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

TRILHA PRINCIPAL

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

Systems Integration Using Web Services, REST and SOAP: A Practical Report

Cristiano M. Garcia, Ramon Abilio
Information Technology Management Department (DGTI)
Federal University of Lavras, Brazil (UFLA)
{cristiano.garcia,ramon.abilio}@dgti.ufla.br

***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 at the Federal University of Lavras, using Simple 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. Introduction

Corporative environments usually have different information systems in its strategic, tactical and operational levels [17]. Usually, it is necessary to share data between those systems and this can be performed through Information Systems Integration (ISI). There are many different approaches for ISI such as "Composite Applications" [14], with focus on techniques or "Distributed Business Processes" [12], with emphasis on organizational processes.

In order to perform an ISI, we can use techniques such as Service Oriented Architecture (SOA), configuring a Service Oriented Integration (SOI).

Several technologies support the implementation of SOA, such as the.NET framework, the *Java 2 Platform, Enterprise Edition* (J2EE) and *Web Services* [10]. *Web Services* represent a form to offer distributed resources connected to the Internet.

Some standards and technologies support the development of Web Services, such as *Representational State Transfer* (REST), an architecture that uses principles that make the Internet scalable [7], and *Simple Object Access Protocol* (SOAP), a widely used communication protocol [21].

Just like in the corporative environments, ISI also exists in academic environments, which can be even more heterogeneous than their corporative counterparts [1][2][4][8][16]. In corporative systems, there can be ERPs (*Enterprise Resource Planning*) with modules that cover activities from several departments. In universities, due to the high heterogeneity of the departments and activities, such as restaurants, distance learning environments, academic control systems and libraries, there usually are many different systems, each one for its specific purpose. In this case, it is necessary to share the data between those systems so that the data is kept consistent.

The Federal University of Lavras (UFLA) has around 20 information and network systems and is an example of how complex the integration of academic information systems can be [8]. Those systems are maintained by the Information Systems Department (CSI), which is subordinate to the Information Technology Management Directorate (DGTI). They are heterogeneous in terms of technology, database and developers (some were developed by the CSI itself and other by third parties).

The integration architecture between information systems and network services was developed using SOAP as a protocol, intending to be [8]: scalable, flexible, monitorable and also, should count with security mechanisms.

The idea of scalability consists on the possibility of easily including new web services; flexibility consists on meeting different institutional requirements; monitorability in the idea that any error could be easily detectable; and the security mechanisms, as the whole integration works with personal, confidential data. The set of web services offered is called Integration Web Services (WSI).

UFLA is again the object of study in this paper, which describes the addition of a new system in its integration environment: a mobile application, called *MinhaUFLA* (myUFLA), to be used by the academic community. Since mobile devices do not have the same amount of resources (energy, data plan, processing and storing capacity) as servers do, the application was designed to be as smallest as possible to the device. In this context, it was decided to perform the integration of this application with the integration environment using REST, as REST inquiries can be from 9 to 30 times faster than SOAP and the XML structure can be up to 3 times larger than JSON [6].

Therefore, it was detected an incompatibility between the communication method used in the mobile device (REST) and the standardized communication method in the existing integration architecture (SOAP). This way, it was necessary to develop a layer to integrate the mobile device and the institutional systems.

The goal of this work is to show how this integration was performed, evaluating positive and negative aspects. Given that the use of mobile devices is growing, the main contribution of this work is to make it possible for institutions undergoing similar situations to analyze their integration architectures and to decide whether or not to add an extra layer for the communication of their mobile apps with the other systems.

The rest of this work is organized as follows. Section II present the related concepts *Web Services*, REST and SOAP. Section III defines the problem. Section IV describes the implemented solution. Section V evaluates the solution and finally, in Section VI, it is presented our conclusions on the whole work.

II. Web Services: SOAP and REST

Service Oriented Architecture is an independent technology paradigm based on patterns which has become one of the main paradigms in distributed systems [19]. SOA aims at organizing and using resources that can be under the responsibility of different organizations, providing a standardized way to offer, discover and interact with functionalities used to achieve consistent effects, with side benefits such as system growth control and global service offer [18].

The main goal of integration using SOA, called Service Oriented Integration (SOI), is to integrate different systems changing little or nothing in their implementation [11]. One of the ways to implement SOI is by using Web Services, which constitute a SOI by offering a service interface that allows for the interaction between service providers and consumers [5].

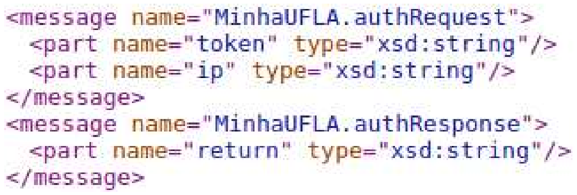
The ways to implement Web Services are [21] SOAP or REST and the most commons standards used to interchange data are the following:

i) *Extensible Markup Language* (XML) [20] – a standard used to describe data in a flexible way; and

ii) *Javascript Object Notation* (JSON) –a language independent format which is less verbose than XML and whose characteristics are the following [13]: legibility (by humans), easy to generate and analyze.

SOAP provides a basic communication standard and uses a document based on XML which is called *Web Services Description Language* (WSDL) to describe the functionalities offered by a WS [21]. The WSDL of the web services provider related to the MinhaUFLA application is generated automatically by the tool called NuSOAP[1].

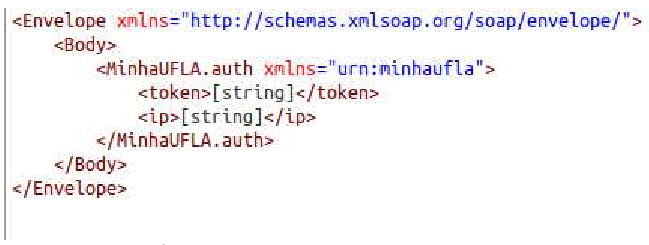
NuSOAP identified the registered Web Services and create message elements concatenating the words *Request* and *Response* to the Web Services names in the WSDL. This way, it identifies the parameters that must be sent to consume the Web Service (message identified with *Request*) and what is returned by the consumed Web Service (message identified with *Response*). For instance, Figure 1 shows the message elements from the service MinhaUFLA.auth. It can be seen that it must be informed the parameters *token* and *ip* when consuming the Web Service (*Request*) and the return value is of type *string* (*Response*).



```
<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. Example of WSDL (partial)

Figure 2 shows an example of SOAP request to the *Web Service* MinhaUFLA.auth (*Request*) and Figure 3 shows an example of SOAP answer to this request (*Response*). In the Figures 2 and 3, it can be seen that there is a structure defined by the WSDL that must be respected for the Web Services to work properly. In Figure 2 we can specifically see that the Wizzdler[2] tool (a tool used to generate request structures based on WSDL documents) generated MinhaUFLA.auth for the request. Through empirical tests, we noticed that MinhaUFLA.auth and MinhaUFLA.authRequest are synonyms within the SOAP request context.



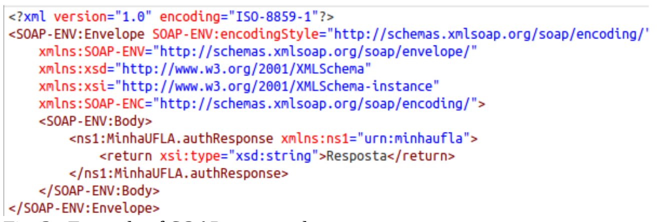
```
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
    <Body>
        <MinhaUFLA.auth xmlns="urn:minhaufla">
            <token>[string]</token>
            <ip>[string]</ip>
        </MinhaUFLA.auth>
    </Body>
</Envelope>
```

Fig. 2. Example of SOAP requisition

REST is an abstraction of the principles that make the World Wide Web scalable and allows us to offer services identified by a *Uniform Resource Identifier* (URI). HTTP methods, such as GET, POST, DELETE and PUT define the operation to be executed over a single or multiple records. For instance, GET returns a record, POST inserts a new record,

[1] https://sourceforge.net/projects/nusoap/
[2] https://chrome.google.com/webstore/detail/wizdler/oebpmncolmhiapingjaagmapififiakb

DELETE removes one and PUT updates an existing record [7].



```
<?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. Example of SOAP return value

Therefore, SOAP requires a XML structure that must be respected within the service request and response and REST allows the communication pattern to be defined by the interested parties [9]. Figures 4 and 5 show the same message using SOAP-XML e REST-JSON. We can infer that the data size will be bigger using SOAP-XML because the message content will be surrounded by the SOAP protocol structure, while using REST-JSON, this does not happen.

Requests using REST-JSON were the best alternative for mobile devices in tests that included also REST-XML and SOAP in different operational systems (iOS 8.0, iOS 8.1, Android 4.4.2 and the iOS simulator) [6]. It was observed that REST-JSON requests are 9 to 30 times faster than SOAP requests, both in processing and transmission time [6]. That is due to the fact that the XML structure (for SOAP and REST-XML) can be up to three times larger than the JSON structure for the same data, besides the fact that the XML processing is more costly in computational terms [6].

III. Problem Definition

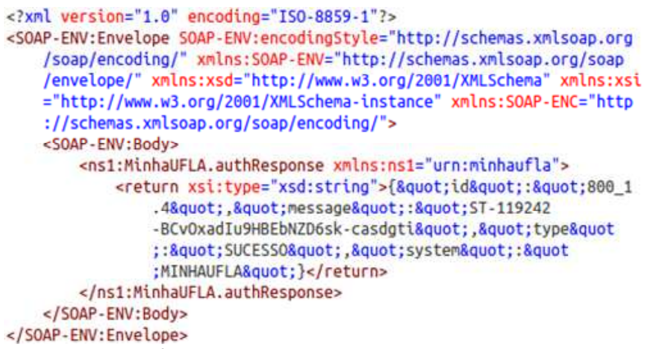
The integration architecture in the Federal University of Lavras (UFLA) was developed using the PHP[3] programming language and designed to [8]: i) have security mechanisms, ii) have the relationship between services and its providers based on the concept that there will be many providers with few services each, grouped according to the systems, which would make it easier to have isolated maintenance; and iii) use SOAP.

Between the end of 2015 and the beginning of 2016, a new app for Android based mobile devices called MinhaUFLA (myUFLA) began to be developed [3]. This app has options that are common to students and professors and options that are specific for each profile. For instance, in the student profile, the user can see the grades in each exam and the geolocation of each classroom; and in the professor profile there are options to check time and place of each class and to insert student attendance information [4]. The app should use data that already exists in the University information systems and that is offered by the WSI.

Therefore, the following issues were analyzed:

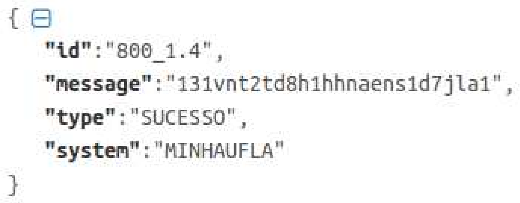
- The mobile app show, get and manipulate data using the integration architecture in order to comply with the security, manutenibility and scalability requirements and to use the existing monitoring mechanisms;
- Given that mobile devices have limited resources when compared to conventional computers, in terms of energy source, Internet bandwidth and processing capability, the MinhaUFLA app was designed to use the minimum processing capability and to transfer as little data as possible through the Internet.

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



```
<?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. Example of a SOAP-XML message.



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

Fig. 5. Example of REST-JSON message.

Based on this analysis, we could see that the integration architecture which uses SOAP and the use of REST-JSON by the applications was not compatible. Hence, the solution was to develop a SOAP/REST integration layer to allow the app to communicate with the WSI through the integration architecture.

IV. Implemented Solution

Given the problem described in Section III, it was designed a SOAP/REST integration layer to allow the integration of the mobile app, receiving REST/JSON requests coming from the app and converting to SOAP for the integration (and vice-versa), as described in the next subsections.

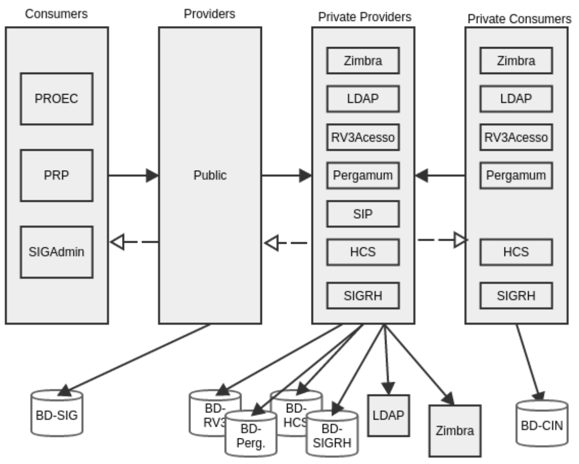


Fig. 6. Integration architecture: providers and consumers

A. Communication Standard between Consumers and Providers

In the integration architecture at UFLA, there is a division between Providers and Consumers. Providers offer Web Services, while Consumers use those Web Services that the others provide.

Each provider is related to a specific system. For instance, Pergamum, the library system, has its own provider and a consumer that uses the data update web services that its provider offers. This relationship can be seen in Figure 6, which represents the integration architecture prior to the MinhaUFLA app.

The communication between providers and consumers is made using a JSON object with 4 attributes: *id*, *message, type* e *system*. This object is encapsulated by the structure intrinsic to using SOAP.

There is also a sequence in the requests to the WSI that must be respected (Figure 7). In the example shown in Figure 7, the consumer authenticates into the provider using a *token* (password) previously defined. Next, the provider returns a JSON object containing a SID (session id). After the authentication, the consumer can access other services using the SID [8].

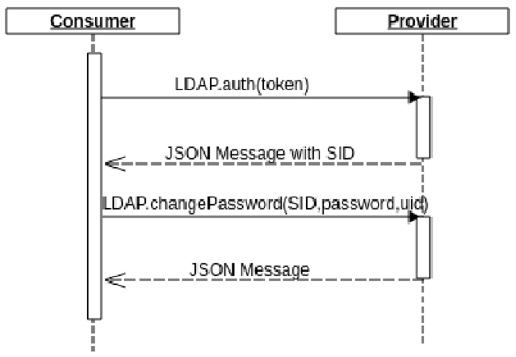


Fig. 7. Communication flow of the authentication of a Consumer and a Provider.

While communicating with the WSI, there must be a request to the *auth* service which corresponds to the authentication with the desired service provider and which considers the *token* (used password) and the origin IP. In the communication between the mobile app and the WSI there can be another step after the authentication which is the *login* service (Figure 8). This step exists only for a few services. While *auth* manages the permission for the app to access the WSI, *login* manages the permission for the user to access certain services inside the application.

B. Integration Layer: REST/SOAP

The integration layer "translates" from REST-JSON to the SOAP protocol used by the WSI and vice-versa. This way, changes within the implementation of the WSI were minimized. When designing this integration layer, called

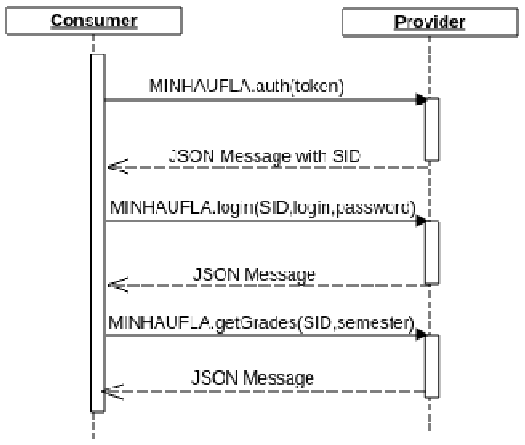


Fig. 8. Communication flow of the web service that returns student grades

within DGTI-UFLA "REST-JSON layer", it was considered the following aspects:

i) service offer complementation by the WSI that offer them using only SOAP; and

ii) security and scalability requirements in case other mobile app initiatives or even other Web or desktop applications need the updated information.

This way, the updated architecture can be represented as seen in Figure 9, where the applications access the REST/SOAP layer which makes a SOAP requests to the WSI - being able to access services from providers such as SIG, Pergamum, RV3/Acesso and HCS, which would return the required information through the same path used in the reverse way, reaching the applications.

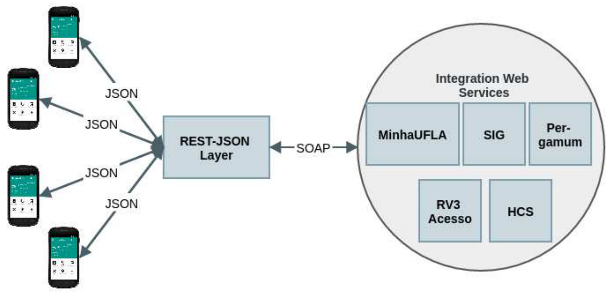


Fig. 9. Architecture using REST-JSON layer

C. Implementation

First, the Web Services for MinhaUFLA as well as the other Web Services for the WSI were developed using SOAP. Such Web Services correspond, for instance to the activities of checking grades in activities of a specific class, remaining credits in the university restaurant and checking academic timetables. These web services should be accessed from the application by the REST-JSON layer.

In order to implement the REST-JSON layer, we used the PHP programming language together with the Laravel framework, in its 5.2 version.

The PHP language was chosen due to an existing standard within DGTI-UFLA on internal systems and the Laravel framework was chosen due to the fact that it creates *Restful* web services.

Even though Laravel has a lot of useful resources to web development, it does not have the native resources to perform SOAP requests. So, in order to perform the requests from the REST-JSON layer to the WSI, it was used an integration package from SOAP to Laravel, called laravel-soap[4].

Since Laravel is a MVC (*Model View Controller*) framework, it is necessary to create a model and a controller. In Laravel it is not mandatory to develop a view, that is, the information can be made available directly from a controller.

Hence, it was developed a generic model called *WSClass* to access the Web Services that use SOAP and a controller specific for the Web Services used by the MinhaUFLA app. The specific web services pass the configuration parameters to the model and the model accesses the SOAP web service. The return from the web services is then returned by the controller, being forwarded to the service consumer application.

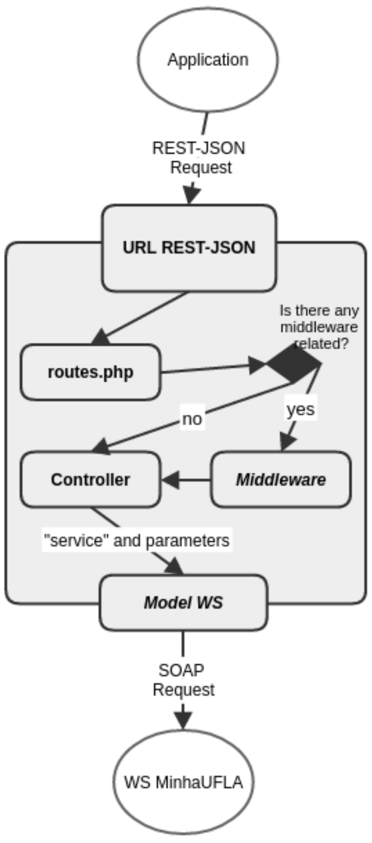


Fig. 10. Flow of a request from the application to the web services related to the MinhaUFLA

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

In the particular Laravel context, it was also developed a middleware, a concept used in Laravel to encapsulate requests, allowing it to execute tasks relating to a specific controller (before or after, depending on the configuration).

This middleware evaluates the existence of a POST variable called *service*, which defines the SOAP web service to invoke. The Laravel routing is flexible and can be configured to use the middleware and to redirect the request to a specific controller. Figure 10 shows the internal flow of the REST-JSON layer.

Considering the flow presented in Figure 10 in a context particular to Laravel for the creation of a new REST-JSON web service, it is necessary to create a new route in the *routes.php* file. Then, it should be created a controller that must instantiate a WS model passing the configuration parameters such as the SOAP provider, the WSDL URL and invoke the *access* method passing the web service name and the required parameters. In case there is any validation to be performed in all requests, such as some type of filter, cryptography, malicious code injection prevention, it is suggested the creation of a new middleware.

V. Evaluation Solution

The goal of this work is to demonstrate how this integration between the mobile app and the integration environment was done, while gathering some interesting data on this integration.

As designed, the monitorability, maintainability and scalability were extended to the layer, making it possible to monitor the integration, even using the REST/JSON layer. This is due to the fact that the layer itself invokes the WSI, which record the requests/responses and use the same physical server configured with HTTPS.

We performed tests in order to verify the average processing time for a request. It was performed 100 requests, with 5 second intervals from, a computer to a test environment installed in a server with the Ubuntu Server 14.04[5] operational system in the same network.

The processing time was measured starting from the request until receiving the response. The measurement was not made from a mobile device because the results would vary with the network that the device used to access the REST-JSON layer, resulting in data that do not accurately reflect how much time the layer costs to the request.

In order to perform the requests, it was used the tools Postman[6] (for REST requests), and Wizdler (for SOAP requests), both extensions to the Google Chrome[7] web browser. The request times were also evaluated using the Tukey HSD test (Honestly Significant Difference) [15], which measures the significant difference between group averages.

Table I shows the averages and standard deviations calculated from the response time measurement of the requests and Figure 11 shows a graphic with the result of the Tukey test.

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

TABLE I
AVERAGE REQUEST TIME

Communication	Average time (ms) + standard deviation
SOAP	64.59 +/- 2.2304
REST-JSON	165.99 +/- 2.2403

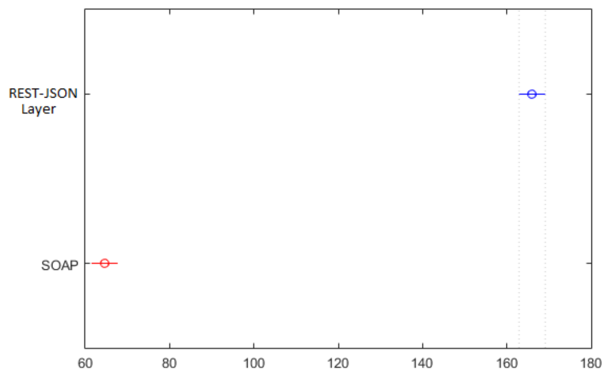


Fig. 11. Tukey Test for the comparison of the averages for the direct SOAP requests and the ones through the REST-JSON. This means that the average time for the SOAP requests are significantly better than the ones using the REST-JSON layer, but do not enjoy the benefits of the latter.

Observing Table I and Figure 11, it can be seen that the implemented solution takes 100 milliseconds longer to process a request. Since the integration architecture became hybrid, using SOAP and REST-JSON, because of the insertion of the REST-JSON layer in the call sequence, this time increase was already expected.

Even though the layer increases the request processing time, this increase represents more processing only on the server side, avoiding processing at the mobile device because the data is returned in the JSON standard.

It is also important to point out that 100 milliseconds is a small, not perceptible time that does not affect the user experience in a negative way.

It was also compared the return sizes for the SOAP and REST/JSON requests using as an example three web services developed to serve the MinhaUFLA app: auth, login and getGrades. Auth returns the smallest data set and getGrades returns the largest. That is, it was compared how much the mobile app consumes in bytes and also, how much it will have to process. The data of this comparison can be seen in Table II and Figure 12.

TABLE II
MESSAGES SIZE (IN BYTES) WHEN USING SOAP AND THE REST-JSON LAYER

Communication	*Web Services*	Size in *bytes*
SOAP	auth	747
REST-JSON	auth	97
SOAP	login	893
REST-JSON	login	179
SOAP	getGrades	2968
REST-JSON	getGrades	1000

The data observed in Figure 12 correspond to the bytes that would be consumed in bandwidth in the mobile device. In average, the messages using the REST/JSON layer use 73%

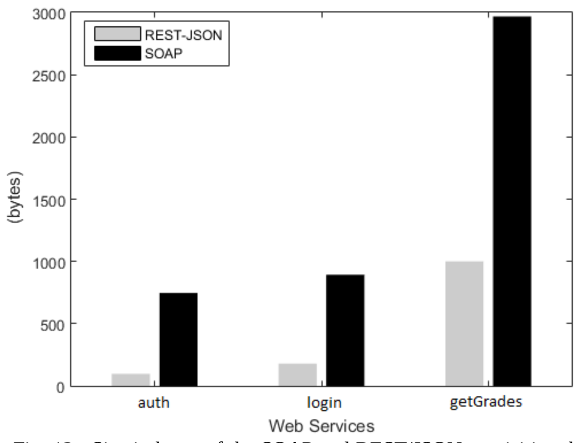


Fig. 12. Size in bytes of the SOAP and REST/JSON requisition, by *Web Service*

less bandwidth than those using SOAP.

The integration architecture using the REST/JSON layer has been in the production environment since September/2016, when the application was officially released. Until November/2016 there were 5600 active installations of the app performing in average 54.000 requests per day. The app has 1788 evaluations in Google Play, achieving an average of 4.8 stars out of the maximum 5[8].

VI. CONCLUSIONS AND FUTURE WORK

Using web services to offer functionality through the web has become more common each day due to its flexibility and security when compared with other ways to offer functionalities and because of its scalability. The most common way to offer this functionality is using SOAP or REST.

At UFLA, there was a set of web services for the systems integration and to offer functionalities. The integration web services use SOAP, because they were designed for the integration of systems with no restrictions concerning processing, Internet and other resources. Nevertheless, with the addition of an institutional mobile app, there was an increase in the complexity of the existing integration on the issues of processing power and data consumption restrictions that apply to mobile devices.

The solution developed, called REST/JSON layer, imposes a increase in average of 100ms (or 0,1 seconds), consumes 73% less bandwidth than its SOAP counterpart and, consequently, required less processing and energy expenditure from the mobile device, because processing XML is slower and requires more CPU than processing JSON [6].

Among the positive points of the REST/JSON layer, we can highlight the following:

- Web service offer to mobile devices with low time cost when compared to a common SOAP request;
- Centralization of the web services offer by the WSI was kept. That is, even though the Web Service was offered using REST, initially the web service would

[8] Data from April/ 2017

be developed and made available through SOAP through the WSI and the REST-JSON would only have the role of transcribing the REST-JSON requests to SOAP and vice versa;

- Mechanisms existing in the integration architecture, such as monitorability, maintainability and scalability were extended to the REST/JSON layer;
- Minimum additional work is necessary for the offer of web services also in REST/JSON (when compared to the necessary work to offer the web services only in SOAP).

As negative points of the REST/JSON layer, we can point out the following:

- Increase of about 0.1 seconds in request time;
- Increase in maintenance complexity, because the layer was developed by using technologies that are different from the ones used in the WSI. Nevertheless, the architecture used offers important positive points in detriment to the less important negative points, as discussed in Section V.

Hence, the problems pointed out after the analysis of the issues were solved with the implementation of the REST/JSON layer. The existing integration architecture changed to allow the integration of the app without interfering in the offer of the web services and in the integration mechanism used until that point, keeping its characteristics and mechanisms of security, monitoring, flexibility and scalability. The consumption of energy and Internet bandwidth by the mobile devices were made in JSON and the SOAP request processing was performed by the application server.

Given that the performance of REST/JSON requests is a lot faster and less burdensome to the CPU, we suggest the following future works:

- To perform new studies on REST-JSON (or other related that may arise);
- The evaluation of the technologies that support HTTP requests;
- To perform internal tests with JSON/REST and other technologies;
- The study of the possibility of rewriting the web services in JSON/REST or in another more efficient technology.

References

[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/>. Accessed in: 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/>. Accessed in: 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

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

[18] Efraim Turban, Dorothy Leidner, Ephraim Mclean, and James Wetherbe. Tecnologia da Informação para Gestão-: Transformando os Negócios na Economia Digital. Bookman, 2010.

[19] 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.

[20] 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.

[21] W3.org. Extensible Markup Language (XML). Disponível em <https://www.w3.org/XML/>. Accessed in: 06 março 2017., 2016.

[22] 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.