Laboratory as a Service (LaaS): a Model for Developing and Implementing Remote Laboratories as Modular Components

Mohamed Tawfik[1], Christophe Salzmann[2], Denis Gillet[2], David Lowe[3], Hamadou Saliah-Hassane[4], Elio Sancristobal[1], and Manuel Castro[1]

[1] Electrical & Computer Engineering Department, Spanish University for Distance Education (UNED), Madrid, Spain

[2] School of Engineering, Swiss Federal Institute of Technology in Lausanne (EPFL), Lausanne, Switzerland

[3] Faculty of Engineering and Information Technologies, University of Sydney, Sydney, Australia

[4] TELUQ, University of Quebec, Montreal, Canada

***Abstract*—This contribution introduces a novel model, Laboratory as a Service (LaaS), for developing remote laboratories as independent component modules and implementing them as a set of loosely-coupled services to be consumed with a high level of abstraction and virtualization. LaaS aims to tackle the common concurrent challenges in remote laboratories developing and implementation such as inter-institutional sharing, interoperability with other heterogeneous systems, coupling with heterogeneous services and learning objects, difficulty of developing, and standardization.**

***Index Terms*—component-based remote laboratories, LaaS, modular remote laboratories.**

I. INTRODUCTION

In the last decade, we have witnessed a significant proliferation of remote laboratories [1, 2], unconstrained by temporal or geographical considerations, in all fields of education—particularly engineering and applied science education—thanks to the exponential revolution of digital technologies. Initial concerns were focused on expanding their application range and dealt with commonplace issues such as security, scheduling, and bandwidth, which eventually, and to a great extent, have been overcome.

Later, the general conclusion from several empirical studies [3-5] was that learning outcomes depends on the exact instructions given to group and the different patterns of work and collaboration regardless of the laboratory format. Thus, current concerns are focused on issues related to remote laboratories delivery format and their pedagogical impact. These issues encompass their integration with heterogeneous educational systems and coupling with other services and learning objects in order to yield a rich scaffold educational environment and hence better learning experience and outcomes. In addition, such integration will promote sharing laboratories across institutions and hence more availability and cost offset.

In response to these needs, this paper proposes a novel model, Laboratory as a Service (LaaS), for developing and implementing modular remote laboratories efficiently with a high level of abstraction and virtualization. The goal is to:

1) Define an organized manner for sharing remote laboratories globally among institutions.
2) Allow wrapping remote laboratories in any heterogeneous application container (e.g., widget, applet, or any Web client) independently of the underlying technology adopted in both, as well as, their coupling and mashing up with heterogeneous services (e.g., learning objects) across the Web.
3) Facilitate maintenance, reusability, and leveraging legacy equipment.
4) Allow interchangeability of components between provider and consumer—seamlessly and programmatically—insofar as consumer could contribute with one or more component instead of the fully-reliance on the provider's equipment and facilities.
5) Promote online experimentation and discovery in either every day's formal or informal contexts.
6) Set principles for a first global standardized design pattern—for remote laboratories development and implementation—to be followed and adopted by remote laboratories developers.

The rest of the paper is structured as follows: *Section II* discusses the previous efforts and the related works, and outlines the novelty of the proposed solution. Prior delving into the description of the LaaS model, *Section III* describes the modular remote laboratories concept. *Section IV* describes the proposed LaaS model which builds on top of the modular remote laboratory concept. *Section V* provides a demonstrative case study example on the implementation of LaaS. As a Proof-of-Concept (POC), in *Section VI* the proposed theory is applied to the real-world by developing the first modular remote prototype. Finally, a conclusion is drawn in Section VII along with the future works.

II. RELATED WORKS

Earlier attempts to deliver remote laboratories as a service can be found in [6-10] . In [6], the functions of commercial instruments based on Virtual Instrument Software Architecture (VISA) and Interchangeable Virtual

Instruments (IVI) were listed in Web Services Description Language (WSDL) files and registered in a Universal Description, Discovery and Integration (UDDI) to be allocated and consumed by Simple Object Access Protocol (SOAP) Web service. A similar approach for controlling instruments online using SOAP Web service is found in [7, 8]. In [9], a simple experiment was developed in LabVIEW and delivered as Representational State Transfer (REST) Web service, to be consumed using Asynchronous JavaScript and XML (AJAX) calls. A similar approach based on REST Web service and AJAX is found in [10].

A distinctive approach was adopted in [11], where a further effort have been realized in order to add intelligence to remote laboratories at the server side and to make little or no assumption about the client. The underlying communications in this approach were realized using Websockets owing to its efficiency and high transmission rate. On the other hand, to promote compatibility with different client applications.

It is also worth noting that the acronym LaaS has been pronounced in the literature for almost three years few times, with two different interpretations. The first interpretation refers to the cloud computing and the Anything as a Service (XaaS) concepts as described in [12-16]. In these approaches, however, the difference between cloud computing and remote access is still blurred. Yet, there is no clear application of the cloud computing principles on real physical laboratories that might be distributed at various universities globally.

The second interpretation—the interpretation assumed in this paper—simply refers to the delivery of remote laboratory as a service that can interoperate with heterogeneous systems and services. The second interpretation is more generic and can be implemented many ways. For instance, in [17], Web service was adopted in conjunction with a proprietary Lab Description Language (LDL) developed by the author in order to achieve interoperability.

Even though, the solution proposed in this paper (LaaS) builds on top of these efforts, it has four main distinctive aspects. The first aspect is that Web service in LaaS is a method and not a solution itself and its adoption is not necessary. For instance, for data streaming (e.g., video and measurement streaming) low level protocol communications are implemented instead. The second and most important aspect is that LaaS goes further beyond abstracting the functions of the laboratories; it implies their development as a set of independent component modules in order to allow interchangeability of components between providers and consumers—seamlessly and programmatically—insofar as consumer could contribute with one or more component instead of the fully-reliance on the provider's equipment and facilities. The third aspect is that LaaS contemplates the future Web and the next generation learning environment in terms of: (1) seamlessly allocating and importing services; (2) bringing objects to the Web; and (3) mashing up and coupling services together—which was possible, in part, thanks to the modular remote laboratories concept. The fourth and last aspect is that LaaS is meant to be a model that addresses the development of remote laboratories, as well as, their implementation process broadly—which entails the relation between consumers, providers, and service broker, as well as, the format of exchanging information and resources between them. As a Proof-of-Concept, a modular remote laboratory was developed successfully and implemented according to the LaaS model.

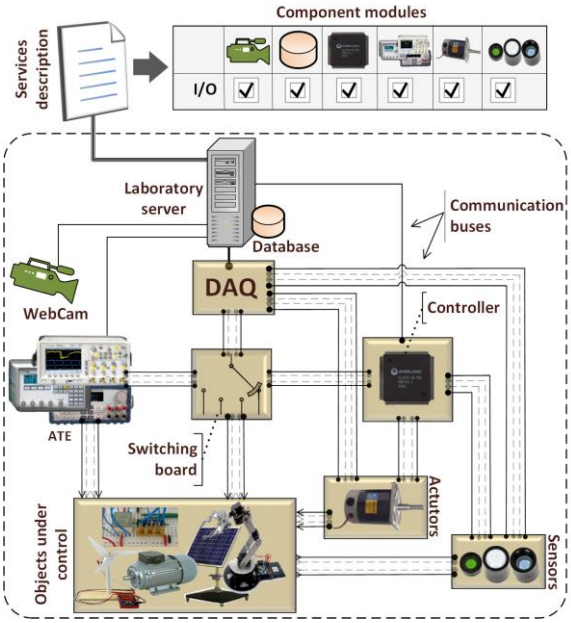

Figure 1. Modular remote laboratories architecture.

III. MODULAR REMOTE LABORATORIES

Consider the generic and common remote laboratory architecture shown in Figure 1. Typically a remote laboratory consists of a laboratory server—which is connected to all the equipment and hosts the control software—in addition to any combination of the on-demand components. The control software can be developed either from scratch using a multipurpose programming language (e.g., Java, C#, or C/C ++) or using a commercial solution, commonly LabVIEW or MATLAB. The Data Acquisition (DAQ) board acts as an interface between the laboratory server and the equipment that don't support direct interface to the computer. A Webcam is used for video live streaming. Commercial Automatic Test Equipment (ATE) is used for specific signal generation or acquiring tasks. Standard connectors are used for connecting components directly to the laboratory server without intermediaries and they encompasses Universal Serial Bus (USB), LAN-based eXtensions for Instrumentation (LXI), PCI eXtensions for Instrumentation (PXI), and AdvancedTCA Extensions for Instrumentation and Test (AXIe). Sensors and actuators are used to convert physical parameters from the objects under control to electrical signals, and vice versa, respectively. A switching board is used for remote switching or wiring any terminals either from the objects under control or the ATE. Some applications might require a controller—in addition to the laboratory server—for a specific task. Commonly used controllers are either microcontrollers or Field-Programmable Gate Arrays (FPGAs) [18, 19].

Modular remote laboratories or component-based remote laboratories are based on interchangeable component modules that expose their I/O terminals or their I/O connectors (i.e., if they physically don't exist or unavailable) in an independent and an abstracted way.

Some components can be modularized and some are fixed and cannot be modularized or interchanged programmatically (e.g., laboratory server and DAQ board). The idea beyond modularizing remote laboratories is to facilitate maintenance, reusability, and interchangeability of components seamlessly and programmatically.

In this sense, if the I/O terminals and connectors of all the component modules of a remote laboratory are provided in a "service description file" in order to allow consumers to get clues on them as shown in Figure 1, the consumer would be able to consume them separately. Furthermore, if one of the component modules is not available and the appropriate I/O connectors are provided instead, the consumer could replace this module with his/her own one instead of the full-reliance on the provider's equipment. For instance, a remote laboratory for image processing may expose an API to allow user to connect his/her camera capture. The image will be transmitted to the laboratory for processing and then return back to the user.

IV. LABORATORY AS A SERVICE (LAAS)

LaaS is a model for developing and implementing modular remote laboratories efficiently. It builds upon the modular remote laboratory concept and implies the delivery of the entire laboratory functions and components in the "service description file" as a set of abstracted services. LaaS follows the Service Oriented Architecture (SOA) and fulfills its essential requirements, which are: (1) interoperability of services regardless of their platform, operating system, and programming language; (2) description of services, their characteristics, and the data that they exchange in a clear and unambiguous manner that allows a potential consumer to allocate and consume them; and (3) access to services by means of standard communication protocols and common format messages. LaaS defines the relation between laboratory providers (i.e., providers of the "service description files"), service broker repository or market place (i.e., Web portal in which "service description files" are indexed), and laboratory consumers (i.e., who build an end-user application upon the provided services). LaaS merges features from cloud computing—in terms of consuming services on-demand with minimum restrictions and higher virtualization—and features from grid computing—in terms of global distribution. LaaS embraces the Web of Things (WoT) in terms of coupling laboratories with heterogeneous services and bringing objects to the Web for all spectrum of needs—in either formal or informal contexts.

From the cloud computing perspective, "service description files" should be registered at an intermediary and provided to users on demand abstracting to him the physical layer and the technical concerns. However, the laboratory hardware would be hosted physically at their distributed providers but allocated and discovered at the broker. Thus, LaaS model is partially cloud and partially distributed and servers of each laboratory will still be located at their provider's institution. The cloud will be a global semantic repository server denominated "market place" as depicted in Figure 2. The market place is a repository that at least creates providers or institutions profiles and supports semantic Web technologies—or at least provides an enhanced search-engine. Once the consumer has imported the LaaS he/she wants, he/she would be able to consume its APIs and communicate directly with the equipment without any extra layers in between.

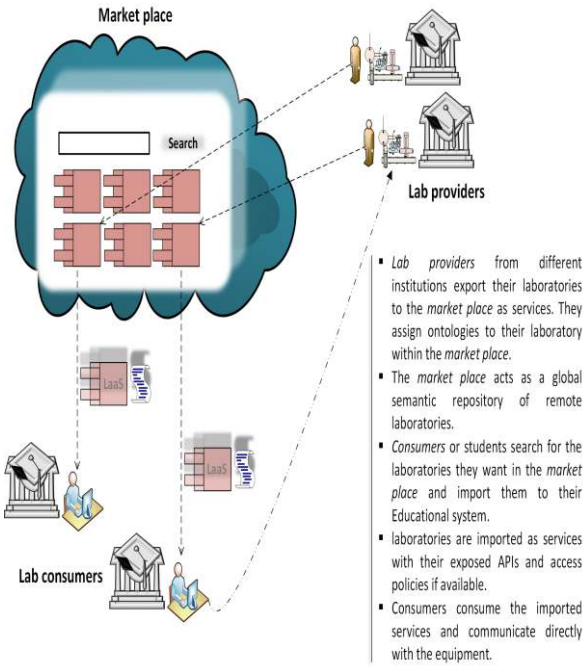

Figure 2. LaaS model.

Developers or consumers are free to develop their own Graphical User Interface (GUI) in form of Widget, applet, or any Web client, using their preferred programming language and technologies upon the provided services in the "service description file". Furthermore, consumer might consume part of the provided services to develop widget as GUI for entering values and the other part for developing another widget as a GUI for drawing graphs from the received measurement. Thus, LaaS will facilitate the creation of mashed-up learning services and customized Personal Learning Environments (PLEs).

External software applications provides GUI to the users by supporting interactive multimedia. Interactive multimedia can be developed by plugin-based solutions such as ActiveX controls, Java applets, and Rich Internet Applications (RIAs). Recently, several approaches endeavored to shift from plug-in solutions towards native Web technologies such as AJAX, and JavaScript [10, 20] in order to achieve maximum compatibility and mobile access. All recent Internet browsers and platforms support Ajax and there is no need to install any software on the client machine. AJAX per se does not provide the rich multimedia and graphical capabilities provided by RIAs. Both solutions usually co-exist and there is no estimation that AJAX would replace RIAs in the near future. Another approach towards open standards is found in [21], where OpenSocial widgets are built using JavaScript and Dynamic HTML (DHTML) in order to render in any widget engine or container.

If the laboratory is made available, it should be accessible unless another session is currently running by another user. Else, the consumer should contact the provider for enquiries. It is left up to the consumer to build his/her own scheduling system for a large scale

deployment with numerous groups and students. Scheduling system is out of the scope of the LaaS model as it focuses on the lowest level implementation for maximum abstraction.

A. *SOA Immplementation*

The most popular middleware technologies for implementing SOA are Common Object Request Broker Architecture (CORBA), Distributed Component Object Model (DCOM), JAVA Remote Method Invocation (RMI), Data Distribution Service (DDS), and Web services. Software specifications—targeting high level business applications—like Java Business Integration (JBI), Windows Communication Foundation (WCF), and Service Component Architecture (SCA) provide a model for composing applications that follow SOA and thus facilitate developing SOA solutions.

DCOM relies on a proprietary binary protocol of Microsoft, which isn't supported by all object models and thus hinders interoperability across platforms. Moreover, DCOM usually requires a highly administered, costly, and complex environment to implement and manage. Similarly, Java Middleware can only be implemented with the presence and requirement of Java Virtual Machine (JVM) in remote and local components of the system involved. CORBA is relatively advantageous as it allows developers to choose almost any language, hardware platform, and networking protocol. However, its development, implementation, and maintenance are very costly and it fails to provide the features of security and versioning. It is also too low-level and complicated and it has a steep learning curve. Therefore, CORBA objects have been hard to reuse effectively. A common major problem of the above mentioned solutions is that if the client has a firewall or a very restrictive server which only enables HTTP connections their communication could become impossible. Other technologies such as DDS, .NET Remoting, and Web services have quickly gained greater industry acceptance and support than predecessors like DCOM, RMI, or CORBA.

Web services have been designed to overcome the problems of platform dependence defining a standard way to exchange information through internet to an unprecedented level. Web services are based on open Web standards and broadly support interoperability across heterogeneous environments. These open standards make Web services indifferent to the operating system, object model, and programming language used. Web services are easily tunneled on HTTP in firewalls and proxies in contrast to DDS and .Net Remoting. By applying performance enhancement techniques, Web services can be the only potential candidate for defining the LaaS model. Web services are either activity-oriented—implemented by SOAP—or resource-oriented—implemented by REST.

REST Web services are preferred for hypermedia systems, e-commerce, and ad-hoc composition over the Web (mashup), while SOAP Web services are preferred in Enterprise application integration (EAI) and in sophisticated Business-to-Business (B2B) applications with a longer lifespan and advanced QoS requirements [22]. The simplicity and the high performance of REST—as well as its homogeneity with Web applications in general and mashup applications in particular—makes it the ideal candidate for implementing the LaaS model. In our case service discovery wouldn't be necessary since its role will be assigned to the market place [23].

Web services, however, are only suitable for transactions or request/response messages. For persistent connections like streaming of data (e.g., Webcam video streaming or measurement streaming), other approaches should be considered to allow server pushing like encoded over Transmission Control Protocol (TCP) or User Datagram Protocol (UDP) and the most recently HTML5 WebSocket APIs— which will help avoiding the reliance on any decoding plugins and promoting the development of Web native applications. WebSocket APIs haven't been officially standardized by the World Wide Web Consortium (W3C) yet. Nevertheless, it is recommended to rely on low level protocols such as TCP and UDP to achieve maximum abstraction and compatibility. Thus, "service description file" should include laboratory/experiment information and all its supported functions and connections.

V. CASE STUDY

The idea of the proposed theories can be briefly resumed in the following demonstrative example. Consider the following scenario shown in Figure 3, where the owner of the modular laboratory provides it as a set of

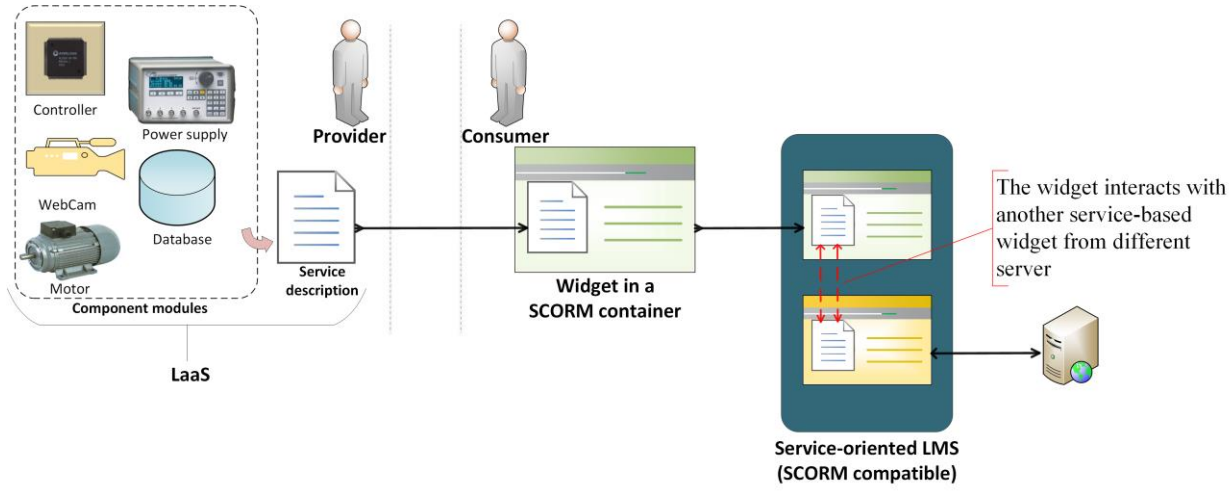

Figure 3. LaaS case study scenario 1.

services in a "service description file", according to the LaaS model, to the consumer.

In this example, the laboratory provides an experiment for implementing a control strategy on an electric motor. User uploads his/her Proportional-Integral-Derivative (PID) control program to the controller, changes the PID parameters (e.g., speed and position), and monitors the feedback effect of different control loops. The laboratory has 5 main component modules: a motor, a controller, a Webcam, a power supply, and a database. All the provided services of the laboratory (i.e., the functions, the I/O terminals, and the I/O connectors) are listed in the "service description file" (e.g., a pdf file) to be read and consumed by the consumer. The service description file is divided into two main parts:

1) The first part contains information such as:
   - A description about the experiment, how it works, objectives, etc.
   - Metadata ontologies to help the mediator or the service broker to list it in the marketplace.
   - Days and hours in which the laboratory is (or not) available if applicable.
   - Access policy and contact or query forms for applying access if applicable.
2) The second part contains all the laboratory functions, I/O terminals, and I/O connectors in form of either Web service or Uniform Resource Identifier (URI) for streaming over low level protocol or Websocket. The Webcam component module will provide an output terminal for video streaming. The controller component module will provide an input terminal to allow user to submit his/her own program code file.

The service broker (or the provider him/herself) indexes the "service description file" in the market place (i.e., not shown in the figure for simplicity). The consumer (or developer) allocates the laboratory in the marketplace and downloads the "service description file". Upon the services provided in that file, the consumer builds his/her own widget application and wraps it in a Sharable Content Object Reference Model (SCORM) container. User, afterwards, will be able to load his SCORM package in any SCORM-compatible LMS. User might not consume all the provided services in the same widget, instead he/she might link some of the provided services to other service-based object. For instance, user might want to introduce the PID parameters though his own widget but monitor the results in an external widget from different server that are specialized in building charts with regard to the received parameters.

Consider the same scenario of the previous example but with some modifications as shown in Figure 4, where the provider doesn't wish to share some of his laboratory component modules, the database and the power supply. In this case, the provider tries as possible to reduce the load on his/her own equipment and facilities and instead leaves it to the consumer to connect his/her own component modules as long as they adhere to the same adopted standard. To accomplish this, the provider instead provides the consumer with standard-based I/O connectors to connect his/her component modules. In this example, the consumer connects his/her database using an Open Database Connectivity (ODBC) connector and connects his/her power supply using IVI/VISA connector. The rest of the procedures are same as the previous example.

This aim of this example was to explain the concept of modular components and interchangeability using two components, however the idea is much broader and can be applicable to any other laboratory component.

VI. Modular Remote Laboratory Prototype

In this section, we apply the proposed theory to the real world by developing the first modular remote laboratory prototype to be delivered according to the LaaS model. The developed laboratory is a motor-tacho laboratory, shown in Figure 5, which consists of a NI USB-6009 DAQ card from National Instruments (www.ni.com), a 28GD11-222E/404E motor-tacho from Portescap (www.portescap.com), and an integrated Webcam. The software was entirely developed using LabVIEW and a numerical control code was developed using MATLAB and imported as an ".m file" into the LabVIEW code using the "LabVIEW MathScript Node".

A. *Experiment Description*

The idea of its experiment is very simple as it aims to emphasize the theory and prove its reliability rather than delving into the technical details of the experiment per se. In the experiment, user feeds the motor with a voltage

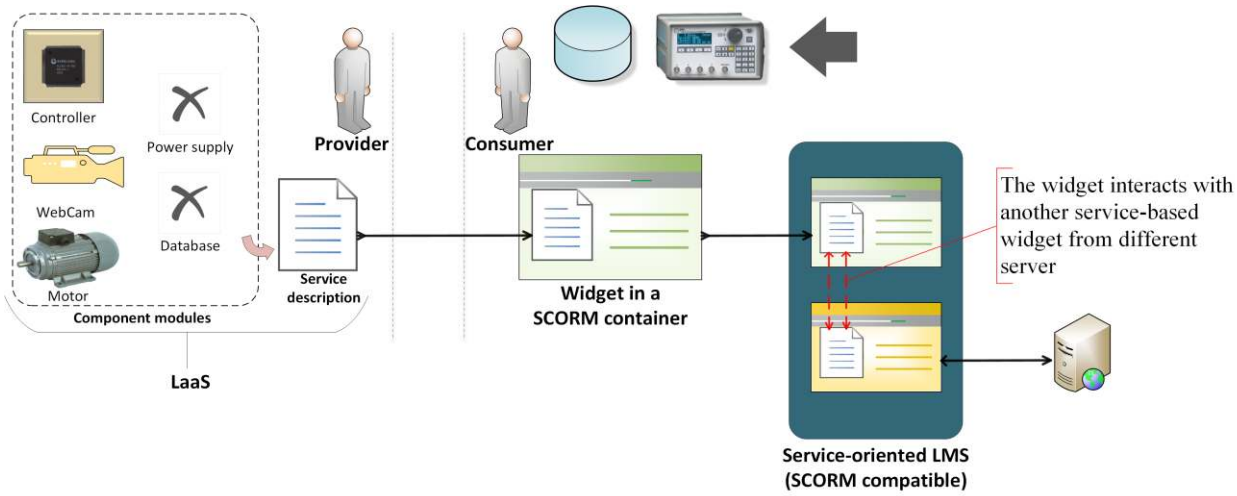

Figure 4. LaaS case study scenario 2.

Figure 5. Motor-tacho laboratory.

rang from 0V to 5V and monitors the corresponding voltage value measured by the tachometer. A control strategy is implement—using MATLAB—so that if the applied voltage is greater than 5V it will be automatically modified and introduced as only 5V. Likewise, if the applied voltage is lower than 0V, it will be introduced as 0V. The tachometer measurement is streamed continuously until the user either stops the experiment or introduces a new input voltage value. Each time the user inputs a value, it is automatically recorded and stored temporarily. Finally, when the user stops the experiment, all the introduced input voltage values—previously stored—are retrieved and copied to his/her database (i.e., not the database of the provider).

B. LaaS immplementation

The motor-tacho laboratory has three component modules as shown in Figure 6, and one of these modules, the database, allows interchangeability using a standard connector, ODBC. This laboratory requires that the consumer connects his/her own ODBC-compliant database to the laboratory server software by sending its Data Source Name (dsn) file in order to be identified. The file is transferred as follows: first, the consumer hosts the ".dsn file" in a Web server and provides the URL of the file to the laboratory server software through a Web service call; then, the laboratory server software copies the information in the ".dsn file" and use it to communicate remotely with the consumer's database.

Web service were created using the "LabVIEW RESTFul Web Service Tool" and through proxy VIs—not the main laboratory software VI—because Web service in LabVIEW cannot run with loops owing to the inherent HTTP latency compared to the loops speed. Thus, in order to keep the main laboratory software VI running and

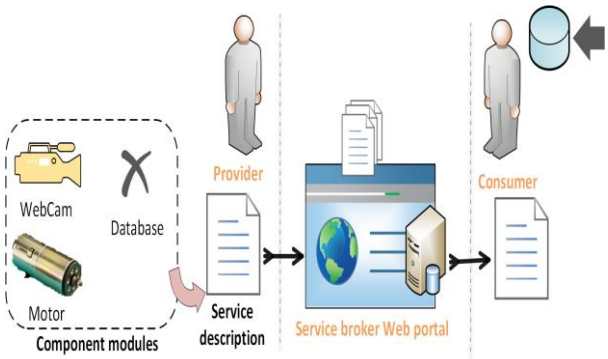

Figure 6. LaaS: motor-tacho laboratory.

accepting sequential calls, Web service should be implemented using proxy VIs that don't contain loops so that Web service calls would be handled by the proxy VIs and the proxy VIs would accordingly communicate with the running laboratory software VI. This will, in turn, allows the provider to visually monitor the main laboratory software VI running and its associated bugs. The communication between the proxy VIs and the laboratory software VI cannot be local even if both run on the same machine because the proxy VIs will be: uploaded to memory, hosted by the "LabVIEW Application Web Server", and deployed independently of the LabVIEW environment. Thus, the communication will be realized on network using "LabVIEW Shared Variables" as illustrated in Figure 7.

In the current example, two proxy VIs will be needed, method1 and method1. Each proxy VI acts as a single Web service method. The default TCP port of the "LabVIEW Application Web Server" is 800.

The tachometer streams the encoded reading on the TCP port 89 once an incoming connection is detected. The Webcam server was developed in LabVIEW using the ActiveX control distribution of VideoCapX (www.videocapx.com). LabVIEW acts as an ActiveX container and sequential methods and properties were created using the "Property Node" and the "Invoke Node" functions in order to allow streaming encoded captures over the TCP port 88.

The "service description file" consists of the following:

- **Description:** A remote laboratory for switching on a permanent magnet DC motor and reading the output using a tachometer.
- **Keywords:** Principles of control, electric machines, DC motor, power engineering, electrical engineering.
- **Provider:** Universidad Nacional de Educación a Distancia (UNED).
- **Contact:** mtawfik@ieec.uned.es, 0034000000.
- **Operation days/hours:** Open for public each week from Friday to Monday from 9am to 9 pm, otherwise contact for enquiry.
- **Additional resources and manuals:** http://...../Motortacho_User_Manuel.pdf., http://youtube.com/watch?V=6..MotortachoRemoteLab.
- **Component modules:** WebCam (provider), Motor-tacho (provider), and database (user, standard=ODBC).
- **Provided services:**

  1) http://192.168.1.66:88>>>for Webcam streaming

  i.e., Encapsulation ASF/WMV, video codec (VP8), audio codec: MPEG audio, caching 600ms.

  2) http://192.168.1.66:89>>>for tachometer reading streaming

  i.e., data is sent as a String and at 400 ms sampling rate.

  3) http://192.168.1.66:80/webServicemethod1/:InputVolt>>>for introducing input voltage value.

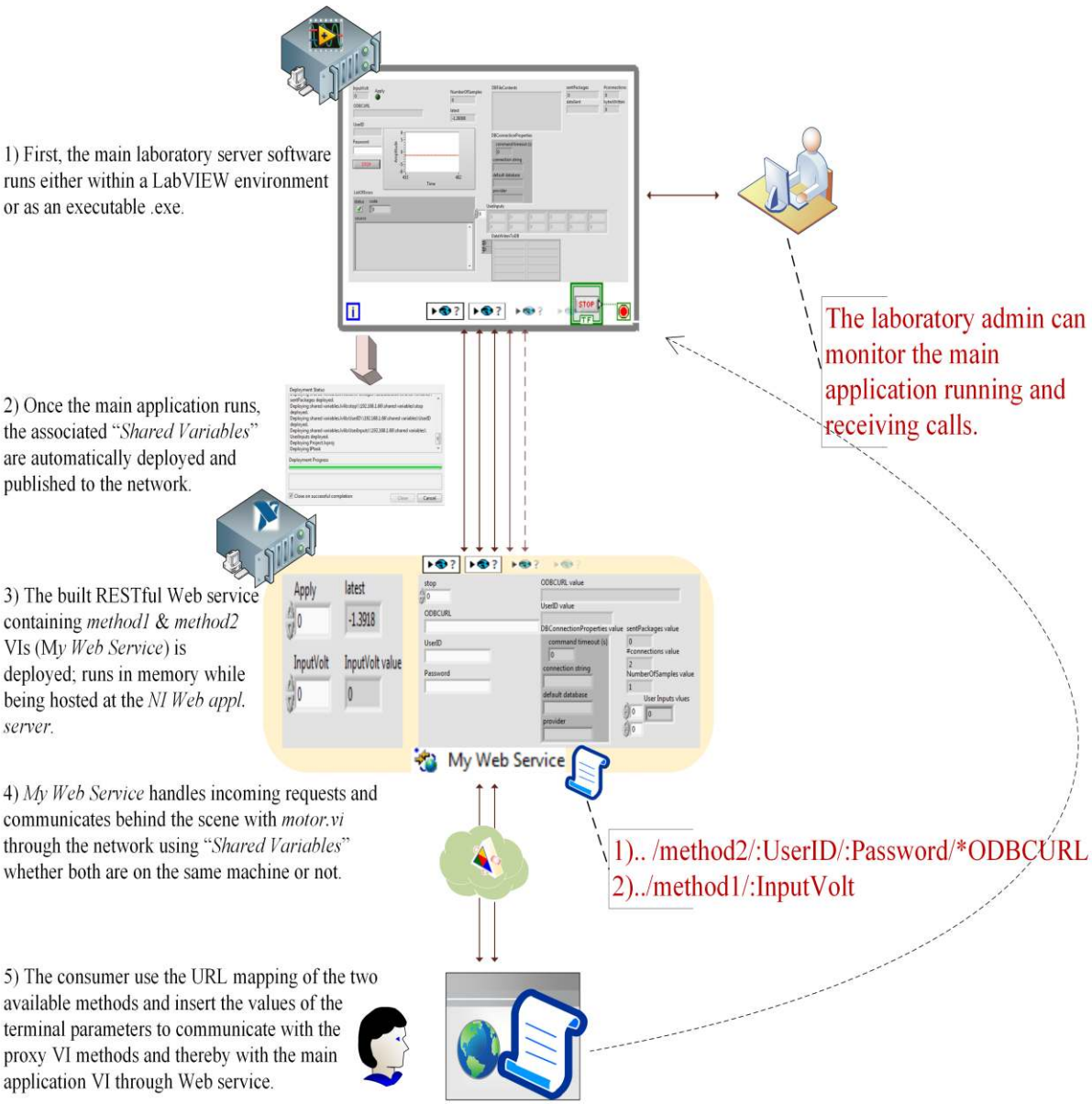

Figure 7. Topology of Web service implementation in LabVIEW.

i.e., as a response, the user gets the latest tachometer reading.

3) http://192.168.1.66:80/webservice/method2/:UserID/:Password/*ODBCURL>>>for introducing the path of the user's .dsn file and his/her database credentials.

i.e., as a response, the session is ended and a list of the input voltage values introduced by the user is copied to his/her database in a new created columns (col1 and col2). Additional information are provided as a response such as: the introduced parameters by him/her except the password value, a list of the voltage input values, database connection information, number of TCP connections and sent packages during the tachometer reading streaming, and number of samples written by the virtual DAQ channel.

C. *Consumption*

As mentioned previously, LaaS doesn't contemplate the way consumers would deploy the provided services in their end-user applications. It only assures the delivery of services in an abstracted way and basing on well-known and accepted standards. Assuming the "service description file" was deposited and indexed in a service broker Web portal, now let's consider the scenario from the consumer's perspective.

After allocating and reading the "service description file", and having understood the laboratory functions and components, we can consume the streaming URIs of the tachometer reading and the Webcam either programmatically or using a decoding client.

We can start a new session by method1 and introduce the following voltage inputs successively, 8, -3, 2, 4, and 6V. Notice that values beyond the maximum voltage value (5V) will apply only the maximum voltage value

(5V) and values beyond the minimum voltage value (0V) will apply only 0V because of the control strategy written in MATLAB.

We finally insert the URL of the ".dsn file" of our ODBC-compliant database call, along with the credentials (e.g., username=root and password=labview), via the Web service method2 (e.g., http://192.168.1.66:8000/webservice/method2/root/labview/192.168.1.22:80/odbctrial2.dsn).

As a result, the session is ended and the following parameters are shown: the database connection properties; the 2D array of the 5 introduced voltage values (i.e., this array is copied to our database); the TCP connections for tachometer reading streaming and the sent packages; and the number of samples written by the virtual DAQ channel. The response is as follows.

```
<Response>
 <Terminal>
  <Name>UserID value</Name>
  <Value>root</Value>
 </Terminal>
 <Terminal>
  <Name>sentPackages value</Name>
  <Value>79</Value>
 </Terminal>
 <Terminal>
  <Name>#connections value</Name>
  <Value>15</Value>
 </Terminal>
 <Terminal>
  <Name>NumberOfSamples value</Name>
  <Value>5</Value>
 </Terminal>
 <Terminal>
  <Name>User Inputs values</Name>
  <Value>
   <DimSize>5</DimSize>
   <DimSize>2</DimSize>
   <Name><Value>0.00</Value></Name>
   <Name><Value>8.00</Value></Name>
   <Name><Value>1.00</Value></Name>
   <Name><Value>-3.00</Value></Name>
   <Name><Value>2.00</Value></Name>
   <Name><Value>2.00</Value></Name>
   <Name><Value>3.00</Value></Name>
   <Name><Value>4.00</Value></Name>
   <Name><Value>4.00</Value></Name>
   <Name><Value>6.00</Value></Name>
  </Value>
 </Terminal>
 <Terminal>
  <Name>ODBCURL value</Name>
  <value>192.168.1.22:80/odbctrial2.dsn</value>
 </Terminal>
 <Terminal>
  <Name>DBConnectionProperties value</Name>
  <Value>
   <Name>command timeout (s)</Name>
   <Value>30</Value>
   <Name>connection string</Name>
   <Value>Provider=MSDASQL.1;User ID=root;</Value>
   <Name>default database</Name>
   <Value>labview</Value>
   <Name>provider</Name>
   <Value>MSDASQL.1</Value>
  </Value>
 </Terminal>
</Response>
```

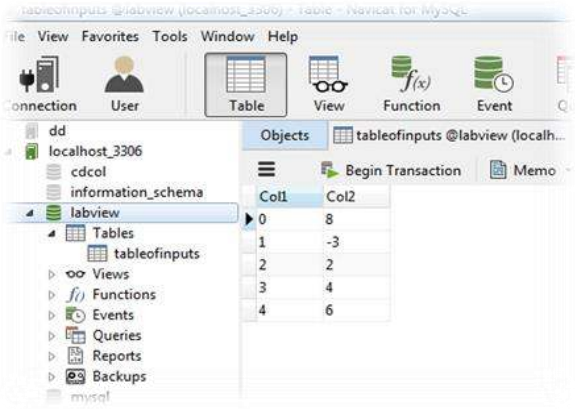

Figure 8. Topology of Web service implementation in LabVIEW.

The create columns (col1 and col2) and the data written to our database (the array of input voltage values; the 5 inserted values) is shown in Figure 8.

Recall that during the execution of the experiment sessions, the administrator or the lab provider can monitor the main application VI running and that the main application should be always running otherwise the calls of the proxy VIs (implemented as Web service) wouldn't get response and consequently wouldn't answer the user's petitions. Figure 9 shows the response of step 4 at the provider's main application VI.

VII. Conclusion and Future Works

In this contribution, two novel concepts were introduced: (1) modular remote laboratories, which aims to convert laboratories into modular components in order to facilitate maintenance, reusability, and interchangeability of components seamlessly and programmatically; and (2) LaaS model: which aims to convert modular remote laboratories into a set of services to be consumed by users with a high level of abstraction and virtualization. It defines, as well, the broader implementation mechanism of these laboratories. A broad case-study example that resumes both concepts was provided. Afterwards, a practical implementation of both concepts was provided, where a simple modular remote laboratory prototype was successfully developed and

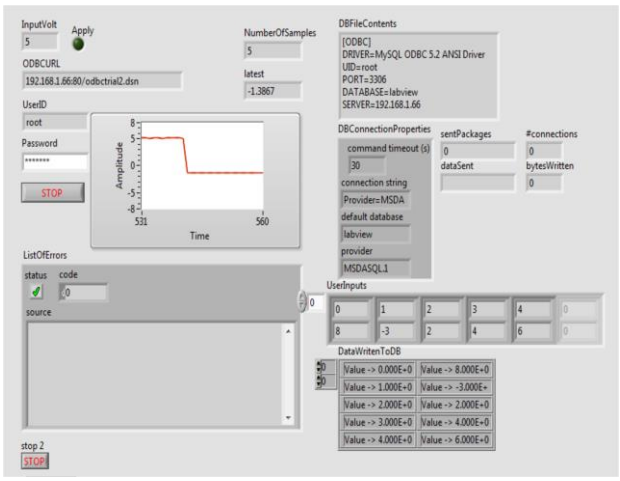

Figure 9. Response to method2 at the main application VI.

consumption results were provided.

From the low level perspective, future works will be focused on expanding the application range and modularizing different kinds of remote laboratories with different components and operation scenarios in order to investigate further issues and discover further solutions. As well, future works will be focused on implementing a scheduling mechanism using extra layers while maintaining the service description file as abstracted as possible in accordance with the premise of the LaaS model.

From the high level perspective, the final goal is to set bases towards an acceptable standard model to which developers and laboratory providers could adhere to. For this purpose, further collaboration will be realized with the IEEE P1876™ Standard for Networked Smart Learning Objects for Online Laboratories Working Group and the Global Online Laboratory Consortium (GOLC) consortium.

ACKNOWLEDGMENT

Authors would like to acknowledge the support of the following projects: e-Madrid (S2009/TIC-1650), RIPLECS (517836-LLP-1-2011-1-ES-ERASMUS-ESMO), PAC (517742-LLP-1-2011-1-BG-ERASMUS-ECUE), MUREE (530332-TEMPUS-1-2012-1-JO-TEMPUS-JPCR), and Go-Lab (FP7-ICT-2011-8/317601). As well, Authors would like to acknowledge the support of the VISIR Community, the GOLC consortium, and the IEEE P1876™ Standard for Networked Smart Learning Objects for Online Laboratories Working Group.

Last but not least, authors would like to acknowledge the project s-Labs (TIN2008-06083-C03-01) for financially supporting the research visit of Mr. Tawfik at UTS and EPFL, which resulted in developing the theory and the prototype.

REFERENCES

[1] M. Tawfik, E. Sancristobal, S. Martin, G. Diaz, J. Peire, and M. Castro, “Expanding the Boundaries of the Classroom: Implementation of Remote Laboratories for Industrial Electronics Disciplines,” *Industrial Electronics Magazine, IEEE,* vol. 7, no. 1, pp. 9, March 19, 2013.

[2] M. Tawfik, E. Sancristobal, S. Martin, R. Gil, G. Diaz, J. Peire*, et al.*, “Virtual Instrument Systems in Reality (VISIR) for Remote Wiring and Measurement of Electronic Circuits on Breadboard,” *Learning Technologies, IEEE Transactions on,* vol. 6, no. 1, pp. 13, March 12 (First Quarter) 2013.

[3] M. F. Aburdene, E. J. Mastascusa, and R. Massengale, "A proposal for a remotely shared control systems laboratory," presented at the Frontiers in Education Conference, 1991. Twenty-First Annual Conference. 'Engineering Education in a New World Order.' Proceedings., West Lafayette, IN, 1991, pp. 589 - 592

[4] I. E. Allen and J. Seaman, "The ninth annual survey of Solan-C: Going the Distance: Online Education in the United States, 2011," The Sloan Consortium, Newburyport 2011.

[5] J. E. Corter, S. K. Esche, C. Chassapis, J. Ma, and J. V. Nickerson, “Process and learning outcomes from remotely-operated, simulated, and hands-on student laboratories,” *Computers & Education,* vol. 57, no. 3, pp. 14, November, 2011.

[6] H. Saliah-Hassane, D. Benslimane, I. D. L. Teja, B. F. L.K., G. Paquette, M. Saad*, et al.*, "A General Framework for Web Services and Grid-Based Technologies for Online Laboratories," presented at the The International Conference on Engineering Education and Research (iCEER) Tainan, Taiwan, 2005, p. 7.

[7] C. D. Capua, A. Liccardo, and R. Morello, "On the Web Service-Based Remote Didactical Laboratory: Further Developments and Improvements," presented at the Instrumentation and Measurement Technology Conference (IMTC), Proceedings of the IEEE (Volume:3) Ottawa, Ont. , 2005, p. 5.

[8] A. Baccigalupi, C. D. Capua, and A. Liccardo, "Overview on Development of Remote Teaching Laboratories: from LabVIEW to Web Services," presented at the Instrumentation and Measurement Technology Conference (IMTC), Sorrento, Italy, 2006, p. 6.

[9] M. Ngolo, L. B. Palma, F. Coito, L. Gomes, and A. Costa, "Architecture for remote laboratories based on REST web services," presented at the E-Learning in Industrial Electronics. ICELIE '09. 3rd IEEE International Conference on Porto 2009, p. 6.

[10] S. Dutta, S. Prakash, D. Estrada, and E. Pop, “A Web Service and Interface for Remote Electronic Device Characterization,” *Education, IEEE Transactions on,* vol. 54, no. 4, pp. 6, November, 2011.

[11] C. Salzmann and D. Gillet, "Smart device paradigm, Standardization for online labs," presented at the Global Engineering Education Conference (EDUCON), IEEE, Berlin, 2013, p. 5.

[12] V. Cheruku, "Integrating Physical Laboratories into a Cloud Environment," Master Thesis, Computer Science, North Carolina State University, Raleigh, North Carolina, 2013.

[13] R. I. Dinita, G. Wilson, A. Winckles, M. Cirstea, and A. Jones, "A Cloud-based Virtual Computing Laboratory for Teaching Computer Networks," presented at the Optimization of Electrical and Electronic Equipment (OPTIM), 13th International Conference on, Brasov, 2013, p. 5.

[14] C. R. S. d. Oliveira and I. N. d. Oliveira, "Uma proposta para a disponibilidade de Laborató-rios de Física como serviços da Computação em Nuvem," presented at the 9th International Information and Telecommunication Technologies Symposium (I2TS'2010) Rio de Janeiro, Brazil, 2010, p. 4.

[15] J. Rugelj, M. Ciglarič, A. Krevl, M. Pančur, and A. Brodnik, "Constructivist Learning Environment in a Cloud," in *Workshop on Learning Technology for Education in Cloud (LTEC'12)*. vol. 173, L. Uden, E. S. Corchado Rodríguez, J. F. De Paz Santana, and F. De la Prieta, Eds., ed: Springer Berlin Heidelberg, 2012, pp. 193-204.

[16] Y. Luo and X. Q. Zhang, “Cloud-Based Platform Building Research of Teaching Resources,” *Applied Mechanics and Materials,* vol. 347-350, pp. 4, August, 2013.

[17] S. Seiler, "Laboratory as a Service A Holistic Framework for Remote and Virtual Labs," Doctoral Thesis, Faculty of Mechanical Engineering-Department of Mechatronics, Tallinn University of Technology, 2012.

[18] M. Tawfik, E. Sancristobal, S. Martin, G. Diaz, and M. Castro, "State-of-the-art Remote Laboratories for Industrial Electronics Applications," presented at the

Technologies Applied to Electronics Teaching (TAEE), 2012, pp. 359-364.

[19] M. Tawfik, E. Sancristobal, S. Martin, R. Gil, G. Diaz, J. Peire*, et al.*, "On the Design of Remote Laboratories," presented at the IEEE Global Engineering Education Conference (EDUCON), Marrakesh, Morocco, 2012, pp. 311-316.

[20] M. Ngolo, L. B. Palma, F. Coito, L. Gomes, and A. Costa, "Architecture for remote laboratories based on REST web services," in *E-Learning in Industrial Electronics, 2009. ICELIE '09. 3rd IEEE International Conference on*, Porto, 2009, p. 6.

[21] B. Evgeny, S. Christophe, and D. Gillet, "Widget-Based Approach for Remote Control Labs," presented at the 9th IFAC Symposium on Advances in Control Education, Resort Automobilist, Russia, 2012.

[22] L. E. Moser and P. M. Melliar-Smith, "Service-Oriented Architecture and Web Services," in *Wiley Encyclopedia of Computer Science and Engineering*, B. W. Wah, Ed., ed: John Wiley & Sons, Inc, 2008, p. 8.

[23] C. Pautasso, O. Zimmermann, and F. Leymann, "RESTful Web Services vs. "Big" Web Services: Making the Right Architectural Decision," presented at the WWW '08 Proceedings of the 17th international conference on World Wide Web Beijing, China, 2008, p. 10.

AUTHORS

M. Tawfik, E. Sancristobal, and M. Castro is UNED, (*email:* mtawfik@ieec.uned,es, elio@ieec.uned.es, and mcastro@ieec.uned.es).

C. Salzmann and D. Gillet is with EPFL (*email:* Christophe.salzmann@epfl.ch and denis.gillet@epfl.ch).

D. Lowe is with University of Sydney (*email:* david.lowe@sydney.edu.au).

H. Saliah-Hassane is with TELUQ | University of Quebec (*email:* saliah@teluq.ca).