UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
INSTITUTO DE MATEMÁTICA
INSTITUTO TÉRCIO PACITTI DE APLICAÇÕES E PESQUISAS COMPUTACIONAIS
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

BRUNO CARLOS DA CUNHA COSTA

Avaliando uma Arquitetura Baseada no Estilo REST

Rio de Janeiro

2014

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
INSTITUTO DE MATEMÁTICA
INSTITUTO TÉRCIO PACITTI DE APLICAÇÕES E PESQUISAS COMPUTACIONAIS
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

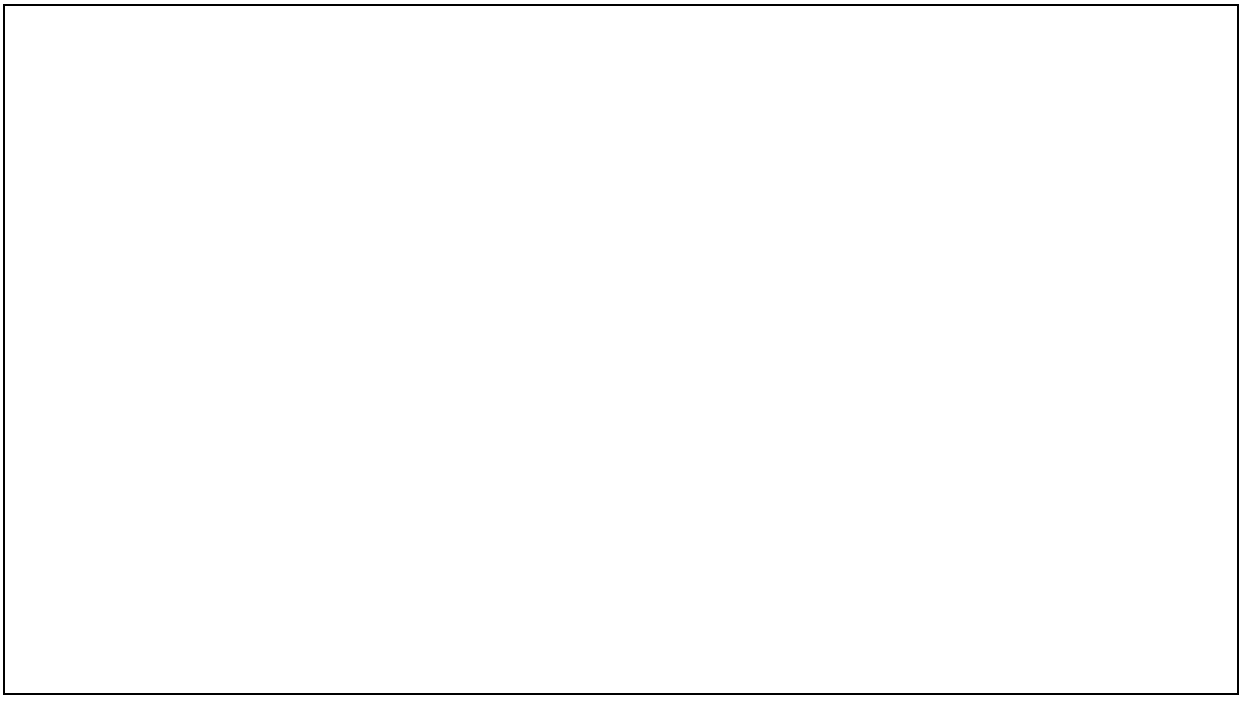
BRUNO CARLOS DA CUNHA COSTA

Avaliando uma Arquitetura Baseada no Estilo REST

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Informática, Universidade Federal do Rio de Janeiro, como requisito parcial à obtenção do título de Mestre em Informática

Orientador: Paulo de Figueiredo Pires

Rio de Janeiro
2014



Bruno Carlos da Cunha Costa

Avaliando uma Arquitetura Baseada no Estilo REST

Dissertação de Mestrado submetida ao Corpo Docente do Programa de Pós-Graduação em Informática da Universidade Federal do Rio de Janeiro e à banca externa convidada como parte dos requisitos necessários para obtenção do título de Mestre em Informática.

Aprovada em: Rio de Janeiro 28 de Fevereiro de 2014.

Prof[a]. Paulo de Figueiredo Pires – Orientador
DSc, PPGI/UFRJ, Brasil

Prof. Leonardo Gresta Paulino Murta
DSc, UFF, Brasil

Prof[a]. Flávia Coimbra Delicato
DSc, PPGI/UFRJ, Brasil

Rio de Janeiro
2014

AGRADECIMENTOS

Ao meu Amigo e Senhor Jesus, e à sua Igreja Católica Apostólica Romana. Sem o conhecimento do sentido da minha vida, nenhum trabalho poderia ter sido realizado.

À minha mãe e senhora Maria de Guadalupe. Totus tuus.

Ao apoio de meus pais, Jania Lúcia e Jorge Luiz. Minha segurança nesta terra.

À minha noiva e futura esposa, Juliana Oliveira. Por ser minha companheira, amiga e o amor da minha vida.

Ao meu irmão Felipe Luiz. Pela sua alegria contagiante que muito me motivou.

À minha vó Nancy, minhas madrinhas e padrinhos, meus tios e tias, primos e primas de minha família Cunha e Costa. Por vários momentos foram suas orientações que me mantiveram no caminho.

Ao meu orientador, Paulo de Figueiredo Pires. Por orientar-me com seu exemplo e seu imenso conhecimento técnico e científico; e à professora Flávia Coimbra Delicato, pela sua confiança em mim que muito me motivou.

Ao professor Leonardo Gresta Paulino Murta. Seus ensinamentos são sempre presentes em minha vida.

Ao Paulo Merson, meu mentor. Pela sua "co-orientação" e exemplo. Sem sua participação, grande parte deste trabalho não seria desenvolvido.

A todos os docentes que tive o prazer de conhecer e aprender no Programa de Pós-Graduação em Informática da UFRJ, por terem contribuído com minha formação profissional e pessoal. Assim como meus companheiros do UBICOMP, o dia a dia com eles é desafiador e enriquecedor.

A todos que contribuíram com este trabalho em especial ao Matthias Naab (Fraunhofer IESE), Daniel Jacobson (Netflix), Casare Pautasso (Università della Svizzera Italiana), Erik Wilde (UC Berkeley School of Information), Felipe Oliveira and Alexandre Saudate (SOA Expert), Luiz Cipriani and Luiz Rocha (Abril Midia), Kleber Bacili (Sensedia), Wanderlei Souza (Apontador.com), engenheiros do Google Maps API e avaliadores arquiteturais do SEI.

Ao convênio UFRJ/CENPES que forneceu apoio financeiro para o desenvolvimento deste trabalho.

Este trabalho foi desenvolvido com recursos financeiros do convênio entre a Universidade Federal do Rio de Janeiro e o Centro de Pesquisas e Desenvolvimento Leopoldo Américo Miguez de Mello da Petrobrás.

“A humildade é o primeiro degrau para a sabedoria”

Sto Thomás de Aquino

“A recompensa de uma obra é tê-la feito; a recompensa de um esforço é ter crescido”

A.-D. Sertillanges

RESUMO

COSTA, Bruno Carlos da Cunha. Avaliando uma Arquitetura Baseada no Estilo REST. Rio de Janeiro, 2014. Dissertação (Mestrado em Informática) - Programa de Pós-Graduação em Informática, Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2014.

O uso de REST, como um estilo arquitetural para a integração de serviços e aplicações traz vários benefícios, porém também apresenta novos desafios e riscos. Particularmente importante entre tais riscos são falhas para a realização efetiva dos requisitos de atributos de qualidade, tais como: segurança, confiabilidade e desempenho. Uma avaliação arquitetural no início do ciclo de vida do desenvolvimento do software pode ajudar a identificar e mitigar esses riscos. Neste trabalho é apresentado um guia contendo diretrizes para auxiliar as atividades de avaliação de arquitetura de sistemas baseados em REST. Essas diretrizes podem ser usadas sistematicamente em conjunto com métodos de avaliação baseados em cenários para analisar questões acerca do design e *tradeoffs* associados aos atributos de qualidade. Para atingir este objetivo uma pesquisa sistemática na literatura e entrevistas com especialistas foram realizadas seguindo um protocolo de pesquisa estabelecido. Para validação, o guia foi enviado a avaliadores arquiteturais do Instituto de Engenharia de Software (SEI) da Universidade Carnegie Mellon, Pittsburgh, EUA. Os avaliadores utilizaram o guia e responderam um questionário, demonstrando a validade das diretrizes. Além disso, neste trabalho também é apresentada uma prova de conceito demonstrando a utilização do guia no método de avaliação arquitetural ATAM.

Palavras-chave: *avaliação de arquitetura de software; diretrizes para avaliação baseada em cenário, rest, atam*

ABSTRACT

COSTA, Bruno Carlos da Cunha. Evaluating an Architecture Based on Representational State Transfer Architecture. Rio de Janeiro, 2014. Dissertação (Mestrado em Informática) - Programa de Pós-Graduação em Informática, Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2014.

The use of Representational State Transfer (REST) as an architectural style for integrating services and applications brings several benefits, but also poses new challenges and risks. Particularly important among those risks are failures to effectively address quality attribute requirements such as security, reliability, and performance. An architecture evaluation early in the software life cycle can identify and help mitigate those risks. In this dissertation we present a guidance to assist architecture evaluation activities in REST-based systems. These guidelines can be systematically used in conjunction with scenario-based evaluation methods to reason about design considerations and trade-offs between quality attributes. To achieve this goal a systematic literature research and expert interviews were done following a search protocol. In order to validate, the guidance was sent to architecture evaluators at Software Engineering Institute (SEI) of Carnegie Mellon University, Pittsburgh, USA. The evaluators used the guidance and answered a questionnaire, demonstrating the validity of the guidelines. In addition, we present in this dissertation a proof of concept demonstrating how the guidance is used in architectural evaluation method ATAM.

Keywords: *software architecture evaluation; scenario-based evaluation guidelines, rest, atam*

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

$ - *Cache*

API - *Application Programming Interface*

ATAM - *Architecture Tradeoff Analysis Method*

CoD - *Code on Demand*

C-S - *Client-Server*

EcoDiF - Ecossistema Web de Dispositivos Físicos

EEML - *Extended Environments Markup Language*

EMML - *Enterprise Mashup Markup Language*

HATEOAS - *Hypermedia as the Engine of Application State*

HTTP - *Hypertext Transfer Protocol*

IEC - *International Electrotechnical Commission*

IoT - *Internet of Things*

ISO - *International Organization for Standardization*

JSON - *JavaScript Object Notation*

L - *Layered System*

QP - Questão de Pesquisa

REST - *Representative State Transfer*

RFC - *Request for Comments*

RG - Requisição para o Guia de Orientação para Avaliação REST

S - *Stateless*

SEI - *Software Engineering Institute*

SOA - *Service-oriented Architecture*

U - *Uniform Interface*

URI - *Uniform resource identifier*

WS-* - Padrões especificados pela *Web Services Interoperability Organization*

XML - *eXtensible Markup Language*

Sumário

1 Introdução

Decisões arquiteturais determinam a capacidade de um sistema de software de satisfazer os requisitos funcionais e atributos de qualidade (Interoperabilidade, Confiabilidade, Desempenho, Segurança, dentre outros). Uma vez que tais decisões estão entre as primeiras a serem tomadas no ciclo de vida de desenvolvimento do software e afetam fases posteriores do processo, o impacto de erros na especificação arquitetural é alto. É necessário, portanto, inspecionar a arquitetura antes da sua implementação, a fim de identificar e reduzir os riscos, principalmente àqueles associados à realização de requisitos de atributos de qualidade. Esta atividade refere-se à avaliação da arquitetura de software [Bass et al. 2012].

A avaliação arquitetural de software caracteriza-se como uma metodologia da Engenharia de Software que propõe diversas atividades para inspeção da conformidade dos requisitos com os elementos arquiteturais de um sistema. A inspeção ocorre através da análise das abordagens de *design* especificadas e documentadas de acordo com os requisitos do software. Tal documentação pode ser descrita, por exemplo, segundo o modelo 4+1 de visão da arquitetura [Kruchten 1995].

Dentre os métodos de avaliação arquitetural existentes, os métodos baseados em cenários [Kazman et al. 1996] inspecionam a arquitetura com base em diversos cenários de interesse definidos por arquitetos, usuários e clientes. Os cenários são descrições do comportamento desejado em um determinado sistema segundo certa condição. Por exemplo, um cenário desejado para um sistema de controle de vendas pela Internet poderia ser: “quando o usuário selecionar a opção para finalizar a compra o sistema deverá efetuar a transação e informar ao usuário em menos de 5 segundos”.

Existem dois momentos para a realização da avaliação arquitetural: antes da implementação (*Early Evaluation*) e depois da implementação (*Late Evaluation*) [Clements et al. 2001; Roy and Graham 2008]. O objetivo da avaliação da arquitetura antes da implementação é assegurar que os componentes especificados atendam aos requisitos, localizando problemas e riscos aos atributos de qualidade antecipadamente à implementação. Inspecionar a arquitetura depois da implementação tem por objetivo assegurar que a arquitetura projetada foi desenvolvida juntamente com os desvios ocasionados, como componentes que não estão de acordo com a especificação.

No contexto dos métodos de avaliação arquitetural baseados em cenários, o método ATAM[1] [Kazman et al. 1998, 2000] é amplamente utilizado atualmente na indústria e academia. Tal método objetiva inspecionar a arquitetura buscando os riscos para a realização dos atributos de qualidade e o *tradeoff*[2] entre eles. O método ATAM propõe a inspeção da arquitetura seguindo nove atividades estabelecidas. Juntamente com os participantes da avaliação, a saber, *stakeholders*[3] e arquitetos responsáveis pelo sistema, a equipe de avaliação executa as atividades do método, confrontando os elementos arquiteturais com os requisitos de atributos de qualidade especificados através de cenários. Nas atividades do método ATAM a inspeção ocorre com base no conhecimento empírico da equipe ou com o auxílio de um guia contendo diretrizes que descrevam os típicos cenários de atributos de qualidade e questões que devem ser feitas na avaliação arquitetural. Estas diretrizes são particularmente importantes quando a arquitetura inspecionada segue algum estilo arquitetural complexo ou de recente especificação.

Os guias são documentos que descrevem informações relevantes para se atingir objetivos em um determinado contexto. Usualmente, no processo de criação de tais guias, as melhores práticas utilizadas por especialistas são documentadas, confrontadas com o conhecimento teórico da área e validadas por profissionais da indústria e academia. Assim, os guias mostram-se como um material valioso, sendo aplicados em diversos processos e atividades.

Guias que apresentam diretrizes acerca de técnicas, métodos e boas práticas são amplamente desenvolvidos e utilizados na Engenharia de Software. No contexto do paradigma orientado a objeto, por exemplo, os padrões de projeto compilados por Gamma et al. (1994) determinam soluções reutilizáveis para problemas recorrentes no desenvolvimento de sistemas orientados a objeto. Diretrizes acerca do projeto e execução de testes são apresentadas por autores como Beck (2002). O guia proposto por Wozniak et al. (2011) apresenta diretrizes para o projeto de sistemas na área automobilística. Outro exemplo, apresentado por Becker e Diaz-Herrera (1994), descreve um guia para a criação de bibliotecas para reuso de componentes de software. Não obstante, o Instituto de Engenharia de Software

[1] Acrônimo para o inglês *Architecture Tradeoff Analysis Method*
[2] Atributos de qualidade que competem entre si. Por exemplo, as decisões de Interoperabilidade podem impactar em questões de Segurança.
[3] Termo amplamente utilizado para indicar qualquer interessado no projeto, por exemplo: arquitetos de software, gerentes de projeto, clientes, usuários, etc.

(SEI)[4] da Universidade Carnegie Mellon, um instituto de referência mundial, usualmente publica guias para o auxílio de profissionais nas subáreas da Engenharia de Software. Um dos guias publicados, apresentado no parágrafo a seguir, tem o foco em avaliação arquitetural de sistemas baseados no estilo SOA[5].

O estilo SOA é um dos estilos mais utilizados atualmente para resolver problemas de integração de sistemas. Ele apresenta uma especificação para a organização de componentes em um contexto orientado a serviços. Devido o risco e o impacto do estilo SOA serem distribuídos entre as aplicações a serem integradas é fundamental a realização de uma avaliação arquitetural antes da implementação do sistema. Com o objetivo de auxiliar avaliadores arquiteturais no tocante às questões de *design* a serem feitas na inspeção arquitetural de sistemas baseados em SOA, o guia "*Evaluating a Service-Oriented Architecture*" [Bianco et al. 2007] foi proposto por avaliadores e pesquisadores do SEI. O documento contém diretrizes sobre considerações de *design* SOA, impactos do estilo arquitetural na realização de atributos de qualidade e *tradeoffs* entre eles. O guia é amplamente utilizado por avaliadores arquiteturais e arquitetos de software [Naab et al. 2013].

Existem duas principais abordagens para a implementação de serviços SOA, a saber, WS-*[6] e REST. A utilização das abordagens também é uma decisão arquitetural que deve ser feita com base nas características do contexto da aplicação (como, por exemplo, utilizar REST para cenários onde dispositivos heterogêneos acessarão o recurso, ou; utilizar WS-* em sistemas distribuídos dentro de uma mesma organização). No guia supracitado, o foco das diretrizes está na abordagem WS-* e poucas informações são apresentadas para a avaliação de sistemas baseados em REST. Além disso, na literatura, não existem guias específicos para a avaliação de arquiteturas baseadas no estilo REST.

1.1 Motivação

REST é um estilo arquitetural para sistemas distribuídos. Esse estilo pode ser empregado como uma abordagem para desenvolver soluções que seguem uma arquitetura orientada a serviços. O estilo REST define um conjunto de princípios que, ao serem adotados, dão origem a sistemas RESTful [Richardson and Ruby 2007]. Os sistemas RESTful são menos acoplados, mais leves, eficientes e flexíveis do que os sistemas Web baseados em

---

[4] *Software Engineering Institute*. Disponível em http://www.sei.cmu.edu/
[5] Acrônimo para o inglês *Service-Oriented Architecture*
[6] Padrões especificados pela *Web Services Interoperability Organization* (WS-I) seguindo o protocolo SOAP (do inglês *Simple Object Access Protocol*, refere-se a um protocolo para troca de informações baseado em XML)

WS-* e podem ser facilmente reutilizados. Além disso, os princípios REST podem ser mapeados nos métodos básicos do protocolo HTTP (GET, POST, UPDATE e DELETE) para criar sistemas CRUD (*Create*, *Read*, *Update*, *Delete*) de uma aplicação RESTful. Os recursos dos sistemas RESTful são identificados e encapsulados por um URI[7]. A utilização do protocolo HTTP, como protocolo de aplicação, admite que os recursos possuam várias representações e permite que os clientes selecionem, dentre as representações disponíveis, aquela que melhor se adeque as necessidades da aplicação.

Muitos sistemas baseados na web hoje em dia usam o estilo REST. REST foi originalmente proposto como um estilo de arquitetura de software para sistemas distribuídos na tese de doutorado de Roy Fielding [Fielding 2000]. Fielding apresenta um conjunto de restrições REST, como a Interface Uniforme e o estilo Cliente-Servidor, descritos como restrições de *design* do estilo arquitetural REST. Usando REST, alguns atributos de qualidade, tais como a interoperabilidade e modificabilidade, são impactados positivamente, ao passo que outros, como desempenho e confiabilidade podem ser afetados negativamente. Ao se avaliar arquiteturas baseadas no estilo REST, avaliadores devem inspecionar elementos e decisões de *design* da arquitetura que influenciam a capacidade do sistema para atender aos requisitos de atributos de qualidade. Esta tarefa exige uma compreensão clara dos *tradeoffs* entre os atributos de qualidade. Neste contexto, uma questão recorrente entre gerentes de projeto, arquitetos e avaliadores arquiteturais é: "qual é o impacto do estilo arquitetural REST na minha arquitetura?" [Naab et al. 2013].

Para responder a esta questão e realizar uma avaliação arquitetural que apresente uma real visão dos riscos das decisões de design REST nos atributos de qualidade, mostra-se necessário um guia com diretrizes que auxiliem avaliadores arquiteturais em suas atividades. Embora existam guias que apresentem algumas informações sobre questões de *design* REST, eles não fornecem uma análise aprofundada sobre o impacto das restrições REST sobre os requisitos de atributos de qualidade nem uma verificação acerca do *tradeoff* entre eles. Além disso, não existem guias disponíveis na literatura que ofereçam uma análise baseada em cenários, mesmo este tipo de análise ser amplamente utilizada por avaliadores arquiteturais.

[7] Acrônimo para o inglês *Uniform resource identifier*: Uma cadeia de caracteres usada para identificar um recurso na web.

1.2 Objetivo

O objetivo desta dissertação é fornecer um guia [Costa et al. 2014] contendo diretrizes para as atividades de avaliação arquitetural baseada em cenários em sistemas que usam o estilo REST, auxiliando avaliadores na mitigação de riscos na realização de atributos de qualidade e análise do *tradeoff* entre eles. São discutidos vários aspectos do projeto e são fornecidas diversas informações acerca de importantes pontos de inspeção. É discutida, também, a forma como a arquitetura deve ser inspecionada pela equipe de avaliação, e são propostos os tipos de perguntas que devem ser feitas durante o processo. Para atingir esse objetivo uma pesquisa sistemática na literatura e entrevistas com especialistas foram realizadas seguindo um protocolo de pesquisa estabelecido. Para validação, o guia foi enviado a experientes avaliadores arquiteturais do SEI, que o utilizaram e responderam a um questionário, demonstrando a validade das diretrizes propostas. Além disso, neste trabalho também é apresentada uma prova de conceito, demonstrando como as diretrizes são utilizadas no método de avaliação arquitetural ATAM.

1.3 Organização do trabalho

O restante deste trabalho está organizado da seguinte forma: o Capítulo 2 apresenta a fundamentação teórica. O Capítulo 3 descreve o protocolo de pesquisa utilizado para desenvolver este trabalho. O Capítulo 4 apresenta os trabalhos relacionados. No Capítulo 5, é apresentado o guia, composto pelos os fundamentos arquiteturais do estilo REST do ponto de vista de um avaliador arquitetural; cenários de atributos de qualidade gerais REST, e; questões de *design* que afetam os atributos de qualidade. A prova de conceito que utiliza as diretrizes apresentadas neste trabalho é descrito no Capítulo 6. No Capítulo 7 são apresentados os resultados da avaliação da utilização do guia por parte de avaliadores arquiteturais. Finalmente, o Capítulo 8 apresenta os trabalhos em andamento e futuros, e as considerações finais.

2 Fundamentação teórica

2.1 Arquitetura de Software e Atributos de Qualidade

A arquitetura de um software está relacionada com a organização fundamental de seus componentes (módulos, camadas, classes, serviços, *threads*, etc.) e os princípios que regem o design, implementação e evolução [ISO/IEC 2011]. Tais princípios definem as funcionalidades, atributos de qualidade e restrições, identificados no levantamento de requisitos, que estarão presentes no sistema. A qualidade do produto de software será o resultado da arquitetura concebida e desenvolvida. A arquitetura, portanto, atua como uma ponte entre os requisitos e a qualidade final do sistema.

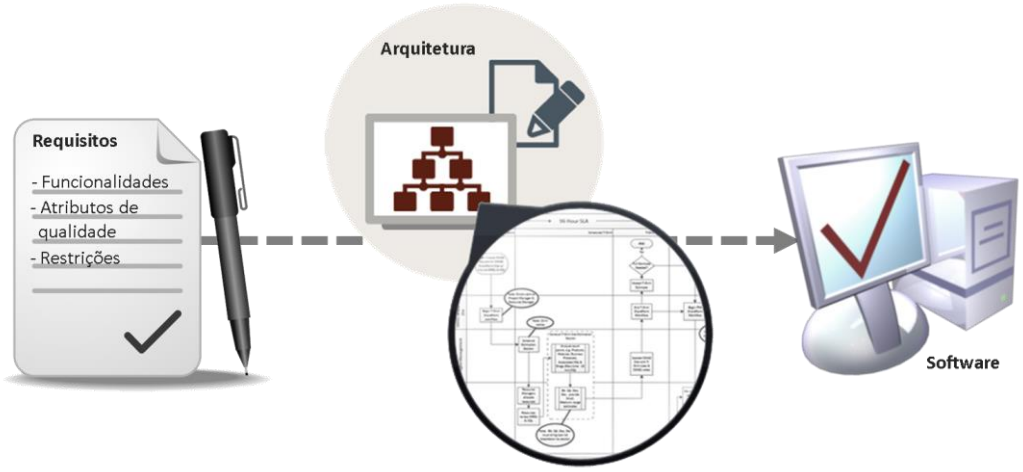


Figura 1. A Arquitetura como ponte entre os Requisitos e o Software

As restrições de um sistema são decisões, usualmente externas à equipe de desenvolvimento, impostas à arquitetura. Por exemplo, a linguagem de programação e outras tecnologias a serem utilizadas. Uma funcionalidade caracteriza-se como uma necessidade a ser atendida do domínio onde o software está sendo construído. A especificação das funcionalidades está entre os primeiros passos da especificação do sistema e orientará grande parte das decisões arquiteturais. Porém, a qualidade final vai além da funcionalidade. Sistemas são frequentemente remodelados não devido à deficiência funcional, mas porque são difíceis de manter, difíceis de escalar, não são portáveis, ou muito lentos [Bass et al. 2012]. Estas dificuldades e problemas estão relacionados aos atributos de qualidade.

Um atributo de qualidade é uma propriedade mensurável ou testável de um sistema que é utilizada para indicar quão bem o software satisfaz aos seus requisitos [Bass et al. 2012]. Por exemplo, na funcionalidade “quando o usuário pressionar o botão o relatório deverá ser

gerado e impresso", o atributo de qualidade *desempenho* poderá estar associado indicando a velocidade que o sistema deve gerar e imprimir o relatório. O documento "*Software engineering -- Product quality*" [ISO/IEC 2007] apresenta cinco atributos de qualidade e suas características, além do atributo funcionalidade:

1. **Confiabilidade**: capacidade do software de manter o nível de desempenho especificado, quando usado sob condições específicas;
2. **Usabilidade**: capacidade do software ser entendido, aprendido e atraente ao usuário, quando usado sob condições específicas;
3. **Desempenho**: capacidade do software de fornecer a velocidade de resposta adequada, relacionado à quantidade de recursos usados, sob condições estabelecidas;
4. **Manutenibilidade**: capacidade do software de ser modificado. As modificações podem incluir correções melhorias ou adaptações do software a mudanças no ambiente ou nos requisitos;
5. **Portabilidade**: capacidade de um produto de software ser transferido de um ambiente para outro;

Além dos atributos supracitados, diversos outros atributos de qualidade são importantes para a qualidade final do produto de software e indicam o nível de satisfação aos requisitos funcionais [Barbacci et al. 1995; Bass et al. 2012; Gorton 2011]:

6. **Interoperabilidade**: capacidade do software de trocar informações úteis e compreensíveis com outros sistemas;
7. **Disponibilidade**: capacidade do software de manter-se em operação mesmo em caso de falhas;
8. **Segurança**: capacidade do software de proteger suas informações de acessos não autorizados, e prover acesso a pessoas e sistemas com autorização de acesso;
9. **Escalabilidade**: capacidade do software de assimilar uma carga crescente de requisições, funcionando corretamente mesmo com alto volume de utilização;
10. **Extensibilidade**: capacidade do software de estender suas funcionalidades;
11. **Adaptabilidade**: capacidade do software de se adaptar a diferentes ambientes sem a necessidade de ações adicionais, como configurações;

12. **Testabilidade**: capacidade para localizar erros, defeitos e falhas no software através de testes;
13. **Auditabilidade**: capacidade do software de manter informações das ações executadas informando quem executou a ação, de onde, como e quando foram executadas.

Os treze atributos de qualidade apresentados fornecem, portanto, uma qualificação para os requisitos funcionais do sistema. Na especificação arquitetural os componentes deverão ser projetados de forma a atender aos requisitos funcionais e atributos de qualidade, e uma avaliação arquitetural poderá ser aplicada para verificar os riscos e problemas presentes na especificação.

2.2 Avaliação Arquitetural de Software

A Verificação e Validação de software tem por objetivo verificar se o software cumpre a sua especificação e atende às necessidades dos usuários e clientes. A Inspeção e Revisão Técnica concentra a inspeção em nível arquitetural e de implementação. Presente nesta disciplina, a Avaliação Arquitetural de Software foca na inspeção dos elementos arquiteturais analisando sua conformidade aos requisitos de atributos de qualidade [Bass et al. 2012].

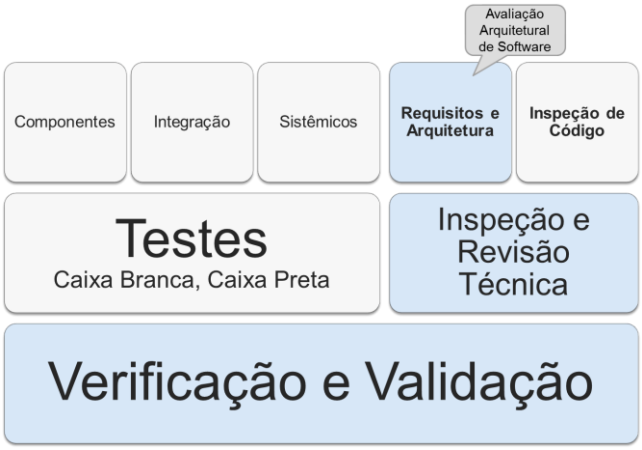


Figura 2. Inspeção e Revisão Técnica na disciplina de Verificação & Validação de Software [Pressman 2009; Sommerville 2010]

A avaliação da arquitetura pode ser realizada em duas oportunidades: antes da implementação (*Early Evaluation*) e depois da implementação (*Late Evaluation*) [Clements et al. 2001; Roy and Graham 2008]. O objetivo da avaliação da arquitetura antes da implementação é assegurar que os componentes especificados atendam aos requisitos, localizando problemas antecipadamente à implementação. Por sua vez, assegurar que a

arquitetura projetada foi implementada, juntamente com os desvios ocasionados, é o objetivo da avaliação arquitetural de software após sua implementação.

Existem diversos métodos de avaliação arquitetural de software [Dobrica and Niemela 2002]. Em comum todos os métodos requerem a participação de *stakeholders*, requisitos de atributos de qualidade, e a documentação da arquitetura [Roy and Graham 2008]. Tais métodos consistem basicamente na descrição de um processo sistemático para a verificação da arquitetura com o objetivo de avaliar a realização da especificação de requisitos. Os principais métodos utilizados para a avaliação arquitetural antes da implementação utilizam cenários como ponto de partida.

2.2.1 Cenários

Um cenário é uma breve descrição que expressa uma instância particular dos atributos de qualidade importantes para o sistema [Kazman et al. 1996]. Atributos de qualidade podem ser descritos em dois tipos de cenários [Bianco et al. 2007]: (i) cenários gerais de atributos de qualidade (descritos nesta dissertação como "cenários gerais"), que são independentes de software e podem ser aplicados a qualquer sistema, e; (ii) cenários concretos de atributos de qualidade (ou apenas "cenários concretos"), que são específicos a um sistema particular. A especificação dos cenários concretos pode ser feita em um formulário com seis itens [Bass et al. 2012], a saber, *Fonte do Estímulo*, *Estímulo*, *Ambiente*, *Artefato*, *Resposta*, *Medição da Resposta*. A Tabela 1 apresenta a descrição de tais itens.

Tabela 1 Formulário de especificação de atributos de qualidade

Fonte do Estímulo	Alguma entidade (um usuário humano, um sistema computacional ou outro atuador) que gerou o estímulo.
Estímulo	Uma condição que requer uma resposta quando chega ao sistema.
Ambiente	As condições para a ocorrência do estímulo.
Artefato	Algum artefato estimulado.
Resposta	Uma atividade realizada como resultado da chegada do estímulo.
Medição de resposta	Quando a resposta ocorre deve ser medida em alguma forma, de modo que o requisito possa ser testado.

Na Tabela 2 é apresentado um exemplo do cenário geral para os atributos de qualidade *Interoperabilidade* e *Disponibilidade*.

Tabela 2 Exemplos da especificação de cenários gerais de atributos de qualidade [Bass et al. 2012]

Atributo	Especificação
Interoperabilidade	**Fonte do estímulo**: Um sistema inicia uma requisição para interoperar com outro sistema; **Estímulo**: Uma requisição para trocar informações entre sistemas; **Artefato**: O sistema que deseja interoperar; **Ambiente**: O sistema que deseja interoperar é descoberto ou é conhecido em tempo de execução; **Resposta**: A requisição para interoperar resulta em troca de informações compreensíveis semanticamente e sintaticamente. Alternativamente a requisição é rejeitada e as entidades responsáveis são notificadas. **Medição de resposta**: O percentual de troca de informações processadas corretamente ou o percentual de requisições rejeitadas.
Disponibilidade	**Fonte do estímulo**: Interna ou externa ao sistema **Estímulo**: Uma falha ocasionada por: (i) omissão, um componente falha em responder a uma entrada; (ii) *crash*, um componente repetidamente falha a responder uma entrada; (iii) lentidão, um componente demora a responder; (iv) resposta, um componente responde com um valor incorreto. **Artefato**: O componente que é requerido para ser disponível. **Ambiente**: O estado do ambiente quando a falha ocorre; **Resposta**: Existem diversas possibilidades de respostas para uma falha de sistema. Por exemplo, uma falha deve ser detectada e isolada corretamente de forma transparente ao usuário. **Medição da resposta**: Especificada por porcentagem ou pelo tempo de detecção e reparação da falha

Cenários gerais podem ser especificados com uma estrutura enxuta bem formatada de estímulo e resposta, denominado por Bass et al. (2011) como *Anti-Scenarios*. Por exemplo:

$$\underbrace{\textit{O sistema recebe um estímulo}}_{\textbf{Estímulo}}\ \text{e}\ \underbrace{\textit{alguma resposta desejável é observada}}_{\textbf{Resposta}}.$$

Por exemplo, o cenário geral para o atributo de qualidade *desempenho* pode ser:

O usuário seleciona a opção e *o sistema responde em menos de 5 seg.*

Estímulo **Resposta**

Os cenários gerais e cenários concretos fornecem a especificação de requisitos para os métodos de avaliação arquitetural que se baseiam em cenários.

2.2.2 Métodos de Avaliação Baseados em Cenários

Métodos de avaliação arquitetural baseados em cenários avaliam a arquitetura do software segundo um conjunto de cenários de interesse. A utilização de cenários tem se mostrado como um solução eficaz para avaliações de arquitetura antes da implementação [Roy and Graham 2008].

Métodos baseados em cenários apresentam seis atividades em comum, descritas abaixo. Algumas atividades podem ser feitas em paralelo, conforme mostrado no diagrama de atividades da Figura 3.

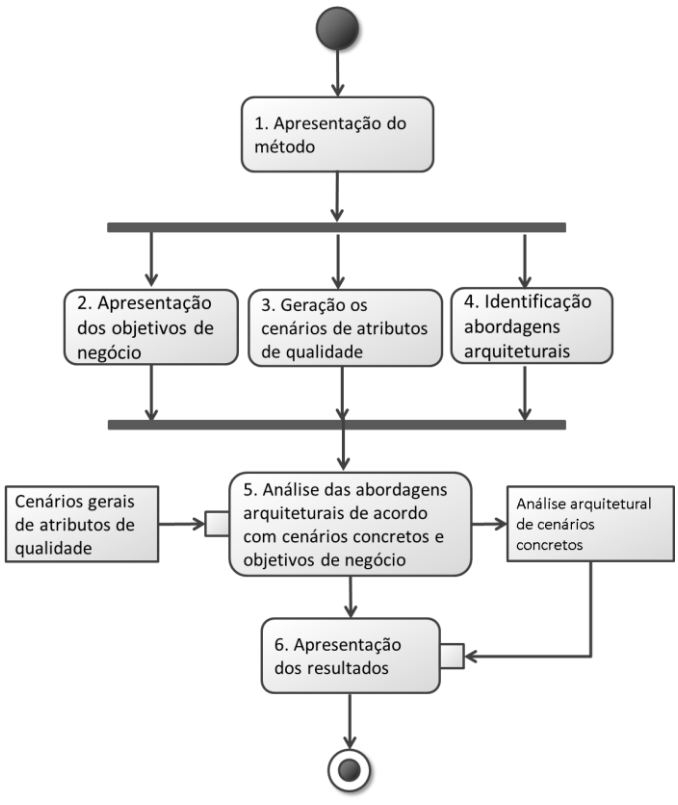


Figura 3 Principais atividades em métodos de avaliação arquitetural baseada em cenários

1. **Apresentação do método**: A equipe de avaliação apresenta as etapas do método, técnicas utilizadas e as saídas do processo de avaliação;
2. **Apresentação dos objetivos de negócio**: Os *stakeholders* apresentam os objetivos de negócio para a arquitetura;
3. **Geração os cenários de atributos de qualidade**: O arquiteto do software apresenta os requisitos de atributos de qualidade descritos como cenários concretos;
4. **Identificação abordagens arquiteturais**: O arquiteto do software apresenta a arquitetura do sistema e a equipe de avaliação organiza as abordagens arquiteturais em itens importantes para a avaliação;
5. **Análise das abordagens arquiteturais de acordo com cenários concretos e objetivos de negócio**: A equipe de avaliação inspeciona as abordagens arquiteturais organizadas na atividade anterior de acordo com os cenários concretos e objetivos de negócio, e gera a *Análise Arquitetural dos cenários*. Nesta atividade, a equipe de avaliação utiliza cenários gerias de atributos de qualidade e confronta-os com as abordagens arquiteturais com questões de *design* baseadas no conhecimento empírico ou em um guia contendo diretrizes para a avaliação arquitetural.
6. **Apresentação dos resultados**: A equipe de avaliação recapitula as atividades realizadas e apresenta a análise arquitetural.

A seguir será demonstrado um método baseado em cenários que apresenta estas atividades em seu processo.

2.2.3 *Architecture Tradeoff Analysis Method* (ATAM)

Um dos métodos de avaliação arquitetural mais utilizado atualmente foi proposto em 1998 pelo SEI. O ATAM [Kazman et al. 1998, 2000] é um método de avaliação arquitetural baseado em cenários que inspeciona a arquitetura com base no *tradeoff* de atributos de qualidade. O método é amplamente utilizado, e, por mais de dez anos tem sido aplicado na indústria como solução típica para avaliação arquitetural [Li and Zheng 2013]. Participam da avaliação ATAM três grupos cooperantes:

- **A equipe de avaliação**: grupo de três a cinco membros coordenados por um avaliador experiente denominado *ATAM Team Leader*[8], externos ao projeto onde a arquitetura será avaliada,
- **Gerentes de projeto e outros *stakeholders***: membros do projeto que tem autoridade para mudanças ou decisões de gerência acerca do projeto, e pessoas interessadas no projeto;
- **Arquitetos**: membros do projeto responsáveis pela arquitetura do software.

O método ATAM foi criado para localizar riscos e *tradeoffs* refletidos em decisões arquiteturais relacionados a requisitos de atributos de qualidade, antes da implementação do sistema. Para isso o método apresenta nove atividades:

1. **Apresentação do método ATAM**: a equipe de avaliação apresenta uma visão geral do ATAM, as técnicas utilizadas e as saídas do processo;
2. **Apresentação dos objetivos de negócio**: os gerentes de projeto apresentam os direcionadores e objetivos de negócio;
3. **Apresentação da arquitetura**: os arquitetos apresentam uma visão geral da arquitetura;
4. **Identificação das abordagens arquiteturais**: os *stakeholders* e arquitetos identificam as abordagens de *design* (estrutura organizacional dos componentes) da arquitetura apresentada na atividade anterior;
5. **Geração da árvore de atributos de qualidade**: os gerentes de projeto e *stakeholders* identificam e priorizam atributos de qualidade em formato de árvore de atributos de qualidade e cenários concretos;
6. **Análise das abordagens arquiteturais**: a equipe de avaliação arquitetural inspeciona as abordagens arquiteturais de acordo com os atributos de qualidade identificados e priorizados na atividade anterior para identificar riscos e *tradeoffs*;
7. ***Brainstorm* para priorizar cenários**: os *stakeholders* priorizam cenários concretos que especificam os atributos de qualidade desejados na aplicação;

[8] Informações disponíveis em: http://certification.cmmiinstitute.com/get-certified/software-architecture/architecture-tradeoff-analysis-method/

8. **Análise das abordagens arquiteturais**: a equipe de avaliação arquitetural continua a identificar riscos e *tradeoffs* nas abordagens para cada cenário concreto priorizado na atividade anterior;
9. **Apresentação dos resultados**: a equipe de avaliação recapitula as atividades do método, as saídas e apresenta os resultados da análise arquitetural.

Na atividade 5 é gerada a árvore de atributos de qualidade. Árvores de atributos de qualidade provêm um mecanismo *top-down* para identificar de maneira eficiente os atributos de qualidade e sua relação com o sistema [Kazman et al. 2000]. Elas auxiliam na construção dos cenários concretos, não sendo obrigatórias para o ATAM. Por exemplo, em um sistema de compras on-line onde o atributo Segurança seja necessário, a árvore de atributos de qualidade será descrita conforme a Figura 4. A Tabela 3 apresenta a especificação do atributo em um cenário concreto criado com base na árvore de atributos de qualidade.

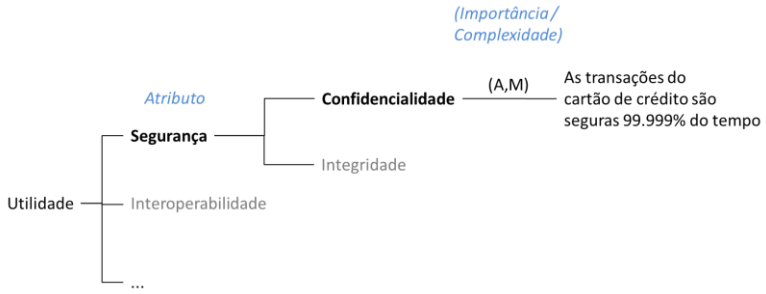


Figura 4 Árvore de atributos de qualidade

A árvore de atividades da Figura 4 contém *Utilidade* como nó raiz. Os nós imediatamente abaixo de utilidade apresentam os atributos de qualidade desejados no sistema. Abaixo dos atributos de qualidade, estão os itens desejados específicos do atributo. Por fim, é descrito como o item do atributo de qualidade será realizado no sistema. Entre o item e a descrição da realização no sistema, é descrito a importância e complexidade em uma escala de alta (A), média (M) ou baixa (B). No exemplo, a confidencialidade apresenta uma importância alta e complexidade média.

Tabela 3 Exemplo de cenário concreto para o atributo de qualidade Segurança.

Atributo	Especificação
Cenário 1. Segurança	**Fonte do estímulo**: Interface de utilização do usuário; **Estímulo**: Um usuário requisita uma compra por cartão de crédito; **Artefato**: Componente de gerenciamento de transações CT1; **Ambiente**: Operação normal;

	Resposta: A requisição é enviada à operadora do cartão de crédito que retorna o status ao componente CT1. CT1 envia a confirmação para a interface, que por sua vez, informa ao usuário; **Medição de resposta**: O sistema confirma o usuário e o valor da transação antes da requisição à operadora. Na resposta, após a efetivação da transação, é verificado o valor debitado e o usuário. O sistema não permite valores ou usuários diferentes daqueles que iniciaram a requisição.

Nas atividades 6 e 8 a equipe de avaliação arquitetural confronta os cenários concretos com as abordagens arquiteturais especificadas. O resultado é um documento descrevendo os riscos para a realização dos atributos de qualidade especificados nos cenários concretos, assim como os *tradeoffs* entre eles. O documento é formado pelos itens: *Resumo do cenário*, *Objetivos de Negócio*, *Atributos de Qualidade*, *Análise Arquitetural*, *Riscos* e *Tradeoffs* [Clements et al. 2001]. A Tabela 4 descreve os itens do documento.

Tabela 4 Documento de avaliação arquitetural.

Resumo do cenário	Um breve resumo do cenário concreto avaliado.
Objetivos de Negócio	Os objetivos de negócio que o cenário concreto objetiva realizar.
Atributo de Qualidade	Atributos de qualidade impactados.
Análise Arquitetural	As principais considerações acerca das abordagens de design em relação ao cenário concreto avaliado.
Riscos	Os riscos que as abordagens de design trazem aos atributos de qualidade.
Tradeoffs	Os *tradeoffs* entre os atributos de qualidade impactados.

Um exemplo do documento de análise arquitetural para o atributo *segurança*, especificado no cenário concreto da Tabela 3, é apresentado na Tabela 5.

Tabela 5 Análise arquitetural do Cenário 1.

Resumo do cenário	Um usuário solicita a compra por cartão de crédito via interface. O componente de gerenciamento de transações envia a solicitação à operadora e responde o status para a interface que, por sua vez, informa ao usuário.

Objetivos de Negócio	Permitir a compra segura com cartão de crédito
Atributo de Qualidade	Segurança, interoperabilidade
Análise Arquitetural	- O sistema especifica um componente para validação do usuário antes da solicitação à operadora do cartão. Esta validação promove segurança.
Riscos	- O módulo de Gerenciamento de Transação não é especificado para operar sob HTTPS.
Tradeoffs	A utilização dos padrões especificados na arquitetura promove segurança, porém impacta negativamente na Interoperabilidade, uma vez que alguns padrões não são amplamente utilizados.

Para a geração do documento de avaliação arquitetural, é comum a utilização de guias com diretrizes que auxiliam os avaliadores acerca da forma como a arquitetura deve ser inspecionada, assim como os tipos de perguntas que devem ser feitas durante o processo.

2.3 Guias

Um guia é um documento que apresenta diretrizes relevantes para um determinado contexto ou problema. Estas diretrizes provêm de duas fontes principais, a saber, conhecimento empírico de especialistas e, da literatura. Os guias têm por especificidade a concisão e completude: devem apresentar um formato que permita tanto uma breve consulta quanto a leitura integral. Além disso, devem apresentar a metodologia utilizada para a compilação das diretrizes e são orientados a contexto ou problema específicos [ALA 2013].

2.3.1 Guia para Avaliação Arquitetural

Na avaliação arquitetural, a inspeção dos riscos à realização dos atributos de qualidade é realizada, em grande parte, com base no conhecimento empírico dos avaliadores. Porém diversos padrões e tecnologias apresentam desafios aos avaliadores, visto serem de recente especificação ou impactarem em muitos atributos de qualidade. Neste contexto os guias tem o objetivo de auxiliar as equipes de avaliação arquitetural, *stakeholders* e arquitetos nas atividades da inspeção. Tais guias trazem diretrizes importantes e específicas para o estilo ou padrão arquitetural utilizado na especificação da arquitetura inspecionada. Em geral, para avaliação arquitetural baseada em cenários tais guias devem apresentar:

1. **Descrição do estilo ou padrão arquitetural sob a ótica da avaliação arquitetural**: apresentam os fundamentos e confrontando-os com os respectivos impactos nos atributos de qualidade;
2. **Questões de *design***: questões que avaliadores devem analisar quando inspecionar a arquitetura;
3. **Cenários gerais de atributos de qualidade**: cenários gerais de atributos de qualidade que demonstram cenários de realização dos atributos de qualidade segundo o estilo arquitetural;
4. **Exemplo de avaliação arquitetural**: uma breve demonstração da utilização do guia em uma avaliação, através da utilização de um método de avaliação arquitetural.

Na avaliação arquitetural, os guias de orientação são utilizados em diversas atividades. No método ATAM, por exemplo, eles são utilizados na atividade 5 (Gerar a árvore de atributos de qualidade), onde *stakeholders* podem utilizar os cenários gerais dos guias para auxiliá-los na geração da árvore e, principalmente, na especificação dos cenários concretos. Nas atividades 6 e 8 (Analisar as abordagens arquiteturais), avaliadores arquiteturais utilizam os guias para auxiliá-los nas questões que devem ser tratadas na arquitetura avaliada.

3 Trabalhos Relacionados

O guia proposto por Becker and Diaz-Herrera (1994) descreve uma abordagem para a avaliação de bibliotecas para reuso de componentes de software de acordo com o *tradeoff design-com-reuso* e *design-para-reuso*. Proposto por Wozniak et al. (2011), o guia AUTOSAR (*AUtomotive Open System ARchitecture*) apresenta diversas diretrizes para análise do processo de *design* de sistemas no contexto automotivo. Ele define uma arquitetura de referência, a distribuição dos recursos de *hardware*, dentre outros. Além disso, propõe atividades que devem estar presentes no processo de desenvolvimento de sistemas de software para automóveis.

O SEI apresenta uma prática comum na publicação de guias. Algumas das principais publicações são: *Guidelines for Software Engineering Education* [Hilburn et al. 1999], que apresenta diretrizes para o currículo de cursos de Engenharia de Software; *Software Acquisition Planning Guidelines* [Novak et al. 2005], que apresenta diretrizes para o plano de aquisição de software, e; *Evaluating a Service-Oriented Architecture* [Bianco et al. 2007], que contém informações técnicas sobre considerações de *design* SOA, impactos do estilo arquitetural na realização de atributos de qualidade e *tradeoffs* entre eles. Tais guias são oferecidos gratuitamente e amplamente utilizados.

Os guias apresentados acima descrevem de forma concisa e completa as informações para os respectivos contextos. Porém, nenhum deles descreve a metodologia para a aquisição das informações utilizadas para a compilação das diretrizes. Este trabalho diferencia dos anteriores por duas contribuições principais: (i) ele apresenta de forma detalhada a metodologia para a construção do guia e, (ii) o foco das diretrizes está no estilo arquitetural REST.

Esta dissertação foi inspirada no trabalho de Bianco et al. (2007), citado acima. O trabalho apresenta um guia contendo diretrizes acerca de considerações de *design* SOA e *tradeoffs* que auxiliam o avaliador arquitetural na identificação e mitigação de riscos. Segundo entrevistas realizadas com avaliadores ATAM do SEI (ver seção 4.2.1), o guia é amplamente utilizado atualmente na indústria e academia. O guia segue uma estrutura definida segundo as atividades do método ATAM. Para isso, apresenta uma introdução do estilo SOA, descrevendo seus princípios do ponto de vista de avaliadores arquiteturais; uma introdução ao método ATAM, descrevendo os participantes do processo, as atividades e resultados esperados; abordagens arquiteturais para a implementação de sistemas baseados no

estilo, nesta seção o trablaho apresenta uma breve descrição acerca do estilo REST; questões de design SOA que afetam atributos de qualidade; cenários gerais de atributos de qualidade no contexto SOA, e; um exemplo de avaliação arquitetural, baseada no método ATAM, em um sistema que utiliza princípios SOA, seguindo as diretrizes descritas no guia. As questões de *design* e cenários de atributos de qualidade focam fortemente na concepção de serviços baseados na abordagem WS-*.

O presente trabalho segue a estrutura do guia proposto por Bianco et al. (2007). Porém, apresenta um diferencial por focar em uma análise aprofundada sobre o impacto das restrições do estilo REST sobre os requisitos de atributos de qualidade. Além disso, é apresentada de forma detalhada a metodologia utilizada para a formulação das questões de *design* e cenários gerais.

O livro de Erl et al. (2012) é considerado como referência na literatura de arquitetura de software sobre abordagens REST para SOA. Ele apresenta diversas diretrizes para especificação de sistemas REST sob uma perspectiva de serviços de negócio. No livro, atributos de qualidade são analisados confrontando-os com princípios REST, apresentados os riscos para a realização dos atributos de qualidade e ações para mitigar tais riscos. Porém, o livro não apresenta uma análise baseada em cenários de atributos de qualidade nem a metodologia para a definição dos pontos de inspeção e riscos.

Algumas formulações do presente trabalho basearam-se no livro de Erl et al. (2012), principalmente no tocante as questões relacionadas às abordagens REST para serviços. Porém, as diretrizes propostas foram confrontadas com uma análise sistemática através de entrevistas com especialistas (ver Capítulo 4). Por fim, as formulações baseadas no livro de Erl et. al foram modificadas ou adaptadas para atender as especificações de um guia genérico para REST (em detrimento de um guia REST no contexto de SOA) e, descritas conforme cenários de atributos de qualidade para serem utilizadas em métodos de avaliação arquitetural.

4 Metodologia

4.1 Introdução

Este capítulo descreve o protocolo para a construção do guia para avaliação arquitetural REST. O protocolo foi definido com três objetivos: (1) compilar as informações relevantes acerca do *design* de sistemas baseados em REST por parte de diversos profissionais da indústria e academia e, da literatura; (2) confrontar as informações compiladas com especialistas, e; (3) validar o guia proposto através de: (i) uma prova de conceito, que apresenta a utilização do guia em um método de avaliação arquitetural, e; (ii) validação do guia completo por experientes avaliadores arquiteturais.

4.2 Atividades

O guia proposto neste trabalho foi desenvolvido seguindo um protocolo proposto ilustrado na Figura 5. As subseções seguintes descrevem cada atividade do protocolo.

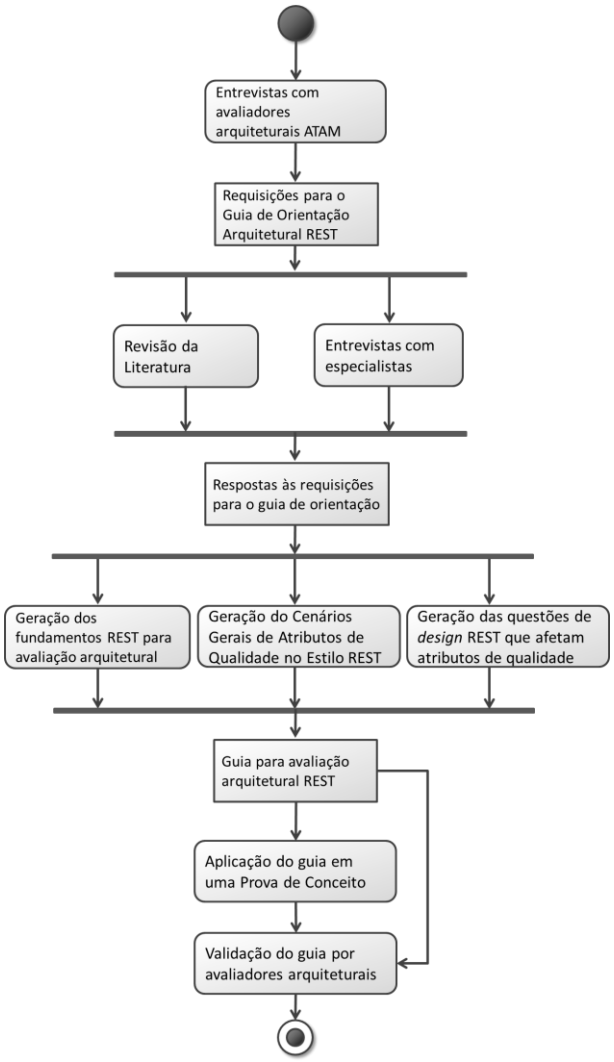


Figura 5 Atividades do Protocolo de Pesquisa

4.2.1 Entrevistas com Avaliadores Arquiteturais

A primeira atividade do protocolo consistiu em entrevistas com experientes avaliadores arquiteturais com objetivo de reunir o conjunto inicial de questões REST que eles considerariam importantes para o guia. Seguindo uma metodologia sistemática [Oishi 2002], foram entrevistados dois avaliadores com experiência de mais de 50 avaliações utilizando o método ATAM. Com dois avaliadores entrevistados, foi possível comparar e analisar suas necessidades comuns. Com o resultado das entrevistas, os avaliadores indicaram que um guia para avaliação arquitetural REST deveria abordar três questões principais. No protocolo, estas questões são chamadas de *Requisições para o Guia para Avaliação Arquitetural REST (RGs)*:

> **RG1**- "Explicar os fundamentos do estilo REST do ponto de vista de um avaliador arquitetural";
>
> **RG2**- "Apresentar os cenários gerais de atributos de qualidade segundo o estilo REST";
>
> **RG3**- "Discutir em detalhes como o REST contribui para os atributos de qualidade e *tradeoffs* entre eles".

Após a definição do conteúdo necessário no guia, foram definidas as fontes de pesquisa onde seriam buscadas as informações: literatura (subseção 4.2.2) e conhecimento empírico de especialistas (subseção 4.2.3).

4.2.2 Revisão da literatura

A metodologia utilizada para a revisão da literatura foi baseada em [Kitchenham et al. 2009] que apresenta um conjunto de atividades bem definidas para a condução de revisão sistemática. Nas subseções seguintes são apresentadas as atividades realizadas.

4.2.2.1 Definição dos objetivos e questões de pesquisa

O objetivo da revisão sistemática é responder às requisições para o guia para avaliação arquitetural REST a partir do conhecimento teórico disponível. As questões de pesquisa (QP), portanto, são derivadas das três requisições RG1, RG2 e RG3:

QP1: “Quais os fundamentos do estilo arquitetural REST sob a ótica da avaliação arquitetural?”;

QP2: “Como os princípios REST impactam nos atributos de qualidade?”.

4.2.2.2 Fontes e *strings* de pesquisa

Após a definição das questões de pesquisa, foi realizada a busca para recuperar toda a literatura relevante que responda às questões de pesquisa definidas. Este objetivo foi alcançado em duas etapas, onde a primeira foi definir as fontes que podem fornecer os estudos relevantes, e o segundo passo foi a definição da *string* de pesquisa.

Para determinar as fontes de pesquisa, foram definidas as principais bibliotecas online utilizadas em Kitchenham et al. (2009). Para evitar a omissão de publicações importantes, o Google Scholar também foi utilizado como uma ferramenta suplementar. Finalmente, antes de iniciar as buscas subsequentes, foi confirmada a disponibilidade de cada biblioteca online. A lista final de fontes pesquisadas abrange as famosas bibliotecas digitais on-line, como mostrado na Tabela 6:

Tabela 6 Fontes de Pesquisa.

Fonte	Tipo	URL
ACM Digital Library	Biblioteca digital	http://dl.acm.org/dl.cfm
IEEE Xplore	Biblioteca digital	http://ieeexplore.ieee.org
ScienceDirect	Biblioteca digital	http://www.sciencedirect.com
SpringerLink	Biblioteca digital	http://link.springer.com/
Google Scholar	Motor de pesquisa	http://scholar.google.com

Depois de completar a lista de fontes, a próxima atividade foi definir os termos de pesquisa, bem como o procedimento para busca nas bibliotecas digitais online. Para criar as *strings* de pesquisa, primeiro foram selecionadas várias palavras-chave relacionadas às questões de pesquisa QP1 e QP2, organizadas em quatro grupos de termos de pesquisa, como mostrado na Tabela 7. Cada grupo contém os termos de busca que são ou sinônimos (diferentes formas da mesma palavra) ou que têm significado semântico similar. Cada grupo tem como objetivo recuperar diferentes conjuntos de estudos: o grupo 1 engloba o termo REST e seu nome completo em inglês; o grupo 2 contém o termo utilizado para pesquisar os atributos de qualidade; o grupo 3 engloba os termos relacionados à arquitetura, e finalmente; o grupo 4 contém os termos relacionados à avaliação arquitetural.

Tabela 7 Termos de Pesquisa Utilizados na Formulação da s*tring* da Pesquisa.

	Grupo 1	**Grupo 2**	**Grupo 3**	**Grupo 4**
Termo 1	rest	quality attribute	design	architecture evaluation
Termo 2	representational state transfer		architecture	architectural evaluation

Com base na organização dos termos de pesquisa, as *strings* de pesquisa ficaram conforme a seguir:

QP1: *(rest OR "representational state transfer") AND ("architecture evaluation" OR "architectural evaluation")*

QP2: *(rest OR "representational state transfer") AND "quality attribute" AND (design or architecture)*

4.2.2.3 Seleção e agrupamento dos artigos

As primeiras pesquisas com o foco em REST iniciaram-se no início dos anos 2000 com a tese de doutorado de Roy Fieldings [Fielding 2000]. Assim, o período utilizado como filtro para a pesquisa foi do ano 2000 a 2013. As *strings* de pesquisa foram utilizadas nas bibliotecas online e no final, foram retornados 384 artigos científicos publicados entre os anos 2000 e 2013, além de 8 livros e 12 relatórios técnicos.

Os artigos retornados foram submetidos a um critério de inclusão. O autor desta dissertação leu sistematicamente o título e resumo dos artigos, livros e relatórios técnicos retornados e selecionou aqueles que atendiam a um critério de seleção definido como: *"apresenta os fundamentos REST assim como sua relação com atributos de qualidade"*. Se não ficasse claro a partir do título e resumo se um artigo ao critério de seleção, todo o artigo era lido. No caso dos livros, somente capítulos relevantes eram revisados. Após este passo foram selecionados 42 artigos científicos, 8 livros e 9 relatórios técnicos.

Os trabalhos selecionados, livros e relatórios técnicos foram sistematicamente estudados para encontrar informações úteis para responder às questões de pesquisa. Em seguida, as informações foram agrupadas e sintetizadas segundo as requisições RG1, RG2 e RG3. A informação coletada neste passo foi utilizada para responder as requisições e auxiliou para a criação do questionário usado na atividade seguinte: entrevistas com especialistas.

4.2.3 Entrevistas com especialistas

Para a realização das entrevistas com especialistas foi selecionada uma amostra contendo diversos profissionais da indústria e pesquisadores com vasta experiência em desenvolvimento de soluções e pesquisa sobre REST. Sua experiência em projetos REST foi a segunda fonte do conhecimento para responder as requisições (RG1, RG2 e RG3) para o guia para avaliação arquitetural REST. A interação com estes especialistas seguiu a metodologia de Oishi (2002).

Inicialmente, foi desenvolvido um questionário com base na revisão da literatura realizada na atividade anterior. O questionário foi disponibilizado on-line e o link enviado para dez especialistas selecionados da indústria e da academia. Todos os especialistas têm trabalhado com REST por mais de 5 anos e têm experiência com 5 a 10 diferentes projetos baseados no estilo. Os entrevistados foram: três pesquisadores, três provedores de serviços REST, dois consultores e dois consumidores de serviços REST. Os pesquisadores têm publicado diversos artigos, participado ou organizado diversas conferências na área, como por exemplo *International Workshops on RESTful Design* (WS-REST)[9]. Provedores trabalham em médias e grandes empresas de TV, filmes pela Internet, meios de comunicação e os mapas on-line. Seus serviços REST são utilizados por dez a trinta milhões de clientes através de mais de mil tipos diferentes de dispositivos. Consultores têm trabalhado em diversas atividades do ciclo de vida do REST, como a concepção, exposição e definição de recursos. Os consumidores de serviços são os desenvolvedores que têm usado extensivamente serviços Web REST a partir de diversos provedores.

Após a aplicação do questionário e analisando as respostas, observou-se que os resultados não eram suficientes para serem usados como base para a geração do guia. Os especialistas indicaram que eles não conseguiram dispender os sugeridos trinta minutos necessários para responder ao questionário. Acredita-se que essa falta de tempo dos entrevistados foi a principal razão para o fraco *feedback* obtido inicialmente. Assim, decidiu-se realizar entrevistas com os especialistas, decisão também motivada pelo bom resultado da interação direta avaliadores arquiteturais da primeira atividade do protocolo. As entrevistas com especialistas também foram planejadas e executadas de acordo com a metodologia proposta por Oishi (2002). Em cada entrevista, as perguntas do questionário eram aplicadas e, com base nas respostas, novas perguntas foram incluídas. Além disso, os resultados da revisão

[9] Disponível em http://ws-rest.org/

sistemática e as respostas de outros entrevistados também eram apresentados com o objetivo de validar as informações e incluir novos dados importantes à entrevista.

4.2.4 Respostas à Avaliação Arquitetural REST

A revisão da literatura e entrevistas com especialistas produziram uma quantidade significativa de dados armazenados em tabelas de entrevistas, documentos de texto, anotações e trechos selecionados da literatura. A próxima atividade do protocolo foi a organização de todas as informações em um documento estruturado para gerar o guia. A seção 5 apresenta o resultado da pesquisa. As informações foram estruturadas de acordo com as requisições RG1, RG2 e RG3 para o guia para avaliação arquitetural REST apresentado nesta dissertação e publicado em nos idiomas inglês e português [Costa et al. 2014].

4.2.5 Validação do guia: Prova de Conceito e Validação do Guia por Avaliadores Arquiteturais ATAM

Com o guia desenvolvido, o próximo passo foi a realização de uma prova de conceito através da utilização do guia em uma avaliação arquitetural em um sistema que utiliza os princípios REST em sua concepção. A Seção 6 apresenta tal avaliação. O objetivo é apresentar como diretrizes podem auxiliar aos avaliadores na inspeção e mitigação de riscos. Por fim, o guia foi enviado a experientes avaliadores arquiteturais ATAM do SEI. Eles utilizaram o guia em cenários reais de avaliação arquitetural e responderam a um questionário que avaliou se o guia atingiu aos seus objetivos. A seção 7 apresenta os resultados da avaliação.

5 Guia para Avaliação Arquitetural de Sistemas Baseados no Estilo REST

5.1 Fundamentos REST para avaliação arquitetural (Requisição RG1)

5.1.1 Introdução

Um dos principais objetivos de uma avaliação arquitetural é inspecionar os riscos para atender aos requisitos de atributos de qualidade em uma arquitetura de software. As próximas seções descrevem os fundamentos do estilo REST, de acordo com o seu impacto nos atributos de qualidade. Os fundamentos apresentados foram baseados nas entrevistas com especialistas e na literatura, com foco em [Erl et al. 2012; Fielding 2000; Richardson and Ruby 2007; Vinoski 2007; Webber et al. 2010; Wilde and Pautasso 2011].

5.1.2 Restrições REST

Representational State Transfer (REST) é um estilo arquitetural de software para sistemas distribuídos [Fielding 2000]. Roy Fielding descreveu seis restrições que definem o estilo REST, cada uma das quais promove um diferente grupo de atributos de qualidade. REST pode ser descrita segundo a expressão (1)

(1) *REST = (C-S, S, $, U, L, CoD)*

Cada variável na expressão é uma restrição. A seguir, será descrita cada restrição juntamente com respectivo impacto sobre os atributos de qualidade e pontos importantes que os avaliadores devem inspecionar para avaliar este impacto. Na Seção 5.3 as restrições serão descritas em questões de design REST.

5.1.2.1 *Client-Server* (C-S, S, $, U, L, CoD)

Cliente-Servidor (do inglês *Client-Server*) é um estilo arquitetural frequentemente utilizado em aplicações baseadas em rede. Esse estilo descreve sistemas distribuídos que envolvem clientes distintos requisitando serviços de componentes do servidor através de uma conexão de rede. Em REST, as requisições são iniciadas por *agentes de usuários* ou *consumidores de serviço* (clientes) e processadas por um *servidor de origem* (servidor), que provê serviços através de uma hierarquia de recursos. O estilo REST tem um forte foco em dissociar cliente e servidor, no entanto, várias arquiteturas mostram desafios para definir quais módulos irão atuar como um cliente ou um servidor. Os principais benefícios do estilo Cliente-Servidor são: a separação de responsabilidades e, evolução e manutenção independente. Avaliadores devem inspecionar a definição da fronteira entre o cliente e o servidor de acordo com a coesão e evolução independente de cada um.

5.1.2.2 *Stateless* (C-S, S, $, U, L, CoD)

A restrição Sem Estado (do inglês *Stateless*) descreve que toda a informação necessária para compreender os dados do estado de conversação entre os servidores de origem e agentes de usuário devem ser incluídas na requisição e mensagens de resposta. Isso significa que, entre as requisições, nenhum estado de conversação é mantido no servidor, o estado é mantido apenas no cliente. Pontos de inspeção importantes estão relacionados ao fluxo de dados de estado do servidor. A restrição Sem Estado permite a replicação de servidores e, consequentemente, promove a disponibilidade, escalabilidade e confiabilidade. Porém, pode diminuir o desempenho devido à necessidade de enviar os dados de estado de conversação embutido nas requisições e mensagens de resposta.

5.1.2.3 *Cache* (C-S, S, $, U, L, CoD)

A restrição *Cache* é adicionada na arquitetura para melhorar seu desempenho. Um elemento de *cache* atua como um mediador entre o cliente e o servidor. O objetivo é evitar a interação de requisição e resposta em transações nas quais a informação já está presente no cache, evitando assim o tráfego de rede. Em soluções multicamadas (restrição *Layered System*, subseção 5.1.2.5), um *cache* pode também estar presente em níveis intermediários. Apesar do *cache* ser um recurso ordinário da World Wide Web, seu uso não é comum em serviços REST quando tais serviços são componentes SOA. Os avaliadores devem inspecionar quais os dados que devem ser armazenados em cache. O grau de aumento da eficiência da rede e, consequentemente, o desempenho do sistema, depende da estratégia de *cache*. No entanto, seu uso pode diminuir a confiabilidade em casos em que o cliente consuma dados antigos que estejam presentes no *cache*.

5.1.2.4 *Uniform Interface* (C-S, S, $, U, L, CoD)

A Interface Uniforme (do inglês *Uniform Interface*) entre os componentes é a característica fundamental que distingue REST de outros estilos baseados em rede [Fielding 2000]. A interface uniforme está intimamente relacionada a Recursos, Identificadores e Representações. Um Recurso é qualquer conceito importante no domínio do sistema que se deseja tornar acessível através de uma interface. Um consumidor de serviço aborda os recursos através de um Identificador único (por exemplo, URI) e uma representação. A Representação de um recurso refere-se a um documento (por exemplo, hipermídia) que traz todas as informações úteis sobre o estado dos recursos e *links* para outros recursos associados. Finalmente, a representação de um recurso deve ser acessada por uma interface simples que

define uma identificação para os recursos e métodos comuns de acesso (por exemplo, métodos HTTP PUT, GET, POST e DELETE). Essa interface representa a interface uniforme. A restrição Interface Uniforme impacta positivamente nos atributos de qualidade de interoperabilidade e descoberta.

5.1.2.5 *Layered System* (C-S, S, $, U, L, CoD)

A restrição do sistema em camadas (do inglês *Layered System*) como descrito por Fielding não é exatamente o padrão de camadas comumente encontrado na literatura sobre padrões. O padrão de arquitetura mais facilmente identificado com a restrição de camadas REST descrito por Fielding é multicamadas (do inglês *multi-tier*). O estilo Multicamadas é uma especialização do estilo cliente-servidor [Clements et al. 2010], onde uma camada intermediária atua como servidor para o nível anterior e como cliente para o nível seguinte. Fielding sugere que elementos em uma camada não devem "ver" além do próximo nível, portanto, contendo complexidade geral do sistema. Sistemas legados, bancos de dados e sistemas de *back-end* em geral, muitas vezes são colocados na camada mais inferior. Elementos em um nível intermediário podem ser de *load-balance*[10] para melhorar a disponibilidade e escalabilidade do sistema. No entanto, a implantação de componentes em várias camadas pode aumentar o tempo de resposta para uma determinada solicitação, devido à comunicação através das camadas. Apesar dos serviços REST simples poderem estar disponíveis no “lado do servidor” em arquiteturas do tipo cliente-servidor, tais serviços são frequentemente encontrados em arquiteturas multicamada desenvolvidas, por exemplo, utilizando Java EE, . NET, ou outra plataforma de implementação adequada para este tipo de aplicação.

5.1.2.6 *Code-on-Demand* (C-S, S, $, U, L, CoD)

Código sob demanda (do inglês *Code-on-Demand*) é uma restrição opcional que permite uma arquitetura dinâmica onde a lógica do agente de usuário pode ser estendida por código recebido do serviço. O código pode ser, por exemplo, um *applet*, função JavaScript utilizada em páginas da web para renderizar *Widgets*[11] ou, validação de entrada de dados de usuários. Por exemplo, considere um cliente que obtém, a partir do servidor, o código que é necessário para executar a criptografia nas mensagens. Código sob demanda é uma forma de viabilizar a execução onde a funcionalidade e o comportamento do agente usuário pode variar

[10] Técnica para distribuir a carga de trabalho uniformemente entre computadores
[11] Componente web de interface gráfica para o usuário

em tempo de execução segundo o código recebido em mensagens de um serviço REST. *CoD* promove extensibilidade. A aplicação no cliente é desprovida de código de execução, porém, a complexidade da implementação do cliente aumenta devido à necessidade de lidar com as adições de código de tempo de execução. A interoperabilidade pode diminuir já que o código recebido deve ser compatível com todos os consumidores do serviço. A segurança também é uma preocupação relacionada ao envio de códigos maliciosos para os agentes de usuário.

5.1.3 REST pragmático

As restrições mencionadas (C-S, S, $,U ,L, CoD) representam o que é chamado nesta dissertação de "REST Puro". Elas não definem uma tecnologia ou plataforma específica para a implementação de sistemas baseados no estilo REST. Uma vez que REST tornou-se um estilo arquitetônico típico para projetar e desenvolver sistemas distribuídos, várias soluções técnicas e frameworks de desenvolvimento têm sido propostos para implementação de serviços REST. Estas soluções são chamadas neste trabalho de "REST pragmático", pois elas se concentram em aspectos práticos de concepção e implementação de serviços que seguem o estilo de arquitetura REST.

REST Pragmático utiliza tecnologias da Internet, principalmente HTTP e hipermídia. As restrições (C-S, S, $,U ,L, CoD) podem ser aplicadas sobre HTTP e tecnologias relacionadas para implementar serviços web que são chamados serviços REST ou serviços RESTful [Richardson and Ruby 2007]. Arquitetos, desenvolvedores e avaliadores de soluções baseadas em REST devem conhecer o funcionamento do protocolo HTTP. O protocolo é especificado principalmente no documento RFC 2616 [Fielding et al. 1999]). No REST Pragmático, a restrição mais importante é a Interface Uniforme e seus conceitos relacionados: Recursos, Identificadores e Representações. Interface Uniforme distingue os serviços REST de outros serviços baseados na web, tais como serviços WS-*. O guia proposto nesta dissertação foi concebido com o foco no REST Pragmático.

5.2 Cenários Gerais de Atributos de Qualidade no Estilo REST (Requisição RG2)

Foram considerados dez atributos de qualidade nos cenários gerais REST extraídos da revisão da literatura e entrevistas, classificados como mais importantes pelos especialistas entrevistados na primeira atividade do protocolo de pesquisa. A descrição dos cenários seguiu a especificação estímulo x resposta (descrita na subseção 2.2.1). Os cenários gerais apresentados na Tabela 8 serão referenciados nas questões de design REST (subseção 5.3) e auxiliaram na construção dos cenários concretos (subseção 6.5).

Tabela 8 Cenários Gerais de Atributos de Qualidade no Contexto REST.

Atributo de Qualidade	Cenário
Interoperabilidade	**I1**- Um consumidor de serviço ‘A’ solicita um recurso ‘R1’ e recebe a representação do estado atual da ‘R1’ na mensagem de resposta. **I2**- Um consumidor de serviço ‘A’ solicita um recurso ‘R1’ em um formato específico (*media type*) e recebe a representação do estado atual de ‘R1’ na mensagem de resposta de acordo com o formato solicitado. **I3**- Um consumidor de serviço ‘A’ solicita um recurso ‘R1’ e é capaz de compreender todas as informações apresentadas na mensagem de resposta.
Confiabilidade	**R1**- Um consumidor de serviço ‘A’ solicita a versão v(n) do recurso ‘R1’, especificando diretamente na URI, e recebe a representação de ‘R1’ na versão v(n) na mensagem de resposta. **R2**- Com base no modelo de domínio, um consumidor de serviços 'A' constrói o URI do recurso 'R1' e recebe a representação do estado atual de 'R1' na mensagem de resposta.
Segurança	**S1**- Um consumidor de serviço ‘A’, com privilégios insuficientes, solicita uma informação confidencial à interface ‘X’ do recurso ‘R1’, ‘X’ nega a solicitação e informa ‘A’ sobre a falta de autorização. **S2**- Um consumidor de serviço autenticado e autorizado ‘A’ solicita um recurso confidencial ‘R1’, que tem acesso, e recebe a representação do estado atual de ‘R1’ na mensagem de resposta.
Testabilidade	**T1**- Um desenvolvedor quer testar um serviço. Se ocorrer um erro ao processar o pedido, o serviço pode ser configurado para fornecer na mensagem de resposta todas as informações para identificar o erro, incluindo o rastreamento da execução.
Desempenho	**P1**- Um consumidor de serviço ‘A’ realiza uma ação em um recurso ‘R1’ e recebe o status de ‘R1’ na mensagem de resposta em menos de n milissegundos.
Disponibilidade	**AV1**- O servidor web onde está os serviços REST é sobrecarregado com uma série de pedidos N% maior do que o normal e os serviços permanecem operantes.

Modificabilidade	**M1**- Um desenvolvedor modifica a lógica interna de um serviço, mas o contrato de serviço (Interface Uniforme) permanece o mesmo, o esforço para efetuar essas mudanças é limitado a N pessoas-dia. **M2**- A estrutura de representação de um recurso ‘R1’ e suas relações com outras alterações de recursos mudam e a identificação de ‘R1’ (URI) não é afetada. **M3**- A representação do recurso ‘R1’ é modificada e é gerada uma nova versão v(n) para o recurso. Os respectivos serviços devem processar corretamente as solicitações tanto para os consumidores de serviço que requisitam a representação de ‘R1’ na versão v(n), quanto para os consumidores de serviço que solicitam a versão v(n-1).
Proteção	**Sa1**- Um consumidor de serviços ‘A’ realiza diversas solicitações usando o método idempotente HTTP PUT e o recurso mantém o mesmo valor da primeira solicitação. **Sa2**- Um consumidor de serviços ‘A’ realiza solicitações usando métodos seguros (como HTTP GET e OPTIONS) e o recurso não é modificado.
Descoberta	**D1**- Um consumidor de serviços ‘A’ solicita um recurso 'R1' e recebe os URIs para encontrar recursos associados na representação de ‘R1’.
Funcionalidade[12]	**F1**- Com base na organização padrão da identificação de recursos na interface de um serviço, o consumidor de serviços ‘A’ pode compor um URI de um recurso. **F2**- Um desenvolvedor deseja desenvolver um serviço que consome recursos e pode encontrar toda a documentação da interface do serviço. **F3**- Um consumidor de serviços 'A' realiza uma ação em um recurso e recebe a sua identificação (URI) no campo HTTP *header Location*. **F4**- Um consumidor de serviço ‘A’ solicita uma representação de apenas alguns atributos de um recurso 'R1', define o número n de páginas e recebe o atual estado de atributos solicitados de 'R1' em n páginas na mensagem de resposta. **F5**- Um consumidor de serviço 'A' quer executar operações em um recurso e 'A' apenas opera sob verbos HTTP (PUT, POST, DELETE,...).

[12] Funcionalidade aqui não se refere a requisitos funcionais, mas como um atributo de qualidade genérico aos serviços REST.

5.3 Questões de *design* REST que afetam atributos de qualidade (Requisição RG3)

5.3.1 Introdução

Em arquiteturas baseadas no estilo REST, várias decisões de *design* apresentam *trade-offs* nos requisitos de atributos de qualidade. Esta seção aborda os seguintes tópicos, compilados a partir do protocolo apresentado e classificados como particularmente relevantes na concepção de sistemas baseados em REST: *Design* de Recursos (subseção 5.3.2); Representação e identificação (subseção 5.3.3), documentação e testes (subseção 5.3.4); Comportamento (subseção 5.3.5), e; Segurança (subsecção 5.3.6). Em cada tópico, são apresentadas as questões de *design* que os avaliadores devem inspecionar. No decorrer das questões, são referenciados os cenários gerais, descritos na subseção 5.2, que são afetados, assim, recomenda-se que esta subseção seja lida com os cenários gerais descritos na subseção 5.2 impressos ou em outro monitor, com o objetivo de facilitar a consulta.

5.3.2 Design de recursos

O *Design* de Recursos relaciona-se com a questão de como projetar os recursos que serão expostos em uma interface de serviço. Ao avaliar o *design* de recursos, as seguintes perguntas podem ajudar a identificar os riscos:

5.3.2.1 Qual é o modelo de domínio da aplicação?

O modelo de domínio representa o conhecimento sobre o domínio da aplicação. Ele descreve as várias entidades, seus atributos, papéis e relacionamentos. O modelo de domínio pode ser representado, por exemplo, através de um diagrama de classe UML. Os avaliadores devem verificar se o modelo de domínio foi validado pelos *stakeholders*. As próximas perguntas e tópicos seguintes estão relacionados com os recursos representados no modelo de domínio da aplicação. Assim, esta etapa é o ponto de partida para a avaliação de sistemas baseados em REST.

5.3.2.2 Quais dados serão expostos como recursos?

Os avaliadores devem ter uma compreensão global acerca das entidades que serão expostas como recursos através de serviços REST.

5.3.2.3 As representações dos recursos são padronizadas em toda a aplicação, departamento ou organização?

A representação de um determinado recurso deve ser a mesma em termos de formato (por exemplo, XML ou JSON) e a estrutura em todos os serviços que lidam com este recurso.

O ideal é que essa padronização esteja presente em toda a empresa ou organização, mas minimamente deve abranger todos os serviços. Se um determinado recurso não é representado por diferentes formatos, o atributo de qualidade interoperabilidade, descrito nos cenário I2 e I3, é severamente prejudicado. O atributo desempenho, descrito no cenário P1, também pode ser afetado negativamente, devido à necessidade de transformações de formato de dados (por exemplo, JSON para XML) ou transformação do modelo de dados (por exemplo, a transformação XSLT de uma definição de esquema para outro).

5.3.2.4 Que tipos de consumidores de serviço vão interagir com os serviços REST?

Avaliadores devem investigar os tipos de componentes de software que atuarão como consumidores dos serviços REST. Eles podem ser páginas web em um navegador, um componente .NET ou Java rodando em uma máquina servidor, aplicações móveis, aplicações *standalone*, ou qualquer outro tipo de programa que pode se comunicar via HTTP. O cenário de interoperabilidade I1 é afetado com base em tais investigações. Também é necessária a investigação das propriedades do canal de comunicação. São consumidores de serviços na mesma rede local ou intranet onde está o serviço REST? Quão rápida e confiável é a conexão? O canal de comunicação é criptografado?

5.3.2.5 Alguma informação confidencial está sendo exposta como um recurso?

O cenário geral S1 descreve os problemas gerais de segurança que devem ser analisados ao projetar recursos. Por isso, os avaliadores devem saber se a confidencialidade dos dados é uma exigência para qualquer serviço REST.

5.3.3 Representação e identificação

As seguintes questões relacionadas à representação e identificação dos recursos podem ajudar a detectar riscos no projeto de serviços REST:

5.3.3.1 Quais os formatos usados para a representação dos recursos?

Um recurso é representado por um documento com um tipo de mídia identificado por um nome (por exemplo, text/html), e declarado no cabeçalho HTTP *Content-Type*. XML (*application/xml*) é um formato de representação muito comum para os recursos, embora outros formatos, como HTML, ATOM e JSON, também são amplamente utilizados. Frameworks de desenvolvimento e plataformas fornecem diferentes níveis de suporte para diferentes formatos. A escolha do formato de representação também afeta o desempenho (cenário P1) e interoperabilidade (cenários I2 e I3). Para alcançar cenários de atributos de

qualidade de interoperabilidade como I2, um serviço REST pode precisar usar diferentes representações de recursos adequados para diferentes dispositivos e sistemas.

5.3.3.2 Foi definido um vocabulário padrão para o recurso?

A representação de recurso deve seguir um vocabulário pré-definido. Para documentos XML, o vocabulário pode ser definido por um esquema XML. O vocabulário também deve ser padronizado em todos os serviços que lidam com esse mesmo recurso, como já discutido em questão 5.3.2.3. Usando um vocabulário conhecido pelos consumidores o atributo interoperabilidade, especificado no cenário I3, é impactado positivamente. Além disso, todas as informações necessárias para entender o recurso devem ser incluídas no pedido e resposta (restrição *Stateless*, seção 5.1.2.2). A restrição HATEOAS[13] [Fielding 2000] é usada para esta finalidade. Esta restrição indica que o documento hipermídia que representa o recurso deve ser utilizado para encontrar a recursos associados. Por exemplo, o seguinte fragmento da representação do recurso de "curso" de uma universidade contém um *link* para um outro recurso que representa os professores que lecionam no curso:

```
<curso>
  <nome>Arquitetura de Software</nome>
  <area>Engenharia de Software</area>
  <professores>
     <uri>/cursos/es/professores</uri>
  </professores>
</courso>
```

HATEOAS afeta positivamente o cenário do atributo descoberta D1. Porém, o tempo de resposta para uma solicitação de serviço diminui porque a representação não precisa trazer todas as informações sobre os recursos vinculados, mas apenas referências (*hyperlinks*). No entanto, um maior número de solicitações de rede pode ser necessário para recuperar informações vinculadas e o cenário do atributo de desempenho P1 pode ser afetado negativamente.

5.3.3.3 Como os URIs são projetados?

Um recurso deve ter pelo menos um URI que faça com que ele seja endereçável via HTTP. Vários princípios de concepção de URIs [Berners-Lee 1996] podem ser utilizados para a especificação de identificadores de recursos REST.

Projetando URIs descritivas relacionado com a hierarquia do modelo de domínio melhora o cenário de confiabilidade R2. Exemplos de URIs descritivo são listados abaixo

[13] Acrônimo para o Inglês “Hypermedia as the Engine of Application State”

- http://www.myweather.com/current/city/Brasilia
- http://www.ufrj.edu/cursos/programacao101

No entanto, URIs não são obrigados a serem sempre descritivos. Os recursos podem ser identificados por IDs numéricos, por exemplo, uma URI pode ser http://www.resoucerslib.com/52545. Neste exemplo, o recurso é identificado pelo ID 52545. A identificação de recursos por ID impacta positivamente o cenário atributo de qualidade modificabilidade M2, mas tem um impacto negativo no requisito de confiabilidade especificado no cenário R2. Outras estratégias para projetar identificadores de recursos podem ser utilizadas. Avaliadores devem inspecionar o *tradeoff* entre URIs descritivas e não-descritivas de acordo com ambos os requisitos de atributos de qualidade de *modificabilidade* e confiabilidade. No entanto, o estabelecimento de um padrão para definir URIs para serviços REST é uma boa prática que impacta positivamente o cenário de funcionalidade F1.

Em geral, os URIs devem usar substantivos e não verbos. De acordo com a restrição interface uniforme, a operação a ser executada deve ser especificada pelo verbo HTTP, não na URI. Por exemplo, ao invés de definir a URI http://www.ufrj.edu/getcourses para a representação dos cursos, a URI deve ser http://www.ufrj.edu/courses com o verbo HTTP GET. Se os verbos forem utilizados na URI, isto pode ter um impacto negativo no cenário de funcionalidade F5.

5.3.3.4 Qual a abordagem para versionamento de recursos?

Representações de recursos são acessadas por sua identificação (URI). Se a representação for modificada, os consumidores de serviços que solicitam este recurso podem ser afetados negativamente (cenário I3). A razão para tal modificação pode ser: (i) uma evolução do modelo de domínio que impacta em representação do recurso, ou; (ii) uma evolução na API. Versionamento de recursos é uma boa prática para resolver este problema e afeta positivamente no cenário de modificabilidade M3. A estratégia utilizada para os recursos de versão pode ser a inclusão do número de versão na URI. Por exemplo:

- http://www.ufrj.edu/cursos/v1/cienciadacomputacao

Outra alternativa é a inclusão do número da versão no cabeçalho HTTP, por exemplo:

Request: *GET /cienciadacomputacao HTTP/1.1*
Accept: *Content-type: application/xml;* ***version=1.0***

Os avaliadores devem inspecionar o *tradeoff* entre as estratégias de versionamento diretamente na URI e no cabeçalho HTTP. Por exemplo, para consumidores de serviços humanos, solicitando recursos através de navegadores web, a versão contida na URI pode ser uma solução melhor e, neste caso, a o cenário do atributo confiabilidade R1 é impactado positivamente.

5.3.3.5 Como são mapeadas as operações nos recursos em verbos HTTP?

Esta questão está relacionada com a questão 5.3.3.3 (Como os URIs são projetados?). Como já mencionado na subseção 5.1.2.4, a restrição interface uniforme entre os componentes é uma característica central em arquiteturas baseadas em REST. A interface deve utilizar métodos HTTP comuns (também chamados de verbos) para indicar a ação a ser executada em um recurso: POST para criar um novo recurso, PUT para atualizar um recurso; DELETE para remover um recurso, OPTIONS para listar quais os métodos são suportados um recurso, entre outros. Ao usar verbos HTTP cenários do atributo funcionalidades, como F5, são afetados positivamente.

Dois conceitos importantes na concepção da exposição do recurso são métodos idempotentes e seguros. Métodos seguros são métodos HTTP que não modificam os recursos (por exemplo, GET e OPTIONS), eles só solicitam representações ou outras informações. Um método HTTP idempotente é um método que pode ser chamado muitas vezes sem resultados diferentes. Por exemplo, considere duas ações que atribuem um valor à variável *var*: (a) var = 5, (b) var++. O primeiro exemplo (a) é idempotente, não importa o número de vezes que método é executado o resultado de *var* sempre será sempre 5. O segundo exemplo (b) não é idempotente. Várias execuções resultarão em diferentes resultados em *var*. Os métodos HTTP PUT e DELETE são idempotentes, POST não é. Cenários do atributo de qualidade proteção Sa1 Sa2 estão relacionadas aos métodos idempotentes e seguros.

5.3.4 Documentação e testes

Consumidores de serviços executam ações (GET, PUT, POST e DELETE) em recursos. Os avaliadores devem inspecionar se existe uma documentação que descreva todas as informações sobre os recursos relacionados à concepção, identificação, representação, segurança, dentre outros. Os agentes de usuário são criados por desenvolvedores que precisam compreender a interface do serviço. Ao avaliar a documentação e testes, as seguintes perguntas ajudam determinar os riscos:

5.3.4.1 Como os recursos são documentados?

É necessário explicar aos consumidores de serviços como acessar e utilizar os recursos e como desenvolver serviços que interagem com a interface do serviço. Uma boa documentação impacta positivamente o cenário de funcionalidade F2.

5.3.4.2 Como os consumidores de serviços podem realizar testes nos recursos?

É importante para os consumidores e desenvolvedores realizarem testes na interface do serviço. Testes dinâmicos podem ser realizados utilizando uma ferramenta de teste denominada "*sandbox*" que tem um comportamento similar ao serviço real. Métodos HTTP, como GET e PUT, podem ser executados na *sandbox* e o desenvolvedor pode analisar sua resposta. Este é um cenário típico para melhorar o cenário T1 do atributo de qualidade testabilidade.

5.3.5 Comportamento

O comportamento está relacionado à interface do serviço, quando os consumidores de serviços executam ações nos recursos. Ao avaliar o comportamento, as seguintes questões de ajuda para determinar os riscos:

5.3.5.1 Quais os códigos de status HTTP presentes na resposta?

A resposta sobre as ações realizadas em recursos deve usar códigos de status padrão HTTP. Os avaliadores devem inspecionar os códigos de status HTTP para operações bem sucedidas e erros que ocorrem em solicitações de recursos. Ao usar códigos de status padrão cenários de atributos de qualidade para o atributo funcionalidade e confiabilidade, como F5 e R2, são impactados positivamente.

5.3.5.2 O cabeçalho de resposta HTTP contém a informação sobre a identificação do recurso?

Quando consumidores de serviços executam ações em um recurso, o serviço deve incluir nas respostas HTTP, no campo de cabeçalho "*Location*", a informação da URI do recurso manipulado. Consumidores de serviços podem identificar a URI do recurso manipulado com a informação da URI no cabeçalho de resposta, afetando positivamente em cenários do atributo funcionalidade como F3.

5.3.5.3 O recurso pode ser comunicar com Open APIs?

O conceito de *Open* API[14] é uma tendência para muitos serviços baseados na web, tais como redes sociais e *e-business*, para publicar serviços através da rede. Se a interface de um serviço permite o uso de *Open* APIs, o atributo funcionalidade (cenário F5) e interoperabilidade (cenários I1 e I3) são impactados positivamente. No entanto, é necessário analisar o *tradeoff* entre tal decisão de projeto e o atributo de qualidade segurança, especificado no cenário S1.

5.3.5.4 É necessário implementar paginação nos recursos?

A representação completa de recursos pode afetar negativamente os cenários de desempenho, como P1. Avaliadores devem analisar se a paginação da representação de recursos é necessária. Uma boa prática é permitir *query strings*[15] em URIs para definir o número e o tamanho da página. Por exemplo, um URI para "cursos da UFRJ", que especifica a visualização de uma página com trinta cursos podem ser: http://www.ufrj.edu/cursos?page=1&size=30. Paginação está relacionada com cenários de desempenho e funcionalidade, como P1 e F4

5.3.5.5 É possível incluir um subconjunto de atributos desejados na URI?

Suponha que um consumidor de serviços quer apenas um subconjunto de atributos e a interface do serviço suporta apenas respostas contendo a representação completa de um recurso. Neste caso, cenários de desempenho e funcionalidade P1 e F4 são influenciados negativamente. Uma boa prática é a de incluir um filtro na URI. Por exemplo, a seguinte URI permite que um consumidor de serviço apenas para recuperar o nome e as informações sobre os cursos da UFRJ: http://www.ufrj.edu/cursos?atributos=nome,informacoes.

5.3.5.6 Como proteger o servidor web de sobrecarga de requisições?

Consumidores de serviços pode realizar um alto índice de solicitações, resultando em sobrecarga do servidor web. Os avaliadores devem examinar as políticas para proteger o servidor, a fim de promover cenários de disponibilidade, como Av1, e desempenho, como P1.

5.3.6 Segurança

Questões de *design* acerca da segurança estão relacionadas à autenticação, autorização e privacidade. Ao avaliar a segurança, as seguintes perguntas ajudar a determinar os riscos:

---

[14] Acrônimo para o ingles “Application Programming Interface”
[15] Uma cadeia de caracteres incluída na URI como parâmetro para o recurso

5.3.6.1 Os recursos privados são projetados de forma segura?

Esta questão está associada à questão de design 5.3.2.2 (Quais dados serão expostos como recursos?). Os avaliadores devem prestar especial atenção aos recursos que são privados, e inspecionar se eles estão expostos incorretamente como recursos públicos. Requisitos de segurança, conforme expresso no cenário geral S1, são normalmente realizados por um *design* correto de recursos.

5.3.6.2 Quais são as políticas de segurança para consumidores de serviço para realizarem ações nos recursos?

Alguns recursos são restritos a um conjunto de consumidores de serviço. É necessário implementar políticas de segurança para garantir a autenticação e autorização de tais consumidores de serviços. Alguns padrões comumente adotados são OAuth (2013) e OpenID (2013). Além disso, é possível usar a autenticação básica HTTP. A versão 2.0 do OAuth provou ser uma boa estratégia para autorização segura em arquiteturas baseadas em REST. Um cenário típico para a segurança é S2.

6 Prova de conceito

6.1 Introdução

Este capítulo apresenta o uso das diretrizes como uma prova de conceito. O objetivo é demonstrar a utilização do guia em uma avaliação arquitetural realizada antes da implementação do sistema (*Early Evaluation*). Para atingir este objetivo, a prova de conceito utilizará o método de avaliação arquitetural ATAM. Seguindo as atividades do método, a apresentação do método ATAM (atividade 1) foi realizada na subseção 2.2.3; foi omitida a geração da árvore de utilidades da atividade 5, uma vez que esta atividade não impacta na prova de conceito, são apresentados apenas os cenários concretos resultantes gerados diretamente na atividade 7. O guia proposto nesta dissertação é utilizado na atividade 8 onde, na análise de cada cenário concreto, são referenciadas as questões de design consideradas. As demais atividades são descritas nas subseções seguintes:

Atividade 2. Apresentação dos Objetivos de Negócio: subseção 6.2

Atividade 3. Apresentação da Arquitetura: subseção 6.3

Atividade 4. Identificação das Abordagens Arquiteturais: subseção 6.4

Atividade 7. *Brainstorm* para Priorizar Cenários: subseção 6.5

Atividade 9. Apresentação dos Resultados: subseção 6.7

6.2 Atividade 2: Apresentação dos Objetivos de Negócio

A arquitetura utilizada para avaliar a orientação proposta é do sistema EcoDiF[16] (Ecossistema Web de dispositivos físicos) [Delicato et al. 2013]. A EcoDiF é uma plataforma Web para conectar dispositivos e produtos com aplicações e/ou usuários finais, a fim de fornecer funcionalidades de controle, visualização, processamento e armazenamento de dados. A EcoDiF atuará como um núcleo de um ecossistema para Internet das Coisas (IoT) [Atzori et al. 2010], oferecendo serviços (de software) focados: (i) na conectividade entre dispositivos e a Internet; (ii) em serviços de aplicação e (iii) em serviços de apoio. A EcoDiF pode ser usada em diversos contextos, tais como aplicações de monitoramento ambiental, de monitoramento de infraestrutura pública, como acompanhamento de trânsito e condições da estrada, bem como para compartilhamento de dispositivos de sensoriamento entre laboratórios acadêmicos.

[16] Disponível em http://ubicomp.nce.ufrj.br/ecodif/index.html

Portanto, a EcoDiF fornece uma API aberta que pode ser utilizada por indivíduos, laboratórios e instituições de pesquisa, e empresas, para acessar os serviços providos pela plataforma. Dentre as operações oferecidas pela API da EcoDiF podemos citar:

- Criação, atualização, visualização e remoção de "ambientes de dados" (denominados *feeds*). Por exemplo: "*Feed* de dados da temperatura no laboratório de pesquisa";
- Criação, atualização, visualização e remoção de "fluxos de dados" (denominados *streams* de dados). Por exemplo: *stream* dos dados enviados por um sensor de temperatura ligado a um dispositivo Arduino[17];
- Criação, atualização, visualização e remoção dados pontuais (denominados *datapoints*). Por exemplo: cada leitura de temperatura do sensor;
- Criação, atualização, visualização e remoção de notificações (denominadas *triggers*). Por exemplo: se a temperatura se elevar acima de 30º, uma mensagem SMS é enviada para o usuário.

Desta forma, a plataforma EcoDiF oferece um serviço voltado para a disponibilização de produtos habilitados para a Internet das Coisas sem a necessidade de ter que se construir qualquer infraestrutura de *backend* para tal.

A EcoDiF é desenvolvida em parceria do Laboratório de Computação Ubíqua (UBICOMP[18]) da Universidade Federal do Rio de Janeiro com o Laboratório de Concepção de Sistemas da Universidade Federal do Rio Grande do Norte. O autor deste trabalho atuou como um dos desenvolvedores da EcoDiF.

6.3 Atividade 3: Apresentação da Arquitetura

A Figura 6 mostra uma visão geral da plataforma e os principais módulos da arquitetura, que serão descritos a seguir.

- **Módulo de Conexão de Dispositivos**: objetiva facilitar a conexão de dispositivos físicos à EcoDiF (e, por consequência, à Internet), de modo que os fabricantes possam configurar seus dispositivos de acordo com a API específica da plataforma, e oferecer um *driver* de conexão para os usuários. Com o *driver*, os usuários podem conectar seus dispositivos facilmente à plataforma;
- **Módulo de Manipulação de Dados**: permite a manipulação, em tempo real, dos dados providos pelos dispositivos conectados à EcoDiF por meio de *drivers* específicos para tais dispositivos;

[17] Plataforma de prototipagem eletrônica de hardware livre
[18] Informações disponíveis em http://ubicomp.nce.ufrj.br/

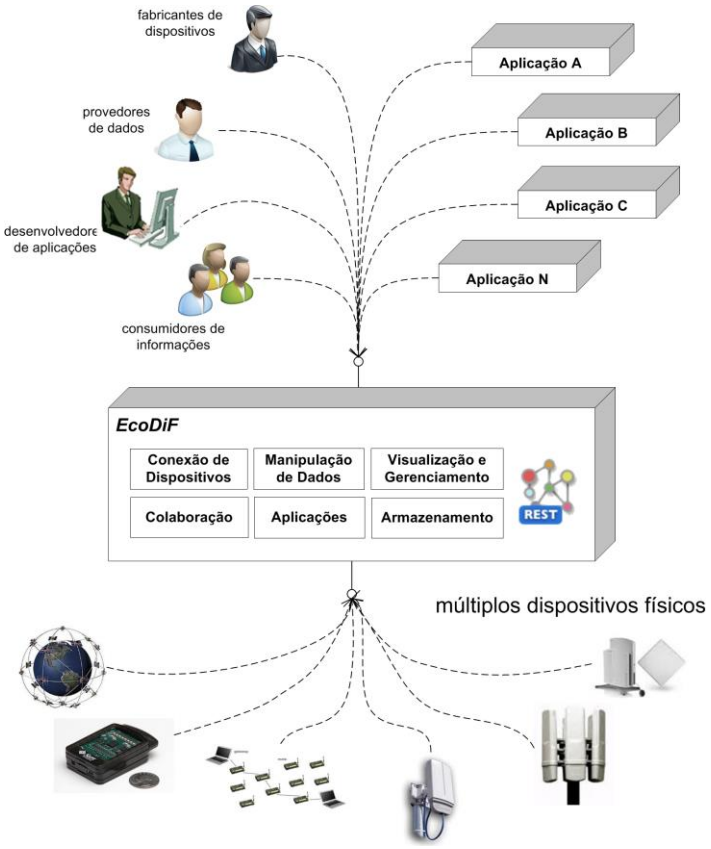


Figura 6 Visão Geral e Componentes Arquiteturais da EcoDiF

- **Módulo de Visualização e Gerenciamento**: provê uma interface Web para que usuários possam gerenciar os dispositivos conectados à EcoDiF (em termos de seu estado e localização), criar alertas e notificações (chamados de *triggers*) e visualizar dados históricos armazenados na plataforma;
- **Módulo de Colaboração**: objetiva facilitar a colaboração entre usuários da EcoDiF, permitindo realizar buscas por dispositivos e aplicações registrados na plataforma a partir de seus metadados (tipo, usuário, localização, etc.);
- **Módulo de Armazenamento**: consiste basicamente de dois repositórios, um para o armazenamento de dados em um banco de dados relacional, e outro para o armazenamento de *scripts* de aplicações no sistema de arquivos;
- **Módulo de Aplicações**: visa prover um modelo e ambiente para programação e execução de aplicações que consumam dados disponíveis na EcoDiF e gerem novos dados/informações que também são disponibilizados na plataforma;

- **Módulo de Serviços Comuns**: agrega mecanismos inerentes a toda a plataforma, tais como segurança, ciclo de vida de aplicações, transações, etc.

A interoperabilidade é um requisito de atributo de qualidade importante para a EcoDiF, assim a arquitetura foi projetada de acordo com os princípios e restrições REST. A EcoDiF emprega os princípios REST para disponibilizar as funcionalidades dos dispositivos físicos na Web utilizando duas abordagens. Na primeira abordagem, são implantados servidores Web embarcados em dispositivos e as funcionalidades desses dispositivos são disponibilizadas na forma de recursos RESTful. Por exemplo, *smartphones* com o sistema operacional Android. Na segunda abordagem, quando um dispositivo não possui recursos de hardware suficientes para executar um servidor embarcado, é possível utilizar outro dispositivo como ponte para disponibilizar as funcionalidades do dispositivo na Web através de uma interface RESTful. Por exemplo, um dispositivo Arduino com um sensor de temperatura é ligado a um computador dotado com um servidor Web para enviar as informações à EcoDiF.

Os *drivers* possuem um papel de suma importância no tocante à integração dos diversos dispositivos à EcoDiF, de modo que a heterogeneidade de tais dispositivos é ocultada para usuários e aplicações que fazem uso dos mesmos e dos dados por eles providos. Essa transparência e interoperabilidade constituem um dos requisitos essenciais observados na literatura com relação a plataformas de middleware no contexto do paradigma de IoT e consolida um dos objetivos principais da própria EcoDiF, que é possibilitar que desenvolvedores e aplicações sejam integrados de forma a criar um ecossistema de IoT a fim de desenvolver novas ideias e produtos de forma orgânica. Com isso, a EcoDiF é capaz de fornecer um acesso padronizado aos dados e serviços providos pelos diversos dispositivos através de interfaces de alto nível, além de promover o reuso de serviços genéricos, que podem ser compostos e configurados para facilitar o desenvolvimento de aplicações de forma mais eficiente para o ambiente altamente distribuído e heterogêneo, característico de IoT. Mais ainda, a complexidade dos dispositivos e do ambiente de rede subjacente é ocultada das aplicações e usuários, livrando-os da tarefa de manipular, de forma explícita, protocolos e serviços de infraestrutura.

No contexto da EcoDiF, os dispositivos a serem integrados são conectados através de *drivers* específicos, desenvolvidos pelos fabricantes de dispositivos especificamente para tais dispositivos, tornando-os assim compatíveis com a API provida pela plataforma. Além disso, usuários que sejam provedores de dados e que desejem disponibilizar na EcoDiF os dados

provenientes dos dispositivos fazem uso de tais drivers. Em essência, os drivers possuem três funções internas básicas: (i) obter os dados coletados pelos dispositivos; (ii) estruturar tais dados utilizando o protocolo específico, utilizado pela EcoDiF para a representação de feeds, e; (iii) enviar tais dados para serem registrados na EcoDiF mediante requisições REST/HTTP utilizando o método PUT.

Diante das funcionalidades oferecidas pelos seus diferentes módulos, a EcoDiF contempla quatro perfis de usuário, a saber:

- **fabricantes de dispositivos**, que desenvolvem drivers para seus dispositivos, tornando-os compatíveis com a API provida pela EcoDiF;
- **provedores de dados**, que desejam disponibilizar os dados providos pelos seus dispositivos na EcoDiF, fazendo uso dos drivers de cada dispositivo;
- **desenvolvedores de aplicações**, que constroem aplicações que usam os dados providos pelos dispositivos integrados à EcoDiF, e;
- **consumidores de informações**, que são usuários que interagem com a EcoDiF para consultar informações (acerca de dispositivos e dados por eles providos, aplicações, etc.) disponíveis na plataforma.

A Figura 7 mostra um diagrama UML de casos de uso que ilustra os atores anteriormente mencionados que interagem com a EcoDiF e os casos de uso por eles realizados e que estão associados às principais funcionalidades providas pela plataforma. Abaixo, a descrição dos casos de uso.

- **UC00 – Gerenciar Usuários**: cadastro de usuários e gerenciamento de *Login* / *Logoff*
- **UC01 – Cadastrar Dispositivos**: criação, edição e exclusão de dispositivos físicos. Além de desenvolvimento e *upload* de *drivers* para a plataforma;
- **UC02 – Conectar Dispositivos**: conexão e desconexão de dispositivos, *download* de *drivers* e visualização de dispositivos conectados;
- **UC03 – Cadastrar *Feeds***: criação, edição, exclusão e consulta de *feeds*;
- **UC04 – Criar Triggers**: criação, edição, exclusão e consulta de *triggers*;
- **UC05 – Gerenciar Aplicações**: criação, edição, exclusão e consulta de aplicações. Além de inclusão e exclusão de *feeds* na aplicação.

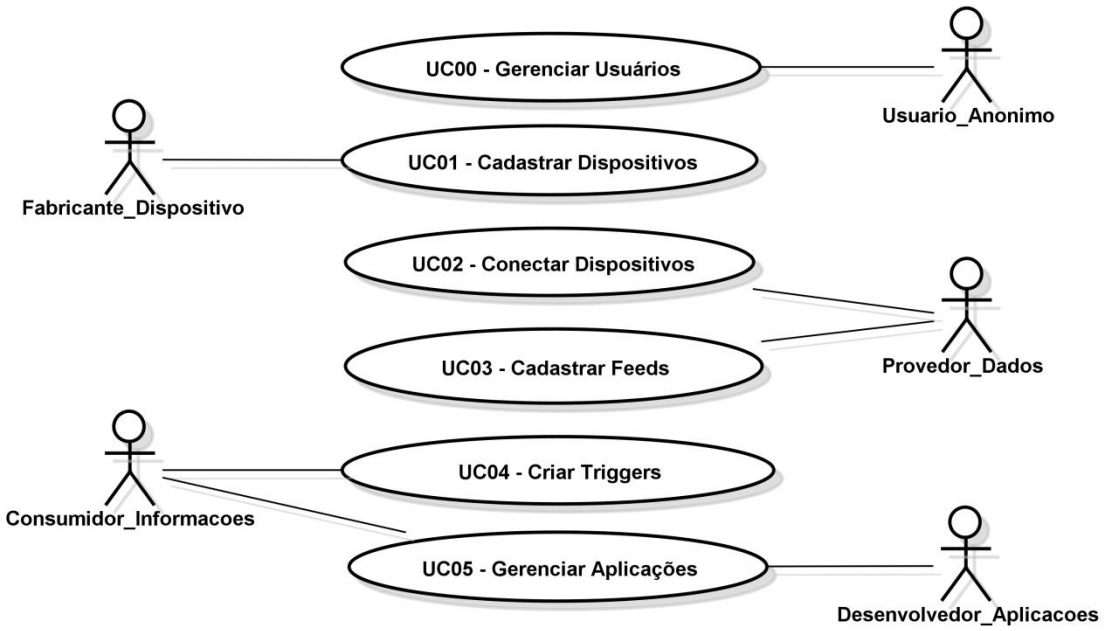


Figura 7 Diagrama UML ilustrando os casos de uso associados às principais funcionalidades providas pela EcoDiF

A Figura 8 apresenta as restrições tecnológicas especificadas para a implementação dos principais módulos que compõem a arquitetura lógica da EcoDiF. Através do Módulo de Visualização e Gerenciamento, por meio de um portal Web a ser implementado utilizando as tecnologias JavaServer Faces (JSF)[19] e Primefaces[20] (ambos de licença open-source), os usuários terão acesso às principais funcionalidades providas pela plataforma, a ser implementada utilizando a linguagem de programação Java e implantada em um servidor de aplicações JBoss[21]. Esse módulo, além das páginas Web propriamente ditas, englobará também as classes responsáveis pela interação das páginas Web e os *managed beans*[22].

Uma vez que a conexão entre os dispositivos integrados e a EcoDiF é habilitada pelos respectivos *drivers*, em conjunto com o Módulo de Conexão de Dispositivos, os dados provenientes de tais dispositivos serão enviados pelos drivers à EcoDiF através de requisições HTTP) fazendo uso da implementação RESTEasy[23] para o estilo arquitetural REST e com

[19] JavaServer Faces Technology. Disponível em: http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html

[20] PrimeFaces. Disponível em: http://primefaces.org/

[21] JBoss Application Server. Disponível em: http://www.jboss.org/jbossas

[22] Na linguagem Java, um managed bean é um objeto que representa um recurso gerenciado por um servidor de aplicação, como o JBoss

[23] RESTEasy. Disponível em: http://www.jboss.org/resteasy/

base nas especificações Java API for RESTful Web Services (JAX-RS)[24]. Esses dados obtidos (*feeds*) serão estruturados utilizando o protocolo EEML[25], que é baseado na linguagem de marcação XML. Uma vez recebidos pela EcoDiF, esses dados em EEML serão tratados pelo Módulo de Manipulação de Dados e efetivamente registrados em um banco de dados relacional MySQL, fazendo uso das especificações da Java Persistence API (JPA)[26], a serem implementadas utilizando o framework Hibernate[27].

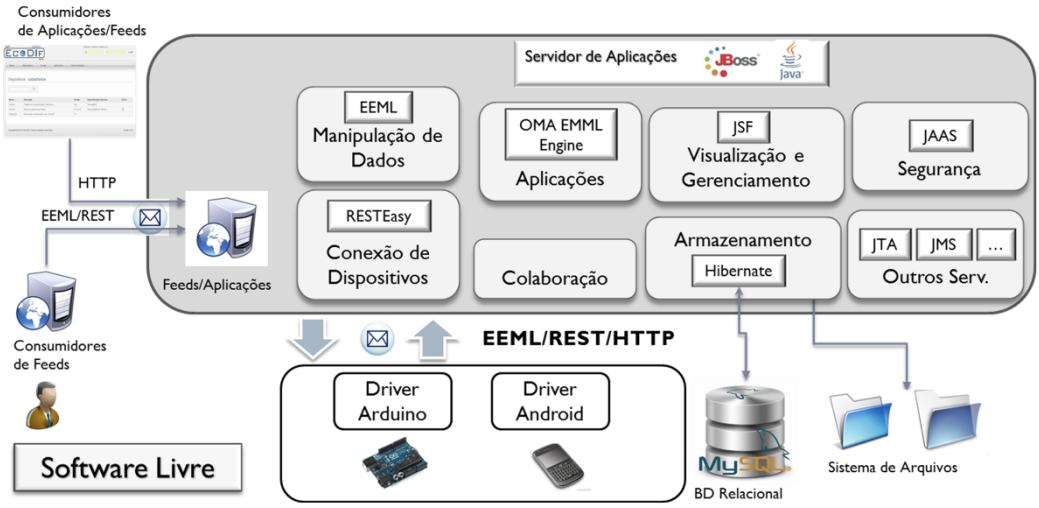


Figura 8 Restrições tecnológicas na especificação dos módulos que compõem a arquitetura da EcoDiF.

Como anteriormente mencionado, usuários que sejam desenvolvedores de aplicações poderão construir, utilizando facilidades providas pelo Módulo de Aplicações, aplicações que façam uso dos dados providos pelos dispositivos integrados à EcoDiF de maneira combinada. Esses dados também serão representados na forma de *feeds*. No contexto da EcoDiF, aplicações são implementadas utilizando a linguagem EMML, uma linguagem aberta também baseada em XML para o desenvolvimento de *mashups*[28] a partir da integração de dados de fontes diversas, tais como serviços web e bancos de dados relacionais. Para a execução de tais aplicações, o Módulo de Aplicações fará o uso de um motor de execução (*engine*) EMML instalado no servidor de aplicações JBoss no qual a EcoDiF será implantada.

---

[24] Java API for RESTful Web Services. Disponível em: http://jax-rs-spec.java.net/

[25] Extended Environments Markup Language (EEML). Disponível em: http://eeml.org/

[26] Java Persistence API. Disponível em: http://www.oracle.com/technetwork/java/javaee/tech/persistence-jsp-140049.html

[27] Hibernate. Disponível em: http://www.hibernate.org/

[28] Um *mashup* é um recurso ou uma aplicação web que usa conteúdo de mais de uma fonte para criar um novo serviço completo

Por fim, o Módulo de Segurança da EcoDiF permitirá o controle de autenticidade, confidencialidade e integridade de usuários para *login/logout* na plataforma, utilizando, para tal, as especificações Java Authentication and Authorization Service (JAAS)[29], implementadas no servidor de aplicações JBoss.

6.4 Atividade 4: Identificação das Abordagens Arquiteturais

A restrição sistema de camadas (*L*) é aplicada pelo agrupamento das responsabilidades em três camadas, conforme ilustrada na Figura 6: interface web, serviços de middleware e dispositivos físicos integrados. A EcoDiF atua como um servidor web e os dispositivos físicos atuam como clientes. A restrição Sem Estado (*S*) é aplicada pois a EcoDiF não mantém o estado das requisições, todas as informações do recurso estão incluídas nas mensagens de resposta. A especificação não utiliza HATEOAS. Código sob demanda (*CoD*) e as restrições de *cache* ($) não são utilizados. Os recursos foram projetados de acordo com o protocolo EEML. O único recurso exposto aos clientes é uma *feed*. Quando um usuário cria uma *feed*, a EcoDiF atribui um identificador (ID) para identificá-la. O padrão de URI para identificar a alimentação é: ecodif.com / api / feed / [ID]. Para solicitar um *feed* é o utilizado o método HTTP GET. O mesmo URI é usado por provedores de usuários para atualizar a *feed* usando HTTP PUT. Quando a *feed* é atualizada, a EcoDiF responde com o código de status HTTP 201, mas não inclui o URI da *feed* atualizada. Para realizar a atualização da *feed* usando PUT, não é necessário incluir as credenciais do usuário no cabeçalho HTTP. Os usuários obtêm suas credenciais no ato do cadastro na plataforma. Se o usuário configura a *feed* como pública, o GET é realizado sem validações de segurança. Porém, mesmo se a *feed* for configurada como privada é possível solicitar sua representação via GET sem credenciais. A *feed* usa unicamente o tipo de mídia *application/xml*. Quando um recurso é solicitado, a EcoDiF envia a representação completa de uma *feed* para o cliente. Não é possível a paginação de recursos nem a seleção de subconjunto de atributos. A EcoDiF não tem versionamento de recursos. A documentação da interface da EcoDiF está disponível no Módulo de Visualização e Gerenciamento, através de uma página web.

6.5 Atividade 7: *Brainstorm* para priorizar cenários

A Tabela 9 mostra os cenários concretos da EcoDiF que são relevantes para uma avaliação de arquitetura baseada em REST. Os cenários foram elencados por gerentes de projeto da EcoDiF, pelo arquiteto da plataforma e *stakeholders*.

---

[29] Java SE Security. Disponível em: http://www.oracle.com/technetwork/java/javase/jaas/index.html

Tabela 9 Cenários Concretos de Avaliação Arquitetural.

Atributo de Qualidade	Cenário Concreto
Cenário 1. Interoperabilidade	**Fonte do estímulo**: *Driver* / Provedor de Dados; **Estímulo**: O provedor envia dados para a EcoDiF; **Artefatos**: *Driver* do dispositivo, Módulo de Conexão de Dispositivos; **Ambiente**: Operação normal; **Resposta**: O *driver* envia dados provindos do sensor do dispositivo e recebe a confirmação de recebimento da interface da EcoDiF; **Medição de resposta**: O sistema responde ao driver informando que recebeu corretamente os dados através do status da requisição.
Cenário 2. Interoperabilidade	**Fonte do estímulo**: Consumidor de Dados; **Estímulo**: Consumidor de Dados solicita dados de *feeds* em um determinado *media type*; **Artefatos**: Módulo de Conexão de Dispositivos; **Ambiente**: Operação normal; **Resposta**: O sistema envia ao usuário a representação do recurso da *feed* solicitada; **Medição de resposta**: O sistema responde para o usuário a representação do recurso no *media type* solicitado / No HTTP header de resposta o sistema informa ao usuário que não possui a funcionalidade de representar o recurso no *media type* solicitado.
Cenário 3. Testabilidade	**Fonte do estímulo**: Fabricante de dispositivo; **Estímulo**: O fabricante de dispositivos deseja criar um novo *driver* e executar testes na API da EcoDiF; **Artefatos**: Módulo de Visualização e Gerenciamento; **Ambiente**: Operação normal; **Resposta**: O sistema provê a documentação para o desenvolvimento do *driver* em uma *sandbox* para testes; **Medição de resposta**: A documentação oferece toda a informação necessária para desenvolver o driver e a *sandbox* oferece todas as funcionalidades necessárias para os testes.
Cenário 4. Confiabilidade	**Fonte do estímulo**: Consumidor de Informações; **Estímulo**: O consumidor de dados solicita uma *feed* em uma versão específica de representação; **Artefatos**: Módulo de Conexão de Dispositivos, módulo de Manipulação de Dados; **Ambiente**: Operação normal; **Resposta**: O sistema envia a representação da *feed* segundo a versão solicitada pelo usuário; **Medição de resposta**: A versão solicitada pelo usuário é enviada pelo sistema.
Cenário 5. Segurança	**Fonte do estímulo**: Provedor de dados; **Estímulo**: O provedor de dados envia valores de sensoriamento para uma *feed*; **Artefatos**: Módulo de Manipulação de Dados, Módulo de Conexão de

	Dispositivos, Módulo de Armazenamento; **Ambiente**: Operação normal; **Resposta**: O sistema valida a autenticidade do usuário e armazena as informações do *feed*; **Medição de resposta**: O sistema não deve permitir o armazenamento de dados dos usuários não autenticados.
Cenário 6. Segurança	**Fonte do estímulo**: Consumidor de dados; **Estímulo**: O consumidor de dados não autenticado solicita a representação de uma *feed* privada; **Artefatos**: Módulo de Manipulação de Dados ; **Ambiente**: Operação normal; **Resposta**: O sistema envia informações da *feed*; **Medição de resposta**: O sistema não deverá enviar informações privadas ao usuário solicitante.
Cenário 7. Desempenho	**Fonte do estímulo**: Consumidor de dados; **Estímulo**: O consumidor de dados solicita a representação de uma *feed*; **Artefatos**: Módulo de Manipulação de Dados; **Ambiente**: Operação normal; **Resposta**: O sistema envia informações da *feed*; **Medição de resposta**: O sistema envia a representação em menos de 2 segundos.
Cenário 8. Disponibilidade	**Fonte do estímulo**: Consumidor de dados; **Estímulo**: O consumidor de dados solicita a representação de uma *feed*; **Artefatos**: Módulo de Manipulação de Dados; **Ambiente**: Servidor sobrecarregado; **Resposta**: O sistema envia informações da *feed*; **Medição de resposta**: O sistema envia a representação em menos de 5 segundos.
Cenário 9. Modificabilidade	**Fonte do estímulo**: Provedor de dados; **Estímulo**: O provedor de dados envia informações sobre uma *feed* no formato EEML na versão n; **Artefatos**: Módulo de Manipulação de Dados; **Ambiente**: Operação normal; **Resposta**: O sistema atualiza as informações da *feed* e responde ao usuário a confirmação da atualização; **Medição de resposta**: O sistema atualiza a *feed* com a representação segundo a versão n.
Cenário 10. Proteção	**Fonte do estímulo**: Consumidor de dados; **Estímulo**: O consumidor de dados solicita informações acerca de uma *feed* através do método GET; **Artefatos**: Módulo de Visualização e Gerenciamento; **Ambiente**: Operação normal; **Resposta**: O sistema envia as informações ao consumidor de dados; **Medição de resposta**: O sistema envia as informações e nenhum recurso é modificado.
Cenário 11. Descoberta	**Fonte do estímulo**: Consumidor de dados; **Estímulo**: O consumidor de dados solicita informações acerca de uma *feed*; **Artefatos**: Módulo de Visualização e Gerenciamento;

	Ambiente: Operação normal; **Resposta**: O sistema envia as informações ao consumidor de dados; **Medição de resposta**: O sistema envia as informações contendo referências a outros recursos associados.
Cenário 12. Funcionalidade	**Fonte do estímulo**: Fabricante de dispositivos; **Estímulo**: Um fabricante deseja desenvolver um novo driver; **Artefatos**: Módulo de Conexão de Dispositivos; **Ambiente**: Operação normal; **Resposta**: O sistema apresenta a documentação para o desenvolvimento do driver; **Medição de resposta**: A documentação contém todas as informações necessárias para o desenvolvimento do driver.

6.6 Atividade 9: Análise das Abordagens Arquiteturais

As abordagens arquiteturais foram inspecionadas usando as orientações referenciadas na Seção 5, onde foram descritas as questões de *design* REST, para cada cenário. Abaixo a descrição detalhada da avaliação de cada cenário concreto.

Tabela 10 Análise Arquitetural do Cenário 1.

Resumo do cenário	Dados do sensor são enviados para EcoDiF pelo *driver*. O sistema atualiza a *feed* e responde ao *driver* em menos de cinco segundos.
Objetivos de Negócio	Permitir a comunicação de dispositivos heterogêneos com a EcoDiF.
Atributo de Qualidade	Interoperabilidade, disponibilidade, desempenho.
Análise Arquitetural	- A utilização do protocolo EEML promove interoperabilidade. (*referência: questão 5.3.3.2 - Foi definido um vocabulário padrão para o recurso?*); - Usando apenas o tipo de mídia XML a interoperabilidade é impactada negativamente (*referência: questão 5.3.3.1 – Quais os formatos usados para a representação dos recursos?*); - O URI é projetado segundo o código da *feed*. Isto promove a extensibilidade, mas impacta negativamente na interoperabilidade (*referência: questão 5.3.3.3 - Como os URIs são projetados?*); - O método utilizado para atualizar os recursos é PUT e é idempotente. Isto promove confiabilidade (*referência: questão 5.3.3.3 - Como os URIs são projetados?*); - A interface do EcoDiF responde com o código HTTP 201, mas não inclui URI do feed em resposta (*referência: questão 5.3.5.2 – O cabeçalho de resposta HTTP contém a informação sobre a identificação do recurso?).*
Riscos	- Drivers que não usam XML não podem enviar informações para EcoDiF; - A requisição não responde o URI do recurso no cabeçalho HTTP *Location*.
Tradeoffs	- O uso de diferentes tipos de mídia promove a interoperabilidade, mas aumenta a complexidade de implementação e validação.

Tabela 11 Análise do Cenário 2.

Resumo do cenário	Consumidores solicitam informações acerca de alguma *feed* e recebem a representação em menos de 1 segundo com o *media type* solicitado.
Objetivos de Negócio	Permitir a fácil visualização dos *feeds*.
Atributo de Qualidade	Interoperabilidade, disponibilidade, adaptabilidade.
Análise Arquitetural	- A URI é projetada pelo código da *feed*. Isto promove extensibilidade, mas afeta negativamente a usabilidade (*referência: questão 5.3.3.3 - Como os URIs são projetados?*); - Não é possível explicitar o tipo de mídia em solicitações. A interoperabilidade é impactada negativamente (*referência: questão 5.3.3.1 – Quais formatos usados para a representação dos recursos?*); - Não é possível explicitar a versão de recursos solicitados. Adaptabilidade é impactado negativamente (*referência: questão 5.3.3.4 – Qual a abordagem para versionamento de recursos?*); - Não é possível incluir um subconjunto de atributos na URI. Desempenho e usabilidade são impactados negativamente (*referência: questão 5.3.5.5 – É possível incluir um subconjunto de atributos desejados na URI?*); - Os recursos não estão ligados a outros. O atributo navegabilidade é impactado negativamente (*referência: questão 5.3.3.2 – Foi definido um vocabulário padrão para o recurso?*).
Riscos	- A especificação não atende a exigência nesse cenário. O sistema só pode produzir respostas XML, não é possível especificar nem a versão nem campos desejados no URI do recurso; HATEOAS não é utilizado, assim, os consumidores de serviço não pode navegar para outros recursos associados.
Tradeoffs	- Adaptabilidade é promovida pelo uso de versões de recursos, mas isso impacta na gestão de configuração de recursos - Explicitando campos solicitação melhorar o desempenho, mas aumenta a complexidade de implementar filtragem na interface do serviço

Tabela 12 Análise do Cenário 3.

Resumo do cenário	Consumidor de dados.
Objetivos de Negócio	Facilita o desenvolvimento de Drivers.
Atributo de Qualidade	Interoperabilidade, disponibilidade, adaptabilidade.
Análise Arquitetural	- A documentação da interface do EcoDiF oferece informações em documentos de texto (*referência: questão 5.3.4.1 – Como os recursos são documentados?*); - A EcoDiF não tem um *sandbox* para testar os drivers (*referência: questão 5.3.4.2 – Como os consumidores de serviços podem realizar testes nos recursos?*).

Riscos	-Sem um *sandbox* para testes, fabricantes de dispositivos não podem analisar o comportamento de resposta ou mensagens de erro antes de disponibilizar o driver para provedores de dados.
Tradeoffs	Desenvolver um *sandbox* é importante para melhorar o desenvolvimento e testes de novos drivers. No entanto, a abordagem *sandbox* afeta negativamente modificabilidade, pois modificações na interface de serviço devem ser feitas também na caixa de areia.

Tabela 13 Análise do Cenário 4.

Resumo do cenário	O consumidor de dados solicita a representação de uma *feed* em uma versão e o sistema envia a representação na versão solicitada;
Objetivos de Negócio	Permitir a interoperabilidade mesmo com diferentes versões de representação das *feeds.*
Atributo de Qualidade	Interoperabilidade, funcionalidade.
Análise Arquitetural	- A especificação não apresenta a possibilidade de solicitação de versões diferentes da representação de recursos (*referência: questão 5.3.3.4 – Qual a abordagem para versionamento de recursos?*); - A URI é projetada utilizando códigos das *feeds*, assim o usuário pode compor facilmente a URI para a solicitação de *feeds*. Isto impacta positivamente no atributo Funcionalidade (*referência: questão 5.3.3.3 - Como os URIs são projetados?*).
Riscos	Por não promover versionamento de recursos, quando ocorrer a modificação da representação, clientes que solicitam o recurso serão prejudicados.
Tradeoffs	URIs descritivas e URIs que utilizam identificadores (*referência: questão 5.3.3.3 - Como os URIs são projetados?*).

Tabela 14 Análise do Cenário 5

Resumo do cenário	O provedor de dados envia os valores de sensoriamento para uma *feed*, o sistema valida o usuário que está enviando os dados e armazena os valores da *feed*.
Objetivos de Negócio	Controlar a autenticidade dos usuários que atualizam as informações das *feeds*.
Atributo de Qualidade	Segurança
Análise Arquitetural	- O sistema não foi projetado para usar nenhum mecanismo de autenticação e autorização (*referência: questão 5.3.6.2 – Quais são as políticas de segurança para consumidores de serviço para realizarem ações nos recursos?*)
Riscos	A especificação não atende a esse cenário.
Tradeoffs	

Tabela 15 Análise do Cenário 6

Resumo do cenário	O consumidor de dados solicita a representação de uma *feed* e o sistema não envia informações privadas ao usuário solicitante.

Objetivos de Negócio	Proteger informações privadas;
Atributo de Qualidade	Segurança
Análise Arquitetural	- Através do protocolo EEML utilizado, é possível definir a visibilidade das *feeds* (*referência: questão 5.3.3.2 – Foi definido um vocabulário padrão para o recurso?*)
Riscos	A especificação não atende a esse cenário. Mesmo com a definição do recurso como privado, o sistema envia tais recursos aos usuários.
Tradeoffs	

Tabela 16 Análise do Cenário 7

Resumo do cenário	O consumidor de dados solicita a representação de uma *feed* e o sistema responde em menos de 2 segundos.
Objetivos de Negócio	Permitir desempenho na interação dos usuários com a EcoDiF
Atributo de Qualidade	Desempenho, Interoperabilidade, Disponibilidade
Análise Arquitetural	- A especificação da URI das *feeds* não permite a paginação de recursos. Assim, *feeds* com muitas informações tenderão a aumentar o tempo de resposta (*referência: questão 5.3.5.4 É necessário implementar paginação nos recursos?*); - A especificação da URI das *feeds* não permite a definição de um subconjunto de atributos desejados. Isto impacta negativamente no desempenho (*referência: questão 5.3.5.5 - É possível incluir um subconjunto de atributos desejados na URI?*); - A especificação não apresenta uma estratégia para tratar sobrecarga de requisições. Se o servidor estiver sobrecarregado, o tempo de resposta da requisição será impacto negativamente (*referência: questão 5.3.5.6 – Como proteger o servidor web de sobrecarga de requisições?*)
Riscos	A especificação atende apenas a requisições de *feeds* com poucos dados. Não existe estratégia para paginação e requisição de subconjunto de atributos.
Tradeoffs	A representação dos dados em múltiplos formatos impacta positivamente na Interoperabilidade, porém é necessário analisar seu impacto no desempenho. Por exemplo: a representação JSON, por ser mais simples, impacta positivamente no desempenho.

Tabela 17 Análise do Cenário 8

Resumo do cenário	O consumidor de dados solicita a representação de uma *feed* e, mesmo com o servidor sobrecarregado, o sistema responde em menos de 5 segundos.
Objetivos de Negócio	Permitir disponibilidade dos serviços mesmo com sobrecarga do servidor.
Atributo de Qualidade	Disponibilidade, segurança

Análise Arquitetural	- A especificação não delimita nenhuma estratégia para proteção do servidor de sobrecarga de requisições (*referência: questão 5.3.5.6 – Como proteger o servidor web de sobrecarga de requisições?*)
Riscos	Sem uma estratégia de proteção de sobrecarga, os serviços poderão responder com instabilidade em situações com onde são realizadas muitas requisições.
Tradeoffs	

Tabela 18 Análise do Cenário 9

Resumo do cenário	O provedor de dados envia informações sobre uma *feed* no formato EEML na versão n, o sistema atualiza as informações da *feed*, e informa ao usuário.
Objetivos de Negócio	Permitir o uso dos serviços mesmo com mudanças na representação das *feeds*.
Atributo de Qualidade	Modificabilidade, testabilidade, interoperabilidade
Análise Arquitetural	- A especificação não define o versionamento de recursos (*referência: questão 5.3.3.4 – Qual a abordagem para versionamento de recursos?*)
Riscos	Sem o versionamento de recursos, qualquer alteração na representação das *feeds* ocasionará em problemas para provedores de serviço que utilizem versões diferentes.
Tradeoffs	Permitindo o versionamento de recursos será necessário a implementação de testes em todas as versões disponíveis. É necessário atentar-se para a necessidade de inclusão de versionamento nos recursos, pois a modificação da representação poderá impactar negativamente na interoperabilidade.

Tabela 19 Análise do Cenário 10

Resumo do cenário	O consumidor de dados solicita informações acerca de uma *feed* através do método GET, o sistema envia a representação da *feed* e não modifica nenhum recurso.
Objetivos de Negócio	Promover proteção aos recursos
Atributo de Qualidade	Proteção
Análise Arquitetural	- A especificação atente ao cenário (*referência: questão 5.3.3.5 – Como são mapeadas as operações nos recursos em verbos HTTP?*).
Riscos	
Tradeoffs	

Tabela 20 Análise do Cenário 11

Resumo do cenário	O consumidor de dados solicita informações acerca de uma *feed*, e o sistema envia as informações contendo referências a outros recursos associados.

Objetivos de Negócio	Maximizar a descoberta e navegação entre as *feeds*.
Atributo de Qualidade	Descoberta, desempenho, disponibilidade
Análise Arquitetural	- A especificação não provê mecanismos de HATEOAS (*referência: questão 5.3.3.2 – Foi definido um vocabulário padrão para o recurso?*).
Riscos	Sem mecanismos de HATEOAS todo o recurso é descrito em uma mesma representação. Isto impacta negativamente no desempenho.
Tradeoffs	Com HATEOAS o atributo Descoberta é impactado positivamente, porém, para a navegabilidade nos recursos relacionados são necessárias novas requisições ao servidor, podendo impactar negativamente na disponibilidade.

Tabela 21 Análise do Cenário 12

Resumo do cenário	Um fabricante deseja desenvolver um novo driver e o sistema apresenta toda a documentação contém todas as informações necessárias para o desenvolvimento do driver.
Objetivos de Negócio	Promover o desenvolvimento de drivers.
Atributo de Qualidade	Funcionalidade
Análise Arquitetural	- A especificação prevê a documentação para todos os métodos da interface (*referência: questão 5.3.4.1 – Como os recursos são documentados?*).
Riscos	Manter a documentação atualizada
Tradeoffs	

Os cenários concretos serão sumarizados e apresentados aos gerentes, arquiteto e *stakeholders* do projeto. A apresentação dos resultados é descrita na próxima atividade.

6.7 Atividade 9: Apresentação dos Resultados

A forma de apresentação dos resultados varia conforme a análise arquitetural. Ela descreve, de forma sintética, os resultados das atividades de inspeção (atividade 8). Com base na análise das abordagens arquiteturais e cenárias concretos, realizadas na atividade anterior, conclui-se que a especificação do sistema EcoDiF apresenta diversos riscos, descritos abaixo.

A representação do recurso é feita apenas com a tecnologia XML, assim sistemas que utilizam outros *media types*, como JSON, não são capazes de se comunicarem com a plataforma, isto impacta negativamente no atributo interoperabilidade. Devido a não referenciar-se a outros recursos associados na representação, a especificação não utiliza a restrição HATEOAS, e a capacidade de descoberta de recursos é impactada negativamente, pois os consumidores não poderão navegar para outros recursos apenas através da

representação da *feed*. Não existe um *sandbox* para testes, assim, fabricantes de dispositivos não são capazes de testar os *drivers* desenvolvidos. Os recursos não são versionados. Um alto risco está relacionado à segurança do sistema, pois não existe um mecanismo de autenticação e autorização para o envio de *feeds*. A velocidade de resposta da requisição tenderá a tornar-se lenta conforme o crescimento das *feeds*. Sem uma estratégia de paginação e seleção de subconjunto de atributos o atributo de qualidade desempenho será fortemente impactado de forma negativa, uma vez que as representações serão muito grandes para trafegarem na rede ou serem exibidas em um browser. Não existe uma estratégia para tratar a sobrecarga do servidor web, impactando negativamente no desempenho.

7 Avaliação do Guia Por Avaliadores Arquiteturais

7.1 Introdução

Além da prova de conceito, foi realizada uma pesquisa no formato de Survey com o objetivo de coletar evidencias empíricas que avaliassem se o guia atingiu aos seus objetivos. Esta pesquisa foi feita com base nas diretrizes propostas por Pfleeger e Kitchenham (2001) e, Kitchenham e Pfleeger (2002a, 2002b).

7.2 Participantes

Os participantes da avaliação do guia são experientes avaliadores líderes ATAM do SEI (SEI ATAM *Team Leader*s). Esta amostra representa uma parcela significativa entre profissionais e especialistas de avaliação arquitetural, uma vez que o SEI, pelo método ATAM, é um dos principais provedores de formação e serviços na área. Além disso, as *Requisições para o Guia de Avaliação REST* provêm de avaliadores do próprio Instituto. Os dois avaliadores entrevistados para a geração das requisições não participaram da avaliação de forma a não influenciarem nas respostas coletadas.

7.3 Processo de avaliação

Os avaliadores (*Team Leaders*) leram o guia e o apresentaram para as suas respectivas equipes que trabalhariam em avaliações de arquiteturas REST. As avaliações arquiteturais foram realizadas em contextos reais, onde o guia foi utilizado. Não foram propostas as atividades nas quais as diretrizes do guia poderiam fornecer melhor auxílio, ao invés disso ficou sob responsabilidade das equipes a determinação das atividades onde o guia seria utilizado. Após a avaliação, o avaliador respondeu a um questionário disponibilizado on-line.

7.4 Validade das questões e do formulário de avaliação

As questões foram definidas com base nas *Requisições para o Guia de Orientação para Avaliação REST* (RG1, RG2 e RG3) (Seção 4) e incluídas no formulário de avaliação. Após a definição das questões, foi feita uma análise no formulário com o objetivo de eliminar o *Researcher Bias* [Kitchenham, B. and Pfleeger 2002a] (questões tendenciosas a favor do resultado esperado). Para atingir a este objetivo, o formulário seguiu 4 estratégias: (i) apenas 5 questões baseadas na escala de Likert[30]; (ii) eliminação de contradições entre as questões; (iii) ordenação imparcial, de forma que uma questão anterior não influenciasse na seguinte, e; (iv)

[30] A Escala Likert é um tipo de escala de resposta psicométrica usada comumente em questionários, e é uma das escalas mais usada em pesquisas de opinião. Ao responderem a um questionário baseado nesta escala, os respondentes especificam seu nível de concordância com uma afirmação.

omissão da proveniência do guia (autores, instituições, etc.), apenas foi explicitado que o guia era fruto de um projeto de pesquisa.

Para cada questão o avaliador deveria atribuir uma nota de 1 a 5, onde 1 caracterizava-se como “discordo totalmente” e 5 “concordo totalmente”. As questões são apresentadas a seguir:

RG1- "Explicar os fundamentos do estilo REST do ponto de vista de um avaliador arquitetural";

> **Questão 1**: O guia REST me ajudou a entender melhor os princípios REST
> **Questão 2**: As questões de design presentes no guia são úteis para decisões em arquiteturas baseadas no estilo REST;

RG2- "Apresentar os cenários gerais de atributos de qualidade segundo o estilo REST";

> **Questão 3**: Os cenários gerais propostos no guia são úteis para gerar a árvore de atributos de qualidade e cenários concretos;

RG3- "Discutir em detalhes como o REST contribui para os atributos de qualidade e *tradeoffs* entre eles".

> **Questão 4**: O guia é útil para analisar as abordagens arquiteturais;
> **Questão 5:** Com o guia eu pude identificar os riscos e tradeoffs na arquitetura analisada melhor do que sem o seu uso.

7.5 Resultados

O questionário foi respondido por 3 avaliadores (*Team Leaders*) arquiteturais que utilizaram o guia com suas equipes em avaliações reais de sistemas baseados no estilo REST. O avaliador A1 tem 4 anos de experiência com avaliações arquiteturais. Os avaliadores A2 e A3 contam com mais de 7 anos de experiência. O avaliador A1 e o avaliador A3 já avaliaram 2 arquiteturas baseadas no estilo REST. Por sua vez, o avaliador A2 já avaliou mais de 7 arquiteturas REST. Todos os avaliadores já utilizaram algum guia para assisti-los nas atividades do método ATAM. A Tabela 22 apresenta os resultados. Além das respostas dos avaliadores (A1, A2 e A3), são apresentadas as médias agrupadas por *Requisição para o Guia de Avaliação REST* (RG).

Tabela 22 Avaliação do guia por avaliadores arquiteturais ATAM

Questão	A1	A2	A3	Média	Média por RG
O guia REST me ajudou a entender melhor os princípios REST	5	3	3	3,7	**RG1**: 4,2
As questões de design presentes no guia são úteis para decisões em arquiteturas baseadas no estilo REST	5	5	4	4,7	
Os cenários gerais propostos no guia são úteis para gerar a árvore de atributos de qualidade	4	3	3	3,3	**RG2**: 3,3
O guia é útil para analisar as abordagens arquiteturais	4	5	4	4,3	**RG3**: 4,15
Com o guia eu pude identificar os riscos e *tradeoffs* na arquitetura analisada melhor do que sem o seu uso	5	4	3	4,0	

7.6 Análise dos Resultados

O objetivo do guia proposto nesta dissertação é auxiliar avaliadores arquiteturais na mitigação de riscos na realização de atributos de qualidade e análise do *tradeoff* entre eles na inspeção de arquiteturas baseadas no estilo REST. Para atingir a este objetivo, buscou-se atender às três requisições (RG1, RG2 e RG3) que foram identificadas através de entrevistas com experientes avaliadores arquiteturais (Seção 4.2.1). A primeira requisição RG1, a saber, "Explicar os fundamentos do estilo REST do ponto de vista de um avaliador arquitetural" foi avaliada pelas questões 1 e 2. A pontuação média das respostas, 4.2, indica que o guia apresentou diretrizes que auxiliaram na compreensão dos fundamentos REST. A requisição RG3, “Discutir em detalhes como o REST contribui para os atributos de qualidade e tradeoffs entre eles”, avaliada pelas questões 4 e 5, obteve o valor médio de 4.15, indicando que as diretrizes demonstraram a influência de REST em atributos de qualidade e o *tradeoff*. Por fim, a requisição R2, “Apresentar os cenários gerais de atributos de qualidade segundo o estilo REST” obteve a média de 3.3. Isto evidenciou que, mesmo sendo necessário reanalisar os cenários gerais de atributos de qualidade de forma a atender melhor a requisição, os cenários gerais de atributos de qualidade foram úteis para a geração dos cenários concretos de atributos de qualidade.

O autor desta dissertação está ciente de que o número baixo da amostra, 3 avaliadores, poderia invalidar os resultados obtidos. Além disso, este número pequeno de participantes não permite muitas conclusões com relação aos objetivos do guia. Porém, conforme apresentado, os participantes contam com uma vasta experiência e trabalham em um dos institutos mais respeitados e referenciados do mundo na área de avaliação arquitetural. Assim, os resultados

obtidos na avaliação mostram maior confiabilidade do que uma avaliação realizada com uma amostra maior, porém, por avaliadores inexperientes.

8 Considerações Finais

A avaliação arquitetural mostra-se como uma prática eficiente para identificar e mitigar riscos na realização de atributos de qualidade. A proposta desta dissertação é fornecer um guia contendo diretrizes para as atividades de avaliação arquitetural baseada em cenários de sistemas que usam o estilo REST, auxiliando avaliadores na mitigação de riscos na realização de atributos de qualidade e análise do *tradeoff* entre eles. Conforme demonstrado na prova de conceito e através da avaliação por avaliadores arquiteturais do SEI, com o auxílio do guia proposto é possível uma identificação mais aprofundada acerca de riscos na realização de atributos de qualidade e o *tradeoff* em arquiteturas baseadas no estilo REST antes da implementação do sistema. Desta forma, mitigando riscos e problemas que poderiam ocorrer durante o ciclo de vida da aplicação.

8.1 Contribuições

A principal contribuição deste trabalho é o guia desenvolvido e já utilizado por equipes de avaliação arquitetural ATAM do SEI. Além do método ATAM, o guia pode ser utilizado em conjunto a outros métodos de avaliação arquitetural. Apesar de possuir maior aderência aos métodos de avaliação arquitetural baseados em cenário, o guia pode ser utilizado para inspecionar outras especificações arquiteturais baseada em REST, auxiliando em questões importantes acerca da realização de atributos de qualidade. Por fim, o guia também pode ser aplicado em treinamentos REST, apresentando as melhores práticas de acordo com requisitos de atributos de qualidade e *tradeoffs* entre eles.

Outra contribuição deste trabalho é a metodologia utilizada para delimitar as questões de design REST de acordo com atributos de qualidade. A metodologia pode ser utilizada para a pesquisa do impacto de princípios arquiteturais de qualquer estilo, padrão ou abordagem arquitetural em atributos de qualidade. Por exemplo, é possível utilizar o protocolo para analisar o impacto da abordagem Domain-driven Design [Evans 2003], que apresenta restrições para o *design* de sistemas baseados no paradigma orientado a objetos, em atributos de qualidade.

8.2 Limitações

Em seu estado atual, o guia de orientação apresenta algumas limitações, sendo as principais:

- Conforme demonstrado na avaliação, uma limitação deste trabalho, apontada por um dos entrevistados, está relacionada aos cenários gerais de atributos de qualidade;
- A metodologia não define uma atividade para validação das questões de design;
- As referências das questões de design da Seção 5.3 aos cenários gerais da Seção 5.2 estão confusas. Ao ler as orientações de design, é necessário retornar aos cenários gerais, o que pode causar confusão no entendimento.

8.3 Trabalhos em andamento e futuros

Diante das limitações elencadas na Seção 8.2, trabalhos futuros poderão ser desenvolvidos a partir deste trabalho, e alguns deles já estão em andamento. Por exemplo, para diminuir a necessidade de consulta dos cenários gerais a partir das questões de design, está sendo desenvolvida uma ferramenta web que dinamicamente apresenta tais associações. Além disso, esta ferramenta web facilitará a consulta rápida nas orientações do guia.

Será feita uma nova pesquisa com os avaliadores arquiteturais que já utilizam o guia de orientação para localizar os pontos de falha nos cenários gerais de atributos de qualidade. Conforme demonstrado na pesquisa, os cenários gerais não atingiram ao resultado esperado. A partir das informações coletadas, os cenários gerais serão novamente compilados.

Outro trabalho futuro importante a ser desenvolvido é a investigação outros elementos que influenciam as questões de design e requisitos de atributos de qualidade para sistemas de REST, tais como: (i) a integração de sistemas legados, (ii) a representação de metadados, e (iii) segurança com SSL. Pretende-se também investigar como Computação em Nuvem apresenta impacto nas questões de *design* REST e cenários de atributos de qualidade gerais.

Referências

ALA. RUSA Guide to Policies and Procedures. 2013. Disponível em: http://www.ala.org/rusa/about/policies/developingguidelines/4developingguidelines. Accesso em: Jan. 2014.

ATZORI, L., IERA, A. AND MORABITO, G. The Internet of Things: A survey. **Computer Networks**, New York, v. 54, n. 15, p. 2787–2805. Out. 2010

BARBACCI, M. R., KLEIN, M. H., LONGSTAFF, T. A. AND WEINSTOCK, C. B.. Quality Attributes. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, **Technical Report CMU/SEI-95-TR-021**. 1995.

BASS, L., CLEMENTS, P. AND KAZMAN, R. **Software Architecture in Practice**. SEI Series in Software Engineering. Addison-Wesley Professional. p. 640. Out. 2012

BASS, L., KLEIN, M. H. AND MORENO, G. Applicability of General Scenarios to the Architecture Tradeoff Analysis Method. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, **Technical Report CMU/SEI-2001-TR-014**, 2001.

BECK, K. **Test Driven Development: By Example**. Addison-Wesley Professional. p. 240. 2002

BECKER, M. AND DIAZ-HERRERA, J. L. Creating domain specific libraries: a methodology and design guidelines. In: THIRD INTERNATIONAL CONFERENCE ON SOFTWARE REUSE: ADVANCES IN SOFTWARE REUSABILITY. **Proceedings of 3rd International Conference on Software Reuse**, 1994.

BERNERS-LEE, T., 1996. Univeral Resource Identifiers - Axioms of Web architecture. Disponível em http://www.w3.org/DesignIssues/Axioms.html. Acesso em: jan 2014.

BIANCO, P., KOTERMANSKI, R. AND MERSON, P. Evaluating a Service-Oriented Architecture. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, **Technical Report CMU/SEI-2007-TR-015**, 2007.

CLEMENTS, P., BACHMANN, F., BASS, L., ET AL. **Documenting Software Architectures: Views and Beyond**. 2ª Edição. Addison-Wesley Professional, 2012

CLEMENTS, P., KAZMAN, R. AND KLEIN, M. **Evaluating Software Architectures: Methods and Case Studies**. Addison-Wesley Professional, 2001

COSTA, B., PIRES, P. F., DELICATO, F. C. AND MERSON, P. Evaluating a Representational State Transfer (REST) Architecture - What is the impact of REST in my architecture? In: ELEVENTH WORKING IEEE/IFIP CONFERENCE ON SOFTWARE ARCHITECTURE, Sydney. **Proceedings of the Eleventh Working IEEE/IFIP Conference on Software Architecture**, p.7 - 11, 2014

DELICATO, F. C., PIRES, P. F., BATISTA, T., ET AL. Towards an IoT ecosystem. In: FIRST INTERNATIONAL WORKSHOP ON SOFTWARE ENGINEERING FOR

SYSTEMS-OF-SYSTEMS, Montpellier, France, 2013. **Proceedings of the First International Workshop on Software Engineering for Systems-of-Systems,** New York: ACM, 2013.

DOBRICA, L. AND NIEMELA, E. A survey on software architecture analysis methods. **IEEE Transactions on Software Engineering**, v. 28, n. 7, p. 638–653. Jul 2002

ERL, T., CARLYLE, B., PAUTASSO, C., ET AL. **SOA with REST: Principles, Patterns & Constraints for Building Enterprise Solutions with REST** (The Prentice Hall Service Technology Series from Thomas Erl). Prentice Hall, 2012

EVANS, E. **Domain-Driven Design: Tackling Complexity in the Heart of Software.** Addison-Wesley Professional, 2003.

FIELDING, R., GETTYS, J., MOGUL, J., ET AL. (1999). **RFC 2616: Hypertext Transfer Protocol -- HTTP/1.1**. Disponível em: http://www.w3.org/Protocols/rfc2616/rfc2616.html. Aesso em: Jan. 2014

FIELDING, R. T.-T. R. N. **Architectural styles and the design of network-based software architectures**. Dissetação de Doutorado, 2000.

GAMMA, E., HELM, R., JOHNSON, R. AND VLISSIDES, J. **Design Patterns: Elements of Reusable Object-Oriented Software**. Addison-Wesley Professional, 1994.

GORTON, I. **Essential Software Architecture**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011

HILBURN, T. B., HISLOP, G., LUTZ, M., MCCRACKEN, M. AND MENGEL, S. (1999). **Guidelines for Software Engineering Education**. Disponível em: http://resources.sei.cmu.edu/library/asset-view.cfm?assetID=13565. Acesso em: Jan. 2014

ISO/IEC (2007). ISO/IEC 9126-1:2001, **Software engineering -- Product quality -- Part 1: Quality model. Multiple**. Distributed through American National Standards Institute (ANSI). p. 32. 2001

ISO/IEC (2011). ISO/IEC FDIS 42010 IEEE P42010/D9. **Systems and software engineering — Architecture description**.

KAZMAN, R., ABOWD, G., BASS, L. AND CLEMENTS, P. (1996). Scenario-based analysis of software architecture. **IEEE Software**, v. 13, n. 6, p. 47–55. 1996

KAZMAN, R., KLEIN, M., BARBACCI, M., ET AL. The architecture tradeoff analysis method. In: **Fourth IEEE International Conference on Engineering of Complex Computer Systems**, Monterey, EUA. PROCEEDINGS OF THE FOURTH IEEE INTERNATIONAL CONFERENCE ON ENGINEERING OF COMPLEX COMPUTER SYSTEMS PROCEEDINGS, 1998.

KAZMAN, R., KLEIN, M. AND CLEMENTS, P. ATAM : Method for Architecture Evaluation. CMUSEI, v. 4, n. August 2000.

KITCHENHAM, B. A. AND PFLEEGER, S. L. Principles of survey research part 2: designing a survey. **ACM SIGSOFT Software Engineering Notes**, v. 27, n. 1, p. 18–20, Jan. 2002

KITCHENHAM, B. A. AND PFLEEGER, S. L. Principles of survey research: part 3: constructing a survey instrument. **ACM SIGSOFT Software Engineering Notes**, v. 27, n. 2, p. 20. Mar. 2002

KITCHENHAM, B., PEARL BRERETON, O., BUDGEN, D., ET AL. Systematic literature reviews in software engineering – A systematic literature review. **Information and Software Technology**, v. 51, n. 1, p. 7–15. 2009

Kitchenham, B. and Pfleeger, S. L. Principles of survey research part 4: questionnaire evaluation. **ACM SIGSOFT Software Engineering Notes**, v. 27, n. 3, p. 20, Mai. 2002

Kitchenham, B. and Pfleeger, S. L. Principles of survey research: part 5: populations and samples. **ACM SIGSOFT Software Engineering Notes**, v. 27, n. 5, p. 17. Set. 2002

Kitchenham, B. and Pfleeger, S. L. Principles of survey research part 6: data analysis. **ACM SIGSOFT Software Engineering Notes**, v. 28, n. 2, p. 24. Mar. 2003

Kruchten, P. B. The 4+1 View Model of architecture. **IEEE Software**, v. 12, n. 6, p. 42–50, 1995

Li, Z. and Zheng, J. Toward industry friendly software architecture evaluation. In: 7TH EUROPEAN CONFERENCE ON SOFTWARE ARCHITECTURE. **Lecture Notes in Computer Science**. Springer Berlin Heidelberg, 2013.

Naab, M., Thorsten, K. and Knodel, J. 2013. **All Architecture Evaluation Is Not the Same: Lessons Learned from More Than 50 Architecture Evaluations in Industry** | SEI Digital Library. Disponível em: http://resources.sei.cmu.edu/library/asset-view.cfm?assetID=48380. Acesso em Jan, 2014.

Novak, W. E., Cohen, J. B., Lattanze, A. J., et al. (2005). **Software Acquisition Planning Guidelines**. Disponível em: http://resources.sei.cmu.edu/library/asset-view.cfm?assetID=7293. Acesso em Kan. 2014

OAuth (2013). **OAuth Community Site**. Disponível em: http://oauth.net/, Acesso em Jan. 2014.

Oishi, S. M. **How to Conduct In-Person Interviews for Surveys**. SAGE Publications, Inc, 2002

OpenID (2013). **OpenID Foundation website**. Disponível em: http://openid.net/, Acesso em Jan. 2014.

Pfleeger, S. L. and Kitchenham, B. A. Principles of survey research: part 1: turning lemons into lemonade. **ACM SIGSOFT Software Engineering Notes**, v. 26, n. 6, p. 16. Nov. 2001

Pressman, R. S**oftware Engineering: A Practitioner's Approach**. McGraw-Hill Science/Engineering/Math: 2009

Richardson, L. and Ruby, S. (2007). R**estful Web Services**.1ª ed. O'Reilly Media: 2007

Roy, B. and Graham, N. Methods for Evaluating Software Architecture: A Survey - **TechRepublic**: 2008.

Sommerville, I. (2010). **Software Engineering**, 9ª ed. Addison-Wesley: 2010

Vinoski, S.. REST Eye for the SOA Guy. **IEEE Internet Computing**, v. 11, n. 1, p. 82–84. Já. 2007

Webber, J., Parastatidis, S. and Robinson, I. **REST in Practice: Hypermedia and Systems Architecture**. O'Reilly Media: 2010

Wilde, E. and Pautasso, C. **REST: From Research to Practice**. New York, NY: Springer: 2011

Wozniak, E., Mraidha, C., Gerard, S. and Terrier, F. A Guidance Framework for the Generation of Implementation Models in the Automotive Domain. In: EUROMICRO CONFERENCE ON SOFTWARE ENGINEERING AND ADVANCED APPLICATIONS. **Proceedings on EUROMICRO Conference on Software Engineering and Advanced Applications**, 2011.

APÊNDICE A – *CHEAT SHEET*

Durante as diversas entrevistas e interações com profissionais, arquitetos, avaliadores arquiteturais e pesquisadores, houve um grande interesse pela leitura deste trabalho. Grande parte dos entrevistados, além dos avaliadores arquiteturais do SEI, solicitaram um documento sumarizando as questões de *design* para uma consulta rápida. Este apêndice apresenta um *cheat sheat* com as questões de design a serem avaliadas ao se avaliar, inspecionar ou especificar uma arquitetura baseada no REST.

Tópico	Questão de design	Atributos de qualidade diretamente afetados
Design de Recursos	Qual é o modelo de domínio da aplicação?	
	Quais dados serão expostos como recursos?	
	As representações dos recursos são padronizadas em toda a aplicação, departamento ou organização?	Interoperabilidade e Desempenho
	Que tipos de consumidores de serviço vão interagir com os serviços REST?	Interoperabilidade e Segurança
	Alguma informação confidencial está sendo exposta como um recurso?	Segurança
Representação e Identificação	Quais os formatos usados para a representação dos recursos?	Desempenho e Interoperabilidade
	Foi definido um vocabulário padrão para o recurso?	Interoperabilidade e Desempenho
	Como os URIs são projetados?	Confiabilidade, Modificabilidade e Funcionalidade
	Qual a abordagem para versionamento de recursos?	Interoperabilidade, Modificabilidade e Confiabilidade
	Como são mapeadas as operações nos recursos em verbos HTTP?	Funcionalidade, Proteção
Documentação e Testes	Como os recursos são documentados?	Funcionalidade
	Como os consumidores de serviços podem realizar testes nos recursos?	Testabilidade
Comportamento	Quais os códigos de status HTTP presentes na resposta?	Funcionalidade e Confiabilidade
	O cabeçalho de resposta HTTP contém a informação sobre a identificação do recurso?	Funcionalidade
	O recurso pode ser comunicar com Open APIs?	Interoperabilidade, Funcionalidade e Segurança
	É necessário implementar paginação nos recursos?	Desempenho e Funcionalidade
	É possível incluir um subconjunto de atributos desejados na URI?	Desempenho e Funcionalidade
	Como proteger o servidor web de sobrecarga de requisições?	Disponibilidade e Desempenho

Segurança	Os recursos privados são projetados de forma segura?	Segurança
	Quais são as políticas de segurança para consumidores de serviço para realizarem ações nos recursos?	Segurança