Revista de Sistemas de Informação da FSMA
n. 19 (2017) pp. 34-41

SISTEMAS DE INFORMAÇÃO
http://www.fsma.edu.br/si/sistemas.html

TRILHA PRINCIPAL

Integração entre Sistemas utilizando Web Services REST e SOAP: Um Relato Prático

Cristiano M. Garcia, Ramon Abilio
Diretoria de Gestão de Tecnologia da Informação (DGTI)
Universidade Federal de Lavras, Brasil (UFLA)
{cristiano.garcia,ramon.abilio}@dgti.ufla.br

***Resumo*— Em ambientes corporativos, é comum a existência de uma variedade de sistemas para automatizar as atividades inerentes à companhia. Em ambientes acadêmicos, isso também acontece. No entanto, a heterogeneidade em ambientes acadêmicos é ainda maior que em ambientes corporativos, pois, normalmente, existem atividades muito especializadas, como: Restaurante Universitário, Biblioteca, processos acadêmicos, processos administrativos e serviços de redes de computadores como: e-mail e autenticação. Para manter a consistência dos dados em todos os sistemas, é necessários que eles estejam integrados. Essa integração acontece na Universidade Federal de Lavras utilizando Arquitetura Orientada a Serviços e utilizando Simple Object Access Protocol (SOAP) como protocolo de comunicação. No entanto, com o desenvolvimento de um aplicativo móvel institucional, foi percebido que essa forma de comunicação é custosa para dispositivos móveis, visto que dispositivos móveis possuem recursos (banda móvel e processamento) limitados. Então, foi desenvolvida uma camada REST-JSON para a integração entre o aplicativo móvel e a arquitetura de integração, de forma que fossem aproveitados os mecanismos da arquitetura de integração existente. Dessa forma, a oferta de funcionalidades da arquitetura de integração foi expandida também para Resource State Transfer (REST), podendo atender novos aplicativos sem que seja necessário modificá-la de forma significativa. Foi constatado que a camada REST-JSON consome cerca de 73% menos banda móvel que SOAP. A camada REST-JSON desenvolvida está em ambiente de produção atendendo a cerca de 5600 instalações do aplicativo que realizam, em média, 54 mil requisições por dia.**

***Palavras-Chave*— Arquitetura Orientada a Serviço, Integração de Sistemas de Informação, Web Services, JSON, SOAP**

***Abstract*—In companies environments, it is normal to exist several systems to ease daily activities. In academic environments, it also happens. However, academic environments may be even more heterogeneous as there are many specialized activities, such as: restaurant, library, academic processes, administrative processes and computer network services, such as email and network authentication. To maintain the data consistency throughout the systems, all the systems must be integrated. This integration was carried out in the Federal University of Lavras by using Simples Object Access Protocol (SOAP) as communication protocol. The development of a new system (mobile application), it was noticed that SOAP is very CPU-intensive and slow, as mobile devices have constraints such as internet and processing. Thus, a REST-JSON layer to integrate mobile application and the integration architecture was developed, benefiting from all the resources the integration architecture had. By using this new layer, the offer of functions from the integration architecture was also expanded to REST, attending to other applications without having to make big changes in the code. It was measured that the REST-JSON layer consumes around 73% less data than SOAP. The REST-JSON layer was released, attending to about 5600 installations of the application that requests the integration around 54000 times a day.**

***Keywords*—Service-Oriented Architecture, Information Systems Integration, Web Services, JSON, SOAP**

I. INTRODUÇÃO

Ambientes corporativos normalmente possuem diferentes sistemas de informação para gerência de dados e informações nos níveis estratégico, tático e operacional [17]. Normalmente, o compartilhamento de dados entre esses sistemas é necessário e pode ser realizado por meio de Integração entre Sistemas de Informação (ISI). Existem abordagens para ISI, como "Aplicações Compostas" [14], com foco em técnicas, ou "Processos de Negócios Distribuídos" [12], com foco nos processos organizacionais.

Para realizar uma ISI, podem ser utilizadas técnicas como Arquitetura Orientada a Serviço, conhecida por SOA (*Service-Oriented Architecture*), configurando assim uma Integração Orientada a Serviço (ou SOI (*Service-Oriented Integration*)).

Diversas tecnologias oferecem suporte à implementação da SOA, como o *framework* .NET, *Java 2 Platform, Enterprise Edition* (J2EE) e *Web Services* [10]. *Web Services* representam uma forma de oferta de recursos distribuídos ligada à Internet.

Alguns padrões e tecnologias oferecem suporte ao desenvolvimento de *Web Services*, como *Representational State Transfer* (REST), arquitetura que utiliza princípios que fazem a internet escalável [7], e *Simple Object Access Protocol* (SOAP), um protocolo de comunicação amplamente utilizado [21].

Da mesma forma que em ambientes corporativos, a ISI também existe em ambientes acadêmicos, que podem ser mais

heterogêneos que os corporativos [1][2][4][8][16]. Em ambientes corporativos, podem ser utilizados ERPs (*Enterprise Resource Planning*) com módulos que abrangem atividades de diversos departamentos dessas instituições. Em universidades, devido à alta heterogeneidade de departamentos e atividades, como restaurantes, plataformas de ensino à distância, sistema de controle acadêmico e biblioteca, geralmente são utilizados diversos sistemas, cada um para seu propósito específico. Neste caso, é necessário compartilhamento de dados entre sistemas e que estes dados sejam consistentes.

A Universidade Federal de Lavras (UFLA), que mantém cerca de 20 sistemas, é um exemplo de quão complexa e heterogênea uma integração em ambientes acadêmicos pode ser [8]. Tais sistemas são mantidos pela Coordenadoria de Sistemas de Informação (CSI), subordinada à Diretoria de Gestão de Tecnologia da Informação (DGTI). Esses sistemas são heterogêneos em relação à tecnologia, banco de dados e desenvolvedores, sendo que alguns sistemas foram desenvolvidos dentro da própria CSI e outros por terceiros. A arquitetura de integração entre sistemas de informação e serviços de rede foi desenvolvida utilizando SOAP como protocolo, e deveria ser [8]: escalável, flexível, monitorável e que contasse com mecanismos de segurança. Escalável no sentido de que os novos *Web Services* pudessem ser facilmente expandidos; flexível para atender diferentes demandas institucionais; monitorável para que qualquer erro pudesse ser facilmente detectado e seguro pois toda a integração trabalha com dados pessoais e sigilosos. O conjunto de *Web Services* ofertados são chamados de WSI, que significa *Web Services* de Integração.

A UFLA também é objeto de estudo deste trabalho, que descreve a entrada de mais um sistema na integração existente na Universidade: um aplicativo móvel destinado à comunidade acadêmica, chamado de MinhaUFLA. Como dispositivos móveis não possuem a mesma quantidade de recursos (energia, plano de dados, capacidade de processamento e armazenamento) de servidores, o aplicativo foi projetado para onerar o mínimo possível o dispositivo. Nesse contexto, optou-se por fazer a integração do aplicativo com os demais sistemas institucionais utilizando REST, pois requisições REST podem ser de 9 a 30 vezes mais rápidas que SOAP e a estrutura do XML pode ser até 3 vezes maior que do JSON [6].

Portanto, passou-se a ter incompatibilidade entre a forma de comunicação definida para ser utilizada no aplicativo móvel (REST) e a forma de comunicação padronizada na arquitetura de integração existente (SOAP). Dessa forma, foi necessário o desenvolvimento de uma camada para integrar aplicativo móvel e sistemas institucionais. O objetivo deste trabalho é demonstrar como essa integração foi realizada, avaliando pontos positivos e negativos. Como o uso de aplicativos é crescente, a principal contribuição deste trabalho é possibilitar que, instituições que passem pela mesma situação, possam analisar suas arquiteturas de integração e optar ou não pela adição de uma camada extra para comunicação de aplicativos com os demais sistemas.

O restante do trabalho está estruturado da seguinte forma: na Seção II, são apresentados conceitos relacionados *Web Services*, REST e SOAP. Na Seção III, está a definição do problema. Na Seção IV, é descrita a solução implementada. Na Seção V, a solução é avaliada e, na Seção VI, são apresentadas as conclusões sobre o trabalho.

II. WEB SERVICES: SOAP E REST

Arquitetura Orientada a Serviço é um paradigma independente de tecnologia, baseado em padrões, e tem sido considerada um dos principais paradigmas em Sistemas Distribuídos [19]. SOA visa organizar e utilizar recursos, que podem estar sob responsabilidade de diferentes organizações, provendo um meio padronizado de oferta, descoberta e interação com funcionalidades usadas pra produzir efeitos consistentes, podendo oferecer benefícios como controle de crescimento de sistemas e oferta global de serviços [18].

A integração utilizando SOA, chamada Integração Orientada a Serviço (SOI), tem como principal objetivo a integração de diversos sistemas modificando pouco ou nada suas implementações [11]. Um dos meios de se implementar SOI é pela utilização de *Web Services*. *Web Services* (WS) constituem uma SOI por meio do provimento de uma interface de serviços que possibilita a interação entre consumidores e provedores de serviços [5].

As formas mais utilizadas para implementar WS são [21]: SOAP ou REST e os padrões utilizados para intercâmbio de dados comumente utilizados são: i) *Extensible Markup Language* (XML) [20] – padrão utilizado para descrever dados de forma flexível; e ii) *Javascript Object Notation* (JSON) – formato independente de linguagem, menos verboso em relação ao XML e tem como características [13]: legibilidade (por parte de humanos), facilidade de geração e análise. SOAP provê um padrão básico de comunicação e utiliza um documento baseado em XML chamado *Web Services Description Language* (WSDL) para descrever as funcionalidades ofertadas por um WS [21]. O WSDL do provedor de *Web Services* relativos ao aplicativo MinhaUFLA é gerado automaticamente pela ferramenta NuSOAP[1].

O NuSOAP identifica os *Web Services* registrados e cria elementos message concatenando as palavras *Request* e *Response* ao nome dos *Web Services* no WSDL. Dessa forma, são identificados os parâmetros que devem ser enviados ao consumir o *Web Service* (*message* identificado com *Request*) e o que é retornado pelo *Web Service* consumido (*message* identificado com *Response*). Por exemplo, a Figura 1 exibe os elementos *message* do serviço MinhaUFLA.auth. Percebe-se que deve-se informar os parâmetros *token* e ip ao se consumir o *Web Service* (*Request*), e o retorno é uma *string* (*Response*).

A Figura 2 apresenta um exemplo de requisição SOAP ao *Web Service* MinhaUFLA.auth (*Request*) e a Figura 3 exibe um exemplo de resposta SOAP a essa requisição (*Response*). Observando-se as Figuras 2 e 3 pode-se perceber que há uma estrutura definida pelo WSDL que deve ser respeitada para que os *Web Services* funcionem corretamente. Na Figura 2

[1] https://sourceforge.net/projects/nusoap/

```
<message name="MinhaUFLA.authRequest">
  <part name="token" type="xsd:string"/>
  <part name="ip" type="xsd:string"/>
</message>
<message name="MinhaUFLA.authResponse">
  <part name="return" type="xsd:string"/>
</message>
```

Fig. 1. Exemplo de WSDL (parcial)

```
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
    <Body>
        <MinhaUFLA.auth xmlns="urn:minhaufla">
            <token>[string]</token>
            <ip>[string]</ip>
        </MinhaUFLA.auth>
    </Body>
</Envelope>
```

Fig. 2. Exemplo de requisição SOAP

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<SOAP-ENV:Envelope SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/">
    <SOAP-ENV:Body>
        <ns1:MinhaUFLA.authResponse xmlns:ns1="urn:minhaufla">
            <return xsi:type="xsd:string">Resposta</return>
        </ns1:MinhaUFLA.authResponse>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Fig. 3. Exemplo de retorno SOAP

especificamente, pode-se perceber que a ferramenta Wizzdler[2] (ferramenta que gera estruturas de requisição com base em documentos WSDL) gerou MinhaUFLA.auth para a requisição. Através de testes empíricos, foi percebido que MinhaUFLA.auth e MinhaUFLA.authRequest são sinônimos dentro do contexto de requisição SOAP.

REST é uma abstração dos princípios que tornam a *World Wide Web* escalável e permite oferecer serviços identificados por uma *Uniform Resource Identifier* (URI). Métodos HTTP, como GET, POST, DELETE e PUT definem a operação a ser executada sobre um registro ou um conjunto de registros. Por exemplo, GET retorna um registro, POST insere um novo registro, DELETE remove e PUT atualiza um registro existente [7].

Portanto, SOAP exige uma estrutura XML que deve ser respeitada na requisição a um serviço e na sua resposta, e REST permite que o padrão de comunicação possa ser definido pelas partes interessadas [9]. As Figura 4 e 5 mostram uma mesma mensagem utilizando SOAP-XML e REST-JSON. Pode-se inferir que o consumo de dados será maior utilizando SOAP-XML, pois o conteúdo da mensagem será envolto pela estrutura do protocolo SOAP, enquanto utilizando REST-JSON isso não acontece.

A utilização de requisições utilizando REST-JSON a partir de dispositivos móveis se mostrou a melhor alternativa em testes envolvendo também REST-XML e SOAP, em diferentes sistemas operacionais (iOS 8.0, iOS 8.1, Android 4.4.2 e simulador iOS) [6]. Foi observado que requisições REST-JSON são de 9 a 30 vezes mais rápidas que requisições SOAP, tanto em processamento quanto em transmissão [6]. Isso se

[2] https://chrome.google.com/webstore/detail/wizdler/oebpmncolmhiapingjaagmapififiakb

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<SOAP-ENV:Envelope SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org
    /soap/encoding/" xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap
    /envelope/" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi
    ="http://www.w3.org/2001/XMLSchema-instance" xmlns:SOAP-ENC="http
    ://schemas.xmlsoap.org/soap/encoding/">
    <SOAP-ENV:Body>
        <ns1:MinhaUFLA.authResponse xmlns:ns1="urn:minhaufla">
            <return xsi:type="xsd:string">{&quot;id&quot;:&quot;800_1
                .4&quot;,&quot;message&quot;:&quot;ST-119242
                -BCvOxadIu9HBEbNZD6sk-casdgti&quot;,&quot;type&quot
                ;:&quot;SUCESSO&quot;,&quot;system&quot;:&quot
                ;MINHAUFLA&quot;}</return>
        </ns1:MinhaUFLA.authResponse>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Fig. 4. Exemplo de Mensagem SOAP-XML

```
{
    "id":"800_1.4",
    "message":"131vnt2td8h1hhnaens1d7jla1",
    "type":"SUCESSO",
    "system":"MINHAUFLA"
}
```

Fig. 5. Exemplo de Mensagem REST-JSON

deve ao fato de que a estrutura do XML (para SOAP e REST-XML) pode ser até 3 vezes maior que a do JSON para os mesmos dados, além do processamento de XML ser mais custoso computacionalmente [6].

III. Definição do Problema

A arquitetura de integração na Universidade Federal de Lavras (UFLA) foi desenvolvida utilizando a linguagem de programação PHP[3] e projetada para [8]: i) contar com mecanismos de segurança; ii) ter a relação entre Provedores de Serviços e Serviços baseada em muitos provedores com poucos serviços, agrupados de acordo com os sistemas, o que facilitaria manutenção isolada; e iii) utilizar SOAP.

Entre o fim de 2015 e o começo de 2016, foi iniciado o desenvolvimento de um aplicativo para dispositivos móveis baseados em Android, chamado de MinhaUFLA [3]. Esse aplicativo possui opções comuns a discentes e docentes e opções específicas para cada perfil, por exemplo: no perfil dos discentes, estão disponíveis notas de avaliações e geolocalização de locais de aula; e, no perfil dos docentes, tem-se opções para consultar horário e local de suas aulas, e realizar controle sobre a frequência dos discentes [3]. O aplicativo deveria consumir dados existentes nos sistemas de informação da Universidade e ofertados pelos WSI.

Portanto, as seguintes questões foram analisadas:

- O aplicativo móvel deveria obter e manipular dados por meio da arquitetura de integração para manter-se os requisitos de segurança, manutenibilidade e escalabilidade, e os mecanismos de monitoramento existentes;
- Como dispositivos móveis dispõem de recursos limitados em relação a computadores convencionais, no que diz respeito à fonte de energia, armazenamento, internet e processamento, o MinhaUFLA foi projetado para realizar o mínimo de

[3] http://www.php.net

processamento e transferir o mínimo possível de dados por meio da internet.

Com base na análise, percebeu-se que a arquitetura de integração, que utiliza SOAP, e a utilização de REST-JSON por parte dos aplicativos móveis eram incompatíveis. Portanto, a solução foi o desenvolvimento de uma camada de integração SOAP/REST para permitir que o aplicativo se comunicasse com os WSI por meio da arquitetura de integração.

IV. SOLUÇÃO IMPLEMENTADA

Diante do problema exposto na Seção III, foi projetada uma camada integradora SOAP/REST que permitisse a integração do aplicativo móvel, recebendo as requisições REST-JSON provenientes do aplicativo e convertendo para SOAP para a integração, e vice-versa, conforme explicado nas Subseções a seguir.

A. Padrão de Comunicação entre Provedores e Consumidores

Na arquitetura de integração existente na UFLA, há a divisão em Provedores e Consumidores. Os Provedores oferecem *Web Services*, enquanto os Consumidores consomem os *Web Services* ofertados pelos provedores. Cada Provedor é relativo a um sistema específico. Por exemplo, o Pergamum, sistema de biblioteca, possui seus próprios Provedor e Consumidor privado que consome *Web Services* de atualização de dados de seu respectivo Provedor. Essa relação pode ser vista na Figura 6, que representa a arquitetura da integração existente na UFLA anterior ao aplicativo MinhaUFLA.

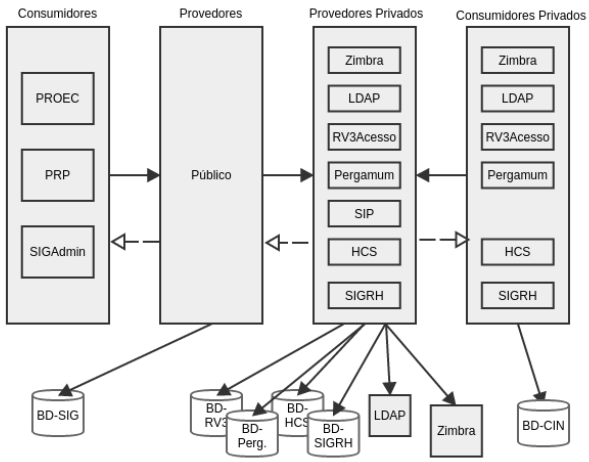

Fig. 6. Arquitetura de integração: Provedores e Consumidores

A comunicação entre Provedores e Consumidores é realizada utilizando um objeto JSON com 4 atributos: *id*, *message*, *type* e *system*. Esse objeto é encapsulado pela estrutura intrínseca da utilização do SOAP. Existe também um fluxo na sequência de requisições aos WSI a ser respeitado (Figura 7). No exemplo mostrado na Figura 7, o Consumidor se autentica no Provedor utilizando um *token* (senha) definido previamente. Na sequência, o Provedor devolve um objeto JSON contendo um SID (ID de sessão). A partir da

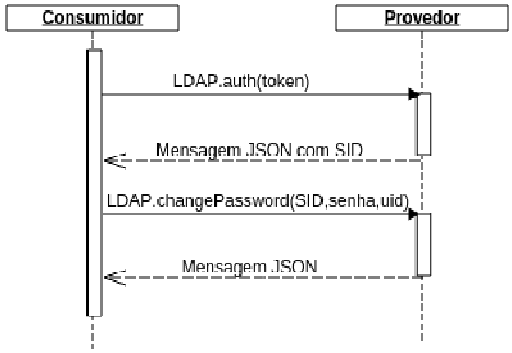

Fig. 7. Fluxo de comunicação da autenticação de um consumidor a um provedor

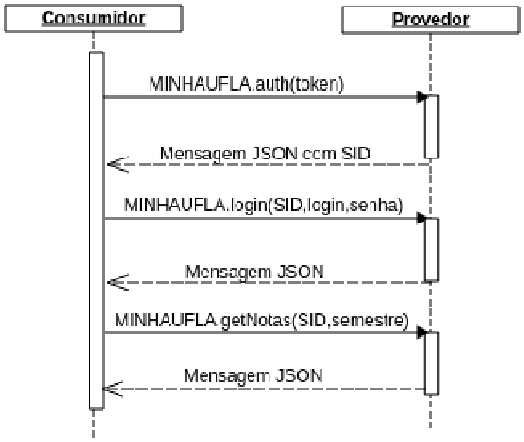

Fig. 8. Fluxo de comunicação com o *web service* de obtenção de notas de um estudante

autenticação, o Consumidor pode acessar outros serviços utilizando o SID [8].

Na comunicação com os WSI, deve haver uma chamada ao serviço "auth", que corresponde à autenticação com o Provedor de Serviços desejado e que considera o *token* (senha utilizada) e o IP de origem. Na comunicação entre o aplicativo móvel e os WSI, pode haver um outro passo, após a autenticação, que é o serviço "login" (Figura 8). Este passo existe apenas para alguns serviços. Enquanto "auth" diz respeito à permissão do aplicativo acessar os WSI, "login" diz respeito à permissão do usuário utilizar certos serviços vinculados ao aplicativo.

B. Camada Integradora: REST/SOAP

A camada integradora realiza a "tradução" REST-JSON para SOAP, protocolo utilizado pelos WSI, e vice-versa. Desta forma, alterações necessárias na implementação dos WSI foram minimizadas. No projeto dessa camada integradora, chamada na DGTI-UFLA de "Camada REST-JSON", considerou-se: i) a complementação da oferta de serviços por parte dos WSI, que os ofertam utilizando apenas o SOAP; e ii) requisitos de segurança e escalabilidade, para o caso de outras iniciativas de aplicativos móveis ou mesmo aplicações Web ou desktop que necessitem de informação atualizada.

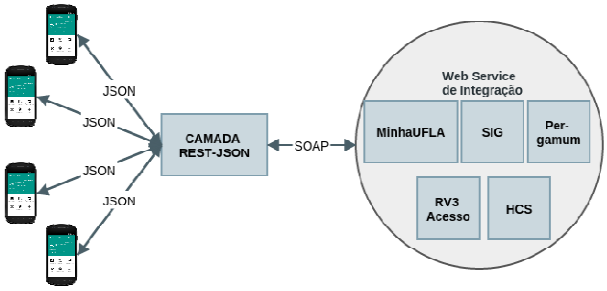

Fig. 9. Arquitetura utilizando a Camada REST-JSON

Dessa forma, a arquitetura atualizada pode ser representada como na Figura 9, em que os aplicativos acessam a Camada REST-JSON, que por sua vez faz requisição SOAP aos WSI – podendo utilizar serviços de provedores como SIG, Pergamum, RV3Acesso e HCS, que então retornariam as informações requisitadas, passando pelo mesmo caminho no sentido inverso, chegando aos aplicativos.

C. *Implementação*

A princípio, os *Web Services* para o MinhaUFLA, bem como os *Web Services* nos WSI, foram desenvolvidos utilizando SOAP. Tais *Web Services* correspondem, por exemplo, às atividades de: consulta a notas de disciplina, consulta a saldo de créditos no restaurante universitário e consulta ao horário acadêmico. Estes *Web Services* deveriam ser acessados a partir do aplicativo por meio da Camada REST-JSON.

Para a implementação da Camada REST-JSON, foi utilizada a linguagem de programação PHP em conjunto com o *framework* Laravel, versão 5.2. A escolha pelo PHP foi devido à uma padronização existente na DGTI-UFLA no que tange sistemas internos e a opção pelo *framework* Laravel foi devido à sua facilidade em criação de *Web Services Restful*.

Apesar de o Laravel possuir uma série de recursos úteis ao desenvolvimento Web, ele não possui recursos nativos para a realização de requisições SOAP. Então, para realizar as requisições da Camada REST-JSON para os WSI, foi utilizado um pacote para integração SOAP com Laravel, chamado laravel-soap[4].

Como o Laravel é um *framework* MVC ((*Model View Controller*)), é necessário a criação de um modelo (*model*) e um controlador (*controller*). No Laravel, não é compulsório o desenvolvimento de uma visão (*view*), ou seja, as informações podem ser disponibilizadas diretamente a partir do controlador. Portanto, foi desenvolvido um modelo genérico, chamado *WSClass*, para acesso a *Web Services* que utilizam SOAP e um controlador específico para os *Web Services* relativos às informações utilizadas pelo aplicativo MinhaUFLA. Os *Web Services* específicos passam os parâmetros de configuração ao modelo e o modelo faz o acesso ao *Web Service* SOAP. O retorno dos *Web Services* é retornado pelo controlador, sendo assim repassado ao aplicativo consumidor de serviços.

Em um contexto particular ao Laravel, foi desenvolvido também um *middleware*, conceito utilizado pelo Laravel para "encapsular" requisições, podendo executar tarefas (antes ou depois, dependendo da configuração) relacionadas a um determinado controlador. Tal *middleware* avalia apenas a existência de uma variável POST chamada *service*, que define o *Web Service* SOAP a ser invocado. O roteamento do Laravel é flexível e pode ser configurado para utilizar *middleware* e redirecionar a requisição para um controlador específico. A Figura 10 apresenta o fluxo interno na Camada REST-JSON.

Considerando o fluxo apresentado na Figura 10 e, em um contexto particular ao Laravel, para a criação de um novo *Web Service* REST-JSON, é necessário criar uma nova rota em routes.php. Depois, cria-se um controlador que deve instanciar um modelo WS passando os parâmetros de configuração, como o nome do provedor SOAP, URL do WSDL e invocar o método "access", passando o nome do *Web Service* e os parâmetros. Caso haja alguma validação que deve ser realizada em todas as requisições, como algum tipo de filtro, criptografia, prevenção de injeção de código malicioso, sugere-se a criação de um novo *middleware*.

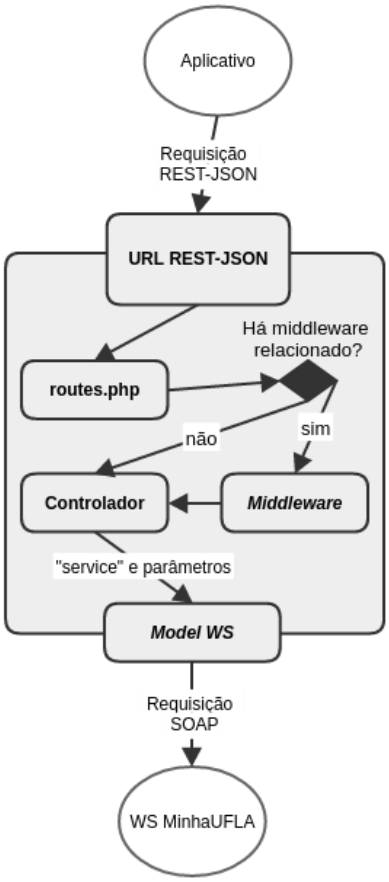

Fig. 10. Fluxo de uma requisição originária do aplicativo

[4] Disponível em https://github.com/artisaninweb/laravel-soap

V. Avaliação da Solução

O objetivo deste trabalho é demonstrar como esta integração entre aplicativo móvel e o ambiente de integração foi realizada, levantando alguns dados interessantes sobre esta integração.

Conforme projetado, as características de monitorabilidade, manutenibilidade e escalabilidade foram estendidas à camada, sendo possível monitorar a integração, mesmo utilizando a camada REST-JSON. Isso acontece pois a própria camada invoca os WSI, que registram as chamadas, e utilizam o mesmo servidor físico configurado com HTTPS.

Foram realizados testes com o intuito de verificar o tempo médio de processamento de uma requisição. Foram realizadas 100 requisições em intervalos de 5 segundos a partir de um computador para um ambiente de testes instalado em um servidor com sistema operacional Ubuntu Server 14.04[5] na mesma rede. O tempo de processamento foi medido partindo da requisição até o recebimento da resposta. A medição não foi realizada a partir de um dispositivo móvel, pois os resultados seriam dependentes da rede que o dispositivo utilizaria para acessar a camada REST-JSON, resultando em números que não expõem quanto tempo a camada impõe à requisição.

Para a realização das requisições, foram utilizadas as ferramentas Postman[6] (para requisições REST), e Wizdler (para requisições SOAP), sendo ambas extensões para o navegador Google Chrome[7]. O tempos de requisição também foram avaliados utilizando o teste de Tukey HSD (*Honestly Significant Difference*) [15], que mede diferença significativa entre médias de grupos.

TABELA I
Tempo médio de execução de Requisições SOAP ou através da Camada REST-JSON

Forma de Comunicação	Tempo médio (ms)
SOAP	64.59 +/- 2.2304
Camada REST-JSON	165.99 +/- 2.2403

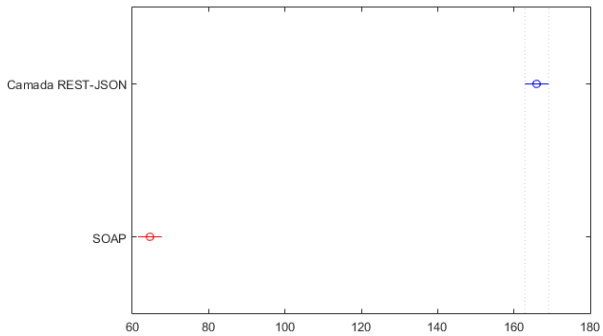

Fig. 11. Teste de Tukey para comparação de médias de requisições SOAP direta e através da Camada REST-JSON, significando que as médias de tempo de requisições SOAP são significativamente melhores que utilizando a Camada REST-JSON. No entanto, sem os benefícios da Camada REST-JSON.

[5] http://releases.ubuntu.com/14.04/
[6] https://www.getpostman.com/
[7] https://www.google.com/chrome/browser/desktop/index.html

A Tabela I apresenta as médias e os devios padrão calculados a partir da medição do tempo de resposta das requisições e a Figura 11 exibe um gráfico com o resultado do teste de Tukey. Observando a Tabela I e a Figura 11, percebe-se que a solução implementada demora 100 milissegundos a mais para processar uma requisição. Como a arquitetura de integração se tornou híbrida utilizando SOAP e REST-JSON, inserindo a camada REST-JSON na sequência de chamadas, esse aumento no tempo era esperado.

Embora a camada aumente o tempo de processamento de requisições, esse aumento representa mais processamento do lado do servidor, evitando assim processamento do dispositivo móvel, pois os dados são retornados no padrão JSON. É importante ressaltar que 100 milissegundos é um tempo pouco perceptível e não atrapalha a experiência do usuário.

TABELA II
Tamanho em *bytes* de mensagens utilizando SOAP e a Camada Rest-JSON

Forma de Comunicação	*Web Services*	Tamanho em *bytes*
SOAP	auth	747
REST-JSON	auth	97
SOAP	login	893
REST-JSON	login	179
SOAP	getNotas	2968
REST-JSON	getNotas	1000

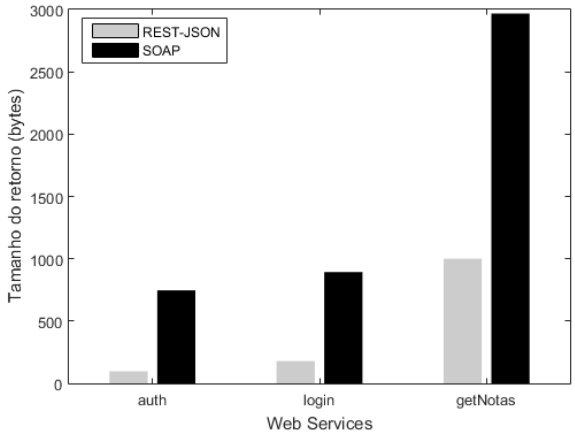

Fig. 12. Tamanho em *bytes* dos retornos de requisições SOAP e REST-JSON, por *Web Service*

Foram comparados também os tamanhos de retornos de requisições SOAP e REST-JSON, utilizando como exemplo três *Web Services* desenvolvidos para atender o aplicativo MinhaUFLA: auth, login e getNotas, sendo auth o Web Service que retorna menos dados, e getNotas o que retorna maior quantidade de dados. Ou seja, foi comparado quanto o aplicativo móvel consome em bytes e, posteriormente, também precisa processar. Os dados desta comparação podem ser visualizados na Tabela II e na Figura 12.

Os dados observados na Figura 12 correspondem aos *bytes* que seriam consumidos da banda móvel do dispositivo. Na média, as mensagens utilizando a Camada REST-JSON consomem 73% menos banda que as mensagens utilizando SOAP.

A arquitetura de integração, utilizando a camada REST-

JSON, encontra-se em ambiente de produção desde setembro de 2016, quando o aplicativo foi lançado oficialmente. Até novembro de 2016, existiam cerca de 5.600 (cinco mil e seiscentas) instalações ativas do aplicativo realizando, em média, 54 mil requisições por dia. O aplicativo conta com 1788 avaliações no Google Play, com média 4.8, com pontuação máxima de 5[8].

VI. Conclusões e Trabalhos Futuros

A utilização de *Web Services* para oferecer funcionalidades por meio da internet tem se tornado cada vez mais comum devido à sua flexibilidade, segurança em relação à outras formas de oferta de funcionalidades, e escalabilidade. As formas mais comuns de se oferecer funcionalidades em forma de *Web Services* é utilizando SOAP ou REST.

Na Universidade Federal de Lavras (UFLA), havia um conjunto de *Web Services* para a integração entre os sistemas e oferta de funcionalidades. Os *Web Services* de Integração na UFLA utilizam SOAP, visto que foram projetados para a integração entre sistemas, sem restrição em relação à processamento, internet, entre outros. No entanto, com a entrada de um aplicativo móvel institucional, houve aumento da complexidade da integração existente no que tange restrição de processamento e consumo de dados em dispositivos móveis.

A solução desenvolvida, chamada de Camada REST-JSON, apesar de impor um aumento de, em média, 100ms (ou 0,1 segundos), consome cerca de 73% menos banda de internet em relação ao que seria consumida utilizando SOAP e, por consequência, menos processamento e menos gasto de energia do dispositivo móvel, visto que processar XML é mais lento e requer mais do CPU que JSON [6].

Dentre os pontos positivos da camada REST-JSON, destacam-se:

- Oferta de *Web Services* para aplicativos móveis com baixo custo temporal em relação à uma requisição SOAP comum;
- Centralização das ofertas de *Web Services* por parte dos WSI mantida. Ou seja, mesmo que a oferta do *Web Service* fosse via REST, inicialmente o *Web Service* seria desenvolvido e disponibilizado via SOAP, por meio do WSI, e a camada REST-JSON teria o papel de transcrever as requisições REST-JSON para SOAP e vice-versa;
- Mecanismos existentes na arquitetura de integração, como monitoramento, manutenibilidade e escalabilidade, foram estendidos à camada REST-JSON;
- Trabalho adicional mínimo é necessário para a oferta de *Web Services* também em REST-JSON (em relação ao trabalho necessário para a oferta de *Web Services* apenas em SOAP).

Como pontos negativos da camada REST-JSON, pode-se salientar:

- Aumento de cerca de 0.1 segundos no tempo de requisição;
- Aumento da complexidade de manutenção, pois a camada foi desenvolvida utilizando tecnologias diferentes daquelas empregadas nos WSI. No entanto, a arquitetura usada fornece pontos positivos importantes, em detrimento de pontos negativos menos importantes, conforme mencionado na Seção V.

Portanto, os problemas apontados após a análise das questões foram solucionados com a implementação da Camada REST-JSON. A arquitetura de integração existente passou a permitir a integração com o aplicativo sem interferir na oferta de *Web Services* e na integração utilizada até o momento, mantendo suas características e mecanismos de segurança, monitoramento, flexibilidade e escalabilidade. O consumo de energia e banda de internet dos dispositivos móveis foram JSON enquanto o processamento da requisição SOAP é feita pelo servidor de aplicação.

Visto que o desempenho de requisições REST-JSON são mais rápidas e menos onerosas ao CPU, como trabalho futuro têm-se:

- Realizar novos estudos em relação a REST-JSON (ou outras tecnologias relacionadas que venham a surgir);
- Avaliação de tecnologias que dêem suporte a requisições HTTP;
- Realização de testes internos com JSON-REST e outras tecnologias;
- Estudo da possibilidade de reescrita dos *Web Services* em JSON-REST (ou outra tecnologia mais eficiente).

[8] Dados de Abril/2017

Referências

[1] Ehab N Alkhanak and Salimah Mokhtar. Using services oriented architecture to improve efficient web-services for postgraduate students. Proceeding of the WASET, pages 68–71, 2009.

[2] VO Andersson, RT dos Santos, ALC Tillmann, and JH Noguez. S. cobalto webservice: Solução para consistência de informações. Resumo Publicado na VIII Workshop de Tecnologia da Informação e Comunicação das IFES, 2014.

[3] S. G. S. Araújo, C. M. Garcia, R. Abílio, and N. Malheiros. Acesso a Serviços e Informações Acadêmicas da Universidade Federal de Lavras em Dispositivos Móveis com a Plataforma Android. X Workshop de Tecnologia da Informação e Comunicação das IFES, 2016.

[4] Carlos Costa, Ana Cristina Melo, Aníbal Fernandes, Luís Mendes Gomes, and Hélia Guerra. Integração de sistemas de informação universitários via web services. In Actas da 5a Conferencia Ibérica de Sistemas y Tecnologias de Información, pages 290–295, 2010.

[5] George Coulouris, Jean Dollimore, Tim Kindberg, and Gordon Blair.Sistemas Distribuídos-: Conceitos e Projeto. Bookman Editora, 2013.

[6] Steven Davelaar. Performance Study – REST vs SOAP for Mobile Applications. Disponível em <http://www.ateam-oracle.com/performancestudy-rest-vs-soap-for-mobile-applications/>. Acesso em: 06 maio 2016., 2015.

[7] Roy T Fielding and Richard N Taylor. Principled design of the modern web architecture. ACM Transactions on Internet Technology (TOIT), 2(2):115–150, 2002.

[8] C. M. Garcia, R. Abilio, and N. Malheiros. Abordagens e Tecnologias para Integração de Sistemas: Um Estudo de Caso na Universidade Federal de Lavras. Revista de Sistemas de Informação da FSMA, (15):11–22, 2015.

[9] Peter Leo Gorski, Luigi Lo Iacono, Hoai Viet Nguyen, and Daniel Behnam Torkian. Service security revisited. In Services Computing (SCC), 2014 IEEE International Conference on, pages 464–471. IEEE, 2014.

[10] Wu He and Li Da Xu. Integration of distributed enterprise applications:a survey. IEEE Transactions on Industrial Informatics, 10(1):35–42, 2014.

[11] B. Hensle, C. Booth, D. Chappelle, J. McDaniels, M. Wilkins, and S. Bennett. Oracle reference architecture-service-oriented integration.

[12] Gregor Hohpe and Bobby Woolf. Enterprise integration patterns: Designing, building, and deploying messaging solutions. Addison-Wesley Professional, 2004.

[13] JSON.org. Introducing JSON. Disponível em <http://www.json.org/>. Acesso em: 06 junho 2016., 1999.

[14] Victor Manuel Moreira Martins. Integração de Sistemas de Informação: Perspectivas, normas e abordagens. PhD thesis, 2006.

[15] John Neter, Michael H Kutner, Christopher J Nachtsheim, and William Wasserman. Applied linear statistical models, volume 4. Irwin Chicago, 1996. R. J. Vidmar. (1992, August). On the use of atmospheric plasmas as electromagnetic reflectors. *IEEE Trans. Plasma Sci.* [Online]. *21(3).* pp. 876–880. Available: http://www.halcyon.com/pub/journals/21ps03-vidmar

[16] Fajar Suryawan. Inter-database synchronization: a low-cost approach to information system integration. 2014.

[17] Efraim Turban, Dorothy Leidner, Ephraim Mclean, and James Wetherbe. Tecnologia da Informação para Gestão-: Transformando os Negócios na Economia Digital. Bookman, 2010.
Mohammad Hadi Valipour, Bavar AmirZafari, Khashayar Niki Maleki,and Negin Daneshpour. A brief survey of software architecture concepts and service oriented architecture. In Computer Science and Information Technology, 2009. ICCSIT 2009. 2nd IEEE International Conference on, pages 34–38. IEEE, 2009.

[18] Willem-Jan van den Heuvel, Olaf Zimmermann, Frank Leymann, Patricia Lago, Ina Schieferdecker, Uwe Zdun, and Paris Avgeriou. Software service engineering: Tenets and challenges. In Proceedings of the 2009 ICSE Workshop on Principles of Engineering Service Oriented Systems, pages 26–33. IEEE Computer Society, 2009.

[19] W3.org. Extensible Markup Language (XML). Disponível em <https://www.w3.org/XML/>. Acesso em: 06 março 2017., 2016.
Michael Zur Muehlen, Jeffrey V Nickerson, and Keith D Swenson. Developing web services choreography standards—the case of rest vs. soap. Decision Support Systems, 40(1):9–29, 2005. information system integration. 2014.