho enquadra-se na primeira vertente.

REST – A construção do Estilo Arquitetural

Tese de doutorado de Roy Thomas Fielding, apresentada no ano 2000 pela *University Of Califórina*, "*Architectural Styles and the Design of Network-based Software Architectures*" (Estilos arquiteturais e o Design de Arquiteturas de Software Baseados em Redes) promove a análise de diversas arquiteturas, tangendo engenharia de software e redes, e então, no capítulo 5, intitulado "*Representational State Transfer* – REST", Fielding (2000) compõe a estruturação deste novo estilo arquitetural, partindo de um estilo que nomeia como "Desconhecido" (*Null*) e adicionando características – por ele nomeadas "restrições" – selecionadas, previamente identificadas nos estudos promovidos nos quatro capítulos iniciais.

> Examinando o impacto de cada restrição, como isso é adicionado ao estilo referenciado, pode-se identificar as propriedades induzidas pelas limitações da *Web*. Restrições adicionais podem então ser aplicadas para formar um novo estilo arquitetural que melhor reflita as propriedades desejadas de uma arquitetura *Web* (Fielding,2000, p.76).

As características selecionadas por Fielding (2000) para compor REST são:

1 – Arquitetura cliente servidor: Fielding (2000) sintetiza a arquitetura cliente - servidor como a resposta a uma requisição na qual o servidor envia os dados em um formato fixo – em termos REST, uma representação –, ocultando todas as informações sobre a verdadeira natureza dos dados, impedindo assim a construção de suposições

sobre a estrutura de dados existentes no lado servidor e facilitando a utilização destes dados no lado cliente. Esta estrutura promove incrementos em termos de segurança e evolução independente das partes envolvidas, simplificando a manutenção e o suporte, porém restringem as funcionalidades por parte do aplicativo cliente ao concentrar a carga de processamento no servidor.

> Ao separar as preocupações de interface de usuário das preocupações de armazenamento de dados, podemos melhorar a portabilidade da interface do usuário em diversas plataformas e elevar a escalabilidade por meio da simplificação dos componentes do servidor (Fielding, 2000, p.45).

Esta atenção focada na portabilidade de interfaces em diferentes plataformas, alicerçada na arquitetura cliente – servidor, fundamenta a operacionalidade conceitual dos *Web service*s e oferecem o tão almejado conceito "write once, run anywhere" (escreva – o código fonte – uma vez, execute em qualquer lugar) vez que a lógica de negócios é uma vez estruturada no lado servidor, disponibilizada via web e acoplada em aplicativos disponibilizados em qualquer plataforma dotada de um aplicativo cliente compatível – web, móvel ou desktop.

2 – Stateless (sem estado) - Um servidor stateless não tem conhecimento sobre as aplicações conectadas a ele, não possui conhecimento do conteúdo dos dados dessas aplicações ou da forma como ela executa suas funções (BERKENBROCK, 2005). Assim cada nova interação é executada de maneira independente das demais, conforme exposto por Fielding (2000): "cada pedido do cliente para o servidor deve conter todas as informações necessárias para compreender o pedido" (FIELDING, 2000, p.79), sendo portanto independente do pedido seguinte ou do precedente.

Essa independência entre requisições, além de prover escalabilidade ao liberar recursos computacionais imediatamente após processamento de uma requisição no lado servidor, permite ainda que tais recursos possam ser distribuídos em servidores diferentes, dividindo a carga de processamento (NGOLO,2009).

3 – Cache – Primeira restrição de adoção facultativa, a tecnologia cache objetiva reduzir a necessidade de interações cliente / servidor, com mesmo par requisição / resposta, e, por consequente, aumentar o nível de eficiência de uso da rede.

Fielding(2000) sugere a adoção da restrição Cache, permitindo o armazenamento no aplicativo cliente de respostas para posterior utilização em caso de requisições equivalentes. O autor percebe nesta tecnologia o potencial de eliminar parcialmente, ou completamente, algumas interações, melhorando a eficiência, a escalabilidade e o desempenho percebidos pelo cliente, através da redução da latência média de uma série de interações.

Entretanto, a adoção de cache reduz a confiabilidade, uma vez que as respostas armazenadas podem não corresponder aos dados atualizados no servidor, tornando crítica a adoção de métodos eficientes de coerência de conteúdo para garantir a integridade dos dados (BERKENBROCK, 2005).

> No caso da *Web*, intermediários como as caches e próxies, podem agir de acordo com a operação executada e contribuir para a

> escalabilidade de toda a infraestrutura, sem afetar a definição da respectiva operação abstrata. Por exemplo, o método *GET*, idempotente por definição, pode ser servido por *caches* intermediárias sem intervenção do servidor responsável pelo recurso. Por outro lado, os métodos *POST, PUT* e *DELETE* só podem ser processados pelo servidor, uma vez que alteram o estado do recurso (NUNES; DAVID, 2004, p.5).

4 – Interface uniforme - A adoção de uma interface uniforme entre os componentes é a característica propulsora do estilo REST e é constituída por quatro restrições: identificação de recursos, manipulação de recursos por meio de representações, mensagens auto descritivas e hipermídia como o motor do estado do aplicativo. Tais definições simplificam a arquitetura e a visibilidade dos recursos do sistema, porém, por generalizar a aplicação, reduzem a eficiência ao não prever otimizações da transferência de informações personalizadas às especificidades de cada aplicação (FIELDING, 2000). Assim, sua constituição foca na otimização de processos para casos comuns da Web, podendo não ser de interessante adoção em alguns casos específicos.

A identificação dos recursos em REST ocorre por meio de um URI (*Uniform Resource Identifiers*, Identificador Uniforme de Recursos) onde cada recurso disponibilizado deve ser exposto por uma identificação global e acessado por meio de URLs (*Uniform Resource Locators*, Localizador Uniforme de Recursos) composto por: nome do protocolo (geralmente mas não restritivo, HTTP), o nome DNS (*Domain Name System* - Sistema de Nomes de Domínios) da máquina em que o recurso está localizado e o nome do arquivo que contém o recurso (TANENBAUM, 2003).

"Depois de expostos os recursos, a interação é realizada utilizando as operações genéricas do protocolo HTTP" (NUNES; DAVID, 2004, p.6). Em REST, a adoção do *Hypertext Transfer Protocol* (Protocolo de Transferência de Hipertexto - HTTP) não é compulsória, entretanto este protocolo, que compõe a camada de aplicação, é a base sobre a qual a internet se estrutura, característica esta que faz com que o HTTP seja o protocolo mais indicado para utilização em *Web services*.

> O HTTP (*HyperText Transfer Protocol*) implementa passagem de metadados através de cabeçalhos, podendo transportar os dados de maneira segura (através da sobreposição de SSL — *Secure Sockets Layer*), sem contar que é praticamente onipresente (todas as páginas de internet são trafegadas utilizando esse protocolo), fazendo-o ter grande aceitação nas diferentes linguagens de programação (SAUDATE, 2013, p.5).

Uma vez identificado o URL, o acesso ao mesmo é realizado por meio do envio do verbo HTTP (método) e, conforme o caso, os parâmetros inerentes àquela operação, definindo assim a requisição desejada. "O símbolo 'verbo' indica o método para ser executado no recurso identificado pelo *Request-URI*" (FIELDING, 1997).

Ao proceder a comunicação entre componentes por meio de uma representação dinâmica de recursos, REST mantem a verdadeira natureza dos dados ocultas no remetente, sem, contudo, restringir as funcionalidades do receptor, uma vez que este pode trabalhar apenas com partes da representação que sejam

relevantes aos seus interesses. Do mesmo modo, mantem-se a capacidade evolutiva das partes envolvidas.

Pilar do estilo arquitetural analisado, cumpre melhor elucidação das representações REST. Uma representação é um conjunto de dados que se associam aos metadados que os descrevem, de forma auto descritiva por meio de pares "nome-valor. As representações REST, por não possuírem um padrão oficializado e universal, se adequam às necessidades de cada projeto ou padrão institucional, o que permite aos desenvolvedores maior flexibilidade para definição das estruturas de representações que serão utilizadas.

Comumente, *Web services* utilizam representações formatadas em XML ou em JSON (JavaScript Object Notation – Notação de Objetos em JavaScript). Não há, porém, qualquer impedimento para adoção de outras notações de representação, ou mesmo de mais de uma notação no mesmo serviço, contanto que ambos - requisitante e requisitado - saibam como manipula-las e seleciona-las.

Concluindo as restrições para provimento de uma interface uniforme nas aplicações REST, HATEOAS – Hypermedia As The Engine Of Application State, Hipermídia como Motor de Estado de Aplicativos –atua como um motor de navegação construído de forma dinâmica e incluído nas representações fornecidas pelo *Web service*. As respostas com HATEOAS apresentam, além dos dados solicitados, hiperlinks dotados de auto vinculação contendo informações claras de acesso além de orientar a navegação entre recursos.

> A API REST deve ser acessada sem o conhecimento prévio para além da URI inicial (marcador) e um conjunto de tipos de mídia padronizados que são apropriados para o público-alvo (ou seja, deverá ser entendido por qualquer cliente que pode usar a API).Daquele ponto em diante, todas as transições de estado de candidatura deve ser impulsionada pela seleção cliente de opções fornecido pelo servidor que estão presentes nas representações recebidas ou implícita pela manipulação do usuário dessas representações. (FIELDING, 2008)

A sensível contribuição que a tecnologia HATEOAS oferece permite a utilização completa dos serviços providos pelo *Web service* possuindo-se, a priori, somente o URL de acesso inicial. Assim, a cada interação o próprio *Web service* fornecerá dados para as interações subsequentes. Estes dados contemplam não somente a URL do próximo serviço, como também a forma como esta interação se dará em termos de composição da requisição – quantidade e ordem de parâmetros, verbo HTTP exigido, etc. A Figura 1 apresenta um exemplo de notação REST dotada de HATEOAS.

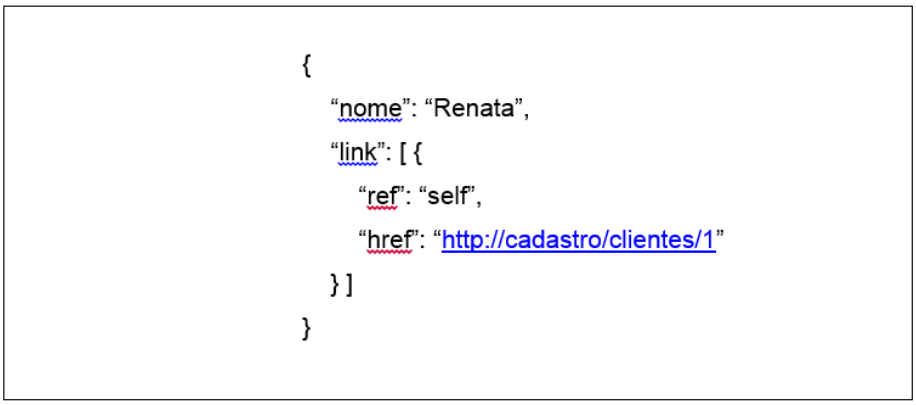


Figura 1- Representação dotada de HATEOAS

Exige atenção o fato de que a adoção da tecnologia HATEOAS não é facultativa ao desenvolvedor de aplicações REST, sendo uma restrição obrigatória para implementação deste estilo arquitetural. "Se o motor do estado do aplicativo (e, portanto, a API) não é conduzido por hipertexto, então não pode ser RESTful e não pode ser uma API REST" (FIELDING, 2008).

5 – Sistema em camadas - Objetivando melhorias operacionais, Fielding (2000) sugere a divisão do sistema em camadas. Esta restrição possibilita a criação de camadas hierárquicas, limitando a ação de cada componente às camadas com as quais a interação se faça necessária, provendo ganhos de segurança, ao encapsular serviços, além de melhorias de desempenho e incrementos de escalabilidade, com melhor balanceamento de carga de processamento e reorganização de componentes intermediários, alocados com melhor eficiência junto aos principais requisitantes.

6 – Código sob demanda – Segunda restrição facultativa, REST permite a funcionalidade opcional de estender código ao aplicativo cliente por meio de *download* e instalação de *applets* e *scripts*.

A utilização de código-sob-demanda acarreta reduções de requisitos de implementação e acréscimos na extensibilidade dos serviços. Por outro lado, há decréscimo de visibilidade, uma vez que parte do processamento ocorre no lado cliente. Deste modo, Fielding (2000) sugere a utilização desta restrição somente em casos nos quais as regras de negócio, que por vezes englobam múltiplas fronteiras organizacionais, ofereçam suporte ao *download* e acoplamento de códigos, e promovam vantagens de funcionalidade.

Modelo de Maturidade de Richardson

Interessado em criar uma classificação para *Web services* REST, Richardson(2008) publicou um trabalho intitulado "*The Maturity Heuristic*", A Maturidade Heurística, por meio do qual propõe estabelecer critérios para distinção dos *Web services* REST em níveis.

Os critérios de maturidade propostos por Richardson (2008) ficaram conhecidos como o "Modelo de Maturidade de Richardson" e propõem a diferenciação dos *Web services* inspirados no estilo arquitetural REST em quatro níveis, conforme retratado na Figura 2.

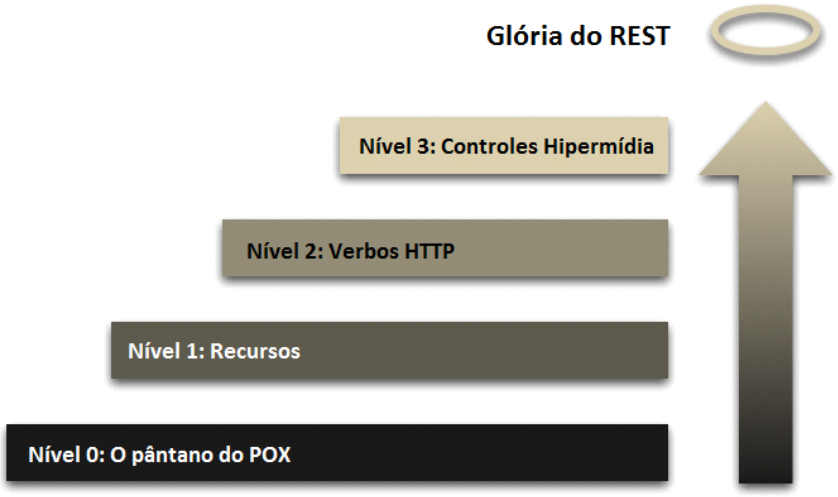


Figura 2 - Modelo de Maturidade de Richardson

Fonte: (FOWLER, 2010) 1

Nota: Adaptado

Cada nível exige a implementação de determinadas restrições REST sendo que o atendimento das implementações requeridas nos níveis anteriores são pré-requisitos ao nível seguinte.

O Nível 0, chamado o Pântano de POX, é suprido pela utilização do HTTP como meio de transporte para provimento de interações entre sistemas remotos.

O Nível 1, Recursos, é atendido pela utilização de identificação de recursos individuais para cada serviço (cada serviço com um URI única).

O Nível 2, Verbos HTTP, exige a utilização de diferentes verbos HTTP (GET/POST/PUT/DELETE/etc.) que direcionados a mesma URI acessam serviços distintos (sobrecarga de métodos).

No Nível 3, Controles Hipermídia, ocorre a implementação de HATEOAS (Hipertexto como Motor de Estado de Aplicativos).

Sob a análise de Richardson (2008), um *Web service* que utilize HTTP como protocolo, URIs únicas para cada serviço, alternância entre serviços pelo verbo HTTP utilizado na requisição e hipertexto dinâmico para navegação entre serviços implementaria a 'Glória do REST' (Richardson, 2008).

Nota-se que a classificação proposta por Richardson(2008) tange *Web services* concebidos sob inspiração do estilo arquitetural REST – e não somente *Web*

services REST, vez que a não implementação de todas as restrições obrigatórias impossibilita a titulação do *Web service* como REST, conforme críticas publicadas por Fielding (2008).

Resultados conceituais

Concluída esta etapa conceitual, nota-se que o termo REST (*Representational State Transfer* - Transferência de Estado Representativo) define com precisão este estilo arquitetural que simplifica a interação entre aplicação cliente e servidor, pela transferência de uma representação, por meio de um par requisição/resposta, geralmente via HTTP, no qual o verbo HTTP, direcionado a um URL, requer a execução do serviço oferecido e a navegação entre serviços ocorre por meio de hipertextos como motor de estado de aplicativos.

Em poder destas ferramentas conceituais, avança-se à implementação prática em um protótipo de *Web service* REST.

PROPOSTA DE IMPLEMENTAÇÃO

Vez que o estilo arquitetural REST é direcionado à implementação em sistemas distribuídos, o protótipo a ser desenvolvido deve se basear na necessidade prática de acesso remoto e proveniente de múltiplos sistemas clientes, cada qual com ampla liberdade de implementação em termos de layout.

Objetivando clareza e aplicabilidade geral, a estrutura proposta para implementação inicial oferece serviços de CRUD – Create, Read, Update e Delete, Criar, ler, atualizar e deletar – em uma base de dados centralizada e gerenciada por um *Web service* REST.

Nota-se que estes mesmos serviços são inerentes às várias aplicações amplamente providas e utilizadas comercialmente, de vendas de ingressos a shows e cinemas, passagens aéreas, bilhetes para estádios, etc. Trata-se portando de um exemplo de implementação com amplo espectro de aplicabilidade.

A escolha pela adoção do estilo arquitetural REST se fundamenta na necessária flexibilidade e compatibilidade com os diversos sistemas clientes que podem se interessar em oferecer aos seus visitantes (usuários) o acesso a este cadastro. Uma vez que cada site, aplicativo para dispositivo móvel ou qualquer que seja a aplicação em questão, possui um sistema informatizado próprio e constituído com tecnologias diversas, a adoção de REST, como já conceituado nas seções anteriores, deve permitir a utilização dos serviços providos pelo *Web service* aos diversos aplicativos clientes.

Uma vez que o objetivo principal no desenvolvimento deste protótipo é a coleta de dados acerca da viabilidade, eficácia e o grau de dificuldade inerentes à implementação do estilo arquitetural REST, optou-se pela simplificação operacional por meio da implementação dos requisitos funcionais fundamentais. Em iterações futuras os requisitos implementados podem se expandir.

LINGUAGENS DE PROGRAMAÇÃO

A linguagem de programação adotada será o PHP - *PHP Hypertext Preprocessor* – em sua versão 5.3.27, disponibilizada em julho de 2013. Esta escolha se baseia no fato do PHP ser uma linguagem de *scripting* de uso geral e popular, de rápido processamento, flexível, escalável, pragmática e amplamente utilizada entre os maiores servidores de conteúdo Web no mundo, a exemplo do Facebook.com, Yahoo.com e Wikipedia.org (CHAPMAN, 2011).

Diversos outros pontos positivos validam a escolha da linguagem PHP, dentre os quais se destacam: a barata e ampla disponibilização de servidores, suporte nos principais IDEs do mercado – no caso em tela utilizou-se o NetBeans 7.4 – e a vasta disponibilidade de *Frameworks* que otimizam o desenvolvimento.

Não obstante, as principais linguagens de programação para Web também são amplamente utilizadas para desenvolvimento de Web services REST, como Java, C# e Rails.

FRAMEWORK

Para desenvolvimento do protótipo de *Web service* REST optou-se pela adoção do *Slim Framework*. Este framework é de licença aberta (*MIT Public License*) e provê características altamente compatíveis com as restrições REST.

Dentre as funcionalidades providas pelo *Slim Framework* se destacam o poderoso roteamento – por meio de métodos HTTP, parâmetros e condições, redirecionamento e paradas, passagem e roteamento de *middleware* –, renderização de modelo personalizadas, *Cookies* seguros com criptografia AES-256, cache HTTP, tratamentos de erros e depuração além da facilidade de configuração.

BANCOS DE DADOS

Para implementação do Protótipo REST optou-se pela adoção do Banco de Dados *MySql*. Tal escolha se fundamenta nas características por ele providas em termos de consistência, alta performance, confiabilidade e facilidade de uso.

Para desenvolvimento da arquitetura do banco de dados utilizou-se o *MySQL Workbench*, ferramenta que permite administração, desenvolvimento e modelagem de bancos de dados.

FLUXOGRAMA DOS SERVIÇOS

Com vista a documentação dos processos, por meio do registro das etapas, tomadas de decisões e relações entre os serviços, optou-se por uma representação gráfica dos elementos, componentes e serviços providos pelo protótipo de *Web service* REST em desenvolvimento. Esta representação é expressa na forma de um fluxograma, escolha que deriva de maneira natural da restrição HATEOAS, vez que os serviços se referenciam por meio de hiperlinks de maneira fluídica e sequencial não-linear.

Este fluxograma (Figura 3) apresenta o início e o fim da interação com o *Web service* nos círculos hachurados. Os serviços são representados na forma de retângulos e cada retângulo representa a execução de um serviço. Os processos de tomada de decisões são representados na forma de losangos.

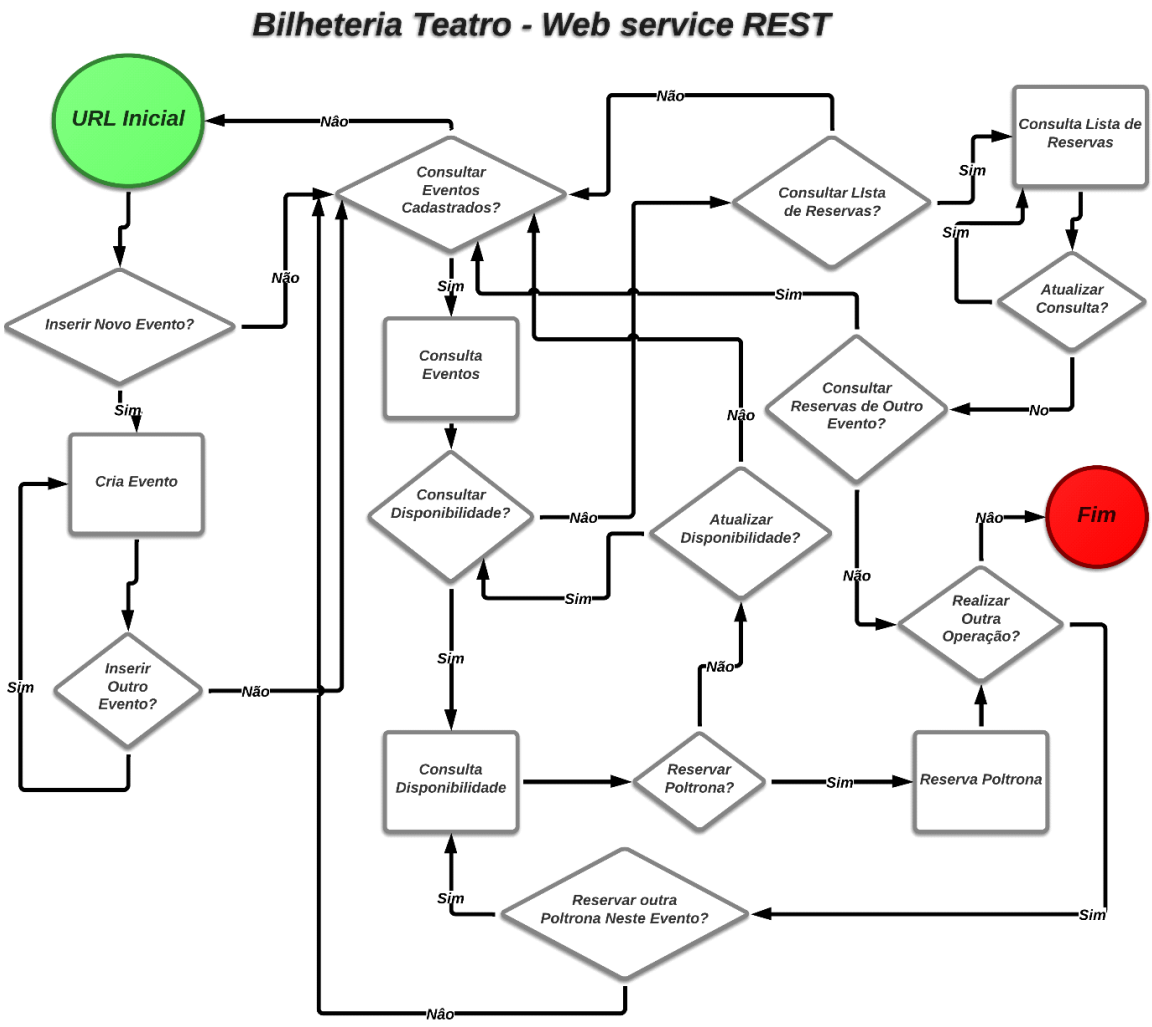
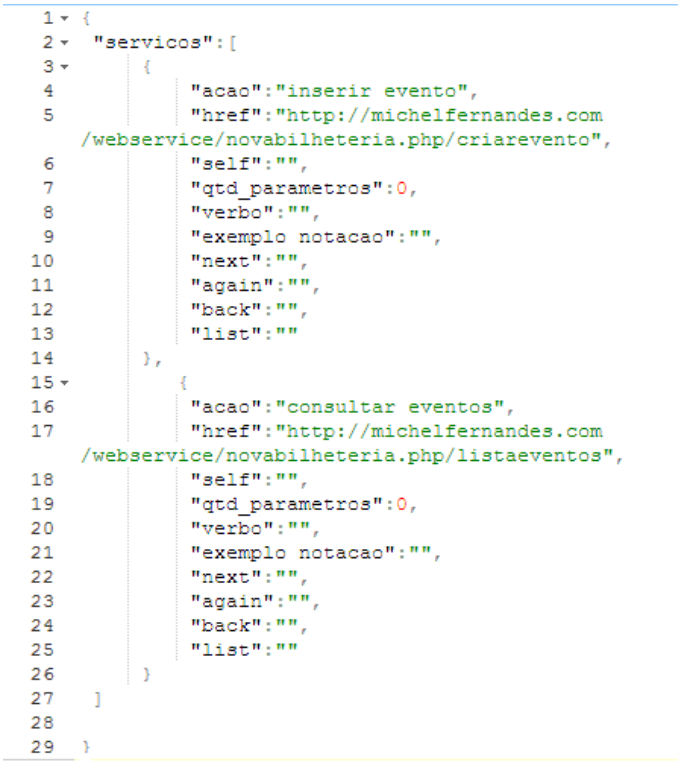


Figura 3 - Fluxograma dos Serviços

ESTRUTURA DE REPRESENTAÇÃO

Vez que não há nenhum padrão absoluto sobre a forma de estruturar as representações REST (MITCHELL, 2013), fica a cargo de cada desenvolvedor a criação da estrutura que melhor atenda aos objetivos da aplicação, contanto que os clientes sejam capazes de interpretar as representações recebidas.

Para o protótipo proposto, optou-se pela criação da estrutura de representação constante na Figura 4 e adotada em todos os serviços providos nesta aplicação.



```
{
 "servicos":[
      {
          "acao":"inserir evento",
          "href":"http://michelfernandes.com
/webservice/novabilheteria.php/criarevento",
          "self":"",
          "qtd_parametros":0,
          "verbo":"",
          "exemplo notacao":"",
          "next":"",
          "again":"",
          "back":"",
          "list":""
      },
         {
          "acao":"consultar eventos",
          "href":"http://michelfernandes.com
/webservice/novabilheteria.php/listaeventos",
          "self":"",
          "qtd_parametros":0,
          "verbo":"",
          "exemplo notacao":"",
          "next":"",
          "again":"",
          "back":"",
          "list":""
      }
  ]

}
```

Figura 4- Representação Json adotada

Esta representação se estrutura na forma de um array de serviços composto por dez parâmetros e expressos em Json. Os parâmetros são:

- Ação: apresentando a ação realizada por aquele serviço;
- Href: contendo o atributo HATEOAS para acesso à ação descrita no atributo anterior;
- Self: apontando os atributos ou informações adicionais àquela requisição;
- Qtd_parâmetros: quantidade de parâmetros exigidos para execução do referido serviço;
- Verbo: verbo HTTP exigido para a requisição;
- Exemplo notação: exemplo de notação para interação com o Web service (este parâmetro somente expõe valores quando se faz necessária ordenação entre os atributos);
- Next: para indicação se é a próxima interação sequencial;
- Again: para indicação de repetição do serviço atual (atualização, etc);
- Back: indicação de que aquele serviço é para voltar ao serviço anterior (em casos de linearidade na interação);

- List: apontando que se trata de uma listagem.

Os parâmetros “ação”, “href”, “self”, “qtd_parametros” e “verbo” são parâmetros funcionais, ou seja, os valores neles contidos são subsidiários às requisições dos serviços referidos.

O parâmetro “exemplo notação” objetiva prover orientações aos desenvolvedores dos aplicativos clientes vez que, quando necessário, apresenta um exemplo de como a URL dinâmica deve ser montada – com os parâmetros necessários – para direcionamento das requisições.

Os parâmetros “next”, “again”, “back” e “list” são fornecidos para fins de orientação lógica em termos de layout. O parâmetro “list” pode, por exemplo, ser utilizado como filtro para orientar o aplicativo cliente na criação de uma tabela.

Em uma sequência de serviços lineares, os parâmetros “next”, “again” e “back” podem ser utilizados como forma de orientar o aplicativo cliente sobre o posicionando dos disparos (links) aos referidos serviços de maneira mais conveniente aos usuários. Um exemplo da utilização destes atributos consta à na Figura 5.

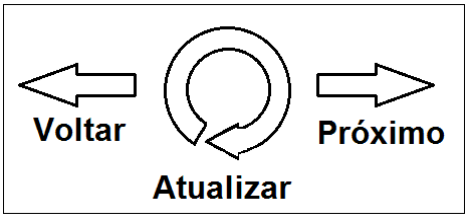


Figura 5- Parâmetros ‘Next’, ‘Again’ e ‘Back’

CLIENTE REST – REQUISITOS ARQUITETURAIS

“Um sistema REST não é somente um servidor REST. Um sistema REST é o conjunto de um servidor REST e um cliente REST capaz de se adaptar às diretrizes que este servidor fornece a cada interação" (SILVEIRA, 2010).

O desenvolvimento do aplicativo cliente REST deve ser tão estruturado em termos conceituais quanto o próprio *Web service* vez que para ser uma aplicação REST tanto o cliente quanto o servidor devem seguir as características deste estilo arquitetural.

> O aplicativo cliente não pode ser pré-configurado com uma sequência rígida de comportamentos padrões. O aplicativo cliente não sabe o que o servidor irá retornar para ele. Talvez ele retorne uma lista vazia. Talvez ele informe que o serviço foi encerrado. Não se pode saber o que o servidor irá retornar. Assim, o aplicativo cliente deve ser adaptável às respostas do servidor (SILVEIRA, 2010).

A necessidade de completa adaptação ao resultado da interação com o *Web service* exige, no aplicativo cliente, uma estrutura capaz de analisar a resposta proveniente do servidor, trata-la e disponibiliza-la ao usuário final de maneira a que este possa realizar as demais requisições necessárias. Deste entendimento resulta que os *Web services* tem por usuários outros desenvolvedores e devem apresentar seus serviços de maneira que aqueles possam integra-los às suas próprias aplicações.

Neste contexto, o aplicativo cliente deve, em última instância, atuar como uma camada intermediária entre o usuário final e o serviço por ele desejado, realizando as interações com o *Web service* e disponibilizando os resultados destas interações – conforme a lógica de cada aplicativo.

RESULTADOS DA IMPLEMENTAÇÃO

Os levantamentos bibliográficos realizados ao longo deste trabalho se mostraram suficientes à compreensão do estilo arquitetural REST e forneceram subsídio ao desenvolvimento da aplicação proposta.

Todas as restrições obrigatórias adotadas por Fielding (2000) puderam ser compreendidas e implementadas, ora pela compreensão direta da tese original, ora pela assistência de outros autores que são referenciados ao longo do texto.

É relevante registrar que grande parte das representações de *Web services* REST (ou que se denominam REST) encontradas na *web* não adotam HATEOAS. A não adoção desta restrição desqualifica a estrutura em termos REST e portanto não podem ser utilizadas como parâmetro de referência.

O desenvolvimento, análise e eliminação de diversas estruturas de representação resultaram na proposta de representação adotada e que é integralmente autoral.

Outra grande barreira superada foi a estruturação do cliente REST. Uma vez que o aplicativo cliente é reformulado dinamicamente a cada interação, não podem haver estruturas fixas e portanto a lógica primaria desta aplicação deve ser constituída em função do tratamento das representações recebidas, o que difere das aplicações tradicionalmente criadas.

Conclusão

O desenvolvimento deste trabalho pode constituir valioso subsídio à compreensão do estilo arquitetural REST em língua portuguesa. Nas pesquisas realizadas, diversas referências a *Web services* REST foram identificas mas após maior aprofundamento conceitual, observou-se que a maioria dessas referências não implementava todas as restrições arquiteturais obrigatórias e portanto não poderiam ser intituladas REST.

Analisada sob tese original de Fielding (2000) e submetida às críticas realizadas por Fielding (2008), a implementação aqui proposta se mostra integralmente coerente

com esse rico estilo arquitetural estando portanto no que Richardson (2008) nomeia como “a glória de REST”.

De maneira sucinta, conclui-se, face a toda literatura consultada e às experiências decorrentes do desenvolvimento do protótipo implementado, que:

1 – REST apresenta baixo nível de dificuldade para implementação, quando o ambiente de desenvolvimento é adequado e os desenvolvedores possuem familiaridade com as restrições conceituais inerentes ao estilo arquitetural;

2 – A adoção de REST para o provimento de interfuncionalidade entre sistemas distribuídos é viável para os casos em que os serviços oferecidos necessitem de escalabilidade, baixo acoplamento, fácil evolução e flexibilidade, onde a valoração desses atributos sobreponha o trabalho de estruturação e análise das representações enriquecidas com hipermídia como motor da aplicação (HATEOAS).

3 – REST é eficaz no provimento de interfuncionalidade entre sistemas distribuídos cumprindo integralmente as funções a que se propõe, conforme demonstrado com sucesso no protótipo construído.

Compreendendo por atingidos os objetivos inicialmente propostos, diversos trabalhos futuros se mostram de interessante exploração.

A primeira proposta tange à verificação da eficácia de REST no provimento de interfuncionalidade entre aplicações constituídas com tecnologias heterogêneas. Conceitualmente a resposta a esse questionamento é positiva, conforme aqui demonstrado, mas implementações práticas enriqueceriam este entendimento.

Um segundo campo de estudo seria a análise de REST em termos de segurança. Seria REST verdadeiramente seguro? Uma vez que a adoção de criptografia não infringe nenhuma restrição arquitetural, esse questionamento também aparenta resultar em uma afirmação, o que seria de maior confiabilidade se demonstrado por meio de implementações.

Mensurações em termos de eficiência também seriam de rica exploração, comparando-se REST com as demais arquiteturas de *Web services* amplamente utilizadas.

Satisfeitos todos objetivos propostos, dá-se por concluído o presente estudo.

REFERÊNCIAS

BERKENBROCK, Carla Diacui Medeiros. **Investigação e Implementação de Estratégias de Notificação de Invalidação para Coerência de Cache em Ambientes de Computação Móvel sem Fio.** 2005. 87 f. Dissertação (Mestrado) - Curso de Mestrado em Ciências da Computação, Universidade Federal de Santa Catarina, Florianópolis, 2005.

CHAPMAN, Roger. **Top 40 Website Programming Languages.** 2011. Disponível em: <http://rogchap.com/2011/09/06/top-40-website-programming-languages/>. Acesso em: 24 dez. 2013.

COULOURIS, George; DOLLIMORE, Jean; KINDBERG, Tim. **Sistemas Distribuídos:** Conceitos e Projeto. 4. ed. Porto Alegre: Bookman, 2007. 788 p.

DEITEL, Paul; DEITEL, Harvey. **Java:** Como Programar. 8. ed. São Paulo: Pearson, 2010. 1144 p. Tradução de: Edson Furmankiewicz.

FIELDING, Roy Thomas et al. **Hypertext Transfer Protocol - HTTP/1.1.** 1997. RFC 2068. Disponível em: <http://www.ietf.org/rfc/rfc2068.txt>. Acesso em: 12 ago. 2013.

_____. **Architectural Styles and the Design of Network-based Software Architectures.** 2000. 162 f. Tese (Doutorado) - Curso de Computer Science, Departamento de Computer Science, University Of California, Irvine, 2000.

_____. **REST APIs must be hypertext-driven.** 2008. Disponível em: <http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven>. Acesso em: 15 jan. 2014.

FOWLER, Martin. **Richardson Maturity Model:** steps toward the glory of REST. 2010. Disponível em: <http://martinfowler.com/articles/richardsonMaturityModel.html>. Acesso em: 10 fev. 2014.

FUGITA, Henrique Shoiti; HIRAMA, Kechi. **SOA:** modelagem, análise e desing. Rio de Janeiro: Elsevier, 2012. 156 p.

MITCHELL, Lorna Jane. **Web Services em PHP:** APIs para a web moderna. São Paulo: Novatec Editora Ltda, 2013. 135 p.

NGOLO, Márcio António Fernandes. **Arquitectura Orientada a Serviços REST para Laboratórios Remotos.** 2009. 95 f. Dissertação (Mestrado) - Curso de Mestrado em Engenharia Electrotécnica e de Computadores, Departamento de Departamento de Engenharia Electrotécnica, Universidade Nova de Lisboa, Lisboa, 2009.

NUNES, Sérgio; DAVID, Gabriel. Uma Arquitectura Web para Serviços Web. **Universidade do Porto,** Porto, p.1-11, 2005.

RICHARDSON, Leonard. **The Maturity Heuristic.** 2009. Disponível em: <http://www.crummy.com/writing/speaking/2008-QCon/act3.html>. Acesso em: 12 fev. 2014.

SAUDATE, Alexandre. **SOA aplicado:** Integrando com web services e além. São Paulo: Casa do Código, 2013. 277 p.

SHARP, John. **Visual C# 2010:** Passo a Passo. Porto Alegre: Bookman, 2011. 780 p.

SILVEIRA, Guilherme. **REST Maturity Model.** 2010. Vídeo. Disponível em: <http://agilenomundoreal.wordpress.com/2010/04/13/videos-sobre-rest/>. Acesso em: 10 jan. 2014.

TANENBAUM, Andrew S.. **Redes de computadores.** 4. ed. Rio de Janeiro: Elsevier, 2003. 945 p.