**UNIVERSIDADE FEDERAL DE SANTA CATARINA**
**DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA**

Ivan Luiz Salvadori

# DESENVOLVIMENTO DE WEB APIS RESTFUL SEMÂNTICAS BASEADAS EM JSON

Florianópolis

2015

Ivan Luiz Salvadori

# DESENVOLVIMENTO DE WEB APIS RESTFUL SEMÂNTICAS BASEADAS EM JSON

Dissertação submetida ao Programa de Pós-Graduação em Ciência da Computação para a obtenção do Grau de Mestre em Ciência da Computação. Orientador: Prof. Frank Augusto Siqueira

Florianópolis

2015

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Salvadori, Ivan Luiz
Desenvolvimento de Web APIs RESTful Semânticas baseadas em JSON / Ivan Luiz Salvadori ; orientador, Frank Augusto Siqueira - Florianópolis, SC, 2015.
158 p.

Dissertação (mestrado) - Universidade Federal de Santa Catarina, Centro Tecnológico. Programa de Pós-Graduação em Ciência da Computação.

Inclui referências

1. Ciência da Computação. 2. Sistemas Distribuídos. 3. Web Services. 4. Web APIs RESTful. 5. Web Semântica. I. Siqueira, Frank Augusto. II. Universidade Federal de Santa Catarina. Programa de Pós-Graduação em Ciência da Computação. III. Título.

Ivan Luiz Salvadori

**DESENVOLVIMENTO DE WEB APIS RESTFUL SEMÂNTICAS BASEADAS EM JSON**

Esta Dissertação foi julgada aprovada para a obtenção do Título de "Mestre em Ciência da Computação", e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.

Florianópolis, 10 de fevereiro 2015.

---

Prof. Dr. Ronaldo dos Santos Mello
Coordenador do Curso

**Banca Examinadora:**

---

Prof. Roberto Willrich
Universidade Federal de Santa Catarina

---

Prof. Frank Augusto Siqueira
Orientador

---

Prof. Lau Cheuk Lung
Universidade Federal de Santa Catarina

Prof. Carlos Alberto Kamienski
Universidade Federal do ABC

## RESUMO

Os princípios arquiteturais REST estão sendo amplamente adotados nas implementações de sistemas distribuídos disponíveis na Web. As implementações REST são realizadas através de Web Services, que são comumente disponibilizados na forma de Web APIs, cujo principal objetivo é o intercâmbio de dados entre aplicações. No entanto, devido à falta de padrões e diretrizes para desenvolvimento, cada implementação REST segue uma linha de desenvolvimento, fato que resulta na dificuldade de construção de aplicações clientes, além de dificultar a composição de Web APIs. Outro desafio é a falta de suporte ao uso de controles hipermídia em representações de recursos que utilizam o formato JSON. Controles hipermídia podem assumir a forma de *links*, que guiam a navegação entre diferentes recursos. Este trabalho propõe uma abordagem de desenvolvimento com os passos necessários para modelar e implementar Web APIs RESTful Semânticas que suportam controles hipermídia em JSON. A abordagem identifica as representações necessárias e realiza anotações semânticas através de associações de propriedades e operações com termos de vocabulários controlados. Além disso, a abordagem busca separar a implementação da camada de integração de dados com as demais camadas do sistema, diminuindo o acoplamento e aumentando a coesão. De forma complementar, este trabalho apresenta um *framework* para o desenvolvimento de Web APIs RESTful baseado na abordagem de desenvolvimento proposta, sendo este compatível com a especificação JAX-RS. O suporte fornecido pelo *framework* proposto possibilita que o desenvolvedor concentre esforços no desenvolvimento do domínio do problema, sem perder tempo com infraestrutura. Com a utilização da abordagem proposta, juntamente com suporte ferramental oferecido pelo *framework*, espera-se proporcionar maior produtividade e qualidade no desenvolvimento de Web APIs alinhadas com os princípios arquiteturais REST.


**Palavras-chave:** RESTful Web API. Descrição Semântica. Controle Hipermídia. Linked Data.

## ABSTRACT

The REST architectural principles are being widely adopted for implementing distributed systems on the Web. REST implementations are developed using Web Services technology and are made available through Web APIs, which have the main goal of interchanging data among applications. However, due the lack of development patterns and guidelines, each REST implementation follows its own development method. As a result, client applications are difficult to build and the composition of Web APIs becomes hard to achieve. Another challenge is the lack of support of hypermedia control in resource representations described using the JSON format. Hypermedia controls can be realized using hyperlinks, which guide navigation among multiple resources. This work proposes a development approach with the required steps for modeling and implementing Semantic RESTful Web APIs with support for hypermedia control in JSON format. The approach identifies resource representations and applies semantic annotations in order to bind properties and operations with terms defined by controlled vocabularies. On top of that, the development approach aims to achieve loose coupling and high cohesion through a clear division between the integration layer and the others layers of the system. In addition, this work proposes a framework based on the JAX-RS specification that provides support for the proposed development approach. The framework allows the software developer to focus on the implementation of the business core, instead of spending time to deal with infrastructure issues. The adoption of the proposed approach, together with the support provided by the framework, aims to improve the quality and productivity achieved by the development of Web APIs that follow the REST architectural principles.


**Keywords:** RESTful Web API. Semantic Description. Hypermedia Control. Linked Data.

## LISTA DE FIGURAS

## LISTA DE TABELAS

## LISTAGENS

## LISTA DE ABREVIATURAS E SIGLAS

**SUMÁRIO**

# 1 INTRODUÇÃO

## 1.1 CONTEXTUALIZAÇÃO

REST (*Representational State Transfer*) foi proposto por Fielding (2000) e reúne uma série de princípios e restrições arquiteturais para o desenvolvimento de aplicações distribuídas. As informações manipuladas pela abordagem REST são organizadas em recursos Web, que constituem um conjunto coeso e mínimo de dados.

Dentre as restrições destacam-se: comportamento *stateless* do servidor e a manipulação de dados através de interface uniforme, com destaque especial para a utilização de controles hipermídia. Web Services REST, também chamadas de Web APIs, são as opções mais comuns para implementar sistemas distribuídos que seguem os princípios arquiteturais REST. Quando uma implementação segue todos estes princípios e restrições, a implementação é considerada RESTful.

Web APIs utilizam o HTTP como protocolo de comunicação, o que implica em obedecer a semântica deste protocolo, ou seja, utilizar corretamente os verbos (GET, POST, PUT e DELETE) e os códigos de *status*. Além de utilizar o protocolo HTTP como base, as implementações de Web APIs utilizam a Web como infraestrutura de desenvolvimento (WEBBER; PARASTATIDIS; ROBINSON, 2010). Utilizar a Web como infraestrutura implica a adoção tecnologias padronizadas, bem como o emprego correto dos protocolos usados pelas aplicações. Os princípio arquiteturais REST diferem do conceito tradicional de serviço ao manipular as informações no formato de recursos. Recursos encapsulam as informações manipuladas por Web APIs em um conjunto coeso e significativo de informações. Mesmo adotando o design de recursos, as implementações REST continuam sendo uma abordagem baseada em serviços.

Uma das razões para utilizar Web APIs é realizar integração de dados. Seu uso possibilita que diferentes aplicações troquem informações com um nível mais baixo de acoplamento, comparado ao compartilhamento de dados através de base de dados. Compartilhamento de bases de dados tem sua aplicação restrita a uma determinada organização, entretanto, com o uso de Wep APIs é possível compartilhar informações em escala global através da Web. Ao disponibilizar dados na Web, perde-se o controle sobre as aplicações clientes, tornando heterogêneo o consumo dos dados e a forma de interação entre cliente e servidor.

Desenvolver Web APIs com foco em consumidores desconhecidos e espalhados pela Web, de certa forma, trata-se de publicação de dados na Web. Independentemente da escala, a integração de dados é melhor realizada quando as informações estão descritas semanticamente, fato que transforma a Web de recursos isolados na Web de Dados (BERNERS-LEE; HENDLER; LASSILA, 2001).

No contexto de Web APIs, as informações são transmitidas em formatos mais adequados para serem consumidas por aplicações, pois os seres humanos não são os principais consumidores. Os formatos mais utilizados são XML e JSON (RICHARDSON; AMUNDSEN; RUBY, 2013, p. 239). Sporny et al. (2013) criaram o formato JSON-LD para suportar descrições semanticas de dados em documentos JSON. Embora JSON-LD represente um grande avanço, limita-se ao desenvolvimento de Web APIs somente-leitura, pois não tem suporte para descrever a semântica necessária para alteração de recursos Web (RICHARDSON; AMUNDSEN; RUBY, 2013). Surge então Hydra (LANTHALER, 2013) como uma proposta para desenvolver Web APIs RESTful com suporte a controles hipermídia, permitindo a leitura e alteração de recursos. No contexto de Web APIs, hipermídia representa controles que permitem a navegação e transformação de estados de recursos, dos quais pode-se destacar os *links* e formulários. Existem ainda *frameworks* que, embora não suportem descrição semântica dos dados, permitem a implementação de Web APIs RESTful que utilizam JSON, como por exemplo Spring HATEOAS (SPRING, 2013) e Apache Isis (BIENVENIDO, 2013).

Apenas suporte de tecnologias e *frameworks* não são suficientes para o desenvolvimento de Web APIs com suporte a controles hipermídia. É necessário definir métodos e diretrizes para guiar a implementação. Neste contexto, Robinson (2011) e Richardson, Amundsen e Ruby (2013) apresentam métodos para projetar Web APIs. Robinson (2011) apresenta a proposta de *Domain Application Protocols* que especializa um protocolo para um determinado domínio. Richardson, Amundsen e Ruby (2013) apresentam um conjunto de passos que guiam o desenvolvimento de Web APIs com uma perspectiva mais ampla.

## 1.2 PROBLEMA DE PESQUISA

Os princípios arquiteturas REST estão em grande parte baseados em representações de recursos que utilizam controles hipermídia. Entretanto, tais controles não são suportados em Web APIs que utilizam o formato de dados JSON. Além disso, não existe uma aborda-

gem que abrange todos os detalhes necessários para o desenvolvimento de Web APIs com suporte a controles hipermídia e descrição semântica. A carência de padrões e modelos faz com que cada Web API seja única, dificultando a comunicação com aplicações clientes (RICHARDSON; AMUNDSEN; RUBY, 2013).

## 1.3 MOTIVAÇÃO E JUSTIFICATIVA

Web APIs que não possuem suporte a controles hipermídia apresentam uma limitação muito importante. Essa limitação pode ser notada quando a Web API sofre alguma modificação, fato que exige a modificação das aplicações clientes. Este fato ocorre devido aos clientes serem fortemente acoplados à implementação do servidor. Web APIs que utilizam controles hipermídia oferecem aos clientes mais flexibilidade a mudanças.

Anotar semanticamente os dados gerenciados pela Web API permite a utilização mais adequada das informações, além de possibilitar o equilíbrio entre comportamento genérico e personalizado dos clientes. São chamadas de anotações semânticas as descrições de recursos que referenciam conceitos de um vocabulário (EUZENAT, 2002). Anotar semanticamente um recurso consiste em associar um ou mais termos de um ou mais vocabulários a um determinado recurso. Outro fato positivo do uso de anotações semânticas é a possibilidade de criação de clientes autônomos capazes de interagir adequadamente sem a intervenção humana. Por fim, a oportunidade de interagir com diferentes Web APIs de forma padronizada resulta em uma integração mais rica, além de facilitar a composição e descoberta de novas Web APIs.

## 1.4 OBJETIVOS

### 1.4.1 Objetivos gerais

Este trabalho tem o objetivo de proporcionar subsídio metodológico e ferramental para o desenvolvimento de Web APIs RESTful que manipulam representações semânticas de recursos. O subsídio metodológico tem por objetivo servir como guia para modelagem de sistemas distribuídos que adotam a abordagem REST. O subsídio ferramental é constituído por um *framework* com suporte a controles hipermídia para representações semânticas no formato JSON-LD.

### 1.4.2 Objetivos específicos

Os objetivos gerais do trabalho podem ser alcançados através dos seguintes objetivos específicos:

- Desenvolver uma abordagem para o desenvolvimento de Web APIs RESTful capazes de manipular representações semânticas. A abordagem deve separar a camada de integração de dados das demais camadas do sistema, a fim de proporcionar flexibilidade e manutenibilidade.

- Reduzir o acoplamento entre Web APIs e aplicações clientes, diminuindo o impacto gerado pela evolução dos sistemas.

- Proporcionar suporte ferramental através de um *framework* compatível com a especificação JAX-RS para a linguagem de programação Java, baseado no padrão JSON-LD e no vocabulário Hydra.

- Implementar mecanismos para geração de documentação de Web APIs para desenvolvedores e agentes de software.

- Criar um meio de classificar e medir a qualidade das implementações de Web APIs RESTful Semânticas.

## 1.5 MÉTODO DE PESQUISA

Este trabalho possui caráter experimental, com objetivo de desenvolver e aprimorar técnicas através da implementação de suporte ferramental e metodológico, elaboração de estudos de caso, implementação e análise dos resultados obtidos. O método utilizado neste trabalho é constituído de:

- **Identificação do problema:** através da revisão bibliográfica nas áreas de descrição semântica, Serviços Web, REST e composição de serviços. Com base nos resultados obtidos da revisão bibliográfica, identificar o estado da arte.

- **Elaboração:** de uma abordagem para o desenvolvimento de Web APIs RESTful Semânticas com base no estado da arte.

- **Desenvolvimento:** de uma ferramenta de apoio para a abordagem proposta. O desenvolvimento contempla também a implementação de um estudo de caso que utiliza a abordagem e o *framework* propostos.

- **Criação:** de um vocabulário capaz de representar o domínio do problema estudado, além de possibilitar a composição de diferentes Web APIs que manipulam informações do mesmo domínio.

- **Análise e avaliação:** a partir de experimentos provenientes da implementação e execução do estudo de caso, com o objetivo de identificar as vantagens, limitações e melhorias.

- **Comunicação dos resultados:** através de artigos científicos e relatórios de pesquisa. O código fonte das implementações realizadas são disponibilizadas através de repositórios públicos de código livre.

## 1.6 ESTRUTURA DA DISSERTAÇÃO

Este trabalho está organizado da seguinte forma:

- **Capítulo 1 - INTRODUÇÃO:** apresenta a contextualização e descreve o problema abordado pelo trabalho, além de apresentar a motivação e justificativa. O capítulo também descreve os objetivos e o método de pesquisa.

- **Capítulo 2 - SERVIÇOS WEB:** apresenta os conceitos de arquitetura orientada a serviços através das abordagens SOAP e REST. Descreve com mais detalhes a abordagem REST, seus fundamentos e restrições arquiteturais, além de ilustrar cenários de utilização.

- **Capítulo 3 - WEB SEMÂNTICA:** apresenta conceitos de publicação de dados na Web, além dos fundamentos e ferramentas para Web Semântica. Descreve o formato de dados JSON-LD, adotado recentemente como padrão W3C, que constitui uma das tecnologias utilizadas por este trabalho.

- **Capítulo 4 - ESTADO DA ARTE:** descreve os trabalhos relacionados relevantes a pesquisa. O estado da arte está organizado em três categorias distintas. A primeira categoria é composta por

abordagens de desenvolvimento de Web APIs. A segunda categoria é constituída por ferramentas utilizadas na implementação de sistemas. Por fim, a terceira categoria apresenta os modelos de maturidade mais utilizados para classificar Web APIs.

- **Capítulo 5 - ABORDAGEM DE DESENVOLVIMENTO:** apresenta uma proposta de abordagem para o desenvolvimento de Web APIs RESTful Semânticas. A abordagem é constituída por um conjunto de etapas que guiam o desenvolvimento de Web APIs em todo seu processo de criação. A abordagem auxilia na organização e descrição semântica das informações e proporciona diretrizes para orquestração das operações sobre os recursos Web.

- **Capítulo 6 - FRAMEWORK JAX-SRS:** apresenta o *framework* JAX-SRS (*Java Framework for Semantic RESTful Services*), como uma alternativa para o desenvolvimento de Web APIs Semânticas para a linguagem de programação Java. JAX-SRS incorpora uma série de conceitos de modelagem de software que resultam em implementações flexíveis e escaláveis de serviços Web semânticos alinhados com os princípios arquiteturais REST.

- **Capítulo 7 - ESTUDO DE CASO:** apresenta um estudo de caso com o objetivo de aplicar a abordagem de desenvolvimento proposta e o suporte ferramental oferecido pelo *framework* JAX-SRS.

- **Capítulo 8 - ANÁLISES e VALIDAÇÕES:** apresenta as análises e validações para as propostas do trabalho. A validação da abordagem de desenvolvimento e do suporte ferramental é realizada através da implantação e execução do estudo de caso. Os resultados obtidos são apresentados e discutidos.

- **Capítulo 9 - CONCLUSÕES:** apresenta as conclusões, contribuições, limitações e trabalhos futuros. Este capítulo apresenta também as publicações realizadas e os artigos em processo de avaliação.

# 2 SERVIÇOS WEB

Serviços Web, mais conhecidos como *Web Services*, permitem o compartilhamento de informações entre sistemas dispostos em diferentes localidades. Dependendo do grau de distribuição, os sistemas podem estar sujeitos à diferentes políticas. Sistemas distribuídos podem utilizar diferentes tecnologias e plataformas. Tais sistemas podem ser construídos com base na Arquitetura Orientada a Serviços, ou SOA (*Service-Oriented Architecture*), que incorpora uma série de conceitos que visam assegurar a interoperabilidade e flexibilidade exigidas por processos de negócios.

Existem duas principais formas de implementação de *Web Services*. Segundo Josuttis (2007), a forma mais tradicional de implementação é através do padrão SOAP (*Simple Object Access Protocol*), entretanto é crescente a adoção de implementações que seguem os princípios arquiteturais REST (*REpresentational State Transfer*). Este capítulo apresenta conceitos sobre *Web Services*, com ênfase nos princípios arquiteturais REST, necessários para o desenvolvimento de Web APIs RESTful Semânticas.

## 2.1 ARQUITETURA ORIENTADA A SERVIÇOS

A arquitetura SOA, pode ser definida como um paradigma para implementação e manutenção de processos de negócio em sistemas distribuídos. Não é uma arquitetura propriamente dita, nem mesmo uma tecnologia específica; a melhor definição seria estilo ou conceito, flexível a ponto de ser aplicável a diferentes circunstâncias; por essa razão é caracterizada como paradigma (JOSUTTIS, 2007).

SOA está fundamentada sobre três pilares: serviços, interoperabilidade e baixo acoplamento. Serviços representam uma funcionalidade de negócio, por exemplo: armazenar ou carregar dados de clientes. Interoperabilidade é a infraestrutura necessária para facilitar a distribuição dos processos de negócio através de sistemas distribuídos, com diferentes plataformas e tecnologias. Entende-se por baixo acoplamento o baixo nível de dependência entre sistemas, pois os impactos causados por evolução e manutenção devem ser minimizados (JOSUTTIS, 2007). SOA pode ser implementado através de *Web Services*. Entretanto, a adoção de SOA não está necessariamente vinculada a uma determinada tecnologia.

SOAP e REST são abordagem que podem utilizadas para a implementação de *Web Services*. SOAP é o modelo mais tradicional e amplamente utilizado por grandes organizações, além de ser um padrão recomendado pelo W3C. REST é uma opção que está em plena expansão, é considerado uma forma mais leve de integração e de baixa curva de aprendizagem. Por ser mais leve, alguns aspectos de segurança e controles transacionais não estão presentes no REST, sendo recomendado em cenários onde o objetivo é apenas proporcionar acesso a dados para sistemas externos (JOSUTTIS, 2007).

## 2.2 SOAP

SOAP (W3C, 2007b) foi o primeiro padrão de *Web Services* desenvolvido, que apesar do nome original (*Simple Object Access Protocol*, em desuso a partir da versão 1.2), não lida com acesso a objetos. Muitos desenvolvedores discordam também do *Simple*, pois consideram sua implementação uma tarefa repleta de desafios.

No padrão SOAP, a comunicação é realizada através do envio de um envelope de dados pelo cliente para o *Web Service*. O envelope (exemplificado na Listagem 1) é um documento no formato XML que carrega internamente o conteúdo da mensagem, e pode conter vários elementos opcionais, relacionados à infraestrutura, para lidar com as questões de segurança, roteamento, dentre outras propriedades.

Listagem 1 – Exemplo de envelope SOAP

```
1 <?xml version="1.0"?>
2 <soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-
      envelope">
3   <soap:Body>
4     ... Conteúdo da mensagem ...
5   </soap:Body>
6 </soap:Envelope>
```

*Web Services* SOAP são descritos por documentos WSDL (*Web Service Description Language*), que constituem contratos ou interfaces de serviços. O documento WSDL é capaz de descrever *Web Services* em diferentes aspectos, como detalhes do serviço (nome do serviço e seus parâmetros) e também detalhes de implantação (protocolo e localização). O documento WSDL descreve a estrutura das informações com XML Schema (exemplificado na Listagem 2) e serviços (exemplificado na Listagem 3), possibilitando que aplicações clientes sejam geradas automaticamente a partir do documento de descrição.

Listagem 2 – Exemplo de descrição de dados em WSDL

```
<types>
  <xsd:element name="getDadosPessoa" type="Pessoa"/>
  <xsd:complexType name="Pessoa">
    <xsd:sequence>
      <xsd:element name="nome" type="xsd:string"/>
      <xsd:element name="sobrenome" type="xsd:string"/>
      <xsd:element name="email" type="xsd:string"/> ...
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
</types>
```

Listagem 3 – Exemplo de descrição de serviços em WSDL

```
<portType name="">
  <operation name="listarPessoa">
    <input message="tns:listarPessoa"/>
    <output message="tns:returns_message"/>
  </operation>
</portType>
<binding name="_webservices" type="tns:">
  <soap:binding style="rpc" transport="http://schemas.
      xmlsoap.org/soap/http"/>
  <operation name="listarPessoa">
    <soap:operation soapAction="listarPessoa"/> ...
  </operation>
</binding>
```

## 2.3 REST

REST é uma coleção de princípios e restrições arquiteturais para o desenvolvimento de aplicações distribuídas na Web. Ele adota o paradigma cliente-servidor, onde as requisições partem inicialmente do cliente e são respondidas pelo servidor (FIELDING, 2000). REST é uma abordagem leve para o desenvolvimento de *Web Services*, que busca simplicidade e baixo acoplamento.

### 2.3.1 Princípios Arquiteturais

Recursos formam a base dos princípios REST e podem ser qualquer informação que se deseje tornar acessível a clientes remotos, e que são endereçados através de um identificador único, denominado URI (*Uniform Resource Identifier*). Recursos podem ser uma lista de filmes em cartaz em um cinema, comentários de um blog, uma página pessoal ou um perfil de um usuário de uma rede social, por exemplo.

Um recurso pode ser identificado por diversas URIs, mas uma URI endereça apenas um recurso. Outra característica importante é que o recurso pode ser representado de diferentes maneiras. A representação de um recurso é uma amostra dos valores de suas propriedades em um determinado momento do tempo. O recurso jamais é acessado diretamente, mas através de uma representação, sendo assim uma URI está sempre associada a pelo menos uma representação.

A Figura 1 apresenta a relação existente entre um recurso, URIs e diferentes representações. As notas finais dos alunos representam um recurso, entretanto, estão disponíveis em quatro formatos (representações) diferentes (HTML, PDF, JSON e XML). Cada representação pode ser acessada através de uma URI distinta, embora o recurso continue sendo apenas um. Não é necessário que exista uma URI exclusiva para cada representação. Pode haver uma única URI associada a diferentes representações (Figura 2). Nesse caso, a representação mais adequada é resultado da negociação entre cliente e servidor.

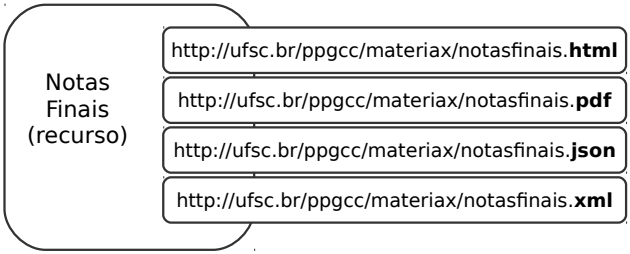


Figura 1 – Relação entre recurso, URIs e representações. Adaptado de (WEBBER; PARASTATIDIS; ROBINSON, 2010)


Figura 2 – Relação entre recurso e representações com URI única. Adaptado de (WEBBER; PARASTATIDIS; ROBINSON, 2010)

Uma das representações de recurso mais utilizadas em *Web Services* REST é o JSON (RICHARDSON; AMUNDSEN; RUBY, 2013). JSON é uma forma textual de representação de dados estruturados em uma coleção de pares no formato de chave/valor (CROCKFORD, 2006). A chave sempre é descrita como texto, e o valor pode ser expresso como texto, numero, booleano, nulo, objeto ou uma sequência ordenada de valores. JSON é muito utilizado no intercâmbio de informações, pois é independente de linguagem de programação, de fácil criação, manipulação e análise (SPORNY et al., 2013). A Listagem 4 mostra um exemplo de arquivo estruturado no formato JSON. Apesar dos benefícios, o formato JSON apresenta limitações. Este formato não possui suporte a controles hipermídia, além de ser difícil integrar dados de diferentes fontes, pois pode haver conflitos e ambiguidades nos pares chave/valor (LANTHALER; GüTL, 2012).

Listagem 4 – Exemplo de JSON

```
1 {
2   "name": "Maria Rosa",
3   "homepage": "http://mariarosa.com.br/",
4   "image": "http://mariarosa.com.br/maria.png"
5 }
```

Um dos principais princípios arquiteturais REST é o estabelecimento de uma interface uniforme entre cliente e servidor. O uso de uma interface uniforme proporciona uma arquitetura simplificada e desacoplada. Uma das formas para estabelecer uma interface uniforme é respeitar a semântica do protocolo utilizado pelo *Web Service*. O HTTP é o protocolo mais utilizados em *Web Services* REST, e respeitar a semântica do protocolo significa utilizar adequadamente os seus verbos e códigos de mensagens. Os verbos HTTP mais utilizados são:

- GET - Obter a representação de um recurso;
- POST - Criar um novo recurso;
- PUT - Alterar um recurso;
- DELETE - Remover um recurso.

Espera-se que os significados dos verbos HTTP sejam respeitados, empregando o verbo adequado para cada ação, embora muitas implementações REST negligenciem esta restrição e utilizam GET para obter, criar, alterar e remover recursos, dentre outras combinações. Outra restrição imposta pelo REST é a correta utilização de códigos de *status* ou mensagens. Todas as requisições tratadas pelo servidor

recebem um código de *status*, que informa ao cliente o resultado da requisição. Os códigos de *status* possuem tamanho fixo de três dígitos e estão organizados da seguinte forma:

- 1XX - Informações;
- 2XX - Sucessos;
- 3XX - Redirecionamentos;
- 4XX - Erros causados pelo cliente;
- 5XX - Erros causados no servidor.

Requisições que resultam em sucesso devem retornar uma resposta com um dos códigos do grupo 2XX. Outro exemplo é quando o cliente faz uma requisição que não possui autorização, onde o código de resposta deve ser 401, indicando ao cliente que o erro é de autenticação inválida. Da mesma forma, quando ocorre algum erro interno do servidor, a resposta deve apresentar o erro 500. Os códigos de *status* do protocolo HTTP fornecem uma maneira adequada de categorizar e padronizar respostas das requisições, e os princípios REST pedem atenção neste aspecto. Maiores detalhes sobre o protocolo HTTP são apresentados em (FIELDING et al., 1999).

Outra restrição arquitetural exige que as requisições contenham todas as informações necessárias para sua execução, sem recorrer a dados armazenados em sessões do usuário. Não é esperado que o servidor mantenha dados na sessão do usuário, tornando a aplicação *stateless*. É responsabilidade da aplicação cliente manter o estado dos recursos para o usuário final, ficando o servidor com a responsabilidade de disponibilizar representações de recursos e alterar seus estados. O comportamento *stateless* implica que o servidor não deve manter nenhuma informação sobre as requisições realizadas. O envio de todas as informações necessárias para a execução das requisições, somado à utilização correta do protocolo utilizado pelo *Web Service* resulta em mensagens auto-descritivas, outro importante princípio arquitetural REST.

O estilo arquitetural REST adota o princípio HATEOAS (*Hypermedia as the Engine of Application State*) (FIELDING, 2000). Este princípio define que as mudanças de estado da aplicação devem ser guiadas através de controles hipermídia (FIELDING, 2000). No contexto de REST, entende-se por aplicação o conjunto de recursos e seus respectivos estados manipulados por um cliente. Controles hipermídia podem assumir a forma de *links*, que guiam a navegação entre diferentes

recursos. Um formulários HTML é outro exemplo de controle hipermídia, que oferece mecanismos para criação ou alteração de informações. HATEOAS implica que as representações de recursos, além de conter dados, devem conter controles hipermídia com transições válidas para um determinado estado.

Uma característica importante dos recursos é a capacidade de se relacionarem com outros recursos através de *links*. *Web Services* REST podem ser projetados de modo a interligar recursos, no lugar de apenas realizar requisições e transportar dados. Por exemplo, uma página pessoal pode ser um recurso que possui *links* para outras páginas, enriquecendo e interligando os recursos. A Figura 3 apresenta o fluxograma de requisições possíveis para um determinado recurso. O ponto de entrada é o recurso *Perfil rede social*, obtido através de uma requisição GET(1). A partir deste perfil, é possível seguir três caminhos distintos. O primeiro caminho é através de GET(2), onde é possível acessar *Outro perfil* da rede social. O segundo caminho, através de GET(3), é possível obter os *Vídeos favoritos* do perfil. A requisição GET(4) é o caminho que leva a uma *Lista de comentários*. A lista possibilita criar, alterar e remover comentários, através das requisições POST(5), PUT(6) e DELETE(7), respectivamente.

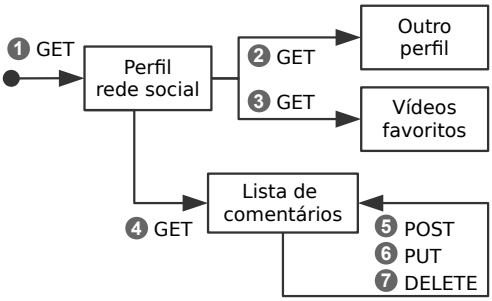


Figura 3 – Fluxograma de requisições

Ocorre neste momento uma mudança de perspectiva, onde o foco deixa de ser em serviços e passa a ser em representações de recursos e suas transições de estado. As transições são as requisições que alteram o estado de um recurso, embora nem todas as requisições causem mudança de estado. Quando o *Web Service* obedece a semântica do protocolo HTTP, fica mais fácil identificar a natureza das requisições, facilitando a utilização por parte das aplicações clientes.

### 2.3.2 Integração de Aplicações com Web APIs

Organizações necessitam interligar sistemas entre diferentes departamentos - por exemplo, o departamento de vendas precisa enviar dados para os setores financeiro, administrativo e logística. Uma solução simples e muito utilizada é através do compartilhamento de bases de dados (Figura 4), onde tabelas são criadas para armazenar dados de diferentes departamentos.

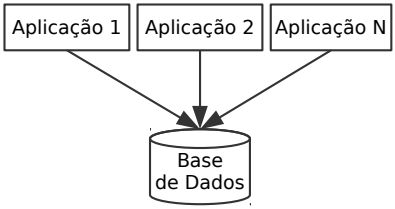


Figura 4 – Aplicações compartilhando base de dados

Esta forma de integração entre aplicações é relativamente simples e rápida de ser implementada, porém apresenta algumas desvantagens. Com a evolução dos sistemas, é inevitável que ocorram alterações (estruturais ou de conteúdo) nas bases de dados. Como diversas aplicações utilizam tabelas em comum, uma alteração pontual no banco de dados pode afetar diversas aplicações, dificultando a evolução e manutenção dos sistemas integrados.

Outra alternativa é realizar a integração através de *Web APIs* (Figura 5). O termo Web API é utilizado para designar *Web Services* que utilizam os princípios arquiteturais REST (RICHARDSON; AMUNDSEN; RUBY, 2013). Neste cenário, cada aplicação possui sua própria base de dados, sem compartilhá-la com demais sistemas. As informações são expostas através de Web APIs, que formam uma camada de integração. Os dados são representados em formatos estabelecidos, geralmente JSON e XML, e transportados através do protocolo HTTP. Com esta abordagem de integração, a Web se torna uma infraestrutura para construção de sistemas distribuídos, conforme discutido detalhadamente por Webber, Parastatidis e Robinson (2010).

Este modelo de integração possui a vantagem de reduzir o acoplamento entre as aplicações, possibilitando que evoluam em ritmos diferentes, sem grandes impactos nas demais aplicações, pois alterações nos modelos de dados não influenciam diretamente na integração.

Outro ponto positivo é a possibilidade da integração se estender para fora dos domínios da organização. Como a base de dados não é mais compartilhada, apenas dados selecionados e protegidos por controle de acesso são disponibilizados, proporcionando maior abrangência de integração. Por outro lado, acrescenta a complexidade de implementação de uma camada extra, responsável por realizar e atender chamadas a outras Web APIs, além converter os dados nos formatos estipulados.

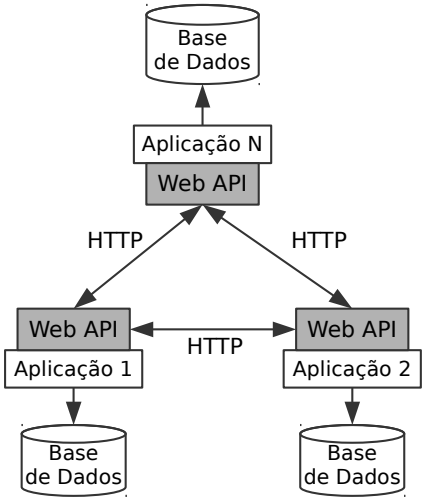


Figura 5 – Aplicações distribuídas através de Web APIs

Na integração através de base de dados, as aplicações podem executar operações de consulta, criação, alteração e remoção de dados. Aplicações baseadas principalmente nestas operações são conhecidas como CRUD (*Create, Read, Update, Delete*). É esperado que a integração através de Web API seja capaz de realizar as mesmas operações. Na realidade, grande parte das Web APIs desenvolvidas atualmente possuem a característica CRUD. Aplicações dessa natureza parecem se encaixar adequadamente ao estilo arquitetural REST. Aplicações CRUD podem facilmente ser construídas utilizando a Web como plataforma de desenvolvimento. Esta consideração baseia-se na utilização do protocolo HTTP, e no uso correto de seus métodos (verbos) GET, POST, PUT, DELETE, dentre outros. Respeitar a semântica dos métodos HTTP é uma das boas práticas (restrições de Fielding), além de ser um dos pilares do desing REST.

### 2.3.3 Descrição de Web APIs

Para interagir com uma Web API, deve-se conhecer os detalhes dos recursos e como realizar requisições. Uma possibilidade é descrever a Web API através de WADL (*Web Application Description Language*). WADL é um documento em formato XML que descreve sintaticamente todos os recursos da Web API, com os detalhes necessários para realizar requisições, além de descrever a estrutura e o formato dos dados. É semelhante ao WSDL dos *Web Services* SOAP, porém o WADL apresenta os detalhes dos recursos Web implementados, enquanto o WSDL descreve os serviços. A Listagem 5 mostra o documento WADL que descreve uma Web API de gerenciamento de pessoas.

Listagem 5 – Exemplo de descrição de recursos Rest

```
<?xml version="1.0" encoding="UTF-8"?>
<application xmlns="http://wadl.dev.java.net/2009/02">
 <grammars>
  <include href="application.wadl/xsd0.xsd">
   <doc title="Generated" xml:lang="en" />
  </include>
 </grammars>
 <resources base="http://localhost/pessoasAPI/">
  <resource path="pessoas">
   <method id="listarTodasPessoas" name="GET">...
   </method>
   <method id="cadastrarNovaPessoa" name="POST">...
   </method>
   <resource path="/{id}">
    <method id="alterarDadosPessoa" name="PUT">...
    </method>
    <method id="carregarDadosPessoa" name="GET">...
    </method>
    <method id="removerPessoa" name="DELETE">...
    </method>
   </resource>
  </resource>
 </resources>
</application>
```

A Figura 6 ilustra os detalhes das operações permitidas para cada recurso. A URL *http://localhost/pessoasAPI/pessoas* possibilita executar duas operações HTTP sobre o recurso endereçado, HTTP GET e HTTP POST. A URL *http://localhost/pessoasAPI/pessoas/{ID}* possibilita executar três operações sobre o recurso endereçado. Não é permitido executar operações com o mesmo método HTTP sobre uma determinada URL. Entretanto, é possível executar diferentes operações em uma mesma URL com métodos HTTP distintos. A URL *http://localhost/pessoasAPI/pessoas/{ID}* é um exemplo de URI *template*, que possibilita construir URLs com campos variáveis. Neste

exemplo, a variável ID representa o identificador único da pessoa, e deve ser substituído pelo valor desejado.

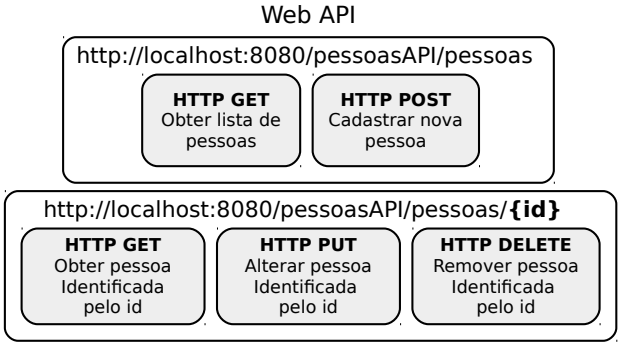


Figura 6 – Recursos e operações

Outra informação importante presente no WADL é a descrição dos dados, referenciada em *include href="application.wadl/xsd0.xsd"*. A estrutura dos dados é descrita em XSD (XML Schema), onde os elementos, propriedades e tipos de dados são detalhados. A Listagem 6 apresenta o XSD dos dados referenciados pelo WADL.

Listagem 6 – Exemplo de descrição de recursos

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    version="1.0">
 <xs:complexType name="pessoa">
  <xs:sequence>
   <xs:element name="amigos" type="pessoa" ... />
   <xs:element name="dataNascimento" type="xs:dateTime" />
   <xs:element name="email" type="xs:string" />
   <xs:element name="genero" type="xs:string" />
   <xs:element name="id" type="xs:int" />
   <xs:element name="linkFoto" type="xs:anyURI" />
   <xs:element name="nome" type="xs:string" minOccurs="0" />
   <xs:element name="posts" type="post" ... />
   <xs:element name="sobrenome" type="xs:string" />
   <xs:element name="telefone" type="xs:string" />
  </xs:sequence>
 </xs:complexType>
 <xs:complexType name="post">
  <xs:sequence>
   <xs:element name="data" type="xs:dateTime" />
   <xs:element name="id" type="xs:int" />
   <xs:element name="mensagem" type="xs:string" />
  </xs:sequence>
 </xs:complexType>
</xs:schema>
```

A Figura 7 apresenta de forma gráfica, exemplos de requisições e respectivos retornos da Web API. As requisições partem de uma aplicação cliente, com destino a Web API, transportando os dados no formato permitido(neste exemplo, JSON). Ao receber a requisição, a Web API executa o serviço e retorna uma resposta ao cliente, que deve conter o código HTTP *Status Code*, além de dados opcionais. Na requisição (1) o cliente solicita a lista de pessoas cadastradas, a Web API responde com *Status Code* 200 e define que *application/json* é o formato de dados da representação retornada pela Web API. A requisição (2) é um exemplo de criação de um novo recurso. O cliente envia os dados através da requisição HTTP POST e a Web API retorna os mesmos dados enviados, com acréscimo do identificador único gerado pela operação bem sucedida. A requisição (3) consulta o recurso criado pela operação (2). As requisições (4) e (5) correspondem respectivamente a alteração e remoção de recurso.

Descrever os recursos e suas operações através de WADL é uma forma de documentação, e serve como guia de utilização de Web APIs, sendo possível a geração automática de aplicações clientes baseadas exclusivamente no documento WADL. Porém o WADL é limitado a descrever apenas sintaticamente os serviços. Não é possível descrever o significado de uma requisição, dados transmitidos, nem mesmo o significado das variáveis dos modelos de URI(*templates*), como por exemplo: */pessoasAPI/pessoas/{ID}*. Neste exemplo, o documento WADL não responde qual o significado de {ID}. Para contornar essa deficiência, documentos auxiliares precisam ser desenvolvidos para guiar o desenvolvimento das aplicações clientes. Estes documentos auxiliares são produzidos geralmente no formato de manuais ou tutoriais, destinados exclusivamente à interpretação humana, impossibilitando a automação da invocação da Web API.

Clientes desenvolvidos através das descrições WADL são extremamente vulneráveis a alterações na Web API. Pequenas alterações (por exemplo: mudanças no formato das URIs, ou mesmo alterações estruturais dos dados) podem gerar incompatibilidades entre cliente e Web API. Outra importante limitação da descrição WADL é que as requisições permitidas para os recursos são listadas sem apontar qualquer ordem de execução. Um exemplo da importância da ordem de execução é percebido com as operações de alterar dados e remover pessoas. Essas operações só podem ser executadas após uma chamada da operação de cadastro. Nota-se que a descrição WADL não é suficiente para desenvolver clientes REST, além de torná-los sensíveis à eventuais mudanças da Web API.

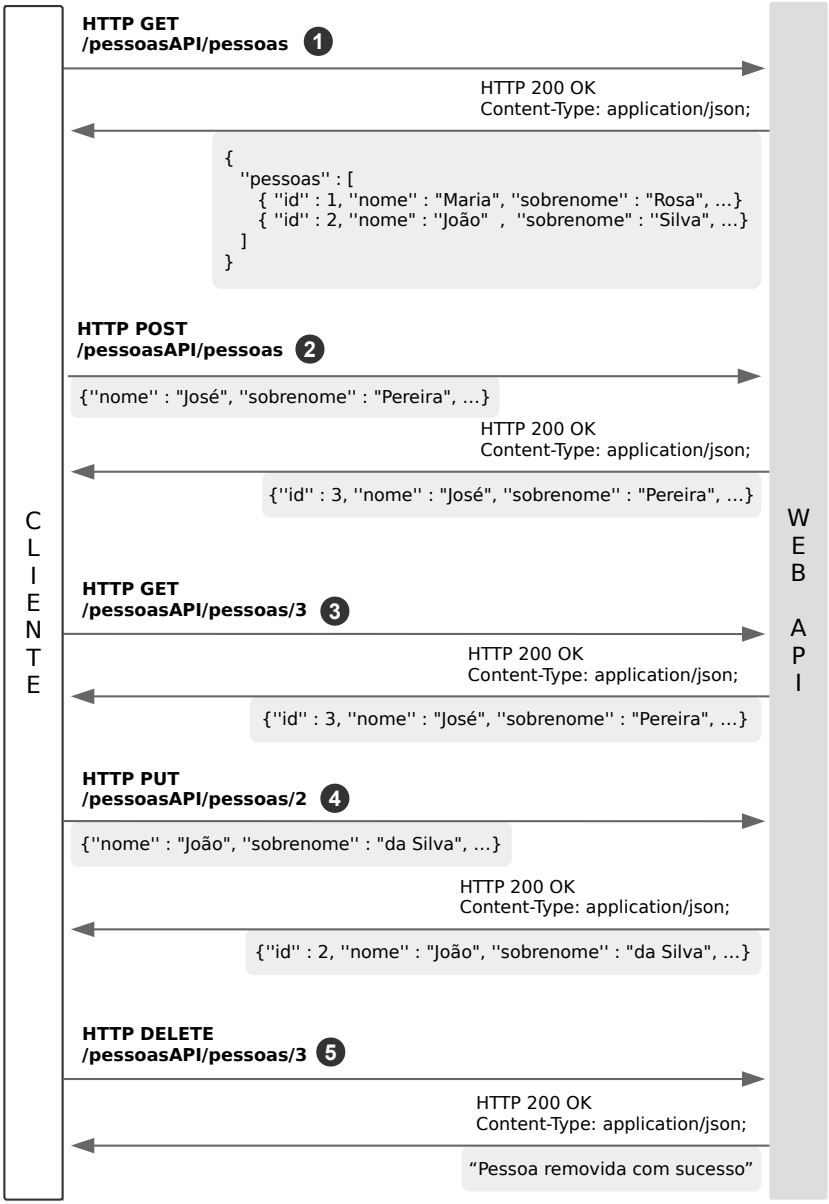


Figura 7 – Exemplos de requisições HTTP para uma Web API

## 2.4 CONSIDERAÇÕES FINAIS

Utilizar representações com suporte a controles hipermídia permite o desenvolvimento de Web API flexíveis, beneficiando as aplicações clientes, diminuindo o grau de dependência com o servidor. Este fato permite que a Web API sofra alterações e evolua sem tornar seus clientes incompatíveis. Embora o uso de hipermídia proporcionar o desenvolvimento de Web API de maior valor, nem todos os problemas são resolvidos, pois esta não oferece, por si só, apoio a clientes autônomos ou à composição automática de serviços e recursos. Para resolver estes problemas é necessário apoio semântico, tanto em nível de recursos quanto de operações. O suporte a controles hipermídia é somente o primeiro passo em direção ao desenvolvimento de Web APIs RESTful semânticas.

# 3 WEB SEMÂNTICA

A maior parte das informações publicadas na Web é destinada aos seres humanos. Entretanto, agentes de software são incapazes de processar adequadamente esse tipo de informação. Documentos produzidos e publicados exclusivamente para a interpretação humana possuem alcance, reuso e integração limitados. Consultas realizadas sobre essas informações resultam em ambiguidades e baixa relevância (FERREIRA FILHO, 2009).

Uma alternativa é descrever o significado das informações publicadas na Web. O conteúdo é apenas uma parte da informação, e o significado deve ser explicitamente descrito para completar a informação. A descrição do significado do conteúdo, denominada descrição semântica, é fundamental para aumentar a expressividade da informação. Documentos descritos semanticamente permitem que softwares interpretem/processem as informações de forma mais adequada. Este capítulo apresenta conceitos e técnicas para descrever semanticamente informações (documentos, dados, serviços, etc) publicados na Web.

## 3.1 PUBLICAÇÃO NA WEB

A Web pode ser entendida como uma plataforma para publicações de documentos. Grandes organizações, agências de pesquisa, industrias e empresas de comércio eletrônico começaram a produzir grandes quantidades de informações. Muitas dessas informações são oriundas de bancos de dados, onde a consulta é realizada através de formulários contidos em páginas Web, também conhecidos como *webforms*. Dados estáticos contidos em páginas Web fazem parte da *surface Web*, enquanto a *deep Web* é formada por páginas Web de conteúdo dinâmico, resultante da pesquisa em *webforms* (CERI et al., 2013). A *deep Web* disponibiliza informações armazenadas em banco de dados. Em outras palavras, torna uma base de dados relacional consultável na Web (BERGMAN, 2001).

Publicar dados através de *webforms* é uma maneira de baixo custo para compartilhar informações de base de dados. Entretanto, realizar a integração de dados através deste modelo de publicação é muito difícil. A primeira proposta de solução para integração de informações foi utilizar *Web Services* com XML. Posteriormente surgiu o paradigma REST juntamente com a representação de dados em JSON. As infor-

mações disponíveis através de Web APIs formam a *deep Web* moderna. De modo geral, os dados expostos por Web APIs não são indexados por mecanismos de buscas. Este obstáculo, na verdade, converge com os propósitos de integração através de Web APIs. É desejável controle na integração de dados, sendo permitido apenas para uso de aplicações Web. Outra possibilidade permitida de consumo de dados de Web APIs é através da técnica de *mashup*, que consiste em combinar dados de diferentes Web APIs para formar um novo serviço (CERI et al., 2013).

O uso de Web APIs facilita o processo de integração de dados entre aplicações, entretanto, as informações publicadas são destinadas para serem consumidas por seres humanos. Agentes autônomos não possuem capacidade de interpretação dos dados publicados, característica que limita o reuso da informação. A grande quantidade de informações distribuídas pela Internet torna difícil encontrar e reutilizar dados relevantes. Entretanto, empresas, governos e centros de pesquisa necessitam de meios de integração de dados eficientes e simplificados. Surge então a Web Semântica, para transformar o enorme repositório de documentos que é a Web em uma rede de informações muito mais compreensível para os computadores (BERNERS-LEE; HENDLER; LASSILA, 2001).

## 3.2 FUNDAMENTOS DA WEB SEMÂNTICA

A Web Semântica é uma extensão da Web atual onde os dados possuem um significado associado, criando a visão de uma rede de informações interligadas. Associar um significado às informações possibilita que pessoas e computadores trabalhem de forma cooperativa (BERNERS-LEE; HENDLER; LASSILA, 2001), proporciona integração e reusabilidade de dados entre aplicações através do compartilhamento global de informações comerciais, científicas e culturais (KASHYAP; BUSSLER; MORAN, 2008). Em outras palavras, a Web Semântica retira dos seres humanos a exclusividade de interpretação das informações.

Tradicionalmente, os documentos publicados são constituídos por um título, alguns parágrafos de texto, frequentemente com imagens ou vídeos e alguns *links* para direcionar a outros documentos (W3C, 2013). Apenas humanos possuem a capacidade para compreender o significado das informações contidas nos documentos. Por outro lado, os computadores estão limitados a interpretar as informações apenas como um emaranhado de textos, *links* e imagens sem sentido nenhum, exemplificado pela Figura 8.

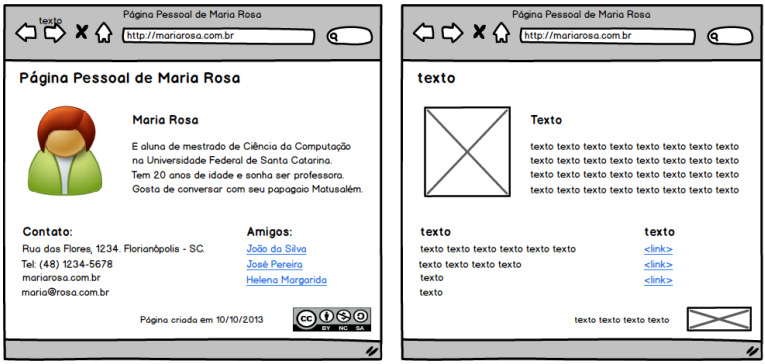


Figura 8 – Interpretações de uma página Web (humano x máquina). Adaptado de (W3C, 2013)

### 3.2.1 Linked Data

A Web original é denominada "Web de Hipertextos" (BIZER; HEATH; Berners-Lee, 2009). Hipertextos são informações em formato digital, que podem conter textos, imagens, vídeos e sons, e permitem ligação (hiperlinks) a outros hipertextos. A evolução da "Web de Hipertextos" é a "Web de Dados", isso implica que as informações publicadas na Web são estruturadas de modo a permitir que agentes de software possam interpretá-las. Estruturar os dados envolve identificar conceitos e elencar suas propriedades e criar relações entre conceitos. É fundamental que os conceitos e propriedades sejam descritos através de tecnologias padronizadas. A padronização proporciona maior integração e consequentemente maior reuso de dados, fato que vai de encontro com a principal motivação de *Linked Data*: A integração racional dos dados publicados na Web.

Com o objetivo de aumentar as oportunidades de integração, Berners-Lee (2006) definiu quatro regras que viabilizam o reuso da informação. A primeira regra é identificar recursos através de URIs, um identificador único de recurso. A segunda regra é associar um endereço HTTP aos recursos, permitindo que sejam localizados na Web. A terceira regra é estruturar os dados com tecnologias padronizadas e recomendadas para Web Semântica, além de estabelecer contextos sobre o relacionamento entre os recursos. A quarta regra é interligar diferentes recursos.

Berners-Lee (2006) criou uma classificação para avaliar a qualidade dos dados publicados na Web. A classificação utiliza ranqueamento, que parte de uma estrela para dados com baixa qualidade, até cinco estrelas para o mais alto nível de qualidade (Figura 9). Para receber uma estrela, basta a informação estar disponível na Web sob licença pública, a informação pode estar representada em qualquer formato, mesmo a digitalização (imagem) de um documento textual. Para receber duas estrelas a informação deve estar em formato legível para agentes de software, podem ser disponibilizados em tabelas do excel (XLS), word (DOC), dentre outros. A terceira estrela se aplica a documentos em formatos abertos como: XLSX, DOCX, arquivos separados por vírgula(CSV), dentre outros. Para receber a quarta estrela, a informação deve ser estruturada nos moldes da Web Semântica, RDF é uma possibilidade (ver em 3.3.1). A quinta estrela é atribuída a informações semanticamente relacionadas. Os dados devem estar vinculados a contextos para enriquecer sua expressividade.

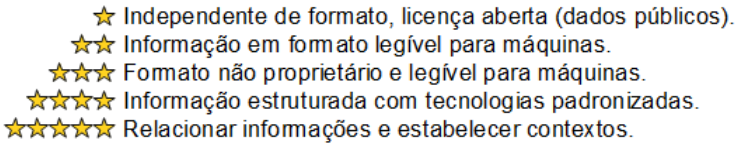


Figura 9 – Classificação Linked Data. Adaptado de (BERNERS-LEE, 2006)

## 3.3 TECNOLOGIAS PARA WEB SEMÂNTICA

Para o sucesso da Web Semântica é fundamental que os dados sejam estruturados. Para poder interpretar e reutilizar esses dados, é preciso seguir padrões. A W3C (*World Wide Web Consortium*) recomenda um conjunto de tecnologias a serem utilizadas. A Figura 10 (A) apresenta a arquitetura da Web Semântica com os padrões estabelecidos. A arquitetura está disposta em camadas, fato que facilita a aplicação parcial, sem a necessidade da implementação total da visão da Web Semântica (FERREIRA FILHO, 2009). A Figura 10 (B) adiciona as tecnologias JSON e JSON-LD (ver em 3.3.4) em suas camadas, além de incluir vocabulários para a descrição de recursos.

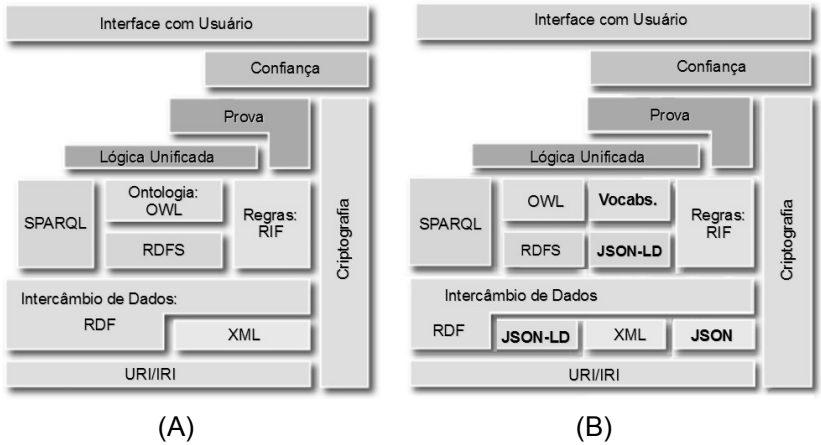


Figura 10 – Arquitetura da Web Semântica. Adaptado de (W3C, 2007a)

### 3.3.1 RDF

RDF (*Resource Description Framework*) é uma linguagem para representar informações na Web. Permite estruturar dados de forma que a informação possar ser processada por computadores e não somente exibida para pessoas. Neste contexto, os dados são denominados *Web resources* ou simplesmente recursos. São metadados que podem ser individualmente identificáveis na Web, por exemplo, nome do autor, titulo e data de modificação de uma determinada página Web. Recursos podem ser também informações especificas sobre o conteúdo da informação, por exemplo, calendário de eventos, informações sobre produtos (preços, especificações, disponibilidade de entrega)(W3C, 2004).

RDF descreve recursos no formato de triplas (sujeito, predicado, objeto). O sujeito é o recurso a ser descrito, o predicado é uma propriedade do sujeito e o objeto é o valor do predicado. A descrição de um recurso através de RDF pode ser vista com um grafo orientado, onde o sujeito e objeto são nodos e o predicado expressa seu relacionamento. A Figura 11 demonstra o grafo dos dados pessoais presentes na página Web da Figura 8. Recursos são descritos através de propriedades, valores são atribuídos para cada propriedade, por exemplo, o recurso *Maria Rosa* tem a propriedade *idade* com valor *20*.

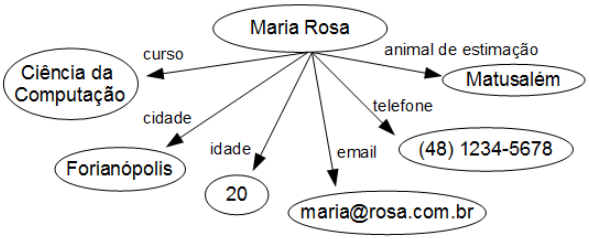


Figura 11 – Exemplo de descrição RDF

A descrição RDF é construída no formato XML, pode estar presente juntamente com o recurso ou em arquivo separado (Listagem 7). O Recurso deve ser identificado por uma URI, as propriedades (predicados) dos recursos também são identificados por URIs, valores podem ser URIs ou literais.

Listagem 7 – Descrição RDF de Maria Rosa

```
1  <?xml version="1.0"?>
2  <rdf:RDF
3    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
4    xmlns:exterms="http://www.example.org/terms/">
5    <rdf:Description rdf:about="http://mariarosa.com.br/#
         mariarosa">
6      <exterms:curso>Ciência da Computação</exterms:curso>
7      <exterms:cidade>Florianópolis</exterms:cidade>
8      <exterms:idade>20</exterms:idade>
9      <exterms:email>maria@rosa.com.br</exterms:email>
10     <exterms:telefone>(48)1234-5678</exterms:telefone>
11     <exterms:pet>Matusalém</exterms:pet>
12   </rdf:Description>
13 </rdf:RDF>
```

A descrição através de RDF limita-se a descrição sintática e individual de recursos. Para um nível de interpretação adequado, é necessário especificar um contexto de aplicação. Sem um contexto definido, a informação não pode ser compreendida sem que haja ambiguidades. RDFS (*Resource Description Framework Schema*) é uma extensão do RDF que permite adicionar relações semânticas entre recursos.

Com RDFS é possível utilizar vocabulários específicos que dão contexto ao recurso, além de organizar os recursos em hierarquias. Definir um vocabulário possibilita que usuários tenham uma pesquisa mais rica e relevante de informações publicadas na Web (SCHEMA.ORG, 2011). Vocabulários controlados organizam e agrupam termos que po-

dem ser associados a um domínio específico (SVENONIUS, 1986). Vocabulários formam uma coleção de termos organizados que facilitam o acesso à informação, podendo ser utilizados para descrever informações a fim de facilitar sua recuperação e interpretação. Um exemplo de vocabulário é FOAF (*Friend Of A Friend*), que expressa relacionamento entre pessoas. Outros vocabulários estão disponíveis como: DBpedia e DublinCore. É possível utilizar um vocabulário para definir um contexto de utilização do recurso, sendo assim, expressar sem ambiguidades o significado de propriedades e relacionamentos de recursos (Figura 12).

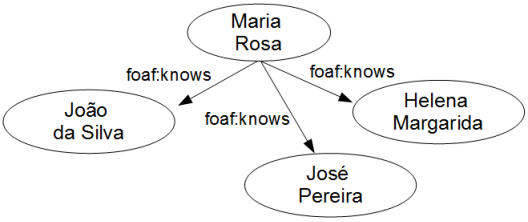


Figura 12 – Exemplo de descrição RDFS FOAF

A Listagem 8 apresenta um exemplo documento XML que descreve a relação de amizade entre pessoas. Os recurso *Maria Rosa* e seus respectivos amigos são descritos e identificados como instâncias de *Person* do vocabulário FOAF. O relacionamento entre eles é descrito como *foaf:knows*. O uso do vocabulário FOAF esclarece o tipo do relacionamento entre os recursos. Associar vocabulários com recursos permite adicionar a informação necessária para expressar significado em um determinado contexto, além de resolver as possíveis ambiguidades.

Listagem 8 – Exemplo de usuo de vocabulario FOAF com RDFS

```
1  <?xml version="1.0"?>
2  <rdf:RDF
3     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
4     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
5     xmlns:foaf="http://xmlns.com/foaf/0.1/">
6     <foaf:Person>
7        <foaf:firstName>Maria</foaf:firstName>
8        <foaf:surname>Rosa</foaf:surname>
9        <foaf:knows>
10          <foaf:Person>
11             <foaf:name>João da Silva</foaf:name>
12          </foaf:Person> ...
13       </foaf:knows>
14    </foaf:Person>
15 </rdf:RDF>
```

### 3.3.2 RDFa

De forma geral, as páginas Web são constituídas por um título, alguns parágrafos de texto, frequentemente com imagens, vídeos e *links* para direcionar para outras páginas (W3C, 2013). O conteúdo das páginas Web é destinado aos seres humanos, entretanto, é possível adicionar marcações em algumas partes da página Web e torná-las interpretáveis para computadores. RDFa (*Resource Description Framework in Attributes*) (W3C, 2013) permite descrever recursos e suas propriedades, reaproveitando a própria estrutura HTML. Com RDFa não há necessidade de separar as descrições RDF, destinadas ao consumo de agentes de software, das informações textuais destinadas ao consumo humano (Figura 13).

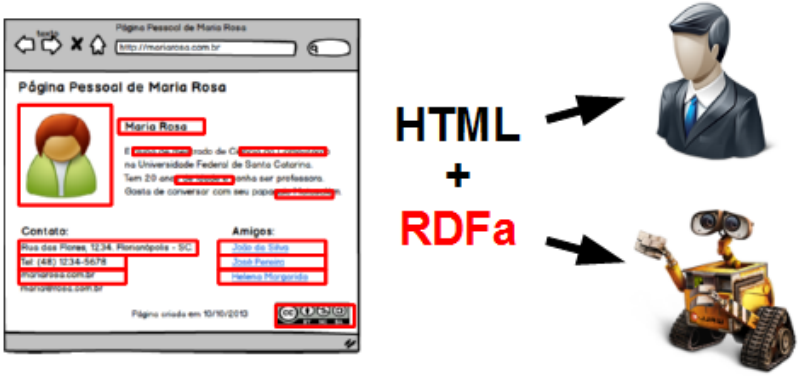


Figura 13 – Diferentes interpretações de uma página Web. Adaptado de (SPORNY, 2012)

A Figura 14 mostra o código fonte de uma página Web com marcações RDFa. O código da descrição RDFa de recursos e suas propriedades estão em destaque. No exemplo, são combinados diferentes vocabulários, que adicionam contextos às informações. O resultado das marcações RDFa pode ser visualizado graficamente[1] pela Figura 15. Diversos recursos foram descritos com suas respectivas propriedades, além de relacionamentos entre recursos.

[1] Gráfico gerado pela ferramenta RDFa play (http://rdfa.info/play/)

```
<html>
  ...
  <div prefix="foaf: http://xmlns.com/foaf/0.1/ chema: http://schema.org/
               ov: http://open.vocab.org/terms/"
       resource="http://mariarosa.com.br#mariarosa" typeof="foaf:Person">
    <img property="schema:image" src="maria.png">
    <h2 property="foaf:name">Maria Rosa</h2>
    <span property="schema:jobTitle">Estudante</span>
    de mestrado de Ciência da Computação na
    <span property="schema:alumniof" resource="#ufsc"> </span>
    <span typeof="schema:EducationalOrganization" resource="#ufsc">
      <span property="schema:name">Universidade Federal de Santa Catarina</span>.
    </span> Tem <span property="foaf:age">20</span>
    anos de idade e sonha ser professora. Gosta de conversar com seu papagaio
    <span property="ov:preferredAnimal">Matusalém</span>.
    <h2>Contato</h2>
    <span property="schema:address" resource="#endereco" typeof="PostalAddress">
       <span property="schema:streetAddress">Rua das Flores, 1234.</span>
       <span property="schema:addressLocality">Florianópolis</span> -
       <span property="schema:addressLocality">SC</span>.
    </span>
    <span property="foaf:phone">Tel: (48)1234-5678</span>
    <span property="schema:url">mariarosa.com.br</span>
    <span property="foaf:mbox">maria@rosa.com.br</span>
    <h2>Amigos</h2>
    <a property="foaf:knows" resource="#joao" typeof="foaf:Person"
       href="http://joaodasilva.com.br">
         <span property="foaf:name">João da Silva</span></a><br>
    <a property="foaf:knows" resource="#jose" typeof="foaf:Person"
       href="http://josepereira.com.br">
         <span property="foaf:name">José Pereira</span></a><br>
    <a property="foaf:knows" resource="#helena" typeof="foaf:Person"
       href="http://helenamargarida.com.br">
         <span property="foaf:name">Helena Margarida</span></a>
  </div>
  ...
</html>
```

Figura 14 – Exemplo de página Web com marcações RDFa

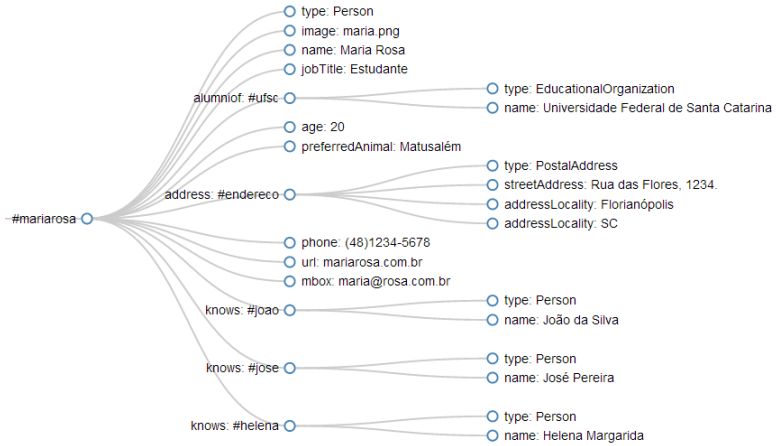


Figura 15 – Representação gráfica de marcações RDFa em HTML

### 3.3.3 Microformats

Microformats são adaptações na semântica HTML para facilitar a publicação, indexação e extração de informações semiestruturadas (KHARE; ÇELIK, 2006). Microformats utilizam a infraestrutura existente do HTML para descrever pessoas, lugares, eventos e outras informações semiestruturadas (KHARE, 2006). Permite adicionar marcações em informações disponibilizadas exclusivamente para a interpretação humana, de modo a tornar a informação legível aos agentes de software.

Microformats limita a descrição semântica a um conjunto predefinido de termos, diferente de RDFa que permite associar informações a termos de vocabulários. Dentre os conjuntos predefinidos de termos estão: *hCard* para a publicação de dados pessoais e de organizações; *hCalendar* para publicação de eventos; *hMedia* para publicação de imagens, vídeos e áudio, dentre outros. A descrição é realizada através do atributo *class*, que é associado a um termo específico. A Listagem 9 apresenta um exemplo de descrição semântica de dados pessoais com Microformats.

Listagem 9 – Exemplo de descrição com Microformats

```
<p class="h-card">
  <span class=p-given-name"Maria</span>
  <span class="p-family-name">Rosa</span>
  <span class="p-country-name">Brasil</span>
  <span class="u-email">maria@rosa.com.br</span>
  <p class="p-adr h-adr">
    <span class="p-street-address">Rua das Flores</span>
    <span class=p-locality">Florianópolis</span>
    <span class=p-region">Santa Catarina</span>
  </p>
  <span class=p-tel">(48) 1234-5678</span>
  <span class=p-organization-name">UFSC</span>
  <span class=p-job-title">estudante</span>
</p>
```

### 3.3.4 JSON-LD

JSON-LD é uma maneira leve para descrever *Linked Data* no formato JSON. Adiciona informações semânticas em documentos JSON existentes, sem necessidade de grandes esforços. Foi projetado para proporcionar simplicidade, compatibilidade e expressividade. JSON-LD é compatível com JSON, ou seja, todo documento JSON-LD é um docu-

mento JSON válido. Essa compatibilidade permite que as bibliotecas e analisadores atuais sejam reutilizados. Seus principais objetivos são possibilitar a construção de Web APIs com recursos semânticos, além de armazenamento de *Linked Data* em *engines* baseadas em JSON (LANTHALER; GüTL, 2012).

JSON-LD permite criar contextos compostos por classes e termos de vocabulários controlados. Os vocabulários são utilizados como fonte de metadados que descrevem o conteúdo dos recursos. JSON-LD permite associar propriedades com termos de diferentes fontes semânticas, além da possibilidade de associar o documento a um conjunto de classes que vincula um significado para o documento como um todo. Para associar vocabulários ao documento, JSON-LD disponibiliza a notação *@context*. JSON-LD permite representar valores de propriedades como *links*, diferenciando URLs de simples valores textuais. Para definir que uma determinada propriedade corresponde a um *link* usa-se a notação*@type: @id*. A informação descrita semanticamente em JSON-LD possui URLs para que o consumidor possa obter mais informações sobre determinado significado. Relacionamentos também podem possuir semântica associada, de modo a estabelecer uma relação semântica entre recursos. Enriquecer semanticamente informações com vocabulários compartilhados e conhecidos resulta em integrações mais ricas, flexíveis e menos vulneráveis a modificações.

A Figura 16 ilustra um exemplo de documento representado com JSON-LD. Entre as linhas 1 a 13 é definido o contexto do documento. O documento é associado ao conceito de *Person* presente nos vocabulários *schema* e *foaf*, através da anotação *@type* (linha 3). Os vocabulários utilizados são declarados nas linhas 4 e 5, e possuem um prefixo e o endereço do vocabulário. As propriedades são associadas aos termos dos vocabulários declarados (linhas 6 a 12). Essa associação permite obter o significado da propriedade através da consulta ao vocabulário corresponde. Para obter mais informações sobre a propriedade *schema:name*, por exemplo, basta visitar a URL do vocabulário juntamente com o termo desejado (*http://schema.org/name*). A anotação *@type* com valor *@id* (linha 10) identifica que a propriedade é representada por um *link*, e não deve ser interpretada como valor textual. A propriedade amigos (linha 12) possui o significado *foaf:knows* que descreve um determinado relacionamento entre recursos. A propriedade *@id* (linha 14) apresenta o identificador único do documento, geralmente representado pela URL do próprio recurso. Com o contexto definido, cada propriedade do documento está vinculada a termos de vocabulários controlados.

A combinação entre o estilo arquitetural REST e os princípios de *Linked Data* proporciona grande avanço para a troca de informações entre Web APIs e agentes autônomos (clientes). Essa combinação baseia-se no enriquecimento semântico de dados, de forma a tornar a informação compreensível aos agentes. Nesse cenário surge o Hydra (LANTHALER, 2013), que provê um vocabulário capaz de representar o significado de transações de estados dos recursos. Isso significa que clientes de Web APIs passam a ter informações suficientes para realizar requisições que modificam o estado dos recursos, através da extensão do formato JSON-LD.

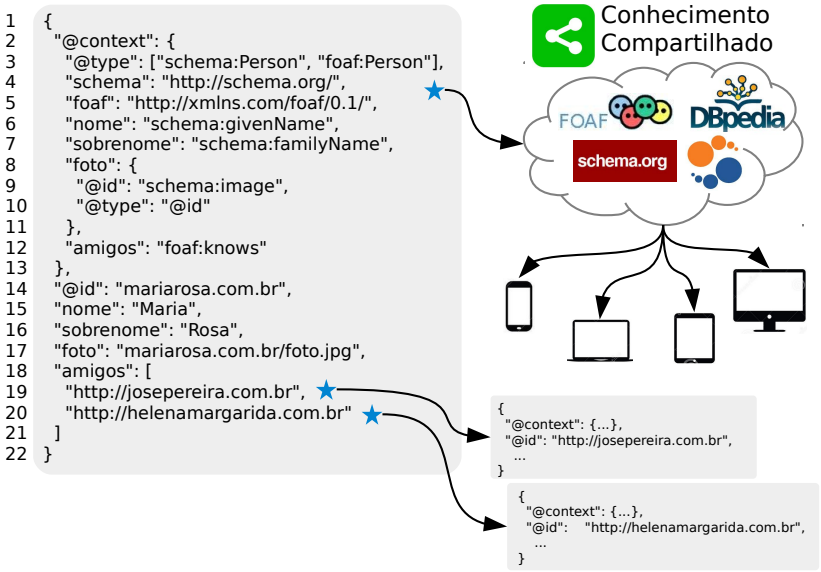


Figura 16 – Documento JSON-LD - diversos vocabulários e *links*

## 3.4 HYDRA

Embora JSON-LD represente um avanço para a representação de controles hipermídia, apenas *links* podem ser vinculados à representações. As demais operações que modificam o estado do recurso não podem ser representadas (RICHARDSON; AMUNDSEN; RUBY, 2013). Hydra propõe uma alternativa que possibilita o uso mais amplo de con-

troles hipermídia, através da ampliação dos conceitos do JSON-LD e de um vocabulário específico para tratar requisições. Hydra adota o conceito de classes para descrever recursos, que contém as propriedades e as operações suportadas. O suporte oferecido pelo Hydra permite a troca de informações no formato JSON, utilizando o contexto semântico do JSON-LD com a adição das instruções necessárias para execução de operações sobre o recurso Web. Além de enriquecer as informações disponibilizadas pela Web API, Hydra oferece uma maneira de descrever todos os recursos gerenciados pela Web API, através da documentação da Web API.

A Listagem 10 mostra um exemplo de documento representado no formato Hydra. O contexto do documento é definido de forma idêntica ao JSON-LD, porém novas notações estendem o poder de representação de controles hipermídia. Entre as linhas 2 e 6 é definido o contexto semântico, que utiliza três vocabulários distintos. Hydra é o primeiro vocabulário utilizado e está associado ao prefixo *hydra* (linha 3). Os outros dois vocabulários são FOAF e Schema.org (linhas 4 e 5), associados aos prefixos *foaf* e *schema*, respectivamente. O elemento *@id* (linha 7) define a semântica associada ao recurso descrito. É possível relacionar um recurso à diversas classes semânticas. No exemplo, o recurso descrito está associado à *foaf:Person* e *schema:Person*. O elemento *@type* define que o recurso representa uma classe Hydra (linha 8). Recursos são descritos no formato de classes, entretanto, Hydra disponibiliza a semântica para descrever *links*, documentação, operações, dentre outros. As propriedades do recurso são agrupadas no elemento *supportedProperty* (linhas 9 a 38). O elemento *@type* (linhas 11, 18, 25 e 32) associa a propriedade do recurso a um termo do vocabulário. O elemento *property* (linhas 12, 19, 26 e 33) apresenta o nome da variável corresponde à propriedade. Hydra permite descrever se a propriedade é obrigatória, apenas leitura ou apenas escrita. Quando uma propriedade é apenas leitura, seu valor não é considerado nas operações que modificam o estado do recurso, enquanto apenas escrita, o valor não é apresentado em operações de leitura do recurso.

Hydra permite descrever operações HTTP que podem ser realizadas sobre os recursos, além de diferenciar a natureza das operações através de palavras reservadas. As operações de criação, alteração e remoção são denominadas *CreateResourceOperation*, *ReplaceResourceOperation* e *DeleteResourceOperation* respectivamente. Hydra permite descrever todos os requisitos necessários para a execução de uma operação. Cada operação informa também o método HTTP que deve ser utilizado, além da URL base para execução. Quando a URL da ope-

ração é omitida, a operação deve ser aplicada sobre a URL do próprio recurso. Através do elemento *expects*, é possível descrever quais são os dados que devem ser enviados para a execução de um determinada operação (linha 43). O recurso resultante de uma operação pode ser descrito através do elemento *returns* (linha 44).

Listagem 10 – Exemplo de documentação Hydra

```
1  {
2    "@context": {
3      "hydra": "http://www.w3.org/ns/hydra/context.jsonld",
4      "foaf": "http://xmlns.com/foaf/0.1/",
5      "schema": "http://schema.org/"
6    },
7    "@id": ["foaf:Person", "schema:Person"],
8    "@type": "hydra:Class",
9    "hydra:supportedProperty": [
10     {
11       "@type": "schema:givenName",
12       "property": "nome",
13       "required": true,
14       "readonly": false,
15       "writeonly": false
16     },
17     {
18       "@type": "schema:familyName",
19       "property": "sobrenome",
20       "required": true,
21       "readonly": false,
22       "writeonly": false
23     },
24     {
25       "@type": "foaf:age",
26       "property": "idade",
27       "required": false,
28       "readonly": false,
29       "writeonly": false
30     },
31     {
32       "@type": "schema:image",
33       "property": "foto",
34       "required": false,
35       "readonly": false,
36       "writeonly": false
37     }
38   ],
39   "hydra:supportedOperation": [
40     {
41       "@type": "ReplaceResourceOperation",
42       "method": "PUT",
43       "expects": ["foaf:Person", "schema:Person"],
44       "returns": ["foaf:Person", "schema:Person"]
45     },
46     {
47       "@type": "DeleteResourceOperation",
48       "method": "DELETE"
49     }
50   ]
51 }
```

## 3.5 CONSIDERAÇÕES FINAIS

Identificar conceitos e propriedades e descrevê-los como recursos é o primeiro passo para construção da Web Semântica. Relacionar recursos com vocabulários conhecidos estabelece um contexto para as informações e aumenta o poder de compreensão de aplicações clientes. Seguir padrões de descrição semântica é fundamental para concretizar a visão da Web Semântica, embora a ausência de padrões bem estabelecidos para todas as camadas da arquitetura seja uma barreira.

A Web Semântica estrutura e enriquece a informação disponibilizada na Web, possibilita que agentes de software sejam capazes de interpretar os dados enriquecidos semanticamente. Além do poder de interpretação, os agentes de software devem ser capazes de consultar e executar serviços sem a intervenção humana. Uma maneira de disponibilizar serviços na Web é através de Web APIs. Implementar mecanismos de forma a tornar autônomo o processo de descoberta e execução de serviços Web é um problema em aberto. Uma alternativa são as Web APIs RESTful baseadas em controles hipermídia, que permitem integração com agentes autônomos e diminuem o acoplamento entre cliente e servidor. Hydra suporta operações que alteram o estado dos recursos Web, dessa forma, amplia a expressividade do JSON-LD e torna-se uma alternativa mais completa para o desenvolvimento de Web APIs Semânticas.

# 4 ESTADO DA ARTE

Este capítulo apresenta o estado da arte sobre o desenvolvimento de Web APIs RESTful Semânticas, que está dividida em: formato de dados, abordagens de desenvolvimento, suporte ferramental e modelos de maturidade.

## 4.1 FORMATO DE DADOS

Existem diferentes maneiras de organizar e disponibilizar as informações trocadas entre clientes e Web APIs. Entretanto, este trabalho aborda apenas os formatos baseados em JSON. Os formatos discutidos são: JSON Schema, JSON Hyper-Schema, JSON HAL, Collection JSON e RESTful Objects.

### 4.1.1 JSON Schema

JSON Schema permite descrever a estrutura de dados de documentos JSON. Semelhante a um contrato, é possível validar, documentar e controlar a interação de dados (IETF, 2013b). É possível representar os dados como *array*, *boolean*, *integer*, *number*, *null*, *object* e *string*. A Listagem 11 ilustra um exemplo de JSON Schema que contém as declarações dos tipos das propriedades, além de dados para validação.

Listagem 11 – Exemplo JSON Schema. Adaptado de (IETF, 2013b)

```
1  {
2      "title": "Uma Pessoa",
3      "type": "object",
4      "properties": {
5          "nome": {
6              "type": "string"
7           },
8          "sobrenome": {
9              "type": "string"
10         },
11         "idade": {
12             "descricao": "Idade em anos",
13             "type": "integer",
14             "minimum": 0
15         }
16     },
17     "required": ["nome", "sobrenome"]
18 }
```

A Figura 17 ilustra as requisições necessárias para interpretar um documento que possui um *schema* associado. A requisição (1) mostra uma operação GET para um determinado recurso, a Web API retorna uma representação no formato JSON com o cabeçalho HTTP contendo um *link*, que implica na existência de informação extra na URI associada, como ilustrado em (2). Para obter a informação extra, o cliente deve realizar outra requisição (3), obtendo como resposta o JSON Schema (4). Em posse do JSON Schema, o cliente tem acesso à estrutura dos dados do documento solicitado, fato que possibilita o primeiro nível de tratamento genérico de dados e desacoplamento entre cliente e Web API.

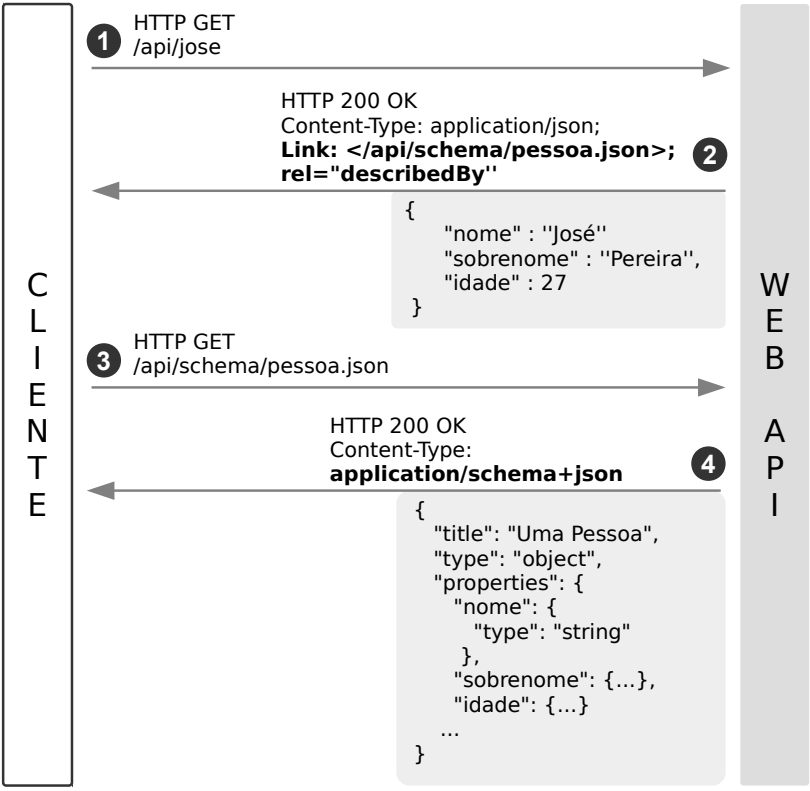


Figura 17 – Requisições para obter JSON Schema

### 4.1.2 JSON Hyper-Schema

JSON Hyper-Schema enriquece a descrição de documentos que utilizam JSON Schema através do suporte a controles hipermídia (IETF, 2013a). É possível vincular *links* e hipertextos e estabelecer suas relações. Entretanto, a existência de tais controles só é percebida por clientes através da consulta e interpretação do *schema*. Não existe a possibilidade de representar propriedades juntamente com *links* no documento JSON (Figura 18).

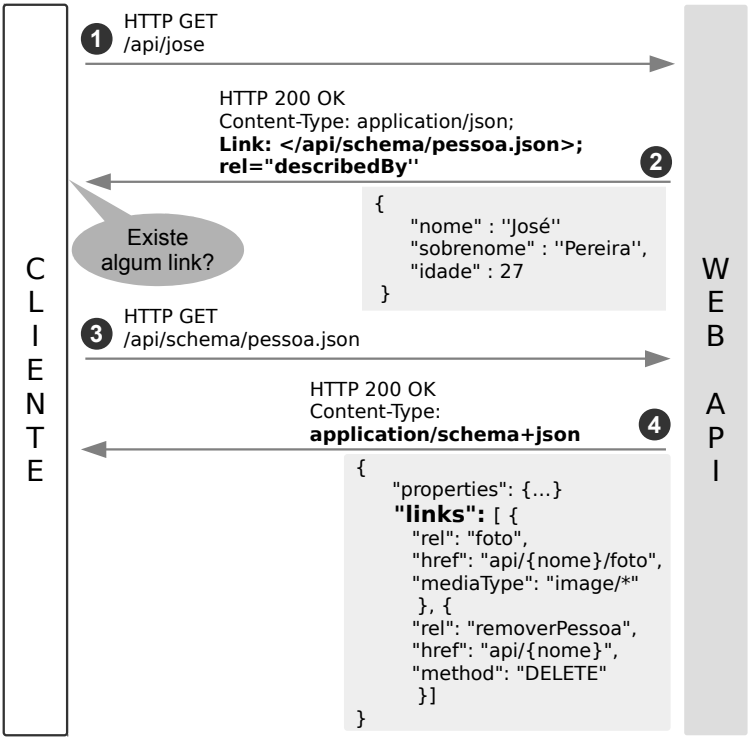


Figura 18 – Obtenção dos links vinculados ao documento JSON

### 4.1.3 JSON HAL

JSON HAL (*Hypermedia Application Language*) consiste em um formato para expressar controles hipermídia em documentos JSON e XML (KELLY, 2013). Documentos podem incorporar *links* diretamente em sua estrutura, uma vez que o formato possui mecanismos para diferenciá-los das demais propriedades. Para que o documento JSON seja interpretado de forma adequada, o cliente deve ser capaz de compreender o formato HAL, estabelecido pelo *HTTP Header Content-Type: application/hal+json;* (Figura 19).

Embora o formato JSON HAL permita representar controles hipermídia diretamente no documento JSON, não é possível realizar a requisição imediata para os *links*, pois as informações necessárias para o acesso são incompletas. Não é possível saber qual método HTTP executar para um determinado *link*, fato que força a criação de documentos responsáveis pela explicação de cada *link* vinculado. De forma geral, a documentação é destinada a seres humanos, que devem interpretar o seu conteúdo para poder interagir com a Web API.

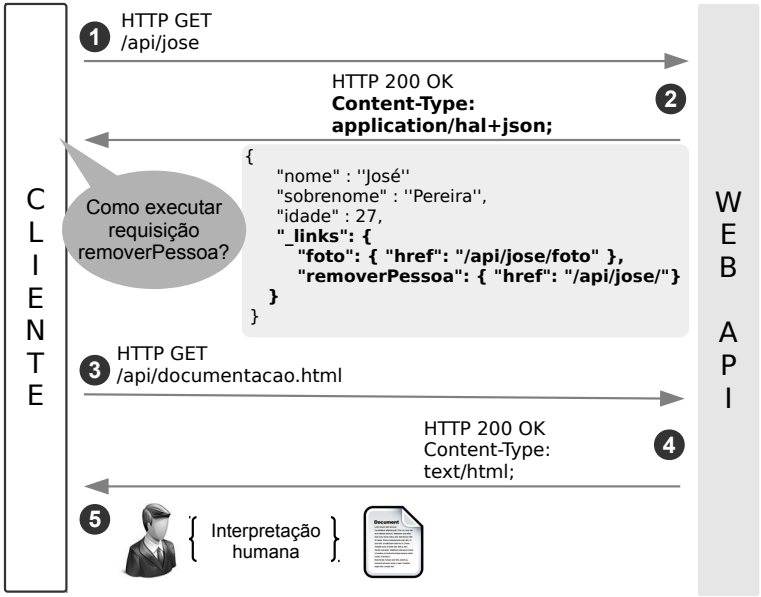


Figura 19 – Exemplo de interação com JSON HAL

### 4.1.4 Collection JSON

Collection JSON é uma formato de dados baseado em JSON capaz de representar listas de recursos (AMUNDSEN, 2013). É constituído de uma coleção de recursos, denominados *items*, onde cada item contém seu *link* de acesso, além dos próprios dados do recurso (Listagem 12). Cada recurso presente na coleção pode conter diversos *links*, ampliando o poder de representação de controles hipermídia.

Listagem 12 – Exemplo de Collection JSON

```
{"collection":{
  "items":[
  {"href":"/api/jose",
   "data":[
    {"name":"nome",
     "value":"José",
     "prompt":"Nome"
    },
    {"name":"sobrenome",
     "value":"Pereira",
     "prompt":"Sobrenome"
    },
    {"name":"idade",
     "value":27,
     "prompt":"Idade"}],
   "links":[
    {"rel":"foto","href":"/api/jose/foto","prompt":"Foto"}]
  },
  {"href":"/api/joao",
    "data":[
    {"name":"nome",
     "value":"João",
     "prompt":"Nome"
    },
    {"name":"sobrenome",
     "value":"Silva",
     "prompt":"Sobrenome"
    },
    {"name":"idade",
     "value":76,
     "prompt":"Idade"
    }],
   "links":[
    {"rel":"foto","href":"/api/joao/foto","prompt":"Foto"}]
   }, ...
}}
```

Para que um documento Collection JSON seja compreendido corretamente, o cliente da Web API deve ser capaz de interpretar o *Content-Type:application/collection+json*, o que requer o conhecimento dos detalhes estruturais da especificação Collection JSON. Além dos *items*, um documento Collection JSON pode conter os elementos: *links, queries, template e error* (Figura 20). O elemento *links* permite rela-

cionar a coleção com outros recursos, através da vinculação de URIs e seus respectivos significados. O elemento *queries* possibilita vincular consultas pré-definidas sobre os itens da coleção. O elemento *template* oferece um modelo de representação que deve ser utilizado para criar novos itens. Por fim, o elemento *error* apresenta as possíveis mensagens de erros e seus respectivos códigos HTTP.

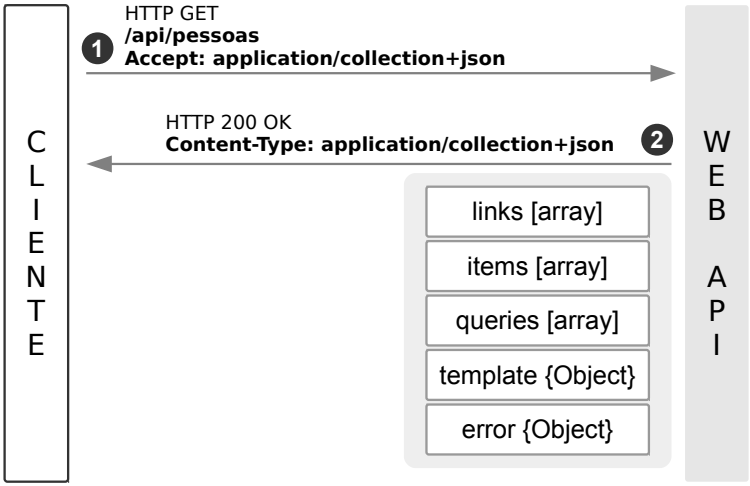


Figura 20 – Estrutura do documento Collection JSON

Recursos representados no formato Collection JSON podem ser manipulados através de operações, que podem ser executadas diretamente sobre a coleção ou sobre os seus itens. A Figura 21 mostra o fluxograma das operações permitidas pelo Collection JSON. Para adicionar um novo item, deve-se executar o método HTTP POST sobre a URI da coleção, enviando uma representação que segue a estrutura definida pelo elemento *template*. É possível realizar consultas pré-definidas sobre os itens da coleção através de diretrizes presentes no elemento *queries*. Cada item da coleção possui uma URI que identifica e endereça o recurso, possibilitando obter mais informações sobre um determinado item da coleção (GET). As operações que modificam (PUT) ou removem (DELETE) um recurso são executadas diretamente na URI do recurso.

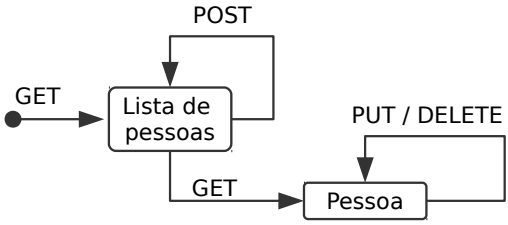


Figura 21 – Fluxograma de operações Collection JSON

### 4.1.5 RESTful Objects

RESTful Objects (HAYWOOD, 2012) é uma especificação para desenvolvimento de aplicações RESTful que permite que clientes tenham acesso aos objetos de domínio através de HTTP, tendo como retorno representações de recursos no formato JSON. As classes de domínio podem expor propriedades, operações e coleções que referenciam outras entidades. Considera que cada classe do domínio da aplicação é um recurso que possui uma representação serializada no formato JSON. Apresenta o conceito de *Object Action*, que representa operações que podem ser executadas através de interfaces RESTful. O documento JSON disponibilizado pela Web API acompanha o cabeçalho HTTP com *Content-type:application/json*, adicionado da informação *profile="urn:org.restfulobjects:repr-types/object*, que adiciona a semântica necessária para manipulação dos controles hipermídia e identifica que o documento utiliza a especificação *RESTful Objetcs* (Figura 22). Outra propriedade presente no cabeçalho HTTP é *x-ro-domain-type*, que permite vincular o objeto de domínio a um conceito ou tipo.

A especificação RESTful Objects permite representar dados e especificar tipos, além de vincular *links* com informações completas de execução. Os tipos de dados suportados são: *string*, *date-time*, *date*, *time*, *utc-millisec*, *big-integer*, *big-decimal*, *blob*, *clob*, *decimal* e *int*. É possível representar controles hipermídia que alteram o estado dos recursos, inclusive a nível individual de propriedades. A Listagem 13 (linhas 20 a 24) exemplifica uma representação que contém *links* que alteram diretamente uma propriedade do recurso. A possibilidade de alteração segmentada de estado descaracteriza o conceito de recurso, pois cria identificadores para propriedades isoladas que deveriam ser manipuladas apenas através do recurso como um todo.

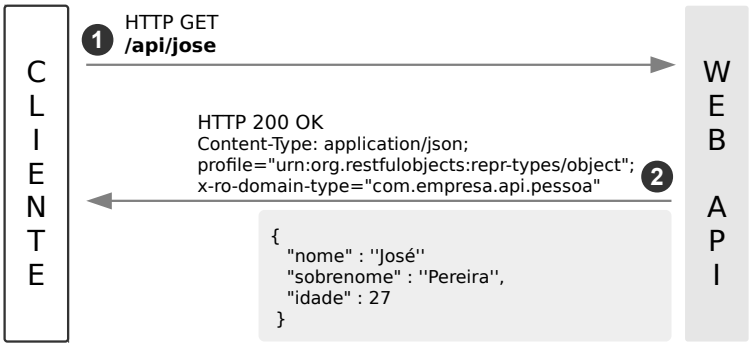


Figura 22 – Exemplo de interação RESTful Objects

Listagem 13 – Exemplo de documento RESTful Objects

```
{
 "nome":{
  "value":"José",
  "format":"string"
 },
 "sobrenome":{
  "value":"Pereira",
  "format":"string"
 },
 "idade":{
  "value":27,
  "format":"int"
 },
 "links": [{
  "rel": ".../attachment;property=\"foto\"",
  "href": "/api/jose/foto",
  "type": "image/jpeg",
  "method": "GET"
 },{
  "rel":".../modify;property=\"nome\"",
  "href":"/api/jose/properties/nome",
  "type":"application/json;profile=\".../object-property\"",
  "method":"PUT",
  "arguments": {"value":"Francisco"}
 },{
  "rel":".../clear;property="\nome\"",
  "href":"/api/jose/properties/nome",
  "type":"application/json;profile=\".../object-property\"",
  "method":"DELETE"
 }]
}
```

## 4.2 ABORDAGENS DE DESENVOLVIMENTO

Esta seção apresenta o estado da arte no domínio de abordagens de desenvolvimento de Web APIs. O primeiro trabalho analisado é *Domain Application Protocol* (ROBINSON, 2011), seguido do Desenvolvimento Baseado em Representações (RICHARDSON; AMUNDSEN; RUBY, 2013).

### 4.2.1 DAP

*Domain Application Protocol* (DAP) é um conjunto de regras e convenções que definem diretrizes para os participantes de sistemas distribuídos alcançarem um objetivo específico de negócio. No contexto de Web APIs, o domínio da aplicação pode ser entendido como uma abstração que contém os formatos de dados, relações entre *links* e tipos de requisições HTTP necessários para a execução de determinada regra de negócio (ROBINSON, 2011). Em aplicações mais simples, por exemplo, baseadas em operações *CRUD*, não é necessário desenvolver um protocolo de domínio da aplicação, embora o protocolo exista de forma implícita.

Robinson (2011) define três passos para desenvolver o protocolo de domínio da aplicação. O primeiro passo implica em modelar o protocolo através máquinas de estados. O segundo passo deve implementar a máquina de estados com a perspectiva de recursos e seus ciclos de vida. O terceiro passo é a documentação da Web API através da utilização de formato de dados, relações entre *links* e tipos de requisições HTTP. A Figura 23 ilustra um exemplo de máquina de estados que representa o protocolo de domínio da aplicação do estudo de caso utilizado como exemplo. A modelagem proporciona o amplo entendimento sobre as transições de estados dos recursos devido às possíveis interações entre clientes e Web API.

A implementação do protocolo de domínio da aplicação depende de um formato de dados com suporte a controles hipermídia. Caso a Web API seja baseada em JSON, é possível utilizar JSON-LD e Hydra como alternativas. O conceito de classes de recursos que caracterizam e estruturam representações e a possibilidade de vincular operações com os detalhes de execução, constituem a essência de um protocolo de domínio da aplicação.

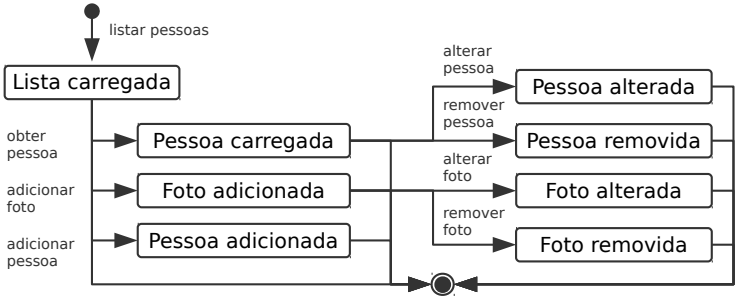


Figura 23 – Máquina de estados - DAP

### 4.2.2 Richardson e Amundsen

Richardson, Amundsen e Ruby (2013) definem uma metodologia composta por 7 passos para guiar o desenvolvimento de Web APIs. O foco da metodologia deixa de ser em recursos e dá ênfase nas representações. O primeiro passo busca identificar e agrupar as informações manipuladas. No exemplo de cadastro de pessoas (Figura 24), pode-se dividir as informações em três representações distintas. A primeira representação é a lista de todas as pessoas cadastradas, a segunda representação é formada individualmente por uma pessoa, e a foto vinculada à pessoa forma a terceira representação.

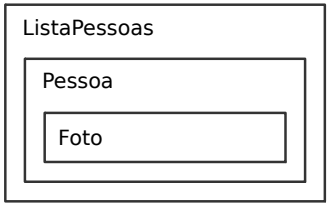


Figura 24 – Identificação das representações - primeiro passo

O segundo passo estabelece as relações entre as representações. A forma sugerida é através de um fluxograma (Figura 25). Cada representação é um elemento e as transições são as requisições permitidas. Por exemplo, a representação *ListaPessoas* permite a criação de novas pessoas através de requisições HTTP POST, e de leitura de uma pessoa

específica por meio de HTTP GET. A representação *Pessoa* permite a alteração e remoção através de HTTP PUT e DELETE respectivamente, além de consultar uma foto vinculada por meio de HTTP GET. A representação *Foto* permite a alteração e remoção através de HTTP PUT e DELETE respectivamente. Com o diagrama é possível identificar a semântica do protocolo (como realizar requisições) e a semântica da aplicação (quais dados são recebidos e enviados).

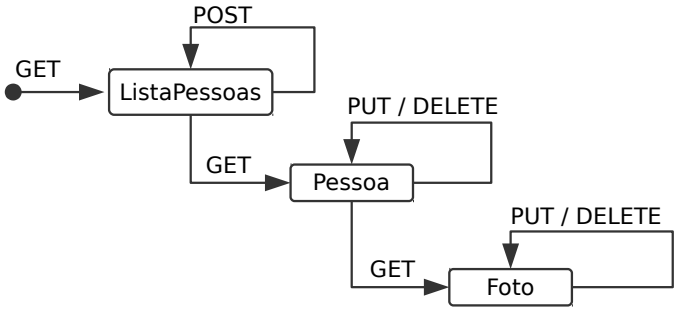


Figura 25 – Fluxograma de operações - segundo passo

O terceiro passo é associar algum significado às propriedades de cada representação. Este passo é importante para remover ambiguidades e proporcionar o entendimento amplo sobre determinada representação. Na prática esta etapa pode ser realizada através a associação das propriedades das representações com classes e propriedades publicadas em vocabulários compartilhados.

O quarto passo dedica-se à escolha adequada de um *Media Type*. O formato mais utilizado atualmente é o JSON, entretanto deve-se reconhecer suas limitações ao suporte de controles hipermídia (RICHARDSON; AMUNDSEN; RUBY, 2013).

O quinto passo está intimamente ligado ao passo quatro, e destina-se à escrita de um *profile*, que é a documentação sobre a semântica da aplicação e a semântica do protocolo. O *profile* pode ser escrito com JSON-LD quando se tratar de Web APIs somente leitura, ou Hydra para implementar transições que alteram o estado dos recursos.

O sexto passo é reservado para a implementação da Web API, sem detalhes ou diretrizes de realização. O último passo é a implantação e publicação da Web API implementada.

## 4.3 SUPORTE FERRAMENTAL

O suporte ferramental é constituído por trabalhos que apresentam soluções para a implementação, descrição semântica e documentação de Web APIs. Os trabalhos relacionados ao suporte ferramental são: Apache Isis, Spring-HATEOAS, o *framework* proposto por John e Rajasree (2012), a especificação JAX-RS, Miredot, Swagger e RestML.

### 4.3.1 Apache Isis

Apache Isis (BIENVENIDO, 2013) é uma implementação Java da especificação RESTful Objects, extensível e customizável, que faz uso de anotações para orientar a exposição de objetos de domínio. Utiliza um modelo arquitetural hexagonal, no qual os objetos de domínio assumem a posição central, enquanto o *framework* se responsabiliza pela persistência, segurança e apresentação (Figura 26). Pela sua natureza extensível e customizável, o *framework* suporta diversas tecnologias, sendo que RESTful Objects é apenas uma delas.

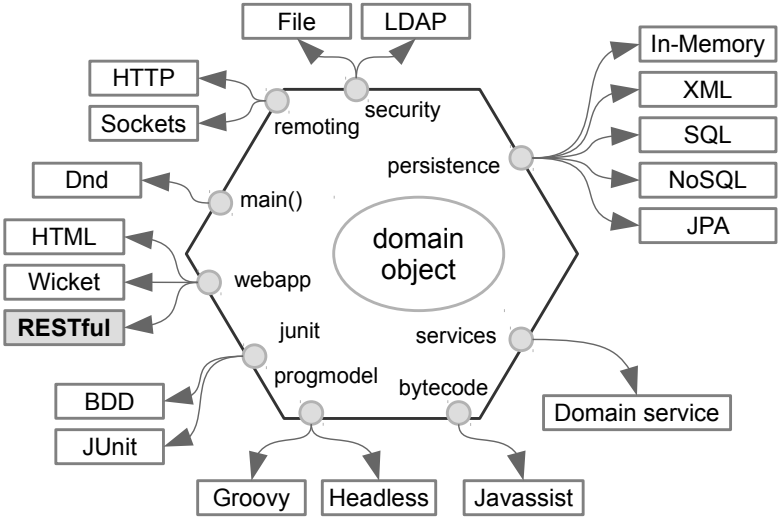


Figura 26 – Arquitetura Apache Isis. Adaptado de (BIENVENIDO, 2013)

Apache Isis é uma solução ampla que abrange diferentes aspectos do sistema. A interface RESTful é apenas um pequeno módulo do *framework*. Outros módulos tratam questões relacionadas à persistência, segurança, testes unitários, comunicação remota, dentre outras. A abrangência do Apache Isis é um fator fundamental na decisão de sua adoção, pois para usufruir das características da integração REST, é imperativo adotar todo modelo proposto pelo *framework*.

### 4.3.2 Spring-HATEOAS

Spring-HATEOAS (SPRING, 2013) é um *framework* destinado ao desenvolvimento de serviços Web baseados em hipermídia na linguagem de programação Java, que permite a criação de representações de recursos REST com controles hipermídia. As representações de recursos são disponibilizadas no formato JSON e enriquecidas semanticamente através da utilização do formato HAL. O *design* proposto pelo Spring-HATEOAS apresenta o conceito de *Resource Representation Class* que descreve apenas as propriedades da representação. As operações ficam separadas em classes de *Resource Controller*, que manipulam as representações e adicionam os *links* necessários (Figura 27).

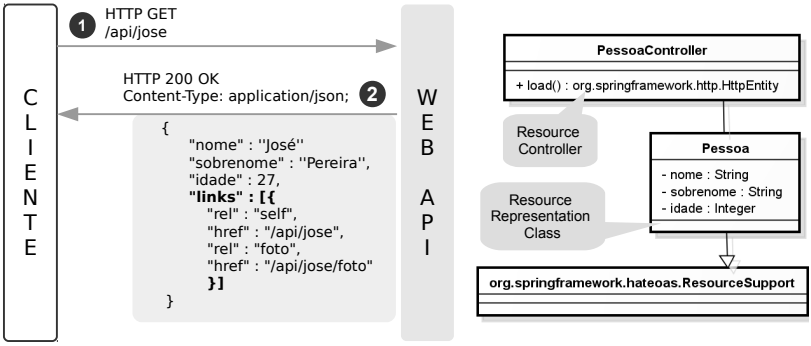


Figura 27 – Exemplo Spring HATEOAS

### 4.3.3 Framework de John e Rajasree

John e Rajasree (2012) propõem um *framework* para descrição, descoberta e composição de serviços RESTful através de anotações semânticas na documentação da Web API. O *framework* realiza anotações em documentos HTML utilizando a combinação entre marcações de *Microformats* e RDFS.

A Figura 28 mostra como o *framework* realiza as anotações semânticas. O elemento *hresource* é a anotação base para a descrição do recurso, onde todas as demais anotações estão encapsuladas. A anotação *name* é utilizada para dar um nome ao recurso, destinado a seres humanos. Através da anotação *url* é possível associar o endereço onde o recurso é disponibilizado. Todos as propriedades do recursos devem ser anotadas com *attribute*. Embora enriquecer semanticamente a documentação de Web APIs proporcione maior integração com agentes autônomos, o cliente fica restrito à documentação HTML como único guia para interação com a Web API. Além de não refletir diretamente a implementação da Web API, a documentação destinada aos agentes de software dependem da existencia de uma documentação para os seres humanos.

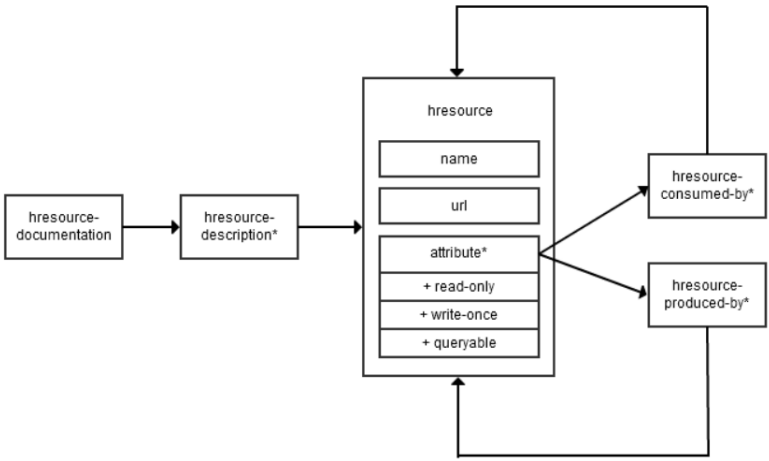


Figura 28 – Visão geral do framework. Fonte: (JOHN; RAJASREE, 2012)

### 4.3.4 Especificação JAX-RS

A especificação JAX-RS (*Java API for RESTful Web Services*) define um conjunto de APIs para o desenvolvimento de serviços Web, empregando a linguagem de programação Java, que obedecem aos princípios arquiteturais REST (ORACLE, 2012). JAX-RS proporciona um conjunto de anotações capazes de expor classes POJOs (*Plain Old Java Objects*) como recursos. A especificação JAX-RS adota o HTTP como protocolo de transporte para as representações, e oferece o suporte para a utilização correta da semântica do protocolo, ou seja, permite utilizar os verbos (GET, POST, PUT, DELETE), formatos (JSON, XML, HTML, etc) e códigos de mensagens (200, 404, 500, etc).

Listagem 14 – Exemplo de utilização de JAX-RS

```
1  @Path("/pessoa/{id}")
2  public class PessoaResource {
3
4      @PathParam("id")
5      private Long id;
6
7      private String nome;
8
9      private String sobrenome;
10
11     private String imagem;
12
13     private List<PessoaResource> amigos;
14
15     @GET
16     @Produces(MediaType.APPLICATION_JSON)
17     public Response carregar() {
18         //codigo para carregar uma pessoa
19     }
20
21     @POST
22     @Consumes(MediaType.APPLICATION_JSON)
23     public Response criar(Pessoa p) {
24         //codigo para criar pessoa p
25     }
26
27     @PUT
28     @Produces(MediaType.APPLICATION_JSON)
29     @Consumes(MediaType.APPLICATION_JSON)
30     public Response atualizar(Pessoa p) {
31         //codigo para alterar pessoa p
32     }
33
34     @DELETE
35     public Response remover() {
36         //codigo para remover uma pessoa
37     }
38 }
```

A Listagem 14 ilustra um exemplo de anotações JAX-RS. Através da anotação *@Path* (linha 1) aplicada sobre uma classe, cria-se um recurso Web, dados de uma pessoa por exemplo, que pode ser acessado através da URI */pessoa/id*. Para tornar o endereçamento dos recursos mais flexível, uma URI pode conter variáveis que são mapeadas para propriedades. A variável é mapeada para uma propriedade da classe através da anotação *@PathParam*, que especifica o nome da variável (linha 4). Quatro métodos são implementados para permitir a manipulação dos recursos, que são anotados com *@GET*, *@POST*, *@PUT* e *@DELETE* (linhas 15, 21, 27 e 34) e criam a correspondência entre os tipos de requisições HTTP e a execução dos métodos anotados. As anotações *@Consumes* e *@Produces* são responsáveis por mapear os tipos de dados de entrada e saída, respectivamente, aceitos por cada método. Os argumentos que constituem a assinatura dos métodos (linhas 23 e 30) são automaticamente mapeados do formato aceito descrito pela anotação *@Consumes* para o argumento referenciado. Cada método retorna um objeto do tipo *Response*, que representa a resposta a ser enviada pelo protocolo HTTP. O objeto *Response* retornado pelo método contém o código de *status* HTTP para a requisição e, opcionalmente, uma representação de recurso vinculado, denominada *entity*. Essa representação, quando presente, é automaticamente mapeada para o formato indicado pela anotação *@Produces*. Caso mais de um formato seja suportado, é escolhido o formato de preferência do cliente, indicado no cabeçalho da requisição HTTP.

### 4.3.5 Miredot

QMINO (2014) apresenta um *framework* denominado Miredot, capaz realizar a geração automática da documentação de Web APIs desenvolvidas em Java com o *framework* JAX-RS. A documentação é gerada através da análise do código fonte da aplicação e resulta em um documento HTML adequado para guiar desenvolvedores de aplicações clientes. A Figura 29 mostra um exemplo de documentação gerado pelo Miredot. No painel localizado a direita, é possível observar os recursos manipulados pela Web API, como por exemplo: *catalog, category, item, product, sales*, dentre outros. Na parte central da documentação, são apresentados os detalhes de cada recurso, como por exemplo: as operações, métodos, descrições, URIs e parâmetros.

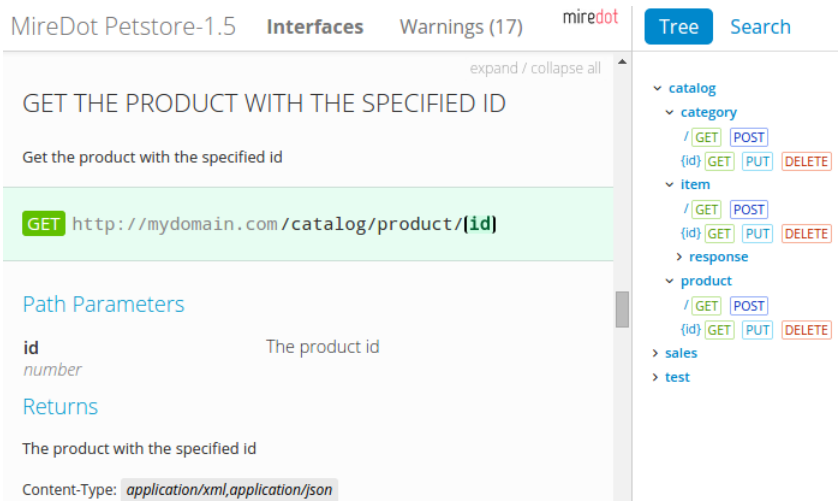


Figura 29 – Exemplo documentação Miredot. Fonte: (QMINO, 2014)

A documentação HTML considera os comentários feitos em *Javadoc*, como mostra a Figura 30. As anotações *@param, @return* e *@summary* são utilizadas para criar a documentação da Web API gerada pelo Miredot. As demais informações do recurso são obtidas através das anotações do *framework* JAX-RS, como por exemplo, *@GET, @Path, @Produces*, entre outras.

```
/**
 * Get the product with the specified id
 *
 * @param productId The product id
 * @return The product with the specified id
 * @summary Get the product with the specified id
 */
@GET
@Path("/product/{id}")
@Produces({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
public Product findProduct(@PathParam("id") Long productId);
```

Figura 30 – Exemplo código Miredot. Fonte: (QMINO, 2014)

### 4.3.6 Swagger

Reverb Technologies (2014) apresenta um *framework* denominado Swagger, capaz de gerar a documentação de Web APIs. Permite que desenvolvedores e agentes de software compreendam como realizar a comunicação com serviços remotos. A documentação é gerada de forma automática através da análise do código fonte da Web API. Swagger é capaz de gerar a documentação de Web APIs desenvolvidas com várias tecnologias, como Java, Go, .Net, PHP, Scala, Ruby, dentre outras. A Figura 31 mostra um exemplo de documentação HTML gerado pelo Swagger. A documentação apresenta os detalhes dos recursos Web manipulados pela Web API, além de permitir a execução das operações na própria página HTML da documentação.

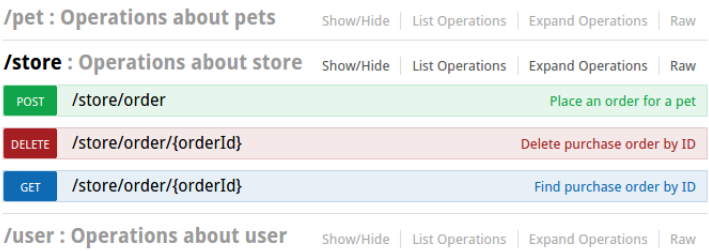


Figura 31 – Documentação Swagger. Fonte: (Reverb Technologies, 2014)

### 4.3.7 RestML

RestML é uma abordagem baseada no paradigma *Model Driven Development* (MDD), e busca facilitar o desenvolvimento de serviços Web. Proposto por Sanchez, Oliveira e Fortes (2014), baseia-se na construção de um modelo que resulta na geração automática de código fonte. Através de um *profile* UML, permite que o modelo desenvolvido possa ser transformado em código fonte que obedece às definições arquiteturais REST. RestML propõe também um outro *profile* UML específico para a plataforma Java EE.

A Figura 32 mostra a modelagem de Web APIs RESTful através do *profile* UML proposto pelo RestML. Classes modeladas com o esteriótipo *Resource* recebem as requisições de aplicações clientes. *Ser-*

*vice* é utilizado em componentes responsáveis pelas regras de negócio. *Representation* descreve um formato de dados específico para um recurso. *Exception* descreve possíveis erros que podem acontecer durante a execução da Web API. *DataAccess* é utilizado em componentes que realizam acesso a dados externos a aplicação.

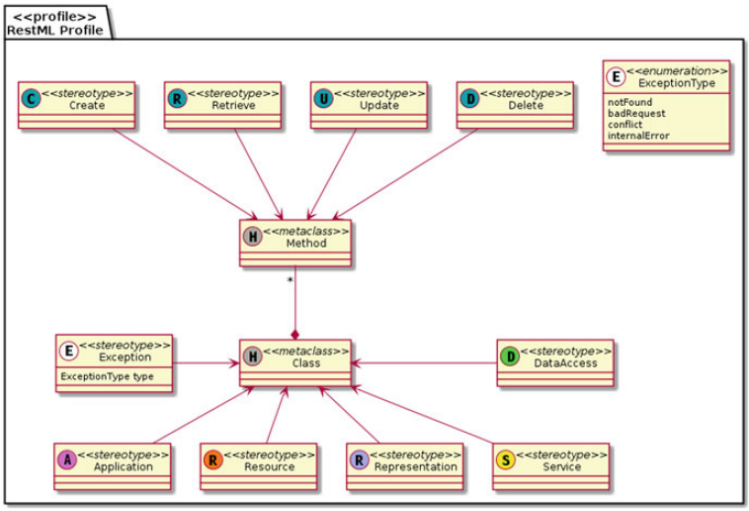


Figura 32 – RestML. Fonte: (SANCHEZ; OLIVEIRA; FORTES, 2014)

## 4.4 MODELOS DE MATURIDADE

Modelos de maturidade são importantes pois oferecem mecanismos para avaliar e comparar a qualidade de Web APIs. *Richardson Maturity Model* e *CoHA* são dois modelos de maturidade que podem ser utilizados para classificar e avaliar implementações de Web APIs.

### 4.4.1 Richardson Maturity Model

*Richarson Maturity Model* (RMM) classifica o *design* de Web APIs em quatro níveis de maturidade (WEBBER; PARASTATIDIS; ROBINSON, 2010). A Figura 33 ilustra os níveis do modelo, partindo do mais baixo nível de maturidade (pântano) a caminho da glória.

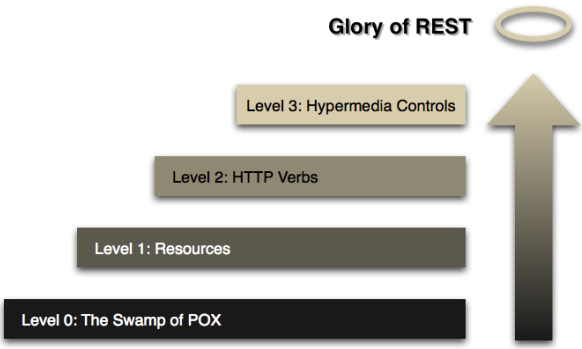


Figura 33 – Níveis de maturidade RMM. Fonte: (FOWLER, 2010)

O nível mais baixo de maturidade consiste na utilização do protocolo HTTP apenas como meio de transporte. Não aplica nenhum dos conceitos Web e não respeita a semântica do protocolo HTTP. O formato mais utilizados é XML, utilizado como uma espécie de envelope, denominado *Plain Old XML* (POX). A comunicação é realizada através do estilo RPC (*Remote Procedure Call*), que concentra todos os serviços em um único ponto de acesso.

O segundo nível de maturidade exige que as informações sejam estruturadas na forma de recursos. As informações manipuladas pela Web API são distribuídas entre diferentes recursos. Cada recurso é identificado e recebe um endereço, permitindo sua manipulação de recursos individuais. Cada recurso disponibiliza um conjunto de operações que são aplicadas sobre os próprios recursos.

O terceiro nível de maturidade refere-se à conformidade e ao respeito à semântica do protocolo utilizado pela Web API. As quatro operações básicas disponibilizadas pelo protocolo HTTP são GET, POST, PUT e DELETE. O terceiro nível de maturidade exige o uso correto dessas operações sobre os recursos. Isto implica que recursos devem ser criados através da operação POST, alterados por PUT, recuperados por GET e removidos por DELETE. Esse nível exige também o uso correto de *status code*, para descrever corretamente as respostas das requisições realizadas.

O quarto nível de maturidade exige a aplicação de HATEOAS, que adiciona controles hipermídia às representações. O uso de controles hipermídia permite aos clientes da Web API manipular os recursos de forma exploratória e desacoplada dos detalhes de implementação.

### 4.4.2 CoHA Maturity Model

*Classification of HTTP-based APIs* (CoHA), proposto por Algermissen (2010), define cinco níveis de maturidade. CoHA é uma modelo de maturidade mais amplo, pois considera implementações de sistemas baseados em HTTP, sem ficar restrito à implementações REST.

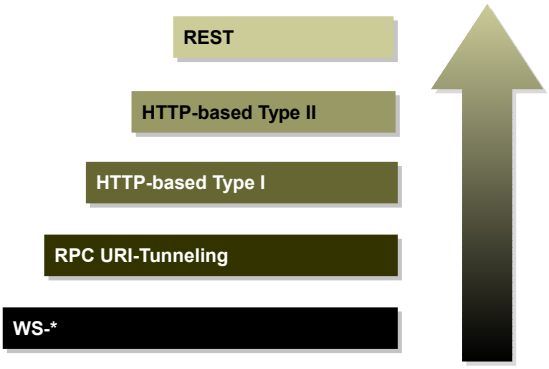


Figura 34 – Níveis de maturidade CoHA

O nível mais baixo de maturidade é denominado *WS-** e se aplica a implementações baseadas na tecnologia SOAP que tratam o protocolo HTTP apenas como mecanismo de transporte de dados. Nesse nível, URIs identificam apenas serviços, todas as informações necessárias para a sua execução são encapsuladas no envelope SOAP. É considerada a pior abordagem do ponto de vista de custo/benefício, pois apresenta alto custo de adoção, possui dependência com fornecedores de tecnologia além de resultar em alto custo para evolução dos sistemas.

O segundo nível é denominado *RPC URI-Tunneling*, o qual exige que as informações sejam estruturadas como recursos, embora as operações não respeitem necessariamente a semântica do protocolo utilizado pela Web API. Embora os recursos sejam identificáveis, esse nível não garante a manipulação de recursos através de várias representações. Esse nível resulta em alto acoplamento entre cliente e servidor e dificulta a evolução independente entre as partes.

O terceiro nível é denominado *HTTP-based Type I* e exige que recursos sejam manipulados de forma a respeitar a semântica do protocolo utilizado pela Web API. Os recursos são manipulados através de representações, permitindo utilizar diferentes formatos de dados. Esse

nível exige o comportamento *stateless* do servidor, ampliando a capacidade de escalabilidade da Web API. Possui baixo custo de adoção, entretanto, possui alto custo de evolução devido ao alto acoplamento entre clientes e servidor.

O quarto nível é denominado *HTTP-based Type II* e exige que as mensagens sejam auto-descritivas. Define que os recursos, operações e formatos de dados sejam descritos no momento do *design*. Web APIs implementadas com esse nível de maturidade são consideradas simples na perspectiva da interação cliente-servidor, pois os recursos são manipulados através de uma interface uniforme.

O quinto nível é denominado REST pois obedece a todas as restrições descritas pelo princípio arquitetural REST. Os recursos são identificáveis e endereçáveis, manipulados através de representações, as mensagens são auto-descritivas e os recursos possuem controles hipermídia. Esse nível de maturidade exige alta curva de aprendizado, dificultado a adoção. Entretanto, apresenta o menor custo de manutenção e evolução, pois as mudanças na Web API não causam impactos nas aplicações clientes.

## 4.5 CONSIDERAÇÕES FINAIS

As tecnologias discutidas neste capítulo oferecem suporte ao desenvolvimento de Web APIs baseadas em JSON, controles hipermídia e descrição semântica. Dentre os formatos de dados apresentados, apenas JSON Objects e JSON HAL possuem *frameworks* que oferecem apoio ao desenvolvimento. Os demais formatos não possuem suporte ferramental necessário para o desenvolvimento prático com produtividade adequada. A carência de ferramentas fica mais evidente quando considerado o suporte para descrição semântica de recursos Web. Nenhuma ferramenta capaz de descrever semanticamente, de forma direta, os recursos manipulado por Web APIs REST, foi encontrado na literatura.

Entre as abordagens de desenvolvimento, DAP apresenta a necessidade do desenvolvimento de um protocolo capaz de representar as características de um domínio específico. A abordagem proposta por Richardson e Amundsen incorpora os conceitos presentes no DAP e apresenta um guia para o desenvolvimento de Web APIs, que contém iniciativas para a descrição semântica dos recursos Web. Para garantir a qualidade dos sistemas distribuídos, é fundamental promover diretrizes de desenvolvimento, suporte ferramental e mecanismos para verificação da qualidade de Web APIs RESTful Semânticas.

# 5 ABORDAGEM DE DESENVOLVIMENTO

Este capítulo apresenta uma proposta de abordagem para o desenvolvimento de Web APIs RESTful semânticas. A abordagem é constituída por um conjunto de etapas que guiam o desenvolvimento de Web APIs em todo seu processo de criação. A abordagem de desenvolvimento auxilia na organização e descrição semântica das informações e proporciona diretrizes para orquestração das operações sobre os recursos Web. Este capítulo apresenta também um modelo de maturidade que contempla os aspectos de *design*, *profile* e descrição semântica, criando dessa forma, mecanismos para avaliar implementações de Web APIs semânticas.

## 5.1 ETAPAS DE MODELAGEM DE WEB APIS

A abordagem proposta nesta dissertação apresenta um conjunto de etapas para a modelagem de Web APIs RESTful Semânticas que manipulam dados baseados em JSON, através do paradigma de orientação a objetos. As etapas são baseadas nas abordagens apresentadas por Richardson, Amundsen e Ruby (2013) e no *Domain Application Protocol* (ROBINSON, 2011). As etapas que constituem a abordagem de desenvolvimento são:

- **Identificação de Recursos e Representações:** busca identificar e agrupar as classes do domínio da aplicação que serão expostas na camada de integração através de representações. Essa etapa tem o principal objetivo de separar a camada de integração das demais camadas do sistema.

- **Descrição Semântica de Dados e Operações:** busca associar as representações a vocabulários controlados e compartilhados. Essa etapa tem o objetivo de proporcionar uma interação autônoma e oferecer mecanismos adequados para que os consumidores interpretem adequadamente as informações manipuladas pela Web API.

- **Utilização do *Design* de Representações Atômicas:** tem o objetivo de definir um modelo de interação entre a Web API e seus clientes, restringindo a manipulação das propriedades das representações de forma isolada.

- **Documentação da Web API utilizando o *profile* Hydra:** busca expor os detalhes das representações manipuladas pela Web API, proporcionando as diretrizes necessárias para que aplicações clientes atinjam seus objetivos.

- **Orquestração de Operações Hydra baseada em Representações:** possibilita que a Web API possa guiar as aplicações clientes em interações complexas que exigem diversas requisições para atingir um determinado objetivo.

### 5.1.1 Identificação de Recursos e Representações

Um dos fatores fundamentais na abordagem proposta é a separação da camada de integração de dados das demais camadas do sistema. A separação permite liberdade para modelar os componentes específicos de negócio (camada de domínio) sem a interferência dos detalhes de integração. Frequentemente, as classes do domínio da aplicação são adaptadas para satisfazer algumas exigências impostas pela integração de dados, como por exemplo, a formatação de dados numéricos e datas. Outra vantagem na separação das camadas é a possibilidade de segmentar a informação de forma adequada para diferentes aplicações clientes. As classes do domínio da aplicação constituem os recursos, e as classes de integração de dados são modeladas na forma de representações, conforme ilustrado na Figura 35.

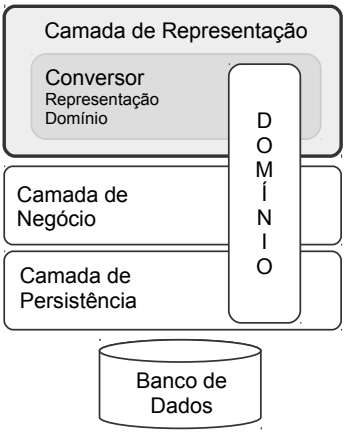


Figura 35 – Separação de camadas

Cada camada existente no sistema possui responsabilidades bem definidas. É importante respeitar essas responsabilidades e não atribuir tarefas inadequadas para uma determinada camada. A camada de domínio da aplicação contém as classes e as regras de negócio do sistema. A camada de domínio é posicionada de forma ortogonal, ou seja, está presente nas demais camadas. A camada de negócio proporciona uma infraestrutura para a execução e interação dos objetos de domínio da aplicação, estabelecendo transações e solicitando a manipulação de dados para a camada de persistência. A camada de persistência é responsável por manter os dados do sistema. A persistência pode ser realizada de diversas formas, através de banco de dados relacionais, armazenamento em memória, dentre outras. A camada de representação é responsável por receber e enviar representações de recursos, encaminhando as solicitações para operações de negócio adequadas. Apenas as classes da camada de representação são disponibilizadas aos consumidores. Os recursos são manipulados indiretamente através de suas respectivas representações. Para que haja a separação de camadas é necessário um conversor capaz de traduzir as representações em objetos de domínio, e também objetos de domínio em representações.

Quando uma aplicação cliente solicita um determinado recurso, a camada de representação solicita a execução da operação de negócio adequada, que por sua vez solicita para a camada de persistência uma determinada consulta ao banco de dados. A camada de persistência obtém o dado e retorna um objeto de domínio da aplicação. A camada de negócio recebe este objeto, aplica as regras adequadas e retorna o objeto para a camada de representação. Ao receber o objeto de domínio, a camada de representação realiza as conversões necessárias e disponibiliza para a aplicação cliente. Quando uma aplicação cliente envia uma representação para a Web API, a camada de representação realiza a conversão para um objeto de domínio e solicita a execução da operação de negócio adequada.

É possível relacionar recursos e representações de diferentes formas, conforme mostra a Figura 36. A primeira forma é quando um recurso é relacionado com apenas uma representação. Nesse caso, todas as suas propriedades e operações fazem parte de uma única representação. A segunda forma é quando um recurso distribui as suas propriedades e operações entre diversas representações, de modo que cada representação contém um subconjunto das características do recurso. Outra possível forma de associação entre um recurso e várias representações ocorre quando busca-se apenas oferecer alternativas de formato de dados. Sendo assim, cada formato de dados é uma classe

de representação de um mesmo recurso. Outro cenário possível é a combinação de diversos recursos em uma única representação. Nessa configuração, uma representação responde por mais de um recurso. As representações proporcionam uma visão particular das classes de domínio da aplicação, com objetivo de realizar a melhor integração de dados possível, sem exercer influência na modelagem das demais camadas do sistema. Ao final da primeira etapa da abordagem, as representações são identificadas e podem ser descritas semanticamente pela próxima etapa.

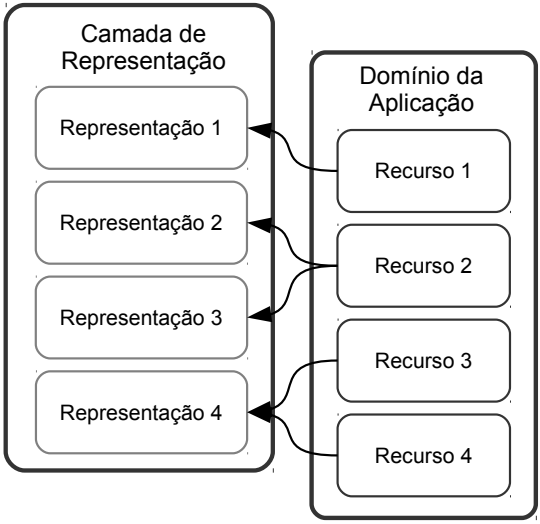


Figura 36 – Relação entre recursos e representações

### 5.1.2 Descrição Semântica

A aplicação dos conceitos da Web Semântica em sistemas que utilizam os princípios arquiteturais REST resulta em Web APIs RESTful Semânticas. Através de Web APIs RESTful Semânticas, é possível desenvolver clientes com maior poder de automação, interoperabilidade e interpretação de informações. Esses clientes são denominados agentes semânticos, e possuem conhecimento suficiente para realizar a comunicação com as Web APIs que manipulam recursos semânticos. Para proporcionar maior poder de interpretação das informações disponibi-

lizadas pela Web API aos agentes semânticos, é necessário descrever as informações em termos de conceitos, regras e papéis através de um domínio compartilhado (KLUSCH, 2008).

A descrição semântica é realizada através da associação entre as representações e vocabulários controlados. O vocabulário é formado por classes semânticas e termos que dão significado a um determinado domínio de aplicação. O vocabulário é compartilhado entre a Web API e agentes semânticos, estabelecendo apenas um acoplamento conceitual. As representações são disponibilizas em JSON-LD, e suas operações são descritas pelo vocabulário Hydra, proporcionando um *profile* legível aos agentes semânticos. A Figura 37 mostra a pilha de tecnologias utilizadas para descrição semântica das representações.

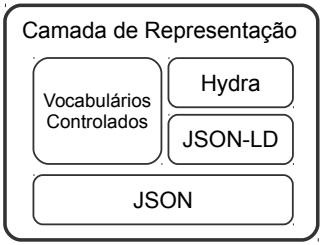


Figura 37 – Pilha de tecnologias para descrição semântica

A etapa de descrição semântica busca associar as representações e suas propriedades à classes semânticas e a termos de vocabulários controlados, respectivamente. As propriedades não estão restritas apenas a associação com termos de vocabulários. É possível associar propriedades com classes semânticas para referenciar outras representações. As operações devem ser associadas ao vocabulário Hydra, permitindo descreve-las com mais detalhes, além de possuir um significado associado.

A Figura 38 mostra um exemplo de descrição semântica de representações. A representação *Pessoa* está associada à classe semântica *Person* e a representação *Empresa* está associada à classe semântica *Organization*, pertencentes ao vocabulário *Schema.org*. As propriedades *nome* e *sobrenome* estão associadas a termos da classe semântica *Person*, entretanto, a propriedade *trabalhaPara* está associada à classe semântica *Organization*, e não a um termo do vocabulário. Na verdade a propriedade *trabalhaPara* é uma referência para outra representação que possui o significado associado à classe semântica *Organization*.

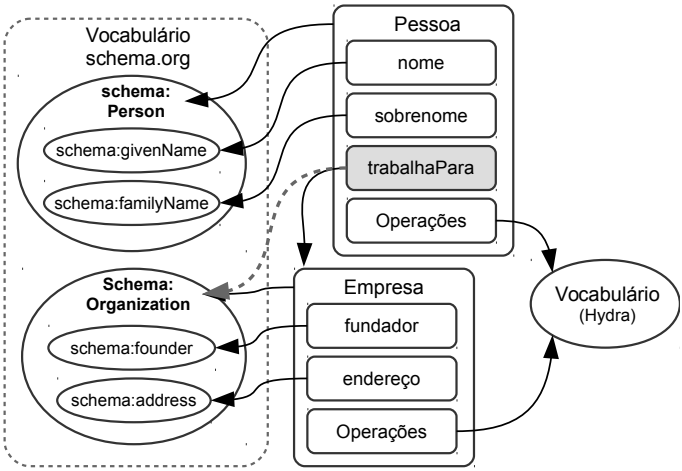


Figura 38 – Descrição semântica de representações

Quando uma propriedade referencia outra representação, o valor da propriedade é uma URL que endereça a representação. Entretanto, pode-se modelar o relacionamento entre representações sem o uso de referências. A Figura 39 mostra um exemplo onde uma propriedade encapsula uma representação, sem utilizar referência. Nesse tipo de relacionamento o valor da propriedade deixa de ser uma URL e incorpora a estrutura da representação relacionada. Com isso, cria-se o conceito de representações não endereçadas, pois não há a necessidade de uma URL associada a uma determinada representação.

Não é necessário que todas as representações identificadas sejam associadas a uma classe semântica. Duas representações podem possuir propriedades associadas a termos de uma única classe semântica. Essa modelagem permite liberdade para organizar as informações da forma mais adequada. A Figura 40 mostra um exemplo onde duas representações estão associadas a uma única classe semântica. A representação *Pessoa* possui três propriedades, entretanto, apenas *nome* e *sobrenome* estão associados diretamente a termos da classe semântica *Person*. A propriedade *dadosContato* é uma referência para outra representação, que possui outras propriedades associadas à mesma classe semântica. As duas representações correspondem a uma mesma classe semântica, e as propriedades podem ser organizadas de diferentes formas com o intuito de melhor atender aos objetivos de negócio e oferecer mais opções de integração de dados.

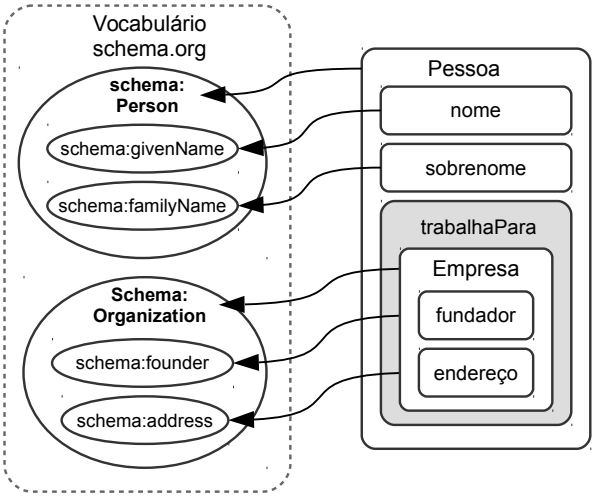


Figura 39 – Representação não endereçada

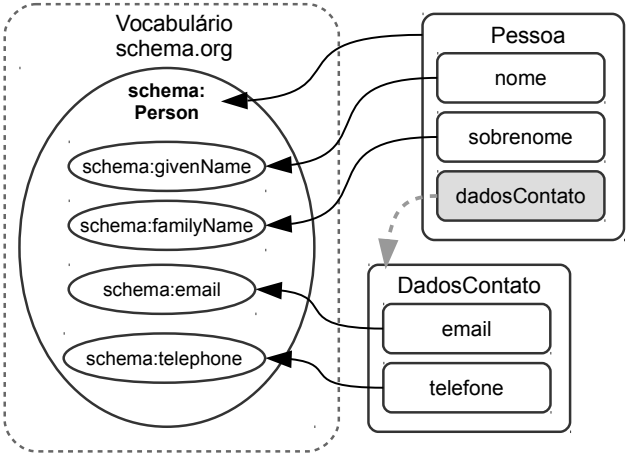


Figura 40 – Duas representações associadas a uma classe semântica

A abordagem de desenvolvimento proposta abrange diversas formas para organizar a informação manipulada pela Web API, entretanto, algumas restrições devem ser respeitadas de forma a não tornar a informação ambígua e dificultar sua interpretação. É permitido associar uma representação a mais de uma classe semântica somente quando existe a relação de equivalência. Esse tipo de associação é utilizado quando existe mais de um vocabulário de domínio de aplicação. Entretanto, não é permitido associar uma representação a diferentes classes semânticas que não estabelecem uma relação de equivalência. A Figura 41 mostra um caso onde uma representação é descrita semanticamente por diferentes classes semânticas. Neste exemplo, as propriedades *nomeChefe* e a *localTrabalho* estão associadas a termos da classe semântica *Organization* por razões de agregação e não por equivalência. Esse tipo de descrição semântica resulta em uma situação de ambiguidade, e não é aceita pela abordagem de desenvolvimento proposta.

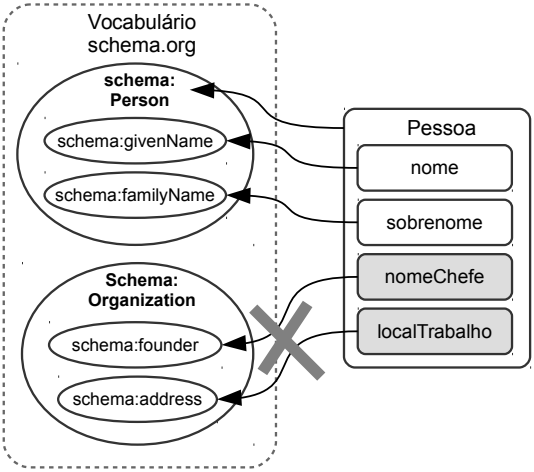


Figura 41 – Descrição semântica incorreta

### 5.1.3 Representações Atômicas

O *design* de Web APIs é um fator muito importante para qualidade da comunicação entre clientes e servidor, pois define como as informações devem ser enviadas, recebidas e interpretadas. As principais formas de modelagem de Web APIs são RPC e *Recursos*.

Web APIs implementadas com o estilo RPC utilizam o protocolo HTTP apenas como transporte para requisições, que são interpretadas como serviços Web disponibilizados através de uma única URI base. O formato da comunicação é fortemente baseado em serviços, que recebem um *payload* de dados e executam uma operação. De forma geral, o *design* RPC não obedece a semântica do protocolo HTTP. Os serviços são executados unicamente através do método HTTP POST e o *status code* é empacotado juntamente com o *payload* de resposta.

O *design* baseado em Recursos agrupa informações para formar um conjunto coeso de dados. Cada recurso recebe um endereço URI, que disponibiliza um conjunto de operações sobre os dados referentes ao próprio recurso. Geralmente as operações limitam-se à criação, leitura, atualização e remoção de recursos, compatíveis com aplicações CRUD.

As melhores modelagens de Web APIs REST são aquelas que aproveitam as características naturais da Web e aplicam corretamente a semântica do protocolo utilizado pela Web API (geralmente HTTP). É possível modelar de diferentes formas Web APIs com foco em recursos. Uma forma que possui várias características positivas é o *design* de *Representações Atômicas*.

O *design* de *Representações Atômicas* restringe a manipulação isolada de propriedades de representações. Nesta modelagem a menor unidade de manipulação é a representação, e as operações disponibilizadas devem obrigatoriamente refletir sobre todas as propriedades.

A Figura 42 mostra um exemplo de representação que não segue o modelo atômico. A representação possui quatro propriedades, entretanto, é possível alterar isoladamente as propriedades *email* e *telefone* através da operação *modificarDadosContato*. Esta modelagem exige a descrição semântica da operação *modificarDadosContato*, pois não pode ser associada à semântica das operações CRUD. Também não é possível usar a referência da semântica do protocolo HTTP, pois existem duas operações que utilizam o método HTTP PUT.

É possível transformar as representações que não seguem o modelo atômico em representações atômicas. A transformação consiste em criar uma nova representação para as propriedades que podem ser alteradas isoladamente. A Figura 43 mostra a modelagem de duas representações atômicas. As propriedades *email* e *telefone* foram realocadas para uma nova representação, de modo a possibilitar que as duas representações sejam manipuladas de forma atômica. Nenhuma semântica é necessária para descrever as operações sobre a representação *DadosContato*, pois a própria semântica do HTTP é suficiente para descrever o comportamento das operações.

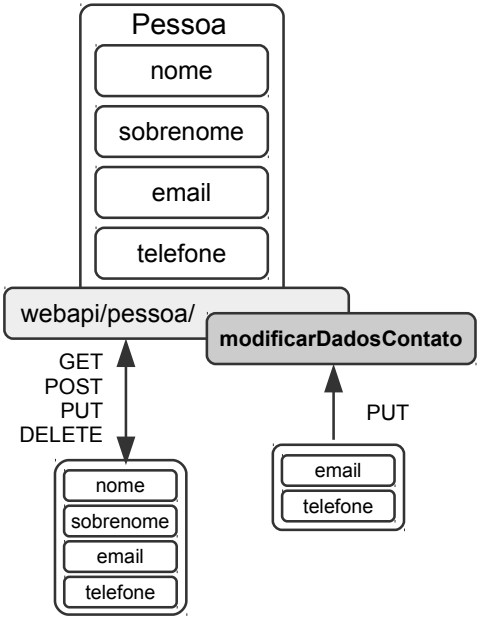


Figura 42 – Exemplo de representação não atômica

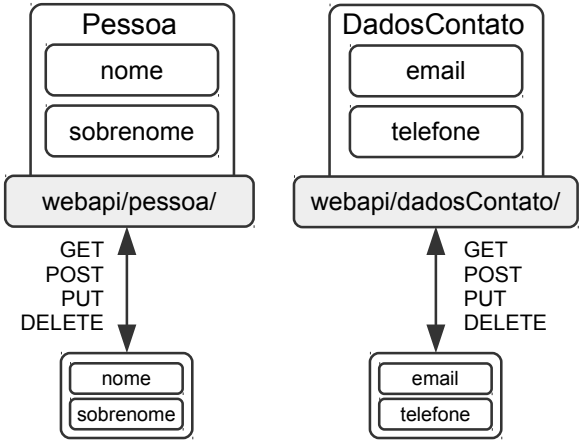


Figura 43 – Exemplo de representação atômica

### 5.1.4 Geração da Documentação

A geração da documentação é uma etapa fundamental para o desenvolvimento de Web APIs semânticas. A documentação deve ser disponibilizada em formato legível para agentes semânticos, além de apresentar os detalhes das representações.

A Listagem 19 mostra um exemplo de documentação gerada no formato Hydra. O contexto semântico (linhas 2 a 5) é constituído pelos vocabulários *hydra* e *schema*, que possuem as referências necessárias para a consulta ao significado dos termos. A propriedade *@type: ApiDocumentation* define o documento JSON como uma Documentação de Web API (linha 6). Entre as linhas 7 a 18, são documentadas as classes suportadas pela Web API. Todas as representações identificadas devem estar presentes na documentação, apresentando os detalhes de suas propriedades e operações.

A documentação deve ser disponibilizada para os agentes semânticos interessados em interagir com a Web API. Dessa forma, recomenda-se que a documentação esteja disponível através da URL base da Web API. Ao ser disponibilizada através da URL base, a documentação assume o papel de *entry point* da Web API, sendo o primeiro recurso a ser consumido por agentes semânticos. É fundamental que esta documentação seja considerada efêmera, ou seja, pode sofrer alterações a qualquer momento. Deve-se utilizar mecanismos de *cache* para orientar os agentes sobre a validade das informações contidas na documentação.

Listagem 15 – Documentação Hydra

```
1  {
2    "@context": {
3     "hydra":"http://www.w3.org/ns/hydra/context.jsonld",
4     "schema":"http://schema.org/"
5    },
6    "@type": "ApiDocumentation",
7    "supportedClasses": [
8      {
9        "@id": "schema:Person",
10       "supportedProperties": [...],
11       "supportedOperations": [...]
12     },
13     {
14       "@id": "schema:Organization",
15       "supportedProperties": [...],
16       "supportedOperations": [...]
17     }, ...
18   ]
19 }
```

Umas das principais funções da documentação da Web API é proporcionar as diretrizes para a execução de operações. A documentação descreve as operações através do vocabulário Hydra, que originalmente define três tipos de operações: *CreateResourceOperation*, *ReplaceResourceOperation* e *DeleteResourceOperation*. A abordagem proposta adiciona a operação *LoadResourceOperation*, destinada a descrever as operações responsáveis pelo carregamento de uma determinada representação, fundamental para guiar os agentes semânticos na obtenção de dados. É possível utilizar operações mais complexas que atendem a determinados objetivos de negócio, que necessitam de uma descrição semântica específica. Para esses casos particulares a abordagem de desenvolvimento adiciona a operação *SupportedOperation*, que pode ser utilizada em tarefas mais complexas e possuir um valor semântico associado. Embora a abordagem estabeleça mecanismos para a modelagem de operações especificas de um domínio, é recomendado que as representações sigam o modelo atômico.

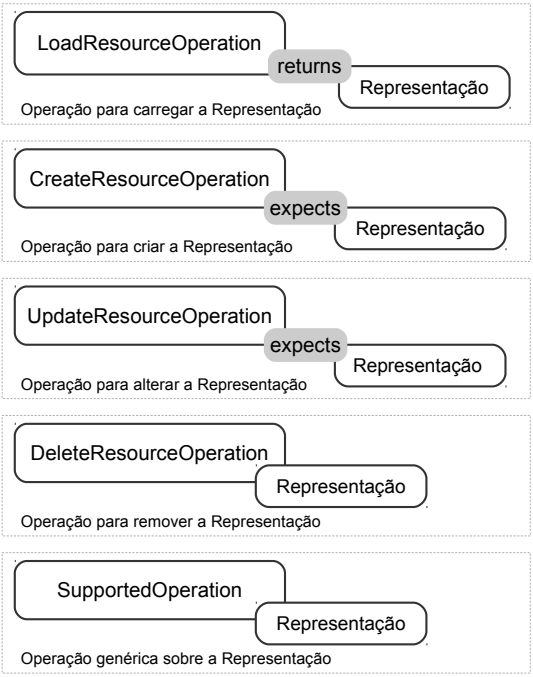


Figura 44 – Semântica das operações Hydra - estendido

A Figura 44 mostra as alternativas que podem ser utilizadas para a descrição semântica das operações. A operação associada ao termo *LoadResourceOperation* que possui como retorno uma determinada representação deve ser interpretada como a operação responsável pelo carregamento do recurso. A operação associada ao termo *CreateResourceOperation* que recebe uma determinada representação deve ser interpretada como responsável pela criação de um novo recurso. O termo *ReplaceResourceOperation* identifica a operação que altera os dados de uma representação de recurso. A operação associada ao termo *DeleteResourceOperation* deve ser interpretada como responsável pela remoção do recurso à ela associado. A operação *SupportedOperation* deve ser associada a um significado semântico de um domínio especifico, e sua interpretação está vinculada a esse vocabulário.

Ao final da etapa de descrição semântica dos dados e operações, as representações devem estar associadas à classes semânticas de vocabulários controlados e compartilhados com as aplicações clientes. As operações identificadas devem estar associadas com o vocabulário Hydra. As representações exigidas e retornadas para a execução das operações também devem ser identificadas. A etapa de *design* de representações atômicas não é obrigatória, apesar de ser recomendada. A etapa de documentação da Web API deve apresentar os detalhes de todas as representações. É necessário que a documentação seja disponibilizada de forma legível a agentes semânticos.

### 5.1.5 Orquestração de Operações

Aplicações complexas geralmente possuem processos complexos, que podem exigir várias requisições envolvendo diferentes recursos. Uma abordagem é encapsular todos os recursos necessários para a execução de um processo complexo em uma única representação, permitindo a realização de todas as etapas em uma única requisição. Esta abordagem é adequada quando a aplicação cliente possui previamente todas as informações necessárias para a execução da etapa.

Alguns processos dependem de informações produzidas ou disponibilizadas pela Web API durante a execução da etapa. Neste cenário, o processo não pode ser executado em uma única requisição, e deve haver um mecanismo para guiar o cliente para a próxima etapa através de controles hipermídia. A orquestração de operações baseada em representações pode ser utilizada para guiar o agente semântico através de operações Hydra descritas em uma determinada representação. A

orquestração de operações pode ser aplicada diretamente na representação do recurso, no cabeçalho HTTP ou na documentação da Web API.

A Figura 45 ilustra um exemplo de orquestração aplicada sobre o cabeçalho HTTP. O agente solicita a criação de um novo recurso com o envio da representação *Pessoa*, através da requisição HTTP POST (1). A Web API responde que o recurso foi criado com sucesso (2) e informa ao agente qual é a próxima etapa. A próxima etapa solicitada pela Web API é a execução da operação *CreateResourceOperation* descrita na representação *DadosContato*, que resulta na criação de um novo recurso. O agente deve ser capaz de executar a operação solicitada, pois os detalhes estão disponibilizados na documentação Hydra da Web API. Dessa forma, o agente finaliza o processo de negócio através da operação (3) e recebe a confirmação de sucesso (4).

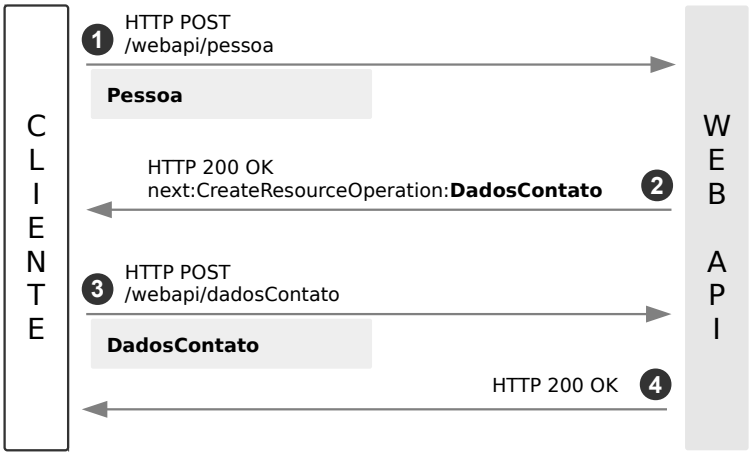


Figura 45 – Orquestração de operações

A orquestração de operações baseada em representações proporciona um mecanismo independente do domínio da aplicação, pois guia a execução de processos complexo através de controles hipermídia. O resultado é a redução do acoplamento entre agentes semânticos e a Web API, pois permite que as operações sejam descobertas durante a execução do processo, caso contrário, o agente deveria ter conhecimento prévio sobre a ordem de execução das operações. As aplicações clientes que não são capazes de interpretar a orquestração, não podem ser guiadas pela Web API na execução de operações complexas.

## 5.2 MODELO DE MATURIDADE $WS^3$ PARA WEB APIS

Segundo Sjoberg, Dyba and Jorgensen (SJOBERG; DYBA; JORGENSEN, 2007), a grande maioria dos estudos realizados é destinada à evolução de novas tecnologias, enquanto os estudos sobre avaliações de tecnologias são menos explorados. Modelos de maturidades podem ser utilizados para proporcionar critérios para avaliar implementações já realizadas, bem como as tecnologias utilizadas. Heitmann et al. (2012) apresentam um estudo empírico que revela o nível de adoção da Web Semântica no desenvolvimento de aplicações, além de identificar os principais desafios tecnológicos. Um dos desafios identificados por Heitmann et al. (2012) é a falta de guias e padrões para o desenvolvimento de aplicações semânticas. Neste sentido, um modelo de maturidade capaz de avaliar aplicações com características semânticas pode ser utilizado como guia de desenvolvimento. Além disso, Tahir e Macdonell (2012) concluem que grande parte das métricas de qualidade são estabelecidas para sistemas Orientados a Objetos. Métricas para outros tipos de sistemas, por exemplo Web, são pouco estudadas. Esta seção apresenta uma proposta de modelo de maturidade capaz de avaliar Web APIs RESTful Semânticas.

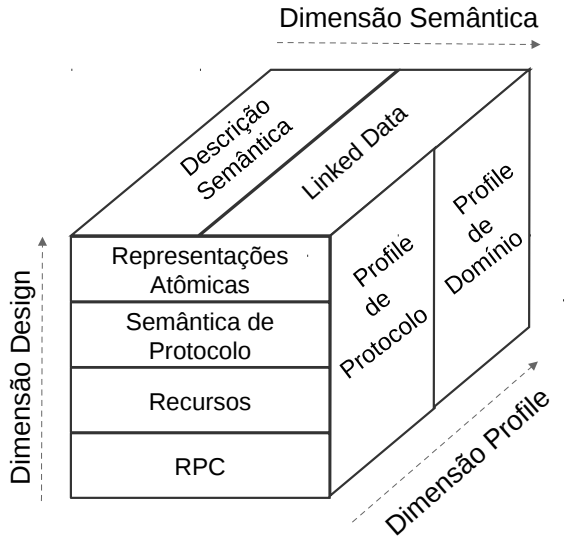


Figura 46 – Modelo de maturidade $WS^3$ para Web APIs

O modelo de maturidade $WS^3$, conforme ilustrado na Figura 46, tem formato cúbico, no qual as faces representam diferentes critérios de qualidade, que estão dispostas em três dimensões. A dimensão *Design* agrupa diferentes formas de modelagem de Web APIs. A dimensão *Profile* é constituída pelos níveis de documentação da Web API. A dimensão *Semântica* descreve o nível de suporte semântico implementado. As três dimensões são independentes e podem estar presentes simultaneamente na mesma implementação.

### 5.2.1 Critérios de Qualidade

Meskens (1996) apresenta uma metodologia para estabelecer um diagnóstico baseado em critérios de qualidade. A metodologia pode auxiliar desenvolvedores a identificar se uma aplicação deve ser reestruturada ou completamente reescrita. Para (MESKENS, 1996), flexibilidade, manutenibilidade e testabilidade são os principais fatores de qualidade. A partir destes fatores, Meskens (1996) identifica diversos critérios de qualidade como consistência, auto-documentação, modularidade e coesão. Meskens (1996) identifica também algumas métricas de qualidade como acoplamento, tamanho, coesão, etc. A metodologia define quatro níveis de abstração. O nível de implementação permite que a linguagem de programação seja abstraída, além de identificar e compreender como os componentes do sistema (arquivos, módulos, variáveis, etc) se relacionam. O nível estrutural apresenta detalhes sobre a influência entre as partes do sistema. O nível funcional ajuda a explicar os relacionamentos lógicos entre os componentes do sistema. Por fim, o nível de domínio oferece uma interpretação sobre o comportamento funcional dos componentes através da especificação de um domínio de aplicação específico.

As métricas mais utilizadas para estabelecer a qualidade dos sistemas são: coesão, acoplamento e complexidade (DAUD; KADIR, 2014) (CHAHAL; SINGH, 2008). Acoplamento é o grau de dependência entre os módulos do sistema. A coesão é reflexo da modelagem dos módulos do sistema. Sistemas que distribuem corretamente as responsabilidades entre os módulos são considerados coesos. A complexidade é derivada do acoplamento e coesão, e pode ser entendida como a dificuldade para compreender e manter um sistema (ZUSE, 1991). Essas métricas são utilizadas nas propostas de Tahir e Macdonell (2012) e Daud e Kadir (2014). Tahir e Macdonell (2012) investigam as características relacionadas a métricas dinâmicas, ou seja, métricas aplicadas durante a

execução de um sistema (CORNELISSEN et al., 2009). Destacam que a complexidade e manutenibilidade de aplicações são os aspectos mais explorados pelas métricas dinâmicas, enquanto que aspectos como testabilidade e reusabilidade recebem pouca atenção. Daud e Kadir (2014) classificam aplicações desenvolvidas com arquitetura SOA. Destacam que a modularidade e baixo acoplamento assumem papel importante em aplicações SOA, pois essas aplicações estabelecem contratos de serviço e comportamento, fato que as diferencia dos demais tipos de sistemas. Cita que o processo de desenvolvimento pode ser divido em quatro fases: requisitos, design, implementação e manutenção. Entretanto, a maior parte das métricas estabelecidas para SOA concentra-se nas fases de implementação e design.

### 5.2.2 Dimensão Design

A *dimensão Design* tem o foco principal em características estruturais da Web API. A *dimensão Design* é baseada em *Richardson Maturity Model* e está dividida em quatro níveis de maturidade. Essa dimensão corresponde à forma como as informações são organizadas. A forma de organização das informações é importante para estabelecer os mecanismos de comunicação entre a Web API e seus clientes. De acordo com a organização das informações e a forma de comunicação é possível estabelecer um nível de confiança entre as partes.

O nível de menor maturidade é atribuído ao *design* RPC, devido ao seu caráter fortemente baseado em serviços e pela discordância com os princípios REST. Implementações modeladas com RPC oferecem o menor nível de confiança para a troca de informações. Os serviços disponíveis devem estar devidamente documentados com detalhes de execução e o significado de cada serviço. Com isso, as implementações com o *design* RPC exigem uma semântica explícita e completa sobre seu funcionamento.

O nível de maturidade *Recursos* é atribuído a implementações que organizam e distribuem as informações em uma unidade coesa, denominada recurso. Agrupar informações na forma de recursos facilita a manipulação da informação. Possibilita definir um conjunto de operações responsáveis pelo gerenciamentos dos dados do recurso. A partir desse nível, as Web APIs devem possuir comportamento *stateless* e os recursos devem ser manipulados através de representações.

*Semântica de Protocolo* é o nível de maturidade que respeita as diretrizes do protocolo utilizado pela Web API. Apesar da *dimen-*

*são Design* abordar principalmente as características estruturais do sistema, o nível *Semântica de Protocolo* aborda aspectos comportamentais, que buscam garantir padrões de comunicação. Além de organizar a informação no formato de recursos, esse nível deve utilizar corretamente os métodos (GET, PUT, POST, DELETE, etc) e os códigos de resposta (200, 404, 303, etc) do protocolo HTTP, tornando as mensagens auto-descritivas. A partir desse nível pode-se considerar que os componentes que constituem o sistema distribuído utilizam uma interface uniforme. A interface uniforme estabelece um nível de confiança mais elevado que os níveis estabelecidos anteriormente. As aplicações clientes têm muito mais conhecimento sobre os mecanismos de comunicação, e podem interagir com a Web API com mais segurança.

*Representações Atômicas* é o *design* que recebe o maior nível de maturidade. Nesse nível, as informações devem estar organizadas na forma de recursos, respeitar a semântica do protocolo e seguir a modelagem de *Representações Atômicas* apresentada na seção 5.1.3. A restrição imposta por essa modelagem implica que os métodos HTTP são aplicados sobre todas as propriedades do recurso. Outra implicação é que o recurso possui apenas uma operação para cada método HTTP. Na perspectiva do protocolo, essa restrição remove qualquer eventual ambiguidade sobre as operações. As restrições impostas pelo *design Representações Atômicas* estabelecem um nível de confiança ainda mais elevado que o oferecido pelo *design Semântica de Protocolo*. Sem as eventuais ambiguidades de operações e a garantia que a execução das operações se aplica a todas as propriedades do recurso, as aplicações clientes podem interagir com muito mais segurança com a Web API.

### 5.2.3 Dimensão Profile

A *dimensão Profile* é baseada no modelo de maturidade CoHA, e possui dois níveis de maturidade que podem ser utilizados em conjunto com as demais dimensões (*Design* e *Semântica*). Um *profile* é responsável pela disponibilização dos detalhes estruturais e comportamentais dos recursos manipulados por Web APIs. O modelo de maturidade $WS^3$ considera somente *profiles* que podem ser interpretados por agentes de software. Essa restrição reforça que a documentação da Web API deve ser utilizada para guiar as aplicações clientes durante a interação, restringindo o uso do *profile* como ferramenta de modelagem. Auto-documentação é um dos critérios apresentados por Meskens (1996) que está diretamente relacionada aos três fatores de qualidade

(flexibilidade, manutenibilidade e testabilidade). O *profile* é uma forma de auto-documentação, com isso, Web APIs que possuem níveis de maturidade associados à *dimensão Profile* possuem essa característica de qualidade.

O nível *Profile de Protocolo* é atribuído a Web APIs que disponibilizam a documentação referente à semântica do protocolo utilizado. Esse nível de maturidade exige que todas as operações sejam descritas e os detalhes de execução sejam apresentados. Dentre os detalhes de execução estão: método HTTP, URI, formato de dados, propriedades obrigatórias, cabeçalhos, etc. Web APIs que disponibilizam um *Profile de Protocolo* proporcionam menor grau de acoplamento, pois é possível realizar a comunicação sem depender de detalhes de implementação.

O nível de maturidade *Profile de Domínio* permite que a documentação descreva os detalhes específicos de um domínio. Nesse nível de maturidade é possível estabelecer um protocolo específico da aplicação, capaz de representar as características de um domínio. Como descrito por Robinson (2011) em *Domain Application Protocol*, permite adotar um formato de dados personalizado, constituído por nomes de variáveis e estruturas próprias de um domínio. Outra possibilidade é definir detalhes de processos específicos como: a ordem de execução de operações, pré-condições, pós-condições, etc. O *Profile de Domínio* também pode ser utilizado para documentar a orquestração de operações baseada em representações apresentada na seção 5.1.5.

### 5.2.4 Dimensão Semântica

A *dimensão Semântica* é o grande diferencial do modelo de maturidade $WS^3$. Está dividida em dois níveis de maturidade que podem ser combinados com as demais dimensões (*Design* e *Profile*). Apesar de *Richardson Maturity Model* e CoHA oferecem mecanismos para classificar Web APIs, não consideram a descrição semântica dos recursos. A descrição semântica de recursos possibilita um elevado nível de confiança na comunicação entre clientes e Web APIs. Um dos resultados da utilização da descrição semântica de recursos é a redução do acoplamento, que passa a ser conceitual, dependendo apenas das convenções estabelecidas pela descrição semântica. O acoplamento conceitual permite modificações na organização dos recursos, desde que o conteúdo e sua respectiva descrição semântica sejam mantidos.

Heitmann et al. (2012) destacam que a adoção da Web Semântica na implementação de sistemas está em expansão. Simplificar a

integração foi identificado como um dos principais motivos para o uso da Web Semântica. Apesar do aumento da adoção de tecnologias semânticas, grande parte das integrações exigem a intervenção humana. Heitmann et al. (2012) apontam que um dos desafios é integrar diferentes fontes de dados heterogêneos. A Web Semântica pode ser utilizada como uma solução para esse tipo de integração. Diante desses fatos, a *dimensão Semântica* do modelo e maturidade $WS^3$ é peça fundamental no processo de classificação e avaliação de Web APIs.

Apesar da *dimensão Semântica* poder ser combinada com as demais dimensões, a descrição semântica é melhor realizada em implementações que possuem níveis de maturidade mais elevados. No caso da descrição semântica de recursos, a Web API deve possuir o nível de maturidade *Recursos* da *dimensão Design*. Embora seja possível utilizar a descrição semântica em modelagens RPC, a combinação entre as dimensões *Design* e *Semântica* fica limitada e pouco natural.

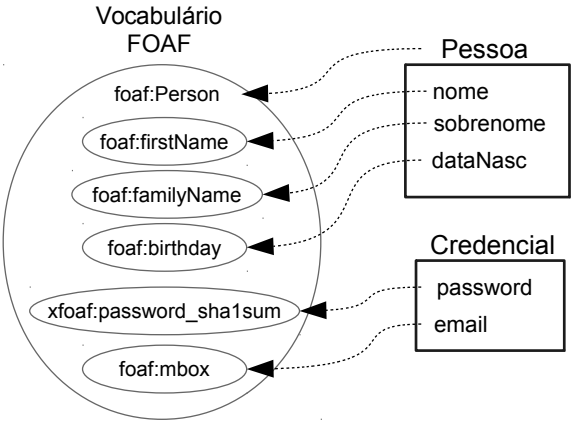


Figura 47 – Exemplo de descrição semântica

O primeiro nível de maturidade da *dimensão Semântica* é denominado *Descrição Semântica*. Esse nível é associado a implementações que manipulam recursos que possuem propriedades e operações descritas semanticamente (Figura 47). A descrição semântica pode ser realizada de diferentes formas, permitindo a manipulação de recursos em diferentes formatos de dados. A forma mais tradicional de descrição semântica é a manipulação de representações que organizam as informações em triplas RDF no formato XML. Entretanto, esse nível

de maturidade permite utilizar representações em JSON-LD ou mesmo documentos HTML enriquecidos com marcações RDFa. A descrição semântica pode ser realizada através de ontologias e OWL ou com a utilização de vocabulários controlados. Web APIs que alcançam o nível de maturidade de *Descrição Semântica* possuem o privilégio de evoluir sem causar grandes impactos em seus clientes. Entretanto, descrever semanticamente as propriedades e operações de recursos tem um custo considerável, pois exige grande esforço de modelagem e desenvolvimento.

O uso de *Linked Data* coloca a Web API no nível mais elevado de maturidade da *dimensão Semântica*. Esse nível de maturidade é importante pois possibilita o compartilhamento e o reuso da informação em larga escala. Nesse nível de maturidade, além de descrever semanticamente os recursos, as Web APIs devem aplicar os princípios de *Linked Data*. Dentre os princípios exigidos estão: usar URIs HTTP como identificador de recursos e estabelecer a semântica entre recursos relacionados, em concordância com os princípios definidos por Bizer, Heath e Berners-Lee (2009). Embora a descrição semântica dos recursos e seus relacionamentos possa ser disponibilizada pela documentação da Web API, *Liked Data* é melhor implementado através de controles hipermídia, pois permite que as aplicações clientes obtenham mais informações sobre os recursos de forma exploratória e sob demanda.

A Figura 48 mostra um exemplo de utilização de *Linked Data*. Neste exemplo, três recursos são descritos semanticamente por dois vocabulários distintos que possuem conceitos equivalentes. Quanto maior a quantidade de vocabulários associados aos recursos, maiores são as chances de as aplicações clientes interpretarem corretamente as informações. O poder de interpretação das informações é ainda maior quando os vocabulários são compartilhados publicamente por diversas outras aplicações, como é o caso dos vocabulários *FOAF*, *Schema.org*, *DublinCore*, *DBPedia*, dentre outros. Entretanto, é possível utilizar vocabulários proprietários para descrever um domínio específico ou não coberto completamento por vocabulários mais populares. Além da descrição semântica das propriedades, o exemplo mostra a descrição semântica dos relacionamentos entre os recursos. Da mesma forma que os termos semânticos, os tipos de relacionamentos são definidos por vocabulários controlados. É possível observar que *Uma pessoa* conhece *Outra pessoa*, que por sua vez é autora de *Um livro*. A utilização de *Linked Data* permite a realização de inferências que resultam em um maior nível de reuso da informação. Por exemplo, é possível deduzir que *Uma pessoa* conhece um escritor. O nível de maturidade *Linked*

*Data* não estabelece um formato padrão de dados, entretanto Web APIs podem utilizar tecnologias padronizadas, como por exemplo, RDF ou JSON-LD, como mostra a Listagem 16.

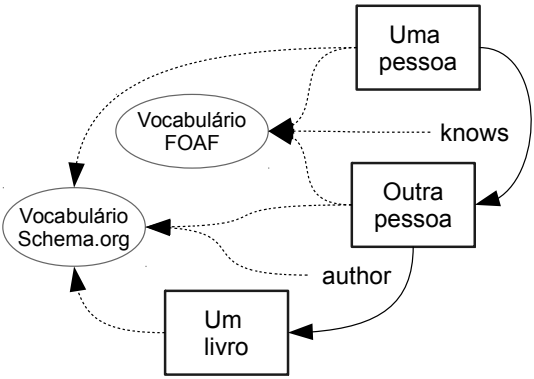


Figura 48 – Exemplo de Linked Data

Listagem 16 – Exemplo de representação em JSON-LD

```
{
  "@context": {
    "foaf": "http://xmlns.com/foaf/0.1/",
    "schema": "http://schema.org/"
  },
  "@type": ["foaf:Person", "schema:Person"],
  "foaf:firstName": "Uma",
  "foaf:familyName": "pessoa",
  "foaf:knows": {
    "@type": ["foaf:Person", "schema:Person"],
    "foaf:firstName": "Outra",
    "foaf:familyName": "pessoa",
    "schema:author": {
      "@type": "schema:Book",
      "schema:name": "Um livro"
    }
  }
}
```

### 5.2.5 Composição e Seleção de Web APIs

Segundo Chahal e Singh (2008) a qualidade é medida pela satisfação dos usuários, através de atributos externos do sistema. Dentre os atributos externos destacam-se: usabilidade, integridade, manute-

nibilidade, confiança, etc. Entretanto, os atributos internos (coesão, acoplamento, reusabilidade) são importantes para a equipe de desenvolvimento. O modelo de maturidade $WS^3$ considera as aplicações clientes da Web API como usuários, e estabelece as características internas e externas da Web API, para atender as expectativas de seus clientes. Para atingir esse objetivo, os fatores, critérios e métricas de qualidade são distribuídos nas três diferentes dimensões do modelo $WS^3$.

O modelo de maturidade $WS^3$ oferece mecanismos de classificação suficientes para o processo de composição e escolha de Web APIs. A Figura 49 mostra um exemplo onde uma aplicação cliente precisa estabelecer uma comunicação com uma determinada Web API. Todas as Web APIs possuem serviços equivalentes, entretanto, implementados de formas diferentes. É necessário que a aplicação cliente escolha uma Web API. Uma forma para facilitar a escolha é utilizar o modelo de maturidade $WS^3$, classificando as Web APIs e identificando os níveis de maturidade atingidos. Numerando os níveis de maturidade de cada dimensão é possível estabelecer uma nomenclatura que representa todas as características de maturidade suportadas pela Web API. Como mostra o exemplo, a *Web API Tradicional* alcança os níveis de maturidade D3-S0-P0, indicando que para *dimensão Design* (D) o nível de maturidade atingido é *Semântica de Protocolo*, e nas dimensões *Semântica* (S) e *Profile* (P) nenhum nível foi alcançado. A partir dos níveis de maturidade informados, a aplicação cliente pode escolher a Web API mais adequada para executar os serviços desejados.

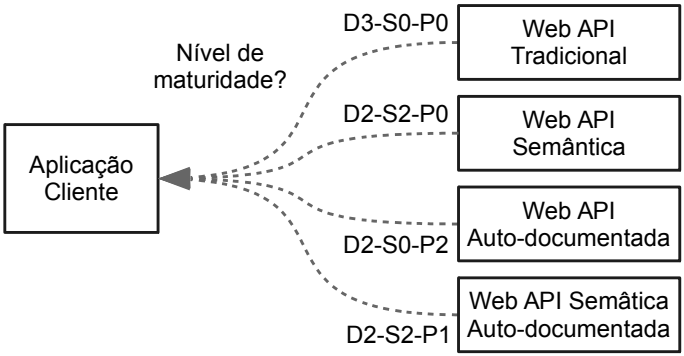


Figura 49 – Escolha e composição de Web APIs

## 5.3 CONSIDERAÇÕES FINAIS

Este capítulo apresentou uma proposta de abordagem para o desenvolvimento de Web APIs RESTful Semânticas. Além de guiar o desenvolvimento, a abordagem apresenta o conceito de *design* de representações atômicas e orquestração de operações baseada em representações. Este capítulo apresentou também o modelo de maturidade $WS^3$, capaz de avaliar implementações de Web APIs que possuem características semânticas. O modelo de maturidade $WS^3$ é formado por três dimensões que consideram o *design*, nível de documentação e suporte semântico.

Apesar da abordagem proposta disponibilizar diretrizes para o desenvolvimento de Web APIs Semânticas, é fundamental a presença de suporte ferramental adequado para a fase de implementação. O capítulo 6 apresenta um *framework* de apoio à implementação de Web APIs que seguem as etapas descritas pela abordagem de desenvolvimento proposta.

# 6 FRAMEWORK JAX-SRS

Este capítulo apresenta o *framework* JAX-SRS (*Java Framework for Semantic RESTful Services*), como uma alternativa para o desenvolvimento de Web APIs semânticas para a linguagem de programação Java. JAX-SRS incorpora uma série de conceitos de modelagem de software que resultam em implementações de serviços Web semânticos alinhados com os princípios arquiteturais REST.

O *framework* JAX-SRS tem como principal objetivo possibilitar o desenvolvimento de Web APIs RESTful com suporte a controles hipermídia e *Linked Data* para representações baseadas em JSON. Com base na especificação JAX-RS, o *framework* proposto oferece suporte ao desenvolvimento através da adição de controles hipermídia e manipulação de representações semânticas. O *framework* JAX-SRS também disponibiliza a documentação da Web API, que descreve as representações dos recursos, suas propriedades e operações suportadas.

## 6.1 CONJUNTO DE ANOTAÇÕES JAX-SRS

A descrição semântica das representações é realizada através de um conjunto de anotações disponibilizado pelo *framework* JAX-SRS (Figura 50). As anotações podem ser aplicadas sobre classes, atributos ou métodos, associando um valor semântico ao elemento anotado. O conjunto de anotações é complementar às anotações JAX-RS e mantém a compatibilidade com essa especificação.

Dentre as anotações que podem ser aplicadas sobre as classes estão: *@SemanticClass*, que associa a classe como uma representação REST; *@Context*, que permite a construção do contexto semântico da representação; *@Vocabulary*, que associa um ou diversos vocabulários ao contexto; e *@SemanticHeaders*, que permite definir cabeçalhos HTTP necessários para a execução das operações declaradas.

As anotações *@SupportedProperty* e *@Link* são aplicadas sobre os atributos da classe e vinculam as propriedades da representação com termos de vocabulários declarados no contexto semântico. A diferença entre as duas anotações é a forma como a propriedade é apresentada; enquanto *@SupportedProperty* sinaliza que o valor da propriedade é parte integrante da representação, *@Link* indica que o valor da propriedade é um *hyperlink*. A anotação *@SupportedCollection* é responsável por definir que a propriedade associada corresponde a uma lista de re-

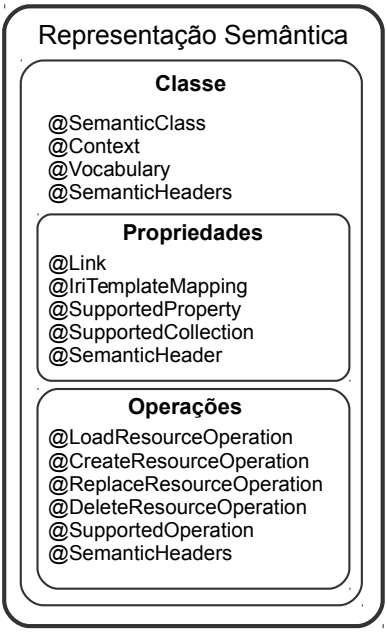


Figura 50 – Conjunto de anotações

cursos. Através da anotação *@IriTemplateMapping* é possível adicionar significado semântico a variáveis presentes em URIs. Da mesma forma que propriedades podem mapear variáveis de URI, propriedades podem também ser associadas a cabeçalhos HTTP. A anotação *@SemanticHeader* permite definir um significado semântico para o mapeamento de cabeçalhos HTTP.

Dentre as anotações que podem ser aplicadas sobre os métodos de classes estão: *@LoadResourceOperation*, *@CreateResourceOperation*, *@ReplaceResourceOperation* e *@DeleteResourceOperation*. Essas anotações, além de possibilitar a vinculação de significado semântico, definem a natureza das operações, que pode ser de carregamento, criação, atualização ou remoção do recurso, respectivamente. A anotação *@SupportedOperation* apenas vincula significado semântico à operação, sem adicionar nenhum significado relativo à sua natureza. Por fim, a anotação *@SemanticHeaders* permite definir os cabeçalhos HTTP necessários para a execução da operação anotada. A Listagem 17 mostra um exemplo de utilização das anotações do *framework* JAX-SRS.

Listagem 17 – Exemplo de utilização de JAX-SRS

```
1  @Path("/pessoa/{id}")
2  @SemanticClass(id="schema:Person")
3  @Context(
4    @Vocabulary(value="schema", url="https://schema.org/"))
5  public class PessoaRepresentation {
6
7      @PathParam("id")
8      @IriTemplateMapping(id="schema:taxID")
9      @SupportedProperty(id="schema:taxID")
10     private Long id;
11
12     @SupportedProperty(id="schema:givenName")
13     private String nome;
14
15     @SupportedProperty(id="schema:familyName")
16     private String sobrenome;
17
18     @SupportedProperty(id="schema:knows")
19     private List<PessoaRepresentation> amigos;
20
21     @Link(id="schema:image")
22     private String imagem;
23
24     @GET
25     @LoadResourceOperation(returnedId="schema:Person")
26     @Produces("application/ld+json")
27     public Response carregar() {
28         //codigo para carregar uma pessoa
29     }
30
31     @POST
32     @CreateResourceOperation(expectedId="schema:Person")
33     @Consumes("application/ld+json")
34     public Response criar(PessoaRepresentation p) {
35         //codigo para criar pessoa p
36     }
37
38     @PUT
39     @ReplaceResourceOperation(expectedId="schema:Person")
40     @Produces("application/ld+json")
41     @Consumes("application/ld+json")
42     public Response atualizar(PessoaRepresentation p) {
43         //codigo para alterar pessoa p
44     }
45
46     @DELETE
47     @DeleteResourceOperation
48     @SemanticHeaders(@SemanticHeader(name = "Authorization",
49                             id = "HTTP BASIC", required = true))
50     public Response remover() {
51         //codigo para remover uma pessoa
52     }
53 }
```

Na linha 2 a representação é associada à classe semântica *Person* do vocabulário *schema.org* através da anotação *@SemanticClass*. Nas linhas 3 e 4 é definido o contexto semântico da representação, onde

o vocabulário *schema.org* é adicionado e associado ao prefixo *schema*. A propriedade *id* (linha 10), que recebe o valor do parâmetro contido no caminho especificado pela anotação *@Path("pessoa/{id}")*, está associada ao termo *taxID* do vocabulário *schema.org* através da anotação *@SupportedProperty* (linha 9). A anotação *@IriTemplateMapping* (linha 8) realiza o mapeamento entre os parâmetros da URI da representação com seus respectivos valores semânticos, permitindo que os agentes possam construir corretamente as URIs dos recursos. As propriedades *nome* e *sobrenome* (linhas 13 e 16) são associadas aos termos *schema:givenName* e *schema:familyName* respectivamente. A propriedade *amigos* (linha 19) contém uma coleção de outras representações que se relacionam através da semântica descrita por *schema:knows*. A propriedade *imagem* (linha 22) é modelada para assumir a forma de um *link* e seu valor está associado ao termo *schema:image*. As operações *carregar, criar e atualizar* (linhas 27, 34 e 42) estão associadas as operações Hydra *LoadResourceOperation, CreateResourceOperation e ReplaceResourceOperation* respectivamente e manipulam representações no formato *application/ld+json*. A operação remover (linha 50) é associada a operação Hydra *DeleteResourceOperation* mas espera que a requisição possua o cabeçalho HTTP denominado *Authorization*.

## 6.2 MÓDULOS DO FRAMEWORK JAX-SRS

O *framework* JAX-SRS é constituído pelo módulo de manipulação de representações JSON-LD (*Provider ld+json*) e também pelo módulo *Gerador de Documentação* da Web API (Figura 51). O *Provider ld+json* é responsável por manipular representações com formato JSON-LD. O módulo *Gerador de Documentação* é responsável por construir o *profile* da Web API nos formatos Hydra e HTML, disponibilizando os detalhes das representações anotadas.

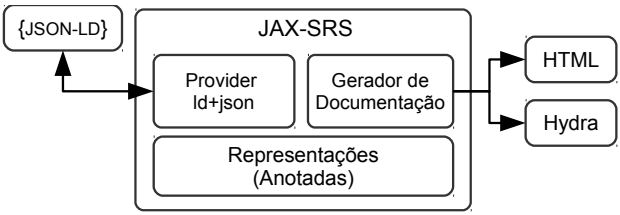


Figura 51 – Módulos JAX-SRS

O *framework* JAX-SRS é uma extensão da especificação JAX-RS, ao qual adiciona a capacidade para manipular representações no formato JSON-LD. A Figura 52 mostra um exemplo de interação entre cliente e Web API envolvendo representações JSON-LD. Na requisição (1) o cliente envia um documento JSON-LD solicitando a criação de um novo recurso. No cabeçalho da requisição HTTP consta o formato da representação, informação necessária para o processamento da operação. Sendo assim, o cliente executa uma requisição HTTP POST informando o cabeçalho *Content-Type: application/ld+json*. A requisição é recebida pelo JAX-RS, que identifica o formato da representação e direciona para um manipulador (*Provider*) adequado. O *framework* JAX-SRS implementa um *Provider* para o formato JSON-LD capaz de realizar a conversão dos documentos enviados pelas aplicações clientes em objetos de representação. A conversão é realizada pelo *Serializador JSON-LD*, que mapeia as propriedades do documento JSON-LD para as propriedades anotadas das classes de representação. O *Provider ld+json* também é capaz de converter representações Java em documentos JSON-LD, disponibilizando representações semânticas para aplicações clientes, como mostra a requisição (2).

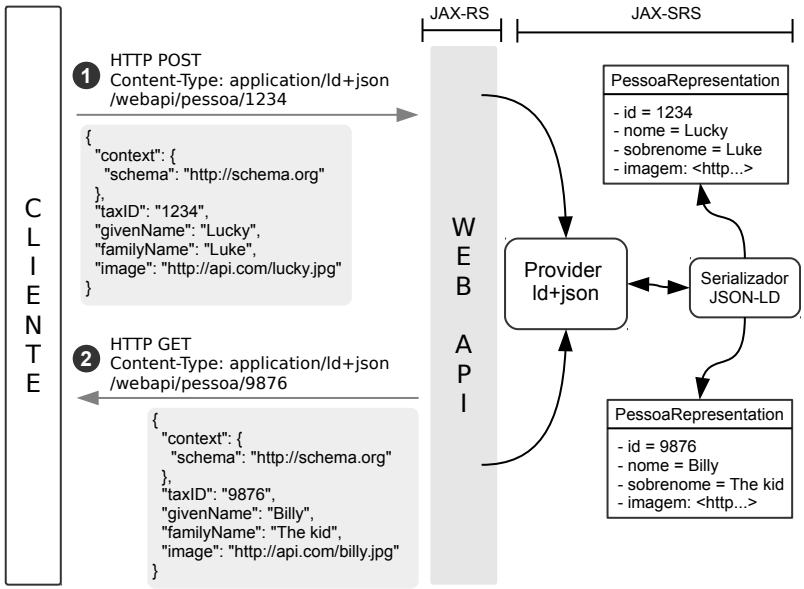


Figura 52 – Provider JSON-LD

O módulo *Gerador de Documentação* é responsável pela construção e disponibilização do *profile* da Web API nos formatos HTML e JSON-LD + Hydra. A documentação é gerada através da identificação das representações anotadas com o *framework* JAX-SRS. O formato HTML é ideal para guiar os desenvolvedores de aplicações clientes, e o formato JSON-LD é adequado para ser interpretado por agentes semânticos.

A Figura 53 (A) mostra um exemplo da documentação no formato JSON-LD gerada pelo *framework*. As representações semânticas são identificadas como *Supported Classes*, que contém os detalhes do contexto semântico, propriedades e operações suportadas. A Figura 53 (B) mostra a mesma documentação no formato HTML, disponibilizando ao desenvolvedor todos os detalhes estruturais e semânticos das representações, além das informações necessárias para a execução das operações.

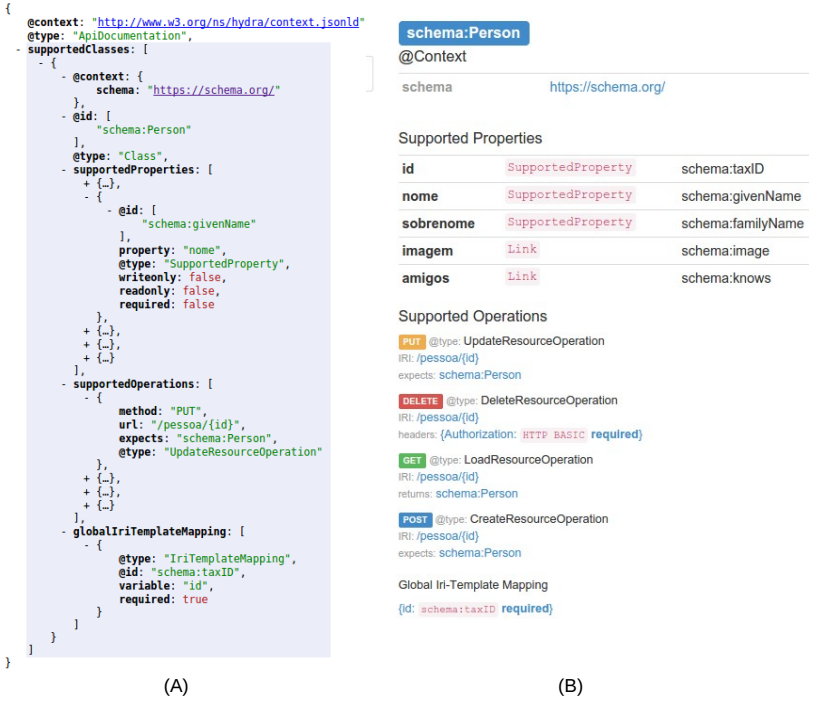


Figura 53 – Documentação gerada pelo JAX-SRS

## 6.3 LIMITAÇÕES

O *framework* JAX-SRS está em constante evolução, e algumas funcionalidades estão implementadas parcialmente, e outras ainda não foram implementadas. O *Serializador JSON-LD* é capaz de realizar o mapeamento entre documentos JSON-LD e representações Java anotadas. Entretanto, o *framework* somente é capaz de converter documentos JSON-LD que possuam representações com a mesma estrutura, ou seja, o documento JSON-LD deve possuir os mesmos atributos que a classe de representação. A versão atual do *framework* JAX-SRS não possui os mecanismos necessários para orientar os agentes semânticos na execução de processos complexos através da orquestração de operações baseada em representações. A única forma disponível para guiar as aplicações clientes é por meio de controles hipermídia presentes nas representações manipuladas pela Web API.

## 6.4 CONSIDERAÇÕES FINAIS

O *framework* proposto possibilita o desenvolvimento de Web APIs mais alinhadas com os princípios arquiteturais REST. Possibilita que as representações contenham dados e controles hipermídia aplicáveis ao atual estado do recurso. Web APIs que manipulam representações com essa característica são denominados *RESTful Hypermedia Driven*, fator fundamental para desenvolvimento de clientes mais independentes da implementação e resistentes a mudanças do servidor, além facilitar a integração com clientes autônomos.

O *framework* JAX-SRS oferece suporte ferramental para o desenvolvimento de Web APIs RESTful com descrição semântica e com controles hipermídia. O principal aspecto explorado é a troca de documentos JSON-LD constituídos não somente por propriedades, mas também por controles hipermídia que permitem que aplicações clientes conheçam as operações disponíveis sobre um determinado recurso Web. A geração automática da documentação da Web API permite que aplicações clientes sejam programadas de forma a explorar com mais autonomia os recursos disponibilizados por Web APIs. Representações com suporte a controles hipermídia, relacionamentos entre recursos através de links e descrição semântica são alguns passos rumo a implementações de Web APIs verdadeiramente RESTful.

# 7 ESTUDO DE CASO

Este capítulo apresenta um estudo de caso com o objetivo de aplicar a abordagem de desenvolvimento proposta e o suporte ferramental oferecido pelo *framework* JAX-SRS. O estudo de caso é constituído por duas Web APIs e três aplicações clientes, como mostra a Figura 54. O estudo de caso busca transformar o site de pessoas procuradas pela Polícia Civil do Estado do Rio Grande do Sul e pelo FBI (*Federal Bureau of Investigation*) em Web APIs RESTful Semânticas. As Web APIs são modeladas de acordo com a abordagem de desenvolvimento proposta e implementadas com o *framework* JAX-SRS. Um vocabulário específico para o domínio das Web APIs foi desenvolvido para servir de base conceitual para a descrição semântica das representações.

Cada Web API possui um cliente implementado de forma tradicional, incapaz de interpretar a semântica das representações. O estudo de caso também apresenta a implementação de um agente semântico capaz de interagir com as duas Web APIs. Este agente semântico interpreta a semântica das representações com base no domínio de aplicação descrito pelo vocabulário compartilhado. O agente semântico também é capaz de interpretar a documentação Hydra e executar as operações em busca de um objetivo previamente programado.

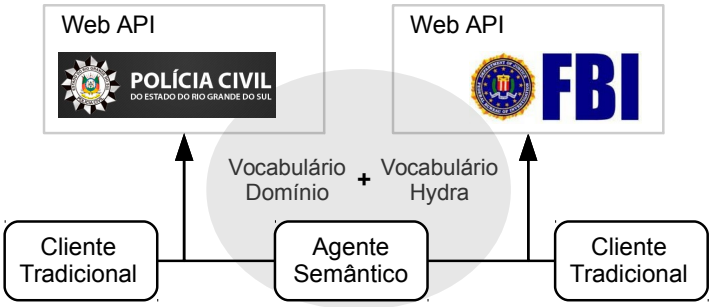


Figura 54 – Visão geral do estudo de caso

## 7.1 VOCABULÁRIO DE DOMÍNIO

A descrição semântica dos dados envolve a associação das propriedades das representações com termos de vocabulários. Para permitir

a descrição semântica adequada, um vocabulário foi desenvolvido para representar o domínio de pessoas procuradas por agências policiais. O vocabulário é disponibilizado através de páginas HTML, e explica em detalhes o significado e o tipo de dados de cada propriedade.

O vocabulário é constituído por três classes semânticas: *Person*, *Criminal* e *Missing*. A classe semântica *Person* (Tabela 1) descreve o significado das propriedades de uma pessoa procurada, porém sem diferenciar entre uma pessoa desaparecida e um criminoso. A classe semântica *Criminal* (Tabela 2) descreve o significado das propriedades específicas de um criminoso, enquanto a classe *Missing* (Tabela 3) descreve o significado das propriedades específicas de uma pessoa desaparecida. Ambas as classes herdam as propriedades da classe semântica *Person*.

Tabela 1 – Classe semântica Person

| **Propriedade** | **Tipo** | **Descrição** |
|---|---|---|
| givenName | Text | The first name of a Person |
| familyName | Text | The last name of a Person |
| image | URL | URL of an image |
| gender | Text | Sex of a Person |
| birthDate | Date | Date of birth |
| placeOfBirth | Text | Place of birth |
| nationality | Text | The country a Person was born |
| jobTitle | Text | Person's occupation |
| scarsAndMarks | Text | Scars, marks and tattos |
| alternateName | Text | An alias |
| height | Text | Height |
| weight | Text | Weight |
| hairColor | Text | Person's hair color |
| eyeColor | Text | Person's eyes color |
| race | Text | Race or ethnicity |

O vocabulário pode ser acessado através do seguinte endereço: *"http://ivansalvadori.com.br/wantedpeople-vocab/"*. O endereço deve ser utilizado como identificador de classes semânticas e seus respectivos termos. Por exemplo, para associar a classe semântica *Person* a uma determinada representação, deve-se utilizar o endereço completo *"http://ivansalvadori.com.br/wantedpeople-vocab/Person"*. Para associar o termo *givenName* a uma propriedade, deve-se utilizar o termo *"http://ivansalvadori.com.br/wantedpeople-vocab/givenName"*.

Tabela 2 – Classe semântica Criminal

| **Propriedade** | **Tipo** | **Descrição** |
|---|---|---|
| protocolNumber | Text | The register protocol |
| summary | Text | A short description |
| crime | Text | Crime commited by a criminal |
| reward | Text | Money offered to tips |
| memberOf | Text | Criminal organization |

Tabela 3 – Classe semântica Missing

| **Propriedade** | **Tipo** | **Descrição** |
|---|---|---|
| protocolNumber | Text | The register protocol |
| summary | Text | A short description |
| reward | Text | Money offered to tips |
| missingDate | Date | Missing date |
| lastSeen | Text | Local last seen |

## 7.2 WEB API - POLÍCIA CIVIL-RS

Esta seção apresenta uma iniciativa para transformar o site de pessoas procuradas pela Polícia Civil do Estado do Rio Grande do Sul em uma Web API RESTful Semântica. A Web API é projetada de acordo com a abordagem de desenvolvimento proposta e implementada com o *framework* JAX-SRS. O levantamento dos requisitos foi realizado através das informações disponibilizadas no site[1] (Figura 55).

Com base na análise das informações disponibilizadas no site, foi elaborada a modelagem do domínio da aplicação. De acordo com a modelagem do domínio, foi aplicada a abordagem de desenvolvimento proposta. O primeiro passo realizado foi a identificação das representações, passo que resulta na separação da camada de integração das demais camadas do sistema. O segundo passo é a descrição semântica das representações identificadas. A descrição semântica é realizada através da associação entre representações e o vocabulário definido na seção 7.1. O terceiro passo, *Design de Representações Atômicas*, deve ser aplicado no momento da modelagem das operações, garantindo que nenhuma propriedade seja manipulada de forma isolada. Por fim, o quarto passo é a criação da documentação da Web API.

[1]Imagem capturada em 30 de Dezembro de 2014 no site oficial da agência - http://www.pc.rs.gov.br/lista/496/procurados


Figura 55 – Site de pessoas procuradas pela Polícia Civil-RS

### 7.2.1 Modelagem do Domínio da Aplicação

O domínio da aplicação é representado pelo do diagrama de classe conceitual ilustrado pela Figura 56. A classe abstrata *Procurado* contém as informações principais de uma pessoa procurada. *Desaparecido* e *Criminoso* herdam as propriedades da classe *Procurado*, e representam uma pessoa desaparecida e um criminoso, respectivamente.

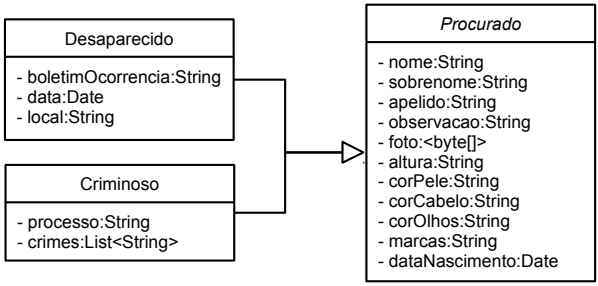


Figura 56 – Diagrama de classe conceitual - Polícia Civil-RS

### 7.2.2 Identificação das Representações

A Figura 57 mostra o diagrama de classe da camada de integração. Com uma modelagem diferente do domínio da aplicação, as classes *DesaparecidoRep* e *CriminosoRep* agrupam as propriedades das classes de domínio *Desaparecido* e *Criminoso* sem o uso de herança. As classes *ListaDesaparecidosRep* e *ListaCriminososRep* constituem representações que agrupam outras representações, através do padrão *Collection* de representações.

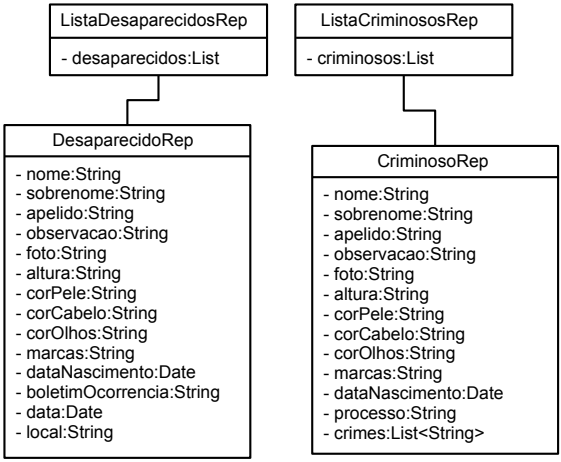


Figura 57 – Diagrama de classe das representações - Polícia Civil-RS

### 7.2.3 Descrição Semântica de Dados

A descrição semântica de dados é a etapa fundamental para possibilitar a correta interpretação e reutilização das informações disponibilizadas pela Web API. Nessa etapa, as representações são associadas à classes semânticas e as propriedades são associadas a termos definidos pelo vocabulário de domínio da aplicação.

A Figura 58 mostra a descrição semântica das representações identificadas. Apesar da Web API possuir quatro representações, apenas as representações *DesaparecidoRep* e *CriminosoRep* são descritas semanticamente. As representações modeladas em forma de coleções (*ListaDesaparecidosRep* e *ListaCriminososRep*) não possuem signifi-

cado associado e são manipuladas através da semântica de operações Hydra. Para facilitar a descrição semântica, o vocabulário é associado ao prefixo *wanted*. A representação *DesaparecidoRep* é associada à classe semântica *wanted:Missing* e a representação *CriminosoRep* é associada à classe semântica *wanted:Criminal*.

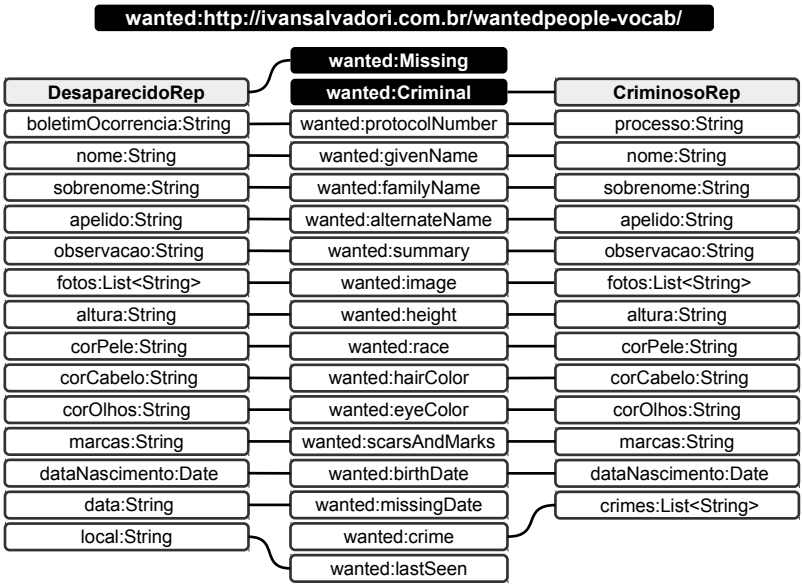


Figura 58 – Descrição semântica dos dados - Polícia Civil-RS

### 7.2.4 Descrição Semântica de Operações

Para descrever semanticamente as operações, primeiramente é preciso identificá-las. Umas das formas mais utilizadas para identificar as operações é através de um fluxograma, ilustrado pela Figura 59. *ListaCriminososRep* é uma coleção de *CriminosoRep* e possui duas operações associadas. A operação GET (1) é responsável pelo carregamento da lista de criminosos. A operação POST (2) é utilizada para cadastrar novos itens. Cada item da coleção possui um *link* para um determinado *CriminosoRep*, representado pela operação GET (3). A alteração ou remoção de um *CriminosoRep* é realizada através das operações PUT (4) e DELETE (5), respectivamente. As representa-

ções *ListaDesaparecidosRep* e *DesaparecidoRep* possuem as operações espelhadas em *ListaCriminososRep* e *CriminosoRep*. As operações são descritas semanticamente com o vocabulário Hydra e os detalhes são apresentados na Tabela 4.

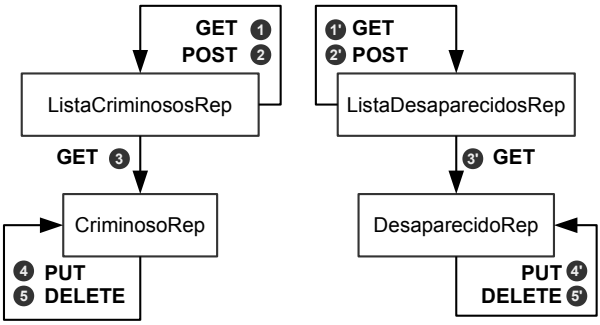


Figura 59 – Fluxograma das operações - Polícia Civil-RS

Tabela 4 – Descrição semântica de operações - Polícia Civil-RS

| **Op** | **Semântica Hydra** | **Retorna** | **Espera** |
|---|---|---|---|
| 1 | LoadResourceOperation | Criminal Collection | - |
| 1' | LoadResourceOperation | Missing Collection | - |
| 2 | CreateResourceOperation | Criminal | Criminal |
| 2' | CreateResourceOperation | Missing | Missing |
| 3 | LoadResourceOperation | Criminal | - |
| 3' | LoadResourceOperation | Missing | - |
| 4 | ReplaceResourceOperation | - | Criminal |
| 4' | ReplaceResourceOperation | - | Missing |
| 5 | DeleteResourceOperation | - | - |
| 5' | DeleteResourceOperation | - | - |

### 7.2.5 Geração da Documentação

A documentação da Web API é resultado direto da implementação da camada de representação. De acordo com a abordagem de desenvolvimento proposta, a documentação utiliza o *profile* Hydra e expõe os detalhes das representações. A documentação da API é gerada automaticamente pelo *framework* JAX-SRS, que disponibiliza o *profile* Hydra nos formatos JSON-LD e HTML.

A Figura 60 mostra a documentação HTML gerada. A documentação apresenta todas as representações implementadas, além dos detalhes da representação *CriminosoRep*. Em *Supported Properties* é apresentada a associação entre as propriedades da representação e os termos do vocabulário de domínio. Em *Supported Operations* são apresentadas as operações que podem ser executas. É possível obter todas as informações necessárias para a execução das operações, como método HTTP, URI, dados de envio e retorno, inclusive os cabeçalhos necessários. O *profile* Hydra no formato JSON-LD está no Apêndice A. O código fonte da Web API está disponível em *http://ivansalvadori.com.br/publicacoes/2015/dissertacaoMestrado/*.

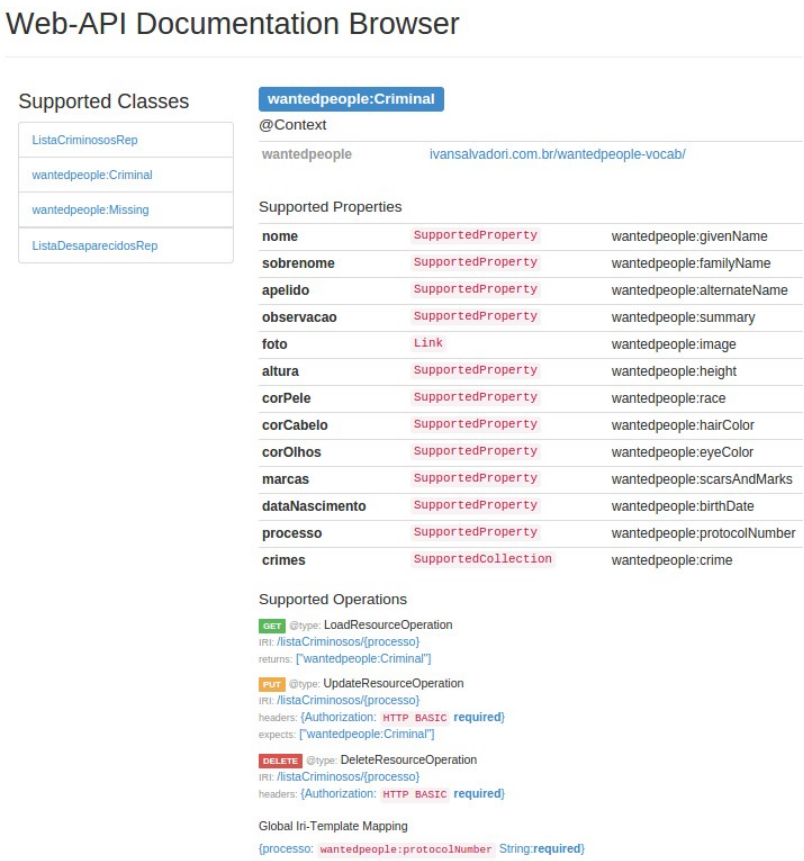


Figura 60 – Documentação HTML da Web API - Polícia Civil-RS

## 7.3 WEB API - FBI

Esta seção tem o objetivo de criar uma Web API RESTful Semântica a partir das informações disponibilizadas no site de pessoas procuradas pelo FBI. O desenvolvimento segue as etapas da abordagem de desenvolvimento proposta, e a implementação é realizada através do *framework* JAX-SRS. A Figura 61 mostra a página oficial de pessoas procuradas pelo FBI [2].


Figura 61 – Site de pessoas procuradas pelo FBI

### 7.3.1 Modelagem do Domínio da Aplicação

Através da análise das informações disponibilizadas pelo site oficial do FBI, foi construído um diagrama de classes conceitual para representar o domínio da aplicação (Figura 62). A classe abstrata *Wanted* contém as informações principais de uma pessoa procurada. A classe

[2] Imagem capturada em 18 de Outubro de 2014 no site oficial do FBI - http://www.fbi.gov/wanted/topten

*Description* contém as informações que descrevem as características físicas, além de dados de origem da pessoa. *Missing* e *Criminal* herdam as propriedades da classe *Wanted*, e representam uma pessoa desaparecida e um criminoso, respectivamente. A classe *CrimeType* é uma enumeração de crimes que podem ser associados a um criminoso.

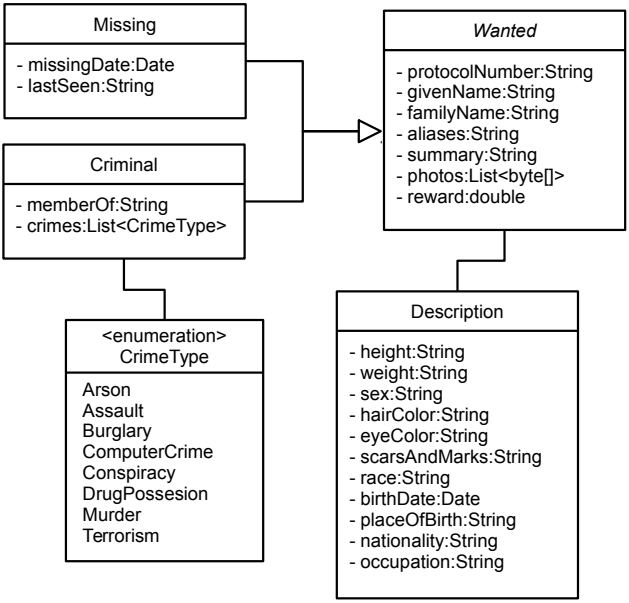


Figura 62 – Diagrama de classe conceitual - FBI

### 7.3.2 Identificação das Representações

A primeira etapa de desenvolvimento é a identificação das representações necessárias para realizar a integração. A Figura 63 mostra o diagrama de classes conceitual da camada de integração. Com uma modelagem diferente do domínio da aplicação, as classes *MissingRep* e *CriminalRep* agrupam as propriedades das classes de domínio *Missing* e *Criminal* sem o uso de herança. As classes *MissingListRep* e *CriminalListRep* constituem representações que agrupam outras representações, através do padrão *Collection* de representações. A classe *CrimeTypeListRep* é modelada como uma coleção que disponibiliza os tipos de crimes através de uma lista de *Strings*.

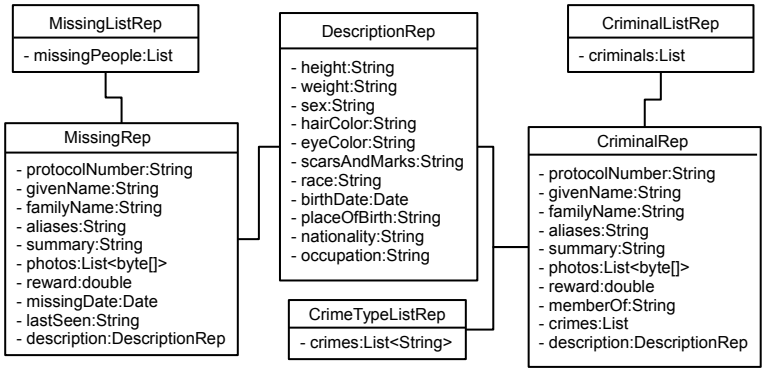


Figura 63 – Diagrama de classe das representações - FBI

### 7.3.3 Descrição Semântica de Dados

A segunda etapa de desenvolvimento é a descrição semântica dos dados e operações. A Figura 64 mostra a associação entre as classes semânticas do vocabulário e as representações. Apenas as propriedades das representações *MissingRep*, *CriminalRep* e *DescriptionRep* são descritas semanticamente pelo vocabulário de domínio. As coleções de representações são descritas com a semântica do vocabulário Hydra.

A representação *MissingRep* é associada à classe semântica *Missing*, enquanto a representação *CriminalRep* é associada à classe semântica *Criminal*. A representação *DescriptionRep* não é associada a nenhuma classe semântica, e suas propriedades fazem parte da classe semântica *Person*. Este é um cenário estabelecido pela abordagem de desenvolvimento proposta, onde uma classe semântica possui termos associados à propriedades de diferentes representações.

### 7.3.4 Descrição Semântica de Operações

A Figura 65 mostra o fluxograma com as operações permitidas sobre as representações identificadas. A representação *CriminalListRep* possui duas operações associadas. A operação GET (1) é responsável pelo carregamento do lista de criminosos. A operação POST (2) é utilizada para cadastrar novas pessoas (*CriminalRep*). A representação *CriminalListRep* possui uma coleção de *CriminalRep*. Cada item da coleção possui o *link* para um criminoso. A partir de *CriminalListRep*

é possível seguir o *link* para um determinado *CriminalRep* através de uma operação GET (3). A alteração ou remoção de um *CriminalRep* é realizada através das operações PUT (4) e DELETE (5), respectivamente. A operação GET (6) é executada para obter a descrição de uma pessoa procurada, que pode ser modificada ou removida através das operações PUT (7) e DELETE (8). *CrimeTypeListRep* permite apenas a consulta dos tipos de crimes através da operação GET (9). *MissingListRep* e *MissingRep* possuem as operações espelhadas em *CriminalListRep* e *CriminalRep*. As operações são associadas aos termos do vocabulário Hydra. Os detalhes da descrição semântica das operações são apresentados na Tabela 5. Com as operações identificadas, pode-se aplicar a terceira etapa que garante a manipulação de representações atômicas.

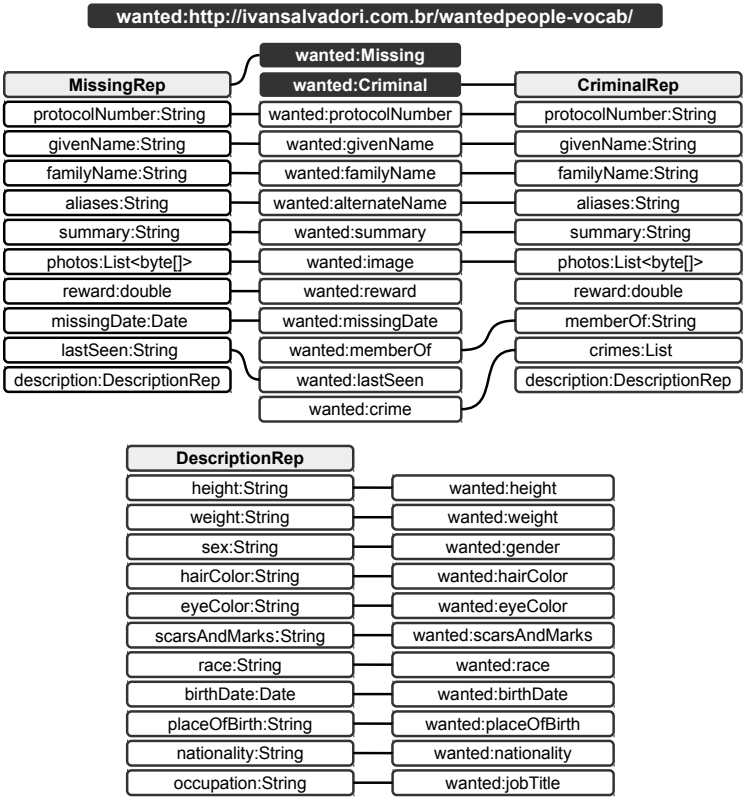


Figura 64 – Descrição semântica dos dados - FBI

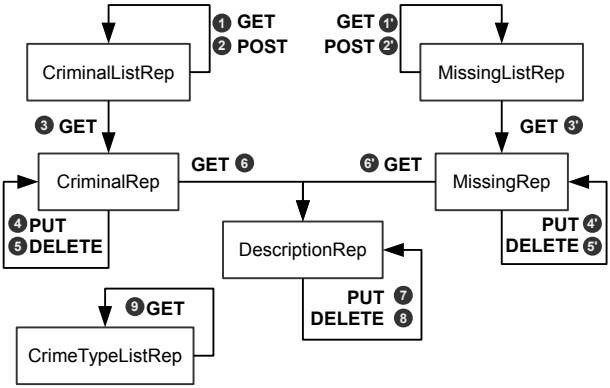


Figura 65 – Fluxograma das operações - FBI

Tabela 5 – Descrição semântica de operações - FBI

| **Op** | **Semântica Hydra** | **Retorna** | **Espera** |
|---|---|---|---|
| 1 | LoadResourceOperation | Criminal Collection | - |
| 1' | LoadResourceOperation | Missing Collection | - |
| 2 | CreateResourceOperation | Criminal | Criminal |
| 2' | CreateResourceOperation | Missing | Missing |
| 3 | LoadResourceOperation | Criminal | - |
| 3' | LoadResourceOperation | Missing | - |
| 4 | ReplaceResourceOperation | - | Criminal |
| 4' | ReplaceResourceOperation | - | Missing |
| 5 | DeleteResourceOperation | - | - |
| 5' | DeleteResourceOperation | - | - |
| 6 | LoadResourceOperation | DescriptionRep | - |
| 6' | LoadResourceOperation | DescriptionRep | - |
| 7 | ReplaceResourceOperation | - | DescriptionRep |
| 8 | DeleteResourceOperation | - | - |
| 9 | LoadResourceOperation | crime Collection | - |

### 7.3.5 Geração da Documentação

A documentação da Web API do FBI é gerada automaticamente pelo *framework* JAX-SRS. A Figura 66 mostra a documentação gerada no formato HTML, com a lista das representações implementadas, além dos detalhes da representação *CriminalRep*. Dentre os detalhes estão: o contexto semântico, as propriedades (*Supported Properties*) e as operações (*Supported Operations*). O *profile* Hydra no formato JSON-LD

está no Apêndice A. O código fonte da Web API está disponível em *http://ivansalvadori.com.br/publicacoes/2015/dissertacaoMestrado/*.

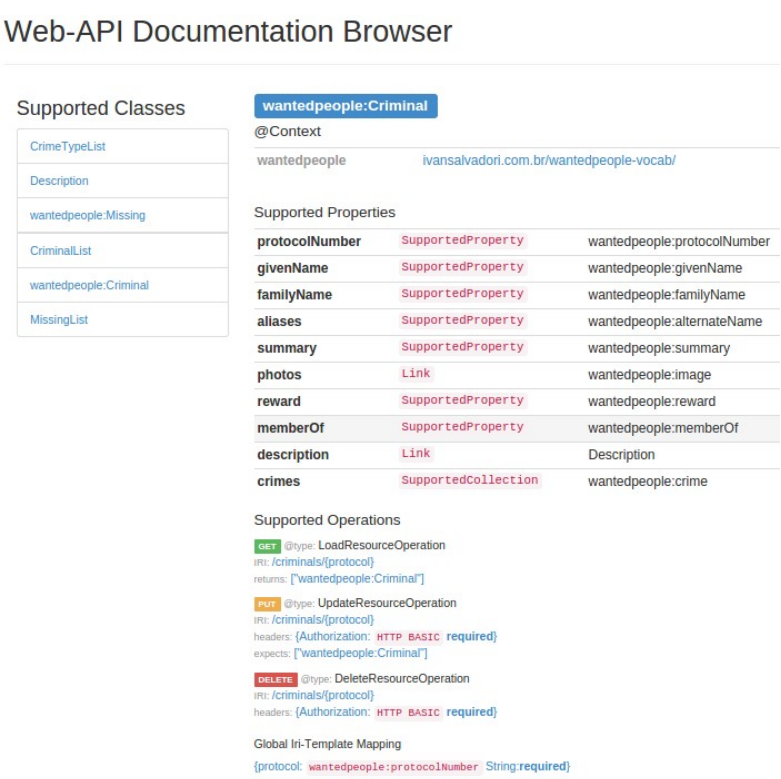


Figura 66 – Documentação HTML da Web API - FBI

## 7.4 APLICAÇÕES CLIENTES

Apesar do desenvolvimento de aplicações clientes estar fora do escopo desse trabalho, o estudo de caso apresenta três exemplos de implementação, para fins de validação da metodologia e do ferramental propostos. Duas aplicações clientes tradicionais foram desenvolvidas, uma para cada Web API. A aplicação cliente tradicional é uma implementação que interage com a Web API sem usufruir das características semânticas oferecidas. Além dos clientes tradicionais, foi desenvolvido

um agente semântico capaz de interagir com as duas Web APIs. Para este estudo de caso, todas as aplicações clientes devem obter a lista de criminosos, em seguida obter todas as informações das pessoas apresentadas na lista. O código fonte das aplicações clientes está disponível em *http://ivansalvadori.com.br/publicacoes/2015/dissertacaoMestrado/*.

As aplicações clientes tracionais foram programadas de forma a conhecer previamente os detalhes da implementação da Web API. Os mecanismos de comunicação implementados no cliente tradicional não são dinâmicos. Isso significa que a estrutura das informações e a forma de execução das operações não podem ser modificadas em tempo de execução. Os clientes tradicionais estão preparados para manipular as representações JSON disponibilizadas pela Web API, entretanto ignoram a descrição semântica. Isso implica que o cliente é capaz de consumir apenas as propriedades previamente conhecidas. As operações também são previamente configuradas com base na implementação da Web API. As URLs, métodos HTTP, variáveis, dados enviados e recebidos são de conhecimento do cliente.

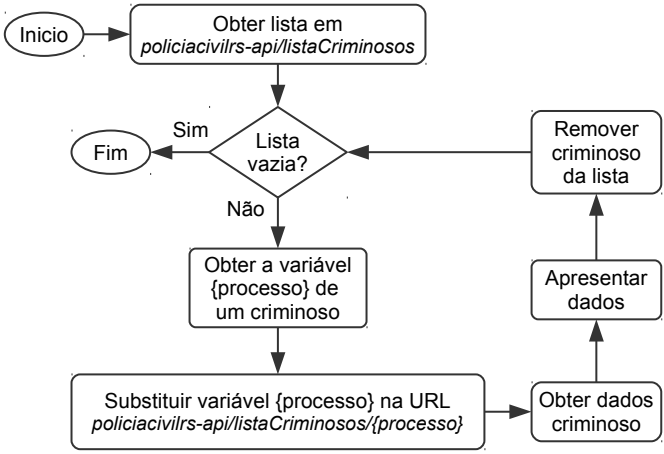


Figura 67 – Fluxo de operações do cliente tradicional - Polícia Civil-RS

O cliente tradicional da Web API da Polícia Civil do Estado do Rio Grande do Sul, cujo fluxo de operações é representado na Figura 67, primeiramente consulta a lista de criminosos através da execução da operação GET aplicada sobre a URL *policiacivilrs-api/listaCriminosos*. O resultado da execução dessa operação é a coleção de criminosos. Cada criminoso contido na lista apresenta o número do processo, que

é utilizado como identificador do item. Para cada item contido na lista, a aplicação cliente executa a consulta da URL *policiacivilrs-api/listaCriminosos/{processo}*, substituindo a variável *processo* pelo número do processo do criminoso. De acordo com a modelagem da Web API, todas as informações do criminoso estão disponíveis nas representação retornadas, finalizando o procedimento.

O cliente tradicional da Web API do FBI, conforme mostra o fluxograma apresentado na Figura 68, obtém a lista de criminosos através da execução da operação GET aplicada sobre a URL *fbi-api/criminals*. Para cada item da lista retornada, duas operações são executadas, pois as informações do criminosos estão distribuídas entre as representações *CriminalRep* e *DescriptionRep*. A primeira operação executada utiliza o método GET na URL *fbi-api/criminals/{protocol}*, substituindo a variável *protocol* pelo valor da variável *protocol* do item da lista. Essa operação resulta nas informações básicas do criminoso. Outras informações são obtidas através da execução da operação GET sobre a URL *fbi-api/criminals/{protocol}/description*, também realizando a substituição da variável *protocol*. Ao término deste laço de repetição, a plicação cliente obtém todas as informações dos criminosos e finaliza o procedimento.

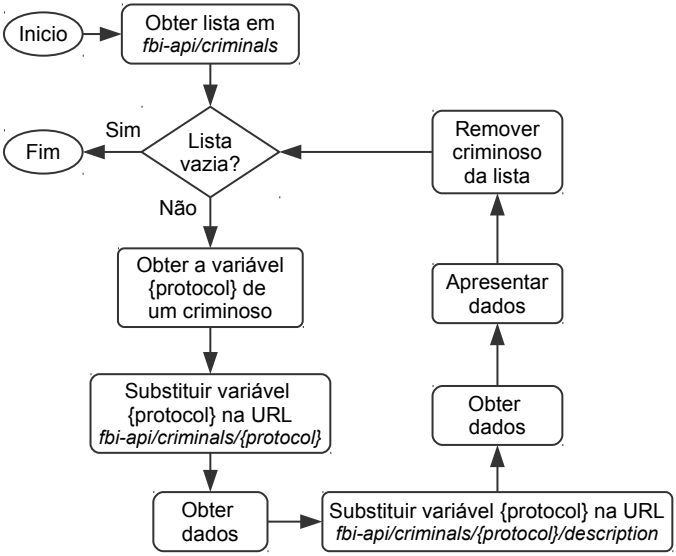


Figura 68 – Fluxo de operações do cliente tradicional - FBI

O agente semântico possui uma abordagem diferenciada. Ele é capaz de interpretar as representações semânticas disponibilizadas por qualquer Web API que utiliza o vocabulário específico do domínio. O agente semântico é capaz de executar operações descritas pelo *profile* Hydra, em busca de um determinado objetivo, como por exemplo, criar, obter, remover ou alterar uma determinada representação.

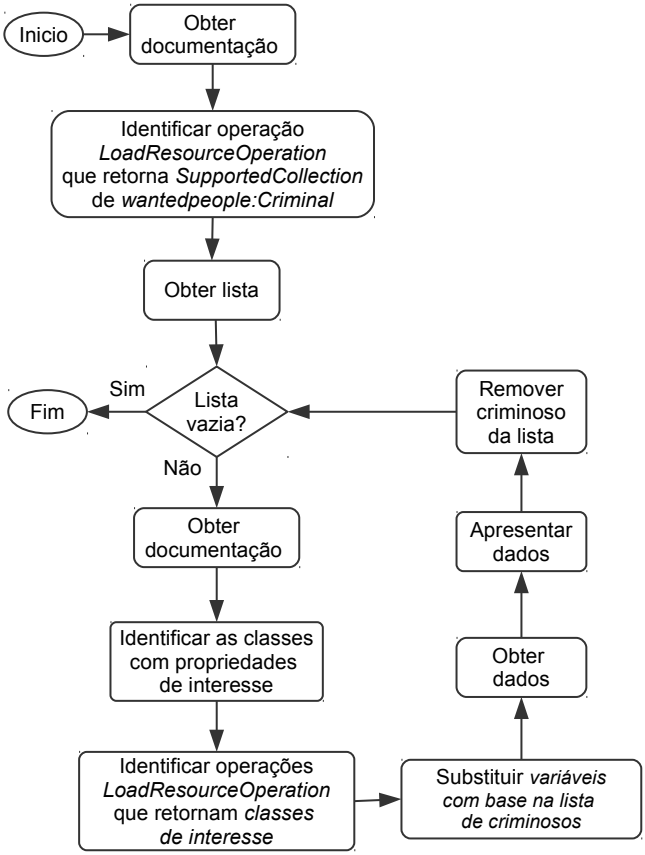


Figura 69 – Fluxo de operações do agente semântico

Neste estudo de caso, o agente semântico, cujo fluxo de operações é especificado na Figura 69, considera a documentação da Web API válida por apenas uma requisição. Ao final da execução de uma operação, a documentação deve ser consultada novamente. Este é um cenário extremamente pessimista, entretanto, representa a execução no

contexto de pior caso. Sendo assim, a primeira ação do agente semântico é obter a documentação da Web API. O próximo passo é obter a lista de criminosos. Para fazê-lo, o agente semântico procura na documentação por uma operação descrita como *LoadResourceOperation* que retorna uma coleção de *Criminal*. Localizada a operação, o agente semântico a executa e mantém o resultado obtido em memória. Para obter as informações dos criminosos, o agente semântico realiza uma busca na documentação pelas propriedades associadas aos termos da classe semântica *Criminal*. A busca tem o objetivo de identificar quais as representações possuem as propriedades desejadas. Após a fase de identificação de representações, o agente semântico localiza as operações do tipo *LoadResourceOperation* que possuem o parâmetro *returns* com as representações de interesse. A fase final é a execução das operações identificadas para todos os itens da lista de criminosos, resolvendo dinamicamente as variáveis presentes nas operações. Ao fim da execução dessas operações, o agente semântico tem acesso a todas as informações dos criminosos e encerra o procedimento.

## 7.5 CONSIDERAÇÕES FINAIS

Este capítulo apresentou um estudo de caso que tem como objetivo aplicar a abordagem de desenvolvimento proposta e o suporte ferramental oferecido pelo *framework* JAX-SRS. O estudo de caso é constituído por duas Web APIs projetadas utilizando a abordagem de desenvolvimento proposta e implementadas com o *framework* JAX-SRS. O estudo de caso apresenta a implementação de três aplicações clientes, dentre elas um agente semântico capaz comunicar-se adequadamente com Web APIs RESTful Semânticas. Os clientes são programados para realizar um conjunto limitado de operações, que envolvem apenas a leitura de informações, entretanto, oferecem a visão necessária para validar as propostas.

# 8 ANÁLISES E VALIDAÇÕES

Este capítulo apresenta as análises e validações para as propostas do trabalho. A validação da abordagem de desenvolvimento é realizada através da execução do estudo de caso apresentado no capítulo 7. As Web APIs do estudo de caso são implementadas com o *framework* JAX-SRS, permitindo a validação do suporte ferramental proposto. Este capítulo apresenta também uma análise comparativa entre o *framework* JAX-SRS e o suporte ferramental apresentado no Estado da Arte (seção 4.3). Os modelos de maturidade RMM e CoHA são analisados e comparados com o modelo de maturidade proposto.

## 8.1 VALIDAÇÃO DA ABORDAGEM DE DESENVOLVIMENTO

Para realizar a validação da abordagem, é necessário que as Web APIs sejam implantadas em um ambiente computacional, de modo a possibilitar a comunicação com as aplicações clientes (Figura 70). O ambiente de execução é constituído por um *Servlet Container Apache Tomcat* versão 7.0.57. As duas Web APIs foram implantadas no mesmo servidor Tomcat, e são executadas simultaneamente. O servidor possui o sistema operacional *Ubuntu Server 14.04 LTS (HVM)*, instalado em uma instância EC2 da nuvem computacional Amazon AWS. A instância escolhida foi a *t2.micro* com processador *Intel Xeon* de alta frequência, operando a 2,5 GHz com Turbo até 3,3 GHz e 1GB de memória principal. As aplicações clientes são executadas fora do ambiente computacional da Amazon.

Um conjunto de dados foi criado para apoiar a execução do estudo de caso. Os dados foram gerados aleatoriamente, com base em um conjunto de nomes, sobrenomes, ocupações, organizações criminosas, características físicas e imagens. Cada Web API possui 100 registros, sendo que cada registro representa uma pessoa procurada. Cada registro é armazenado em um arquivo distinto, e as informações são estruturadas no formato JSON. As imagens utilizadas no conjunto de dados da Web API da Polícia Civil do Estado do Rio Grande do Sul (PC-RS) foram extraídas do site da própria agência e complementadas com imagens do site[1] de pessoas procuras disponibilizado pela Secretaria de Segurança do Rio de Janeiro. As imagens utilizadas nos dados da Web API do FBI foram extraídas do site oficial da agência e ofusca-

[1] http://www.procurados.org.br/

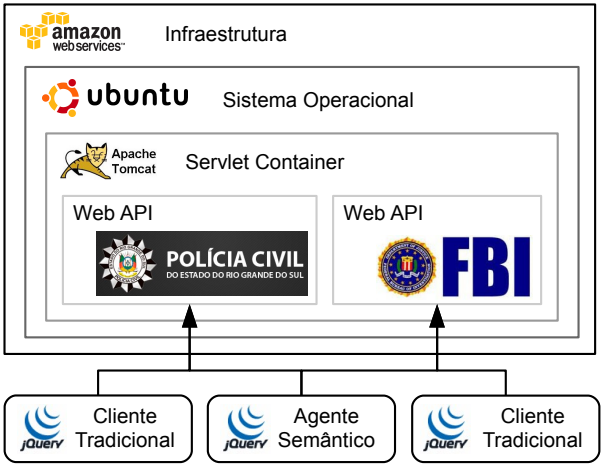


Figura 70 – Visão geral da infraestrutura - estudo de caso

das para preservar a identidade das pessoas. As imagens são codificadas em *BASE64* e armazenadas juntamente com as demais informações da pessoa procurada, no mesmo arquivo JSON. O conjunto de dados da Web API da PC-RS possui 2.284,50 KB, enquanto os dados utilizados pela Web API do FBI possuem 364,70 KB. A diferença de tamanho do conjunto de dados é devido à qualidade das imagens extraídas dos sites. Todos os dados utilizados no experimento podem ser obtidos em *http://ivansalvadori.com.br/publicacoes/2015/dissertacaoMestrado/*.

O próximo passo foi a execução das aplicações clientes. Primeiramente foi executado o cliente tradicional da Web API da PC-RS, resultando em 101 requisições. A primeira requisição obtém a lista de pessoas procuradas. As demais requisições obtêm as informações de cada pessoa presente na lista. Em seguida foi executado o agente semântico sobre a Web API da PC-RS. A execução resultou em 203 requisições, sendo: uma requisição para obter a lista de procurados, 100 requisições para consultar os dados de cada pessoa contida na lista e 102 requisições para consultar a documentação da Web API. O grande número de requisições para a documentação é devido ao agente semântico considerar sua validade por apenas uma requisição.

A execução do cliente tradicional da Web API do FBI resultou em 201 requisições. A primeira requisição é destinada a obter a lista de criminosos. As demais requisições buscam os dados de cada criminoso presente na lista. A Web API do FBI foi modelada para distribuir

as informações dos procurados em duas representações (*CriminalRep* e *DescriptionRep*), sendo assim, exige o dobro de requisições. Por fim é executado o agente semântico sobre a API do FBI. A execução resulta em 403 requisições, sendo: uma requisição para lista de criminosos, 200 requisições para os dados das pessoas e 202 requisições para obter a documentação da Web API.

Cada aplicação cliente foi executada dez vezes, de modo a possibilitar o cálculo de uma média de tempo necessário para a execução das requisições. A Tabela 6 apresenta os dados obtidos pelas sucessivas execuções das aplicações clientes. O cliente tradicional da Web API da PC-RS leva em média 24,08 segundos para executar 101 requisições, enquanto o agente semântico demanda 57,99 segundos para executar 203 requisições. A execução do cliente tradicional da Web API do FBI exige em média 50,51 segundos para executar 201 requisições, enquanto o agente semântico exige 148,73 segundos para executar 403 requisições.

Tabela 6 – Média do tempo(segundos) de requisições em série

| PC-RS Tradicional | PC-RS Semântico | FBI Tradicional | FBI Semântico |
|---|---|---|---|
| 25,94 | 48,47 | 51,64 | 140,56 |
| 21,26 | 47,99 | 49,79 | 143,35 |
| 23,68 | 57,90 | 52,18 | 138,05 |
| 24,64 | 57,50 | 50,07 | 154,25 |
| 23,73 | 56,17 | 51,45 | 137,75 |
| 23,46 | 57,59 | 48,92 | 149,94 |
| 24,71 | 61,00 | 52,98 | 160,74 |
| 28,88 | 58,92 | 48,20 | 159,12 |
| 22,33 | 62,66 | 51,11 | 165,32 |
| 22,19 | 71,74 | 48,77 | 138,20 |
| **24,08** | **57,99** | **50,51** | **148,73** |

Os dados apresentados na Tabela 6 compreendem execuções realizadas sequencialmente. Entretanto, é possível executar as requisições em paralelo, reduzindo drasticamente o tempo de execução. A Tabela 7 apresenta os dados da execução das aplicações clientes utilizando requisições paralelas. Dessa forma, foi possível reduzir o tempo de execução das aplicações clientes em mais de 80%. Para ativar a opção de requisições paralelas, os clientes implementados em *jQuery* foram configurados para utilizar a propriedade *async: true* na configuração *$.ajaxSetup*.

Tabela 7 – Média do tempo(segundos) de requisições em paralelo

| PC-RS Tradicional | PC-RS Semântico | FBI Tradicional | FBI Semântico |
|---|---|---|---|
| 4,17 | 10,16 | 7,40 | 14,24 |
| 4,04 | 8,67 | 7,19 | 14,75 |
| 3,95 | 9,31 | 7,33 | 14,23 |
| 4,01 | 8,67 | 6,78 | 14,31 |
| 4,11 | 9,28 | 7,08 | 14,80 |
| 4,02 | 10,20 | 7,32 | 14,22 |
| 3,91 | 9,34 | 6,94 | 15,04 |
| 4,00 | 8,28 | 7,26 | 14,61 |
| 4,07 | 7,63 | 7,20 | 12,67 |
| 4,03 | 7,45 | 7,11 | 12,83 |
| **4,03** | **8,89** | **7,16** | **14,17** |

A Figura 71 mostra a comparação do tempo de execução das aplicações clientes nas suas diferentes abordagens. A comparação foi realizada com base nos dados obtidos pela execução de requisições em paralelo. É possível observar um aumento de 120,6% no tempo de execução do agente semântico em relação ao cliente tradicional para a Web API da PC-RS. O percentual de aumento do tempo de execução do agente semântico para a Web API do FBI é de 97,9%.

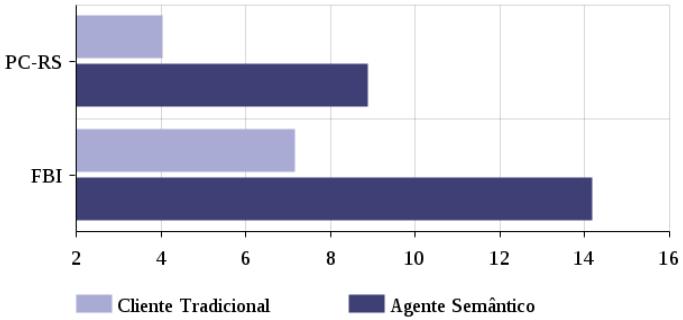


Figura 71 – Tempo de execução - cliente tradicional x semântico

Ao final da execução das aplicações clientes, os dados retornados pelas Web APIs foram coletados e analisados[2]. A Tabela 8 mostra a

[2]Dados de cabeçalhos HTTP não foram considerados.

quantidade de informações recebidas por cada cliente. A quantidade de informações sobre as pessoas procuradas (coluna Dados) é independente do cliente ser tradicional ou semântico, pois as Web APIs sempre retornam representações JSON-LD enriquecidas com descrição semântica e controles hipermídia. O mesmo ocorre com a lista de pessoas procuradas (coluna Lista). A coluna API-DOC mostra a quantidade de dados correspondentes à documentação da Web API, resultante de todas as requisições necessárias para a execução do agente semântico.

Tabela 8 – Informações retornadas pelas Web APIs

| **Cliente** | **Dados** | **Lista** | **API-DOC** | **Total** |
|---|---|---|---|---|
| PC-RS tradicional | 2.335,40 KB | 17,90 KB | 0,00 KB | 2.353,30 KB |
| PC-RS semântico | 2.335,40 KB | 17,90 KB | 599,90 KB | 2.953,20 KB |
| FBI tradicional | 547,10 KB | 16,70 KB | 0,00 KB | 563,80 KB |
| FBI semântico | 547,10 KB | 16,70 KB | 1.704,70 KB | 2.268,50 KB |

É possível notar a grande quantidade de dados correspondentes à documentação da Web API. Os clientes tradicionais não realizam nenhuma requisição para obter a documentação. Entretanto, o agente semântico, pelo menos na perspectiva de pior caso, apresentou grande consumo da documentação. A Figura 72 mostra a comparação da transferência de dados das aplicações clientes nas suas diferentes abordagens. O percentual de aumento dos dados do agente semântico sobre a Web API da PC-RS representa 25,49% em relação ao cliente tradicional. A execução do agente semântico sobre a Web API do FBI resultou em um aumento de 302,36% em relação ao cliente tradicional.

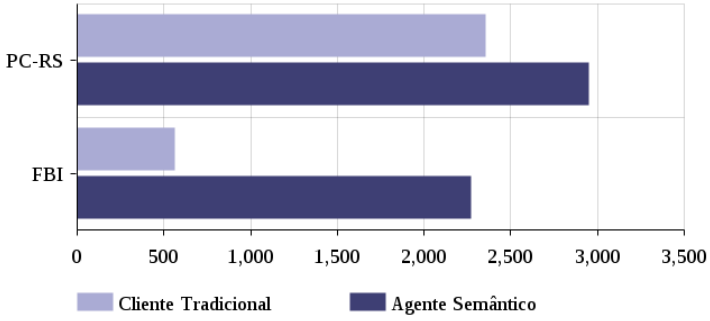


Figura 72 – Transferência de dados - cliente tradicional x semântico

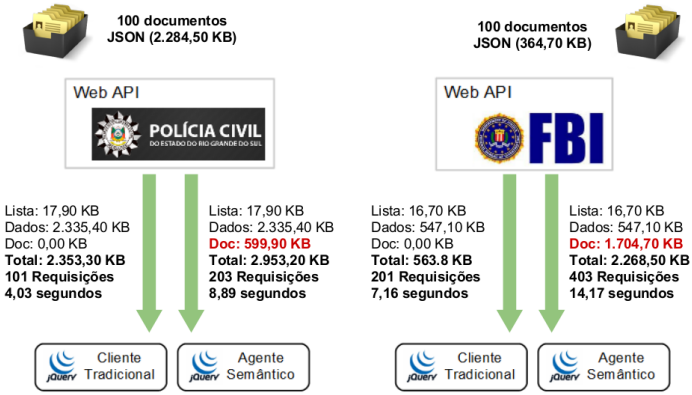


Figura 73 – Visão geral da execução do estudo de caso

A fonte de dados utilizado pelas Web APIs é formada por arquivos no formato JSON. Entretanto, quando esses dados são disponibilizados para as aplicações clientes, a Web API adiciona a descrição semântica e controles hipermídia, resultado em representações JSON-LD. No caso da Web API da PC-RS, o conjunto de dados original é composto por 100 arquivos que ocupam 2.284,50 KB. Ao final da execução do cliente, a Web API disponibilizou 100 representações JSON-LD que somam 2.335,40 KB, representando um acréscimo de 50,90 KB no tamanho dos dados transferidos. O mesmo ocorreu com a Web API do FBI, que originalmente possui 100 arquivos que ocupam 364,70 KB, que resultaram em 200 representações JSON-LD que somam 547,10 KB, totalizando um acréscimo de 182,40 KB. A Figura 73 mostra a visão geral da execução do experimento. As Figuras 74 e 75 mostram o percentual de cada tipo de dado retornado na execução do agente semântico sobre a Web API da PC-RS e FBI, respectivamente.

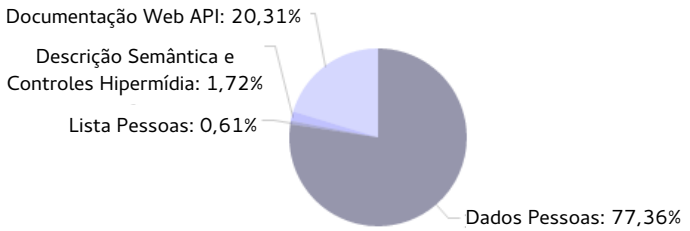


Figura 74 – Distribuição dos dados retornados - Web API da PC-RS

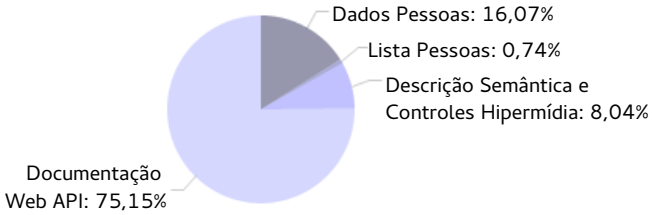


Figura 75 – Distribuição dos dados retornados - Web API do FBI

## 8.2 AVALIAÇÃO DO SUPORTE FERRAMENTAL

A Tabela 9 apresenta uma comparação entre as ferramentas apresentadas no Estado da Arte e o *framework* JAX-SRS. A comparação é realizada através da avaliação das características das ferramentas com base no modelo de maturidade $WS^3$.

Tabela 9 – Comparação entre ferramentas

| Trabalho Relacionado | Dimensão Desing | Dimensão Semântica | Dimensão Profile |
|---|---|---|---|
| Spring-HATEOAS | Todos os níveis | Não suporta | Não suporta |
| Apache Isis | Todos os níveis | Não suporta | Não suporta |
| John e Rajasree | Não se aplica | Descrição | Protocol |
| RestML | Todos os níveis | Não suporta | Protocol |
| Swagger | Não se aplica | Não suporta | Protocol |
| Miredot | Não se aplica | Não suporta | Protocol |
| JAX-SRS | Todos os níveis | Linked Data | Protocol |

Sob o ponto de vista da *dimensão Design* do modelo de maturidade $WS^3$, as ferramentas Spring-HATEOAS, Apache Isis, RestML e JAX-SRS podem atingir todos os níveis de maturidade previstos pelo modelo. Entretanto, John e Rajasree, Swagger e Miredot não podem ser avaliados pela *dimensão Design*, pois são utilizados diretamente na documentação da Web API, etapa posterior ao processo de modelagem.

Sob a perspectiva da *dimensão Semântica*, apenas John e Rajasree e JAX-SRS oferecem algum tipo de suporte semântico. Entretanto, o suporte oferecido por John e Rajasree não corresponde diretamente à implementação da Web API, mas sobre a sua documentação gerada em formato HTML. De posse da documentação, o *framework* realiza ano-

tações semânticas no documento, ampliando o entendimento da Web API para agentes de softwares e desenvolvedores. Por outro lado, o *framework* JAX-SRS atinge o nível de maturidade mais elevado do modelo $WS^3$. Além de permitir a descrição semântica das representações, o *framework* JAX-SRS aplica os princípios de *Linked Data* definidos por Berners-Lee (2006) e Bizer, Heath e Berners-Lee (2009).

Na *dimensão Profile*, apenas os *frameworks* Swagger e JAX-SRS realizam a geração e disponibilização de documentação de forma automática. Embora o *framework* Miredot realize a geração automática da documentação, a disponibilização não é direta, pois é realizada no momento de compilação da Web API e o resultado não é disponibilizado para consulta no contexto da aplicação. Em outras palavras, apenas o desenvolvedor da Web API tem a posse da documentação, enquanto a publicação deve ser feita através de um procedimento manual. A geração da documentação do RestML é baseada na construção de um modelo UML, que pode ser visto como uma forma de documentação da Web API. Entretanto, esse modelo pode apresentar detalhes de implementação que não são do interesse dos desenvolvedores de aplicações clientes. Os *frameworks* Spring-HATEOAS e Apache Isis não possuem mecanismos para geração da documentação da Web API. Nenhuma ferramenta atingiu o nível mais elevado da *dimensão Profile* do modelo de maturidade $WS^3$.

## 8.3 AVALIAÇÃO DO MODELO DE MATURIDADE PROPOSTO

Richardson Maturity Model e CoHA oferecem mecanismos para classificação de Web APIs. Através dos níveis de maturidades, é possível identificar as principais características que devem estar presentes em Web APIs, resultando em um guia de desenvolvimento.

Apesar de ser muito utilizado como referência para qualidade de Web APIs, RMM não descreve em detalhes quais são os passos necessários para alcançar os níveis mais elevados de maturidade. RMM é considerado controverso pois considera poucos atributos de qualidade no processo de classificação (STRAUCH; SCHREIER, 2012). CoHA é uma alternativa mais abrangente, pois aplica-se a sistemas distribuídos baseados em HTTP, e não apenas a implementações REST, e considera aspectos como: performance, visibilidade, custos, efeitos de evolução e manutenção, dentre outros. Por outro lado, categoriza Web APIs REST em um único nível que exige total adequação aos princípios arquiteturais REST.

O modelo de maturidade proposto nesta dissertação adiciona características que não são consideradas, ou são pouco exploradas pelos modelos RMM e CoHA, como o uso de *profile* e Web Semântica em Web APIs. O uso de *profiles* é fundamental para o desenvolvimento de aplicações corporativas que tratam de processos complexos, enquanto a Web Semântica oferece mecanismos que melhoram a interpretação e o reuso de dados. O nível de maturidade *HTTP-based Type I* do modelo CoHA exige o uso de WADL para descrever os recursos disponibilizados por Web APIs. WADL pode ser considerado uma abordagem para implementação de *profile*, entretanto, não é capaz de descrever a ordem ou orquestração de chamadas para realização de processos complexos exigidos por determinados domínios específicos de aplicação. Como apresentado na seção 5.1.5, aplicações complexas geralmente possuem processos complexos, que podem exigir várias requisições envolvendo diferentes recursos. Dessa forma, o modelo de maturidade CoHA utiliza uma linguagem de *profile* incompleta para Web APIs RESTful Semânticas.

Outro aspecto pouco explorado pelos modelos RMM e CoHA é uso de controles hipermídia. Embora seja considerado fundamental para atingir os níveis mais elevados de maturidade, os modelos não apresentam detalhes ou possíveis abordagens de implementação de controles hipermídia. Apenas exigir a presença de controles hipermídia nas representações é uma questão muito vaga. Um dos principais problemas é encontrar um formato de dados adequado para representar os mecanismos de transição de estado da aplicação. Os formatos mais tradicionais como HTML e XML possuem suporte a controles hipermídia, entretanto, não são adequados para todos os tipos de aplicações. Mesmo o formato JSON-LD não possui suporte completo, uma vez que permite apenas representar *links* entre recursos. O modelo de maturidade proposto apresenta uma alternativa de implementação ao relacionar o princípio HATEOAS com o contexto de Web Semântica, com o contexto de *profile*, com a abordagem de orquestração de operações baseada em representações e a utilização de JSON-LD com o vocabulário Hydra.

Na avaliação realizada por Heitmann (HEITMANN et al., 2012), apenas 11% das aplicações que utilizam tecnologias de Web Semântica realizam integrações totalmente automática. Uma vez que o modelo de maturidade $WS^3$ contempla três diferentes dimensões, a documentação da Web API pode ser utilizada em conjunto com as tecnologias semânticas, e aplicadas sobre recursos bem modelados, permitindo a integração de sistemas sem a intervenção humana.

# 9 CONCLUSÕES

Este trabalho apresentou propostas para o desenvolvimento de Web APIs RESTful capazes de manipular representações semânticas de recursos. As propostas abordam questões metodológicas e ferramentais que auxiliam na modelagem e implementação dos sistemas, buscando melhorar a qualidade das Web APIs desenvolvidas. Entretanto, é necessário mecanismos adequados para medir e classificar as implementações. Sendo assim, foi criado um modelo de maturidade capaz de avaliar Web APIs que possuem características semânticas.

## 9.1 CONTRIBUIÇÕES

A principal contribuição deste trabalho é a abordagem para o desenvolvimento de Web APIs RESTful Semânticas. Por meio da abordagem de desenvolvimento proposta foi possível traçar diretrizes que orientam a modelagem de Web APIs, contemplando aspectos de *design* de software, descrição semântica e documentação da camada de integração de dados. As diretrizes de *design* valorizam a separação de camadas e responsabilidades. A descrição semântica de representações permite que as informações sejam interpretadas corretamente. A documentação possibilita aos clientes conhecer os detalhes estruturais dos recursos manipulados pela Web API.

O resultado da utilização da abordagem e do suporte ferramental propostos pode ser visto através do estudo de caso apresentado. As duas Web APIs do estudo de caso foram modeladas e implementadas de formas diferentes, embora manipulem informações do mesmo domínio de aplicação. A descrição semântica permitiu ao agente semântico interpretar corretamente as representações das duas Web APIs. As características semânticas das Web APIs proporcionaram ao agente semântico interpretar corretamente as representações, mesmo com com propriedades e estruturas diferentes. O agente semântico explorou a documentação de modo a executar as operações previamente programadas, embora implementadas de diferentes formas pelas Web APIs. Sendo assim, foi possível reduzir o acoplamento entre Web APIs e aplicações clientes, alcançando um dos objetivos do trabalho.

Outra importante contribuição foi o suporte ferramental oferecido pelo *framework* JAX-SRS, que reduziu o esforço para adicionar informações semânticas aos recursos, contribuindo para o aumento de

produtividade e qualidade no desenvolvimento de Web APIs semânticas. Por meio de anotações é possível descrever semanticamente as representações manipuladas, além de gerar e disponibilizar automaticamente a documentação da Web API. A geração automática da documentação permite que aplicações clientes sejam programadas de forma a explorar com mais autonomia os recursos disponibilizados. Além da descrição semântica, o *framework* JAX-SRS adiciona controles hipermídia nas representações, permitindo que as aplicações clientes tenham as informações suficientes para explorar outros recursos e executar operações desejadas.

Apesar de a abordagem de desenvolvimento proposta, juntamente com a utilização do *framework* JAX-SRS, representar uma iniciativa para o desenvolvimento de Web APIs RESTful Semânticas, existe um acréscimo significativo de complexidade e esforço no desenvolvimento. A adoção da abordagem exige a presença de uma base de conhecimento, com o objetivo de descrever semanticamente as informações manipuladas pela Web API. A implementação realizada com o auxilio do *framework* JAX-SRS exige a utilização do conjunto de anotações sobre as propriedades e operações das classes de representação, com o objetivo de adicionar a semântica necessária para a correta interpretação das informações por parte das aplicações clientes.

O acréscimo de complexidade e de esforço de desenvolvimento, resultantes da adoção da abordagem e do *framework* propostos, podem tornar o desenvolvimento inviável. A adoção das propostas é recomendada no desenvolvimento de Web APIs que são frequentemente modificadas, ou aplicações clientes que não são desenvolvidas pela mesma organização ou equipe responsável pela Web API. Outra possível utilização da abordagem e *framework* propostos ocorre quando diversas Web APIs são combinadas para produzir um resultado comum. Nesse cenário, as implementações das Web APIs podem ser realizadas por diferentes times de desenvolvedores ou organizações. Entretanto, seguindo as diretrizes que constituem a abordagem de desenvolvimento proposta, é possível estabelecer uma integração adequada.

Por fim, o modelo de maturidade proposto é uma contribuição importante para classificar implementações de Web APIs que manipulam representações semânticas. Ao considerar aspectos como *design*, *profile* e *Web Semântica*, o modelo de maturidade proposto nesta dissertação contempla um número maior de características analisadas no processo de classificação, em comparação aos modelos de maturidade estudados.

## 9.2 LIMITAÇÕES E TRABALHOS FUTUROS

A abordagem de desenvolvimento proposta descreve semanticamente apenas a camada de representação, entretanto a descrição semântica pode ser aplicada nas demais camadas do sistema, tornando a aplicação totalmente semântica. Outra limitação da abordagem está relacionada à orquestração de operações baseada em representações, que é apresentada apenas de modo conceitual. Trabalhos futuros podem explorar maneiras de implementar os conceitos de orquestração definidos. Uma alternativa é a extensão do vocabulário Hydra de forma a contemplar os mecanismos de orquestração.

O *framework* JAX-SRS está em um estado funcional, entretanto, melhorias podem ser realizadas. Uma possível melhoria é possibilitar a manipulação de presentações RDF, uma vez que JSON-LD é compatível com este formato. O padrão JSON-LD permite vincular o contexto semântico em outra presentação, entretanto o *framework* JAX-SRS não possui essa funcionalidade. Apesar do suporte ferramental oferecer apoio para a aplicação da abordagem de desenvolvimento proposta, os mecanismos de orquestração não são contemplados pelo *framework*. Melhorias visuais podem ser realizadas na documentação Web para tornar a experiência dos desenvolvedores mais agradável.

O estudo de caso explorou apenas operações de leitura das informações disponibilizadas pelas Web APIs. Embora suficientes para o experimento realizado, a execução de operações que modificam o estado dos recursos oferece oportunidades para novas análises. Trabalhos futuros podem explorar o desenvolvimento de agentes semânticos capazes de interpretar todas as operações Hydra, estabelecendo uma comunicação adequada com diferentes Web API de um determinado domínio, visando alta performance e segurança.

Trabalhos futuros podem aprimorar detalhes sobre a composição de Web APIs. Com a descrição semântica dos recursos Web possibilitada pelo JSON-LD, juntamente com o poder de representação de controles hipermídia proporcionado pelo Hydra, é possível que recursos sejam distribuídos entre diversas Web APIs, exigindo mecanismos de descoberta de recursos. Neste contexto, a classificação da implementação através do modelo de maturidade proposto pode ser utilizada como métrica para estabelecer os níveis de confiança da comunicação no processo de composição de Web APIs.

## 9.3 PUBLICAÇÕES

A pesquisa apresentada neste trabalho resultou nas seguintes publicações:

- A Framework for Semantic Description of RESTful Web APIs. IEEE International Conference on Web Services (ICWS), 2014, Anchorage, USA.
- Desenvolvimento de Web APIs RESTful Semânticas. Linked Open Data Brasil, 2014, Florianópolis.

Além das publicações mencionadas, dois outros artigos ainda estão sendo produzidos a partir dos resultados desta dissertação. O primeiro deles, intitulado "*A Maturity Model for Semantic RESTful Web APIs*", que descreve o modelo de maturidade $WS^3$, foi submetido para publicação em um evento internacional. O segundo artigo consiste em uma compilação de todos os resultados obtidos ao longo do mestrado, e será submetido para publicação em um periódico internacional.

# REFERÊNCIAS

ALGERMISSEN, J. **Classification of HTTP-based APIs**. 2010. Disponível em: <http://nordsc.com/ext/classification_of_http_based_apis.html>.

AMUNDSEN, M. **Collection+JSON**. 02 2013. Disponível em: <http://amundsen.com/media-types/collection/>. Acesso em: 08/12/2013.

BERGMAN, M. K. The deep web: Surfacing hidden value. **Journal of Electronic Publishing**, v. 7, n. 1, 2001. Disponível em: <http://dx.doi.org/10.3998/3336451.0007.104>.

BERNERS-LEE, T. **Linked Data**: Personal view only. imperfect but published. 07 2006. Disponível em: <http://www.w3.org/DesignIssues/LinkedData.htm>. Acesso em: 20/10/2013.

BERNERS-LEE, T.; HENDLER, J.; LASSILA, O. The semantic web. **Scientific American**, v. 284, n. 5, p. 34–43, maio 2001.

BIENVENIDO, D. **Apache Isis: Java Framework for Domain-Driven Design**. 2013. Disponível em: <http://www.infoq.com/news/2013/01/apache-isis-java-domain-driven>. Acesso em: 05/03/2014.

BIZER, C.; HEATH, T.; Berners-Lee, T. Linked data - the story so far. **Int. J. Semantic Web Inf. Syst.**, v. 5, n. 3, p. 1–22, 2009.

CERI, S. et al. **Web Information Retrieval**. [S.l.]: Springer, 2013. (Data-centric systems and applications).

CHAHAL, K. K.; SINGH, H. A metrics based approach to evaluate design of software components. In: **Global Software Engineering, 2008. ICGSE 2008. IEEE International Conference on**. [S.l.: s.n.], 2008. p. 269–272.

CORNELISSEN, B. et al. A systematic survey of program comprehension through dynamic analysis. **Software Engineering, IEEE Transactions on**, v. 35, n. 5, p. 684–702, Sept 2009. ISSN 0098-5589.

CROCKFORD, D. **RFC 4627 - The application/json Media Type for JavaScript Object Notation (JSON)**. [S.l.], 2006. Disponível em: <http://tools.ietf.org/html/rfc4627>.

DAUD, N. N.; KADIR, W. W. Static and dynamic classifications for soa structural attributes metrics. In: **Software Engineering Conference (MySEC), 2014 8th Malaysian**. [S.l.: s.n.], 2014. p. 130–135.

EUZENAT, J. Eight questions about semantic web annotations. **Intelligent Systems, IEEE**, v. 17, n. 2, p. 55–62, March 2002. ISSN 1541-1672.

FERREIRA FILHO, O. F. **Serviços Semânticos: Uma Abordagem RESTful**. Dissertação (Mestrado) — Universidade de São Paulo, 2009.

FIELDING, R. et al. **RFC 2616, Hypertext Transfer Protocol – HTTP/1.1**. [S.l.], 1999. Disponível em: <http://www.rfc.net/rfc2616.html>.

FIELDING, R. T. **REST: Architectural Styles and the Design of Network-based Software Architectures**. Tese (Doctoral dissertation) — University of California, Irvine, 2000. Disponível em: <http://www.ics.uci.edu/ fielding/pubs/dissertation/top.htm>.

FOWLER, M. **Richardson Maturity Model**: Steps toward the glory of rest. 2010. Disponível em: <http://martinfowler.com/articles/richardsonMaturityModel.html>. Acesso em: 16/11/2014.

HAYWOOD, D. **Restful Objects Specification(v1.0.0)**. [S.l.], 2012. Disponível em: <http://restfulobjects.org>.

HEITMANN, B. et al. An empirically grounded conceptual architecture for applications on the web of data. **Trans. Sys. Man Cyber Part C**, IEEE Press, Piscataway, NJ, USA, v. 42, n. 1, p. 51–60, jan. 2012. ISSN 1094-6977. Disponível em: <http://dx.doi.org/10.1109/TSMCC.2011.2145370>.

IETF. **JSON Hyper-Schema: Hypertext definitions for JSON Schema json-schema-hypermedia**. 2013. Disponível em: <http://json-schema.org/latest/json-schema-hypermedia.html>. Acesso em: 03/03/2014.

IETF. **JSON Schema: core definitions and terminology json-schema-core**. 2013. Disponível em: <http://json-schema.org/latest/json-schema-core.html>. Acesso em: 03/03/2014.

JOHN, D.; RAJASREE, M. S. A framework for the description, discovery and composition of restful semantic web services. In: **Proceedings of the Second International Conference on Computational Science, Engineering and Information Technology**. New York, NY, USA: ACM, 2012. (CCSEIT '12), p. 88–93. ISBN 978-1-4503-1310-0. Disponível em: <http://doi.acm.org/10.1145/2393216.2393232>.

JOSUTTIS, N. M. **SOA in Practice: The Art of Distributed System Design**. Beijing: O'Reilly, 2007. ISBN 978-0-596-52955-0.

KASHYAP, V.; BUSSLER, C.; MORAN, M. **The Semantic Web, Semantics for Data and Services on the Web**. Berlin, Heidelberg: Springer-Verlag, 2008. ISBN 978-3-540-76451-9.

KELLY, M. **HAL - Hypertext Application Language**: A lean hypermedia type. 2013. Disponível em: <http://stateless.co/hal_specification.html>. Acesso em: 03/03/2014.

KHARE, R. Microformats: the next (small) thing on the semantic web? **Internet Computing, IEEE**, v. 10, n. 1, p. 68–75, Jan 2006. ISSN 1089-7801.

KHARE, R.; ÇELIK, T. Microformats: A pragmatic path to the semantic web. In: **Proceedings of the 15th International Conference on World Wide Web**. New York, NY, USA: ACM, 2006. (WWW '06), p. 865–866. ISBN 1-59593-323-9. Disponível em: <http://doi.acm.org/10.1145/1135777.1135917>.

KLUSCH, M. Semantic web service description. In: SCHUMACHER, M.; SCHULDT, H.; HELIN, H. (Ed.). **CASCOM: Intelligent Service Coordination in the Semantic Web**. [S.l.]: Birkhäuser Basel, 2008. (Whitestein Series in Software Agent Technologies and Autonomic Computing), p. 31–57. ISBN 978-3-7643-8574-3.

LANTHALER, M. Creating 3rd generation web apis with hydra. In: **Proceedings of the 22Nd International Conference on World Wide Web Companion**. Republic and Canton of Geneva, Switzerland: International World

Wide Web Conferences Steering Committee, 2013. (WWW '13 Companion), p. 35–38. ISBN 978-1-4503-2038-2. Disponível em: <http://dl.acm.org/citation.cfm?id=2487788.2487799>.

LANTHALER, M.; GüTL, C. On using json-ld to create evolvable restful services. In: ALARCóN, R.; PAUTASSO, C.; WILDE, E. (Ed.). **WS-REST**. [S.l.]: ACM, 2012. p. 25–32. ISBN 978-1-4503-1190-8.

MESKENS, N. Software quality analysis system: a new approach. In: **Industrial Electronics, Control, and Instrumentation, 1996., Proceedings of the 1996 IEEE IECON 22nd International Conference on**. [S.l.: s.n.], 1996. v. 3, p. 1406–1411 vol.3.

ORACLE. **JAX-RS: Java API for RESTful Web Services - Version 2.0 Public Review (Second Edition)**. [S.l.], 2012. Disponível em: <https://jcp.org/aboutJava/communityprocess/final/jsr339/index.html>.

QMINO. **Miredot**: Reference manual. 2014. Disponível em: <http://www.miredot.com/docs/manual/>. Acesso em: 08/11/2014.

Reverb Technologies. **The Swagger Specification**. 2014. Disponível em: <https://github.com/wordnik/swagger-spec>. Acesso em: 08/11/2014.

RICHARDSON, L.; AMUNDSEN, M.; RUBY, S. **Restful Web Apis**. Oreilly & Associates Incorporated, 2013. ISBN 9781449358068. Disponível em: <http://books.google.com.br/books?id=i3a7mAEACAAJ>.

ROBINSON, I. **RESTful Domain Application Protocols.** [S.l.]: Springer, 2011. 61-91 p. ISBN 978-1-4419-8302-2.

SANCHEZ, R. V. V.; OLIVEIRA, R. R. d.; FORTES, R. P. d. M. Restml: Modeling restful web services. In: PAUTASSO, C.; WILDE, E.; ALARCON, R. (Ed.). **REST: Advanced Research Topics and Practical Applications**. [S.l.]: Springer New York, 2014. p. 125–143. ISBN 978-1-4614-9298-6.

SCHEMA.ORG. **Getting started with schema.org**. 2011. Disponível em: <http://schema.org/docs/gs.html>. Acesso em: 14/10/2013.

SJOBERG, D. I. K.; DYBA, T.; JORGENSEN, M. The future of empirical methods in software engineering research. In: **2007 Future of Software Engineering**. Washington, DC, USA: IEEE Computer Society, 2007. (FOSE '07), p. 358–378. ISBN 0-7695-2829-5.

SPORNY, M. **What is JSON-LD?** 06 2012. Disponível em: <http://www.youtube.com/watch?v=vioCbTo3C-4>. Acesso em: 22/10/2013.

SPORNY, M. et al. **JSON-LD 1.0 A JSON-based Serialization for Linked Data**. [S.l.], 10 2013. Disponível em: <http://json-ld.org/spec/latest/json-ld/>.

SPRING. **Spring Hateoas - Reference**. 05 2013. Disponível em: <http://projects.spring.io/spring-hateoas/>. Acesso em: 08/01/2014.

STRAUCH, J.; SCHREIER, S. Restify: From rpcs to restful http design. In: **Proceedings of the Third International Workshop on RESTful Design**. New York, NY, USA: ACM, 2012. (WS-REST '12), p. 11–18. ISBN 978-1-4503-1190-8. Disponível em: <http://doi.acm.org/10.1145/2307819.2307824>.

SVENONIUS, E. Unanswered questions in the design of controlled vocabularies. **Journal of the American Society for Information Science**, Wiley Subscription Services, Inc., A Wiley Company, v. 37, n. 5, p. 331–340, 1986. ISSN 1097-4571. Disponível em: <http://dx.doi.org/10.1002/(SICI)1097-4571(198609)37:5<331::AID-ASI8>3.0.CO;2-E>.

TAHIR, A.; MACDONELL, S. A systematic mapping study on dynamic metrics and software quality. In: **Software Maintenance (ICSM), 2012 28th IEEE International Conference on**. [S.l.: s.n.], 2012. p. 326–335. ISSN 1063-6773.

W3C. **RDF Primer**. [S.l.], 2 2004. Disponível em: <http://www.w3.org/TR/rdf-primer/>.

W3C. **Latest Layercake Diagram**. 2007. Disponível em: <http://www.w3.org/2007/03/layerCake-small.png>. Acesso em: 28/10/2013.

W3C. **SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)**. [S.l.], 4 2007. Disponível em: <http://www.w3.org/TR/soap12-part1/>.

W3C. **RDFa 1.1 Primer - Second Edition**: Rich structured data markup for web documents. [S.l.], 8 2013. Disponível em: <http://www.w3.org/TR/rdfa-primer/>.

WEBBER, J.; PARASTATIDIS, S.; ROBINSON, I. **REST in Practice: Hypermedia and Systems Architecture**. [S.l.]: O'Reilly, 2010. ISBN 978-0-596-80582-1.

ZUSE, H. **Software Complexity: Measures and Methods**. Hawthorne, NJ, USA: Walter de Gruyter & Co., 1991. ISBN 0-89925-640-6.

# APÊNDICE A – Documentação Hydra - Estudo de Caso

Listagem 18 – Documentação Hydra - Web API da PC-RS

```
{
  "@context": {
    "hydra":"http://www.w3.org/ns/hydra/context.jsonld",
    "wantedpeople":"ivansalvadori.com.br/wantedpeople-vocab/"
  },
  "@type": "ApiDocumentation",
  "supportedClasses": [{
    "@id": ["wantedpeople:Criminal"],
    "@type": "Class",
    "supportedProperties": [{
      "@id": ["wantedpeople:givenName"],
      "property": "nome",
      "@type": "SupportedProperty"
    }, {
      "@id": ["wantedpeople:familyName"],
      "property": "sobrenome",
      "@type": "SupportedProperty"
    }, {
      "@id": ["wantedpeople:alternateName"],
      "property": "apelido",
      "@type": "SupportedProperty"
    }, {
      "@id": ["wantedpeople:summary"],
      "property": "observacao",
      "@type": "SupportedProperty"
    }, ...],
    "supportedOperations": [{
      "method": "GET",
      "url": "/listaCriminosos/{processo}",
      "returns": ["wantedpeople:Criminal"],
      "@type": "LoadResourceOperation"
    }, ...],
    "globalIriTemplateMapping": [{
      "@type": "IriTemplateMapping",
      "@id": "wantedpeople:protocolNumber",
      "variable": "processo"
    }]
  }, {
    "@id": [
      "ListaCriminososRep"
    ],
    "@type": "Class",
    "supportedProperties": [{
      "@id": ["wantedpeople:Criminal"],
      "property": "criminals",
      "@type": "SupportedCollection"
    }],
    "supportedOperations": [{
      "method": "GET",
      "url": "listaCriminosos",
      "returns": ["ListaCriminososRep"],
      "@type": "LoadResourceOperation"
    }, ...]
  }, ...]
}
```

Listagem 19 – Documentação Hydra - Web API do FBI

```
{
  "@context": {
   "hydra":"http://www.w3.org/ns/hydra/context.jsonld",
   "wantedpeople":"ivansalvadori.com.br/wantedpeople-vocab/"
  },
  "@type": "ApiDocumentation",
  "supportedClasses": [{
    "@id": ["CriminalListRep"],
    "supportedProperties": [{
      "@id": ["wantedpeople:Criminal"],
      "property": "criminals",
      "@type": "SupportedCollection"
    }],
    "supportedOperations": [{
      "method": "GET",
      "url": "criminals",
      "returns": ["CriminalListRep"],
      "@type": "LoadResourceOperation"
    }, ...]
  }, {
    "@id":["Description","wantedpeople:Criminal"],
    "supportedProperties": [{
      "@id": ["wantedpeople:height",
      "property": "height",
      "@type": "SupportedProperty"
    }, {
      "@id": ["wantedpeople:weight"],
      "property": "weight",
      "@type": "SupportedProperty"
    }, ...],
    "supportedOperations": [{
      "method": "GET",
      "url": "criminals/{protocolNumber}/description",
      "returns": ["Description","wantedpeople:Criminal"],
      "@type": "LoadResourceOperation"
    }, ...],
    "globalIriTemplateMapping": [{
      "@type": "IriTemplateMapping",
      "@id": "wantedpeople:protocolNumber",
      "variable": "protocolNumber"
    }]
  }, {
    "@id": ["wantedpeople:Criminal",
    "supportedProperties": [{
      "@id": ["wantedpeople:protocolNumber"],
      "property": "protocolNumber",
      "@type": "SupportedProperty"
    }, {
      "@id": ["wantedpeople:givenName"],
      "property": "givenName",
      "@type": "SupportedProperty"
    }, ...],
    "supportedOperations": [{
      "method": "GET",
      "url": "/criminals/{protocolNumber}",
      "returns": ["wantedpeople:Criminal"],
      "@type": "LoadResourceOperation"
    }, ...]
  }]
}
```