Using IoT-Type Metadata and Smart Web Design to Create User Interfaces Automatically

Luis de la Torre, Jesus Chacon, Dictino Chaos, Ruben Heradio and Rajarathnam Chandramouli

***Abstract*—The advent of the Internet of Things has generated loads of data from the devices that are now connected to the Internet. While the majority of the data corresponds to measurements done by these devices, there is a second type of information (the metadata) that provides information about the devices themselves. Most of this metadata is still underused, when used at all. On the other hand, the graphical user interfaces that allow operating and/or monitoring the connected devices from a computer or smartphone, are usually programmed from zero. However, the metadata that describes the main properties of the devices (i.e. inputs, outputs, precision, range, etc.) can be used along with smart web design techniques to automatically create these interfaces. This article proposes a framework to achieve this, and presents an application example consisting of an online lab of a servo-motor.**

***Index Terms*—Internet of Things, Online labs, Metadata**

I. Introduction

The main idea behind the *Internet of Things* (IoT) [1], [2] is to extend the Internet network so that it goes from connecting just the devices that have traditionally been in the network (computers, smartphones, printers...) to connect a much wider variety of things (such as all types of sensors, blinds, refrigerators, or locks). While this idea may now sound simple, it only became a real possibility with the right combination of mature enough technologies in: 1) WiFi and satellite communications, 2) microprocessors and 3) batteries.

However, connecting so many different things to the Internet has three consequences that must be taken into account [3]. The first one is that a huge amount of new data and communication transmissions are now appearing, and this may be difficult to digest by agents and/or the network, respectively. The second one is the variety of the data formats, which complicates even more the work of agents to read and process all the information generated by the new things that are now being connected. The third one is, for those applications that require it, the need of new *User Interfaces* (UIs) to present the information to human users and/or allow them to operate these things. This paper tackles this last issue.

A video intercom that allows you to see on your phone who is calling at your door and to open it, if you want, from wherever you may be, is a good example of an IoT application that requires both visualization of information (the video stream) and operation (a way to open the door). For this, a UI (running on the phone in this example) is required to both display the video and allow the user to open the door. In this case, the UI could be rather simple, as the bare minimum would only require a dedicated (usually rectangular) space to broadcast the video stream and see who is calling at the door, and a button to open the door if desired. However, other applications require much more complex designs and functionalities [4], [5]. An example could be a process line in a factory that can be remotely supervised and operated. In such a context, information from many different sources (cameras, proximity sensors, RFID readers...) should be displayed in the UI. Similarly, one single button would be completely insufficient to control the operation of the process line, where many actuators (motors, electrovalves, switches...) may play a crucial role.

In an IoT context, each of the elements, whether they are sensors or actuators, is an individual entity that is able to broadcast information and/or receive operation commands autonomously. These IoT devices usually provide some metadata [6] to allow other agents to: 1) know which services are available and 2) invoke them. Whether this agent is the final user (for example, an end-user UI) or an intermediate element (for example, an IoT gateway), what remains important here is that a more or less detailed information about the capabilities and communication features of the devices is available. This information, when it is complete enough, can be used to build web UIs automatically when combined with modern smart web design techniques [7], [8]. This, in turn, has the potential to solve, or at least alleviate, the third issue that appeared with the advent of the IoT, mentioned two paragraphs above.

Online Labs (OLs) [9], [10], which are virtual or real lab resources that are made available for use online, may be considered a particular use case or application of the IoT, as the basic idea of an OL is to connect lab things to the Internet. While not all OLs use IoT devices and/or approaches, they present the potential to do so. More importantly, a vast majority of legacy OLs could be adapted or converted, solely with software tools, to present IoT-type functionalities, in the sense of communications and metadata availability. Similarly, an OL can also be considered a simplified, or small scale, Industry 4.0 scenario, where the connected industrial plant or process is replaced by the lab equipment, which, in many cases, is a smaller and simpler version of an actual industrial plant (especially for Electrical Engineering and Control Engineering lab activities).

Luis de la Torre and Dictino Chaos are with the Departamento de Informática y Automática of UNED, Madrid, Spain (e-mail: {ldelatorre, dchaos}@dia.uned.es)

Jesus Chacon is with the Departamento de Arquitectura de Computadores y Automática of Universidad Complutense de Madrid, Madrid, Spain (e-mail: jeschaco@ucm.es)

Ruben Heradio is with the Departamento de Ingeniería del Software y Sistemas Informáticos of Madrid, Spain (e-mail: rheradio@issi.uned.es)

Rajarathnam Chandramouli is with the Department of Electrical and Computer Engineering of the Stevens Institute of Technology, New Jersey, USA (e-mail: mouli@ieee.org)

The rest of the work is organized as follows. Section II analyzes the related work in the literature regarding the automatic generation of UIs, both for a general context and in the online labs' field. Section III presents the solution we propose to achieve the automatic generation of UIs. While contextualized for OLs, the tools and methods presented in this section can be immediately applied to more general IoT scenarios. Section IV gives an example that uses the proposed solution with a servo-motor online laboratory. Section V discusses the conclusions of this work. Finally, Section VI provides the links to the public repositories where our solution and motivational example are available.

II. Related Work

An influential seminal work on UI generation was the SYNGRAPH system [11]. Many ulterior systems have followed SYNGRAPH's approach of producing UIs tailored to different situations by processing input abstract functional specifications. For example, the SUPPLE system [12], [13] renders UIs customized for specific devices and user models by optimizing a cost function that considers the interface's functional input specification. Other systems, such as SUPPLE++ [14] and MyUI [15], enable accessible interfaces adapted to users with low vision, with mouse impaired dexterity, etc. As a result, the focus of much research has been on defining appropriate *User Interface Description Languages* (UIDLs) that support specifying interfaces abstractly and concisely. For instance, Navarre et al. [16] provide a comparative analysis of 25 UIDLs. Examples of this trend in the IoT domain are [17], [18], [19], [20].

In contrast, other approaches (including ours) look for taking advantage of *things'* metadata to generate their interface automatically. For example, the *World Wide Web Consortium* (W3C) published the *Web of Things* (WoT) architecture and the *Thing Description* (TD) model to tackle the heterogeneity of the hardware and protocols involved in complex IoT infrastructures. Through their TDs, *Web Things* (WTs) can expose their properties in well-known web protocols. [7], [8] propose to generate WTs' UIs from their TDs. In the online experimentation area, the *Smart Device Specification* (SDS) for OLs [21] can be considered as a TD equivalent. The SDS of an OL describes what services the lab provides and, as shown in [22], [23], it can be used to generate basic UIs.

Here, we develop some ideas briefly proposed in [24]. This paper (i) describes the metadata specification of the Remote Interoperability Protocol (RIP) [25], (ii) explains how to generate OLs' UIs from RIP metadata, and (iii) provides a fully-functional open-source implementation of our approach. It is worth noting the generality of our work, applicable not only in online experimentation but also in many other contexts.

III. Proposed Solution

Adding IoT-type metadata to OLs, so that they can be considered "smart", has become a trend in recent years. The first works to address this were [21], [26], which presented a specification to turn traditional OLs into smart OLs. A subsequent effort that resulted in an IEEE standard for online learning objects was [27]. Similarly, our RIP protocol follows the same ideas of the two previous works and extends them with additional features and metadata information.

It is important to keep in mind that while RIP was initially designed for OLs, it can be applied to other use cases comprehended by cyber-physical systems and the IoT. When RIP is applied to OLs, We talk about "experiments" and "lab", but we could simply change the terms to "processes" and "industrial plant", respectively, when RIP is used in an industrial context, or to "applications" and "IoT devices" when it is used in an IoT scenario. Therefore, the proposed solution has the potential to be applied to a wide variety of domains that include: lab work for education and/or research, IoT applications such as smart homes, smart cities, smart agriculture, etc. and Industry 4.0.

In this section, an introduction to RIP and to its essential features is first presented. The last part shows how these features can be used to build a UI automatically, and describes our proposal on how a smart client, that supports RIP, can implement this possibility.

A. Introduction to RIP & to its main features

The objective of RIP is to offer a simple, yet powerful, communication solution usable from web clients. As such, RIP only uses pure HTTP standard protocols. RIP exposes metadata and input and output methods and variables related to a control program (for example, a LabVIEW VI) defined in a remote computer to monitor and manipulate some physical devices.

Next, a description of RIP's main features and of the data this protocol transfers is given.

1) Definition of experiments and general metadata: Existing OLs can incorporate RIP by simply installing a *RIP server* implementation in the lab computer or board. Several experiments can be defined within one single server, and each experiment can be associated with just one single OL model/equipment or with different ones. An experiment gets defined by the next required data, which needs to be specified in the RIP server:

- *Simulation model or control program*: Each OL experiment requires a simulation model (for virtual labs) or a control program (for real labs) that runs the mathematical simulation or handles the connection to the hardware, respectively.
- *Path to the simulation model or the control program*: Experiments must all define the path to the simulation model or the control program within the machine that hosts the RIP-enabled OL.
- *Experiment identifier*: This identifier unequivocally pinpoints a particular experiment defined in the RIP server.

While the previous data is mandatory, the following optional metadata may also be provided for each experiment:

- *Authors*: Experiments may include a list of creators or authors.
- *Description*: Experiments may include a description that provides some details about the experiment scope, possible experimental tasks, objectives, and so on.

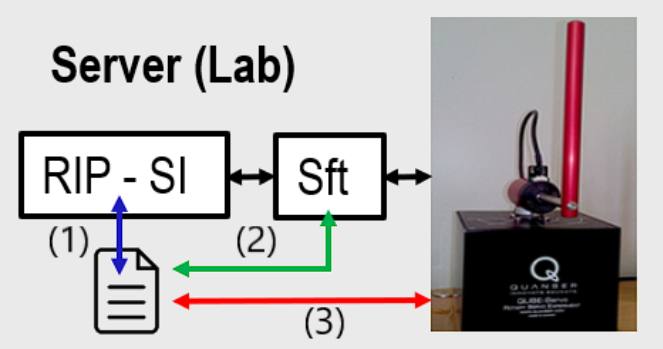

Fig. 1: Possible metadata sources: (1) custom-defined information in the configuration files of the RIP server implementation (RIP-SI), (2) lab software (Sft); either a simulation model or a control program, and (3) lab plant with IoT devices.

- *Keywords*: Experiments may include a list of keywords or related terms.
- *Cameras*: Finally, experiments may also include a list of accessible URLs that stream the video grabbed by the associated cameras. This is highly recommended for real online labs, but not used in virtual labs.

2) Gather information of input and output variables: The RIP server has a list of well-defined input and output variables from the simulation model or control program associated with each defined experiment. In this work, we understand a variable is well-defined when it provides, at least, the following information:

- *Name*: The name of the variable.
- *Description*: A brief description of what the variable is and its purpose.
- *Nature*: Whether it is an output variable, an input variable, or both.
- *Type (or format)*: Defines if the variable is a boolean, a number, a string or an array.
- *Min value*: The minimum value the variable can take. For booleans, *"false"*; for strings, and empty string (*""*); in any other case, a number.
- *Max value*: The maximum value the variable can take. For booleans, *"true"*; for strings, and empty string; in any other case, a number.
- *Precision*: The minimum quantity in which the value of a variable can be modified. For booleans or strings, an empty string; in any other case, a number.

If the simulation models or the control programs allow it, the RIP server will build the list of well-defined variables automatically from the information contained in the software. This possibility depends on the features offered by the software used in the lab computer and on whether RIP provides support for such software. Examples that currently meet both requirements are LabVIEW and MATLAB. Another option is that RIP receives this information from supported IoT devices that provide metadata. That would be the case for experiments that use real equipment, when such equipment includes these kinds of devices. If this list cannot be built automatically by the RIP server, either from the information received from the simulation model/control program or from the IoT devices themselves, RIP also provides a way to allow a human user (normally, the lab owner) to create such list for each OL experiment that requires it. Fig. 1 illustrates all these possibilities.

3) Transfer the metadata: There are two levels of metadata an agent can obtain from a RIP server. In both cases, agents can retrieve this information through a simple REST call:

- *General*: This corresponds to general information about all the experiments defined in the RIP server and about the available communication method to get more details about them.
- *Related to an experiment*: This corresponds to the information about the input and output variables (see Section III-A2). It also gets the metadata associated with that particular experiment (see Section III-A1). Last but not least, it also includes information about the built-in methods provided by RIP that an agent can call to communicate with the OL for manipulating it and reading its state (see next).

4) Write and read variables: An agent may use RIP's built-in protocol methods to manipulate the OL experiment and read its states, which is done by reading/writing output/input variables from/to its corresponding simulation model or control program. Just like in the previous case, a simple REST call is all an agent needs to perform the write and read operations.

B. Use of RIP and smart web design techniques to automatically create UIs

While RIP was originally defined to provide simple and reliable communications between web client apps and lab resources, it can also be used as a key piece to build these client applications automatically.

Thanks to the metadata provided by RIP (see Section III-A2), a smart client app can determine the complete list and features of inputs and outputs, and their associated methods to write and read them, respectively. For this, the minimum and maximum values allowed for each input and output lab variable, along with their precision, name, format, description, and available REST methods are used. Fig. 2 extends Fig. 1 to depict this process.

An important detail is that all processes illustrated in Fig. 2 take place in runtime. This way, whenever a change in the lab modifies the metadata, the HTML5 client application can be updated automatically. This means that when a user accesses the webpage in which the lab client app is embedded, it will already display the required changes. If the OL client application is open when the changes take place on the server-side, the user only needs to refresh the webpage to get the UI updated version. A couple of change examples that would produce this effect are: adding a motor to the system (which would result in having a new input), or adjusting the precision of a sensor (which would change the associated metadata for that output and so, the way it should be displayed).

The following two sections provide more details on how Steps 3 and 4 are performed to achieve the automatic generation of UIs.

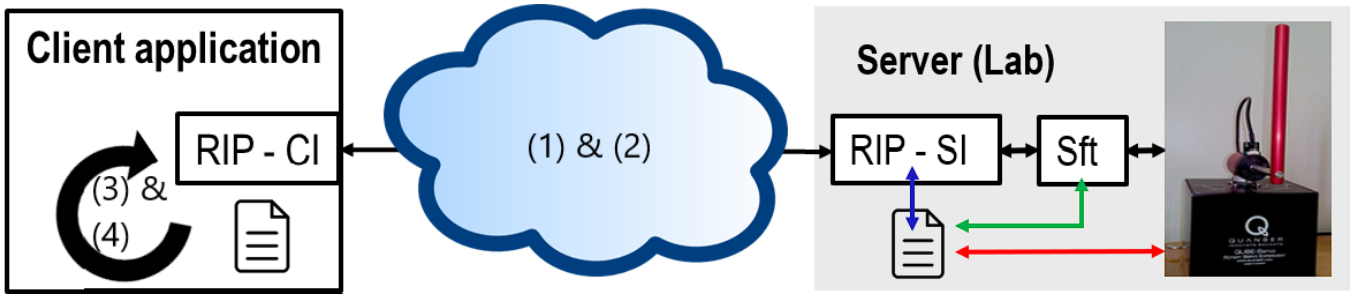

Fig. 2: Use of RIP to automatically generate the UI for an OL. (1) The client, with a RIP client implementation (RIP-CI), sends a request to ask the RIP server about the information available for an OL. (2) The RIP server sends the metadata (variables' minimum, maximum and precision values, as well as their format, names, description and methods for reading/writing them). Here, the RIP server also sends the URLs from which lab images can be grabbed, if any. (3) The client application reads the received metadata and uses the information to specify the HTML UI elements (buttons, sliders, labels, etc.). (4) The client creates all the required HTML elements and injects them into the web app on-the-fly.

1) Using the metadata to define the HTML UI elements: The part of the metadata gathered and exchanged by RIP and used to build the UI, has the following JSON format and content for a system with N outputs and M inputs:

```
{
  "outputs":{
    "list":[
      {
        "name":"output_1",
        "description":
                  "description_output_1",
        "type":"type_output_1",
        "min":"min_output_1",
        "max":"max_output_1",
        "precision":"precision_output_1"
      },
      //...
      {
        "name":"output_N",
        "description":
                  "description_output_N",
        "type":"type_output_N",
        "min":"min_output_N",
        "max":"max_output_N",
        "precision":"precision_output_N"
      }
    ],
    "methods":[
      {
        //...
      }
    ]
  },
  "inputs":{
    "list":[
      {
        "name":"input_1",
        "description":
                  "description_input_1",
        "type":"type_input_1",
        "min":"min_input_1",
        "max":"max_input_1",
        "precision":"precision_input_1"
      },
      //...
      {
        "name":"input_M",
        "description":
                  "description_input_M",
        "type":"type_input_M",
        "min":"min_input_M",
        "max":"max_input_M",
        "precision":"precision_input_M"
      }
    ],
    "methods":[
      {
        //...
      }
    ]
  }
}
```

For the sake of simplicity and briefness, the *methods* metadata in the *outputs* and *inputs* fields are not included. However, it contains all the information a client needs to create the REST calls to: 1) subscribe for getting updates on the values of the outputs, and 2) send requests for writing/modifying the inputs. Interested readers can find all the details in [25].

We extended the RIP client implementation to use the above information for building the UI elements of the web app automatically. The UI elements are defined with different HTML tags, depending on whether the variable is an input or an output, and depending on its type (provided in the metadata). The basic procedure consists of filling the HTML tag attributes with the information received as metadata. Table 1 shows which UI elements are built for each case, as well as the HTML tags that get automatically generated and injected in the web app.

While many possibilities are valid here, we found the UI elements in Table 1 offer a good solution for users to read

Nature	Type	UI elements	Basic HTML code
output	boolean	button	<input readonly type="button">
	number	label (x2)	<label>
	string	label (x2)	<label>
input	boolean	button	<input type="button">
	number	label	<label>
		input field	<input type="number">
		slider	<input type="range">
	string	label	<label>
		input field	<input type="text">

TABLE I: UI elements and HTML code correspondence for the different inputs and outputs.

and enter the data of an OL, and an example of the result can be found in Section IV. While it is not in the scope of this paper to explain in detail how all the HTML tags are generated based on the metadata, we present one of the cases. Before that, however, a few comments are needed to further explain the previous table.

- *Output booleans* do not use a <*label*> tag because the label (consisting on the *name* of the variable as it is provided by the metadata) is added to the <*input*> tag itself through its *value* attribute. To prevent users from interacting with this element, the *readonly* property is added to the tag.
- *Output numbers and strings* use two labels. The first one takes the value of the variable's *name*, as provided by the metadata. The second one takes the value of the variable's *value*, as provided by RIP's REST interface.
- *Input numbers* use a label (again, for the variable's *name*), and a numeric and a range input field, which are used to provide two different ways of entering data in the web app to write a value in an input variable.
- *Input strings* use a label (same use as in the previous cases) and a (text) input field to allow users to enter the new string they want to write in the variable.

One of the most illustrative examples of HTML code generated by the RIP client implementation, is the case of a slider created for an input number variable. For a slider generated for an input variable *i*, where $1 \leq i \leq M$, the HTML code is (following the naming convention used in the metadata example given at the beginning of this subsection):

```
<input min="min_input_i"
       value="initial_value_of_input_i"
       max="max_input_i"
       step="precision_input_i"
       type="range"
       title="description_input_i"
       onchange="rip.set(['input_i'],
                 [this.value]);"
/>
```

The *min* and *max* attributes of the HTML tag fix the endings of the generated slider to *min_input_i* and *max_input_i*, preventing the user to set a value out of such range. The *step* attribute fixes the minimum change the values of the slider can take and so, it must fit the precision of the input. The *title* attribute provides a tooltip for the element, displayed when the user leaves the mouse on top of it, and would show the description provided in the metadata for the input *description_input_i*. The initial value set for the slider in the UI is specified through the *value* attribute. This value is obtained by RIP at the moment of establishing the connection with the remote system. Finally, the *onchange* instruction is used to get the value of the slider when the user changes it (this is done through *this.value*) and send it to the remote system using a REST call (implemented in RIP in the *set()* method). This method takes two parameters: 1) the name of the input to be modified (*input_i*), and 2) the new value the input must take (*this.value*). A web app not using RIP could still build the required REST calls on the fly simply using the information provided by the metadata in the *methods* field for *inputs* and *outputs*.

2) Arranging the HTML UI elements: Once the individual HTML UI elements are defined, the next step is to arrange and place them in the web app to form a friendly UI[1]. There are many valid choices to do this, and the program we developed allows users to choose between some options to change the final layout. Nevertheless, by default, the UI is built according to the following main rules:

1) The control panel, containing all the elements for inputs and outputs, is placed below whatever view elements have been added in the web application manually (if any).
2) This panel is divided into two sections (<*div*> tags): the first one contains all the inputs, while the second one contains all the outputs. These sections are placed in two different rows by default.
3) Both sections present the *overflow* HTML property to allow some of its elements to move to a new row when the space is not enough to accommodate them in one single row. This is especially useful in order to make the UI usable from mobile devices with smaller screens.
4) The elements are placed in each section from left to right following the order they are placed in the metadata.
5) For inputs/outputs that produce several UI elements, the label is place in the same column (not the same row) as the input/output elements.

The configuration options implemented to give the user some control on how the resulting UI will look like, are two: 1) to change the location of the control panel (point 1 in the list above) and 2) to select whether the labels should appear in the same row or in the same column (point 5 in the above list). When the first configuration is set to place the control panel on the right, or on the left, the input and output sections are placed in columns instead of in rows (point 2) and they are filled with the elements as they appear in the metadata from top to bottom, instead of from left to right (point 4). Fig. 3 shows a configuration window developed by the authors to facilitate users the process of selecting between the implemented options.

IV. Case Study: Servo-motor Online Lab

Servo-motors are commonly used in control engineering courses for controlling their position and/or velocity, normally by means of a PID. Moreover, several OLs [28], [29] have been

[1]We consider a UI to be "friendly" when it is responsive, its elements are grouped by functionality, and the UI presents no usability issues.

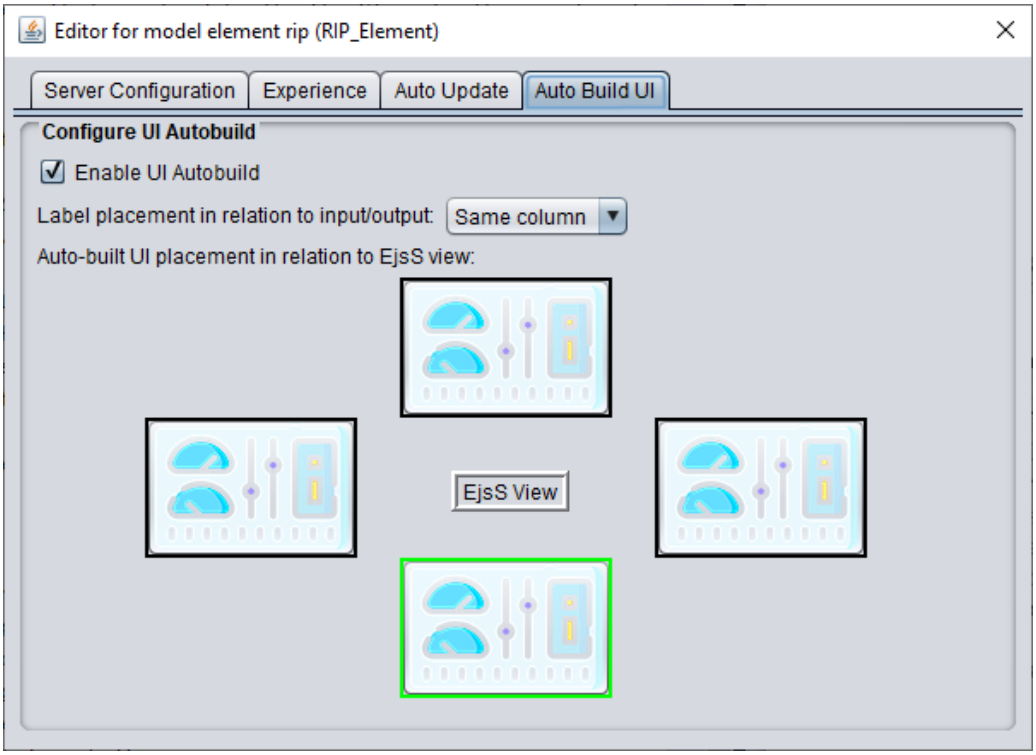

Fig. 3: Configuration window to select the parameters that change the layout of the resulting UI.

proposed to make them remotely accessible 24/7. Therefore, this system provides a great example for this section.

The example presented here uses a Virtual Instrument (VI)[2], created by National Instruments (developers of LabVIEW), as the OL's simulation model. VIs are how programs in LabVIEW are called, and include two parts: the block diagram (where the model is prepared using a visual programming language similar to Simulink) and the front panel, where the UI is built. Since this example uses a virtual model and no hardware is involved, the origin of the metadata, as represented in Fig. 1, is the second case, which, in this case, is the VI. The reason to use an existing VI, created by a third-party, instead of creating a new one ourselves, is to demonstrate that our solution can be applied to many available examples out there with little to no effort. The only modifications done to National Instruments' VI were to reconvert elements that used a certain type (clusters) not currently supported by RIP, and remove repeated inputs/outputs. This process took the authors less than 10 minutes. Fig. 4 shows the VI's front panel after it was modified to meet the requirements imposed by RIP. Initially, this lab could only be used on-site, sitting in front of the computer where the VI and LabVIEW are installed and run. Thanks to the use of RIP, and the web client app that is built automatically, the lab becomes remotely operable from a webpage, following the architecture presented in Fig. 2.

Table 2 presents the inputs and outputs of the modified VI (and so, the data exchanged between the lab server and the web app client), where:

inputs	Disturbance	Setpoint	Reset control	Kc	Ti	Td
outputs	Position	Voltage	Time			

TABLE II: Inputs and outputs for the the servo-motor position control example by National Instruments.

- For the inputs: *Disturbance*, *Setpoint* and *Reset* represent the disturbance a user can enter to the system, the desired setpoint (desired position, in this case) and a control to reset the controller, respectively. *Kc*, *Td*, and *Ti* represent the three parameters of the PID controller. An additional input (*stop*) is always added by RIP to give the user a way to disconnect the client from the server and stop the process running there. It does not have to be defined in the control program or the simulation model (as it happens in this example), which is why it is not listed in this table.
- For the outputs: *Position*, *Voltage*, and *Time* represent the actual position of the motor (in radians) at a given time, the input voltage to the motor (the control signal in this example), and the current time, respectively.

For this example, the web application including the RIP Javascript client was prepared in Easy Javascript Simulations (EJS)[3], a free and open-source authoring tool widely used to develop online labs. Fig. 5 presents the main panel in EJS to setup the web app and gives an idea on how fast and effortless is to prepare a web app so that it generates its UI automatically. The configuration window in Fig. 3 is included in EJS to make it very simple to specify the desired layout for the UI. It is important to highlight that EJS is not required, but rather used as a tool where we integrated RIP to avoid users having to

[2]The VI used for the presented example is *SimEx DC Motor Position Control with PID.vi*, which comes with NI's Control & Simulation toolkit.

[3]https://www.um.es/fem/EjsWiki/

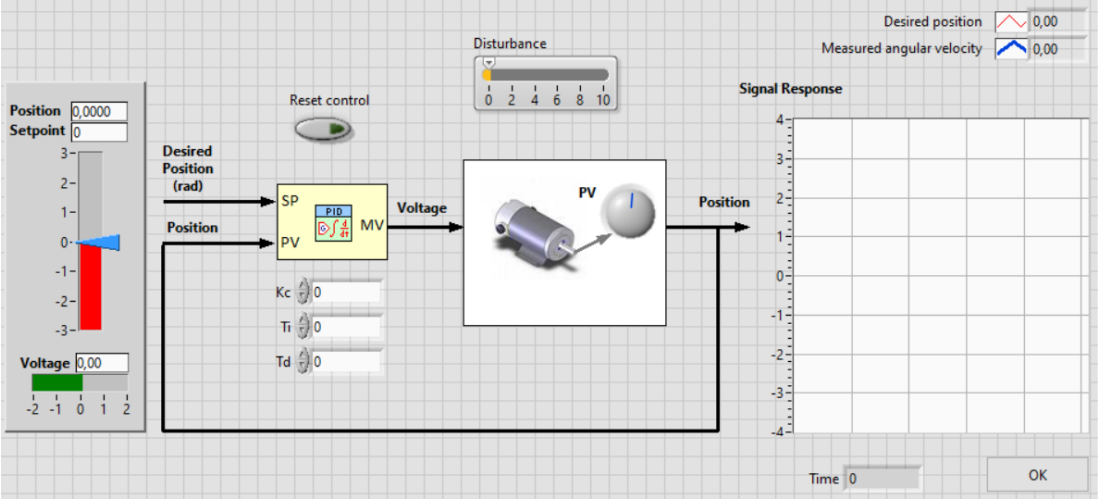

Fig. 4: Modified version of National Instruments' VI example for position control of a servo-motor. Required changes were: 1) replace the cluster control representing the PID with three float controls for each of the PID parameters, and 2) remove all repeated controls/indicators in any of their forms. The "Time" output was also added, not because it is required, but to explicitly provide this important information in the resulting web app.

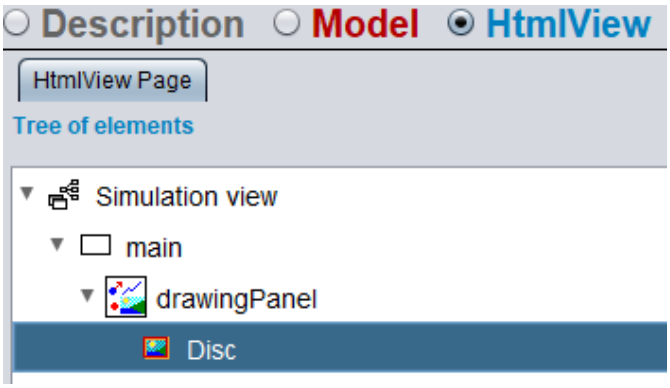

Fig. 5: Setup of the servo-motor web application in EJS. Defining the UI is reduced to inserting a main and a drawing panel (which correspond to <*div*> tags in HTML), and an image of a disc to represent the motor in the virtual lab.

write one single line of code when preparing the bare-bones of the web application that will build its UI automatically.

When the resulting web app, generated by EJS, is opened and the RIP server is not running or reachable, the web browser displays a simple UI with no functionality. Since only the motor image was defined when building the web app in EJS, that is all that appears (see the top image in Fig. 6). However, when the RIP server is running and reachable, the RIP client in the web app gets all the required metadata to automatically build the controls and indicators in the UI, and the result is a functional app (bottom image in Fig. 6).

As expected, the bottom image in Fig. 6 shows the UI elements to read and write the data in the OL, and listed in Table 2. Not only these components are displayed in the UI, but they are also fully functional and capable of receiving and sending the data when expected.

As discussed in Section III-B2, the automatic UI builder accepts some layout configurations. In EJS, this is done through a configuration window where users can choose where to place the control panel at, and whether they prefer labels to appear in the same column or in the same row as the inputs/outputs. Fig. 7 shows an example of the UI built automatically with a different selection of parameters. Additionally, this example uses different CSS options than the ones that come with EJS by default to show a more up-to-date UI.

An aspect that is usually very important in OLs is plotting and visualizing data in graphs. However, the application generated in this example does not offer this option. The reason for this is twofold. First, representing data in a graph usually requires selecting a pair of variables that makes sense plotting one against the other. It also requires some knowledge to decide which of the two variables should be better placed in the y-axis, and which one in the x-axis. Therefore, automating this process is not possible unless the metadata received from RIP (or any other agent) includes some information about the relation between the different input and output variables. Second, previous works (such as [30]) provide a compatible solution for plotting data in OL web apps other than creating a UI in which pre-built graphs are already embedded. Actually, [30] relies on the same tools (RIP for the lab side and HTML5 for the client-side) and mechanism (exchanging and processing the lab metadata) to achieve this. These works propose giving end-users (normally students or researchers) the power to select the data to be plotted (variables for the x and y axes), the labels, the scale and so on, instead of forcing users to work with predefined graphs that are already integrated within the OL web application. This, in turn, offers a better learning or research experience, as users are free to plot whatever data they want, whenever they need it.

V. Conclusions

This work presented the tools and methodology to automatically build user interfaces for web applications that connect to remote resources. These resources can be either virtual (for

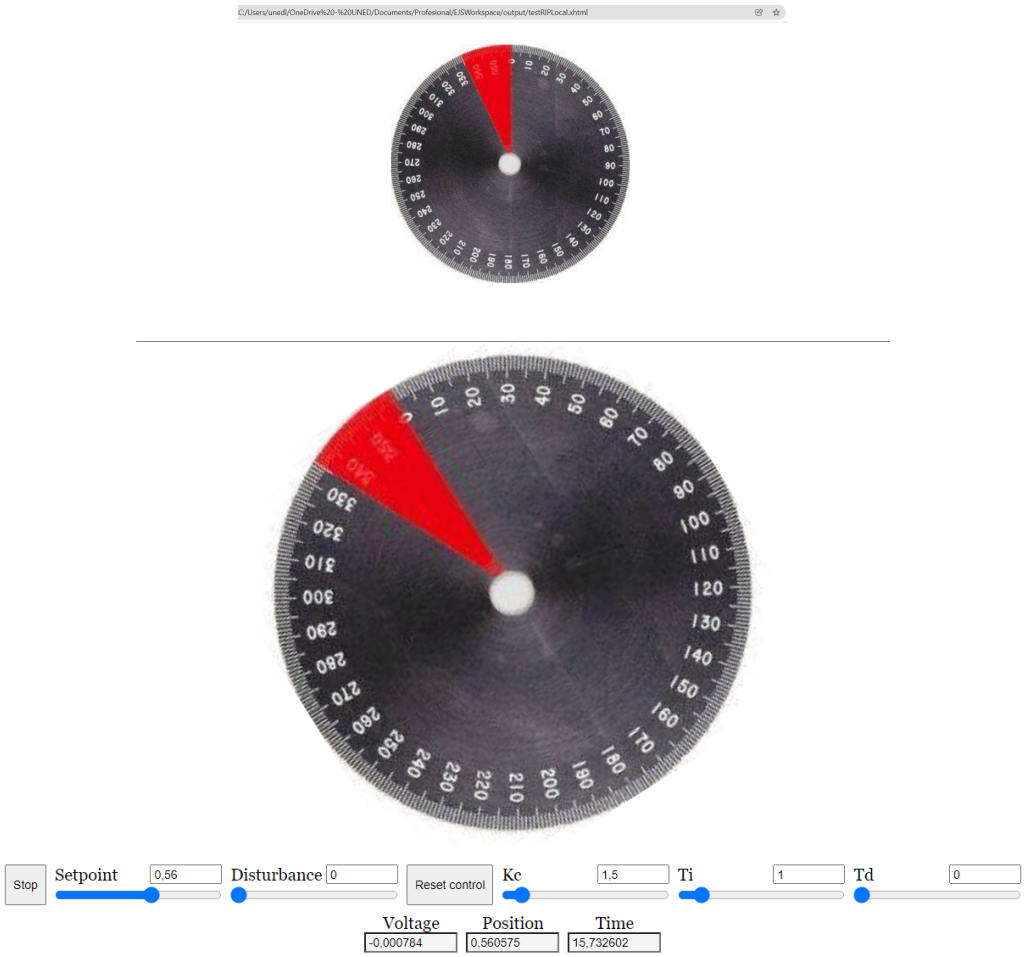

Fig. 6: Web app when the metadata is not available and the UI cannot be built automatically (top), and when it is (bottom). The *Stop* button is added by RIP automatically even if it is not defined in the server, and the server executes its action when the user presses this button in the app or when the client disconnects. This example uses EJS' default CSS configuration.

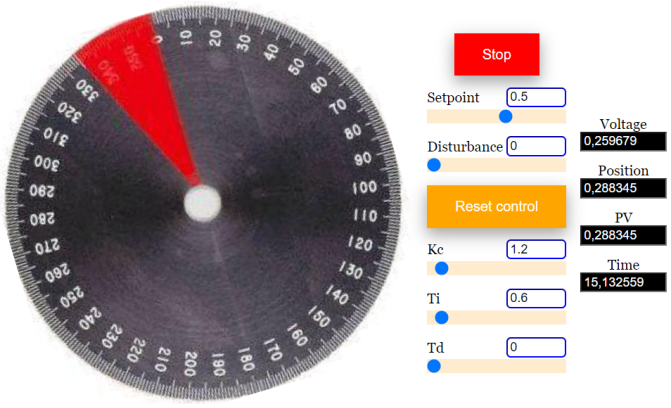

Fig. 7: Web app automatically built with different parameters and using a preset CSS different to EJS' default one.

simulation programs running in a computer, for example) or real (for IoT and other hardware devices).

The tools and methodology proposed to achieve this are based on: 1) the use of IoT-type metadata that contains all the relevant information of the remote resources to be monitored and/or operated from the web app, and 2) smart web design techniques to build the HTML+Javascript interface. By obtaining the metadata from the remote resources and parsing their information, the basic HTML5 user interface elements can be built and arranged automatically in the web app using, for example, fixed layouts that may, too, accept some additional custom configuration parameters.

The overall process is illustrated with an example of a servo-motor online lab, widely used in engineering education. However, it could be applied in many other applications, and even contexts, such as IoT, smart homes, and Industry 4.0.

The main advantage of the proposed method is that it saves time and effort in developing the user interfaces for the web applications required to interact with remotely available resources, although more work is needed in terms of how these user interfaces can be built to offer more complex and user-friendly configurations.

Current limitations of the presented solution include: 1) units are not specifically considered in the metadata and have to be added in a variable's name if they are required, 2) complex types (such as arrays) are not supported, 3) options to select between different UI layouts are still very limited, as shown in Fig. 3, and 4) it has only been tested with

fairly simple processes/lab resources so far. However, future work in both the RIP specification and the RIP client implementation will address the previous first three shortcomings. Additionally, the authors consider applying machine learning techniques to build the UI automatically, instead of using fixed HTML+CSS layouts. Finally, our plans also include testing the proposed solution with more complex equipment with a higher resemblance to industrial plants.

VI. Material

Following open science's best practices[4], our software artifacts are available publicly.

- The RIP server implementation that supports the solution described in Section III is available at https://github.com/Nebulous-Systems/rip-server_labview.
- EJS with the RIP client implementation that includes the UI automatic build feature, is available at https://gitlab.com/ejsS/tool
- The case study presented in Section IV is available at https://github.com/Nebulous-Systems/Servo-Automatic-UI-Generation-Example

VII. Acknowledgements

The work that led to this publication has received funding from the European Union's Horizon 2020 research and innovation programme funding "Next Generation Internet initiative (NGI)", within the framework of the NGI Explorers Project (grant agreement no. 825183).

References

[1] L. Atzori, A. Iera, G. Morabito, The internet of things: A survey, Computer Networks 54 (15) (2010) 2787–2805.

[2] P. Sethi, S. Sarangi, Internet of things: Architectures, protocols, and applications, Journal of Electrical and Computer Engineering 2017 (2017).

[3] J. Gubbi, R. Buyya, S. Marusic, M. Palaniswami, Internet of things (iot): A vision, architectural elements, and future directions, Future Generation Computer Systems 29 (7) (2013) 1645–1660.

[4] A. Zanella, N. Bui, A. Castellani, L. Vangelista, M. Zorzi, Internet of things for smart cities, IEEE Internet of Things Journal 1 (1) (2014) 22–32.

[5] E. Sisinni, A. Saifullah, S. Han, U. Jennehag, M. Gidlund, Industrial internet of things: Challenges, opportunities, and directions, IEEE Transactions on Industrial Informatics 14 (11) (2018) 4724–4734.

[6] R. Hegarty, J. Haggerty, Presence metadata in the internet of things challenges and opportunities, in: International Conference on Information Systems Security and Privacy (ICISSP), Valletta, Malta, 2020.

[7] L. Smirek, G. Zimmermann, M. Beigl, Adaptive user interfaces as an approach for an accessible web of things, in: Proceedings of the Seventh International Workshop on the Web of Things (WoT), Stuttgart, Germany, 2016.

[8] J. D. H. Bezerra, C. T. de Souza, A model-based approach to generate reactive and customizable user interfaces for the web of things, in: Brazillian Symposium on Multimedia and the Web (WebMedia), Rio de Janeiro, Brazil, 2019.

[9] R. Heradio, L. de la Torre, D. Galan, F. J. Cabrerizo, E. Herrera-Viedma, S. Dormido, Virtual and remote labs in education: a bibliometric analysis, Computers & Education 98 (2016) 14–38.

[10] L. La Torre, J. Saenz, D. Chaos, J. Sanchez, S. Dormido, A Master Course on Automatic Control Based on the Use of Online Labs, in: IFAC Symposium on Advances in Control Education (ACE), Philadelphia, Pennsylvania, USA, 2019.

[11] D. R. Olsen, E. P. Dempsey, SYNGRAPH: A Graphical User Interface Generator, SIGGRAPH Comput. Graph. 17 (3) (1983) 43–50.

[12] K. Gajos, D. S. Weld, Supple: Automatically generating user interfaces, in: International Conference on Intelligent User Interfaces (IUI), Funchal, Madeira, Portugal, 2004.

[13] K. Gajos, D. Weld, J. Wobbrock, Automatically generating personalized user interfaces with supple, Artificial Intelligence 174 (12-13) (2010) 910–950.

[14] K. Z. Gajos, J. O. Wobbrock, D. S. Weld, Automatically generating user interfaces adapted to users' motor and vision capabilities, in: ACM Symposium on User Interface Software and Technology (UIST), Newport, Rhode Island, USA, 2007.

[15] M. Peissner, D. Häbe, D. Janssen, T. Sellner, MyUI: Generating Accessible User Interfaces from Multimodal Design Patterns, in: ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS), Copenhagen, Denmark, 2012.

[16] D. Navarre, P. Palanque, J.-F. Ladry, E. Barboni, Icos: A model-based user interface description technique dedicated to interactive systems addressing usability, reliability and scalability, ACM Transactions on Computer-Human Interaction 16 (4) (2009).

[17] S. Mayer, A. Tschofen, A. Dey, F. Mattern, User interfaces for smart things - a generative approach with semantic interaction descriptions, ACM Transactions on Computer-Human Interaction 21 (2) (2014).

[18] M. Brambilla, E. Umuhoza, R. Acerbis, Model-driven development of user interfaces for iot systems via domain-specific components and patterns, Journal of Internet Services and Applications 8 (1) (2017).

[19] S. Mansour, M. Karam, A functionality-driven design of a centralized home automation internet of things framework with a dynamically generated web interface, in: International Conference on Software Engineering and Data Engineering (SEDE), San Diego, USA, 2017.

[20] L. Henschen, J. Lee, R. Guthmann, Automatic Generation of Human-Computer Interfaces from BACnet Descriptions, in: Distributed, Ambient and Pervasive Interactions: Understanding Human (DAPI), Las Vegas, NV, USA, 2018.

[21] C. Salzmann, S. Govaerts, W. Halimi, , D. Gillet, The smart device specification for remote labs, in: 12th International Conference on Remote Engineering and Virtual Instrumentation (REV), 2015, pp. 199 – 208.

[22] W. Halimi, C. Salzmann, H. Jamkojian, D. Gillet, Enabling the automatic generation of user interfaces for remote laboratories, in: International Conference on Remote Engineering and Virtual Instrumentation (REV), New York, USA, 2017.

[23] L. F. Zapata-Rivera, M. M. Larrondo-Petrie, C. P. Weinthal, Generation of multiple interfaces for hybrid online laboratory experiments based on smart laboratory learning objects, in: IEEE Frontiers in Education Conference (FIE), Cincinnati, USA, 2019.

[24] L. de la Torre, L. T. Neustock, G. K. Herring, J. Chacon, F. J. G. Clemente, L. Hesselink, Automatic generation and easy deployment of digitized laboratories, IEEE Transactions on Industrial Informatics 16 (12) (2020) 7328–7337.

[25] L. de la Torre, J. Chacon, D. Chaos, Remote interoperability protocol specification (2019).
URL https://doi.org/10.5281/zenodo.2644242

[26] C. Salzmann, S. Govaerts, W. Halimi, D. Gillet, The smart device specification for remote labs, International Journal of Online and Biomedical Engineering 11 (2015) 20–29.

[27] IEEE Std 1876-2019, IEEE Standard for Networked Smart Learning Objects for Online Laboratories, section 4: First layer of standardization: Lab as a Service (LaaS) and Section 5: Recommended practices for using online labs as learning objects (2019).

[28] M. Guinaldo, J. Sanchez Moreno, S. Dormido, A packet-based network control system architecture for teleoperation and remote laboratories, in: IEEE Conference on Decision and Control (CDC), Atlanta, GA, USA, 2010.

[29] F. Osses, R. Ortega, C. Muñoz-Poblete, Remote laboratory for position control using a Quanser Servo SRV02 operated via Moodle, in: IEEE International Conference on Automation/XXIII Congress of the Chilean Association of Automatic Control (ICA-ACCA), Concepcion, Chile, 2018.

[30] D. Galan, D. Chaos, L. de la Torre, E. Aranda-Escolastico, R. Heradio, Customized online laboratory experiments: A general tool and its application to the furuta inverted pendulum, IEEE Control Systems Magazine 39 (5) (2019) 75–87.

[4] https://ec.europa.eu/info/research-and-innovation/strategy/strategy-2020-2024/our-digital-future/open-science_en

Luis de la Torre is Associate Professor at the Computer Science and Automatic Control Department, UNED. de la Torre received the M.Sc. degree in physics from the Complutense University of Madrid, Madrid, Spain, and the Ph.D. degree in computer science from UNED, Madrid. His current research interests include virtual and remote labs, distance education, and http protocols and technologies for networked control systems with event-based control techniques, and he has published over 20 articles in international journals in these and other topics. He has been a visiting scholar at Stanford University and the Swiss Federal Institute of Technology in Lausanne (EPFL) among others.

Jesus Chacon received the M.Sc. degree in Automation and Industrial Electronics Engineering from the University of Cordoba, Cordoba, Spain, in 2009, and the Ph.D degree in Control and Systems Engineering from the Universidad Nacional de Educacion a Distancia (UNED), Madrid, in 2014. He is an Assistant Professor with the Department of Computer Architecture and Automatic Control, Universidad Complutense de Madrid (UCM). His research interests include simulation and control of event-based systems, communication protocols, and control engineering education, and he has published over 15 articles in international journals in these and other topics.

Dictino Chaos received the B.D. degree in physics from the Complutense University of Madrid, Madrid, Spain, in 2004, and the Ph.D. degree in computer engineering and automatic control from the National Distance Education University (UNED), Madrid, Spain, in 2010. He is currently an Associate Professor with the Department of Computer Science and Automatic Control, UNED. His research interests include nonlinear control, tracking, point stabilization, path following of underactuated vehicles, the stability of switched systems, machine vision, and robotics. Dr. Chaos received an award with a special prize for the best thesis.

Ruben Heradio is Full Professor at the Software and Systems Engineering Department, UNED. He received the M.Sc. degree in computer science from the Polytechnic University of Madrid, Spain, in 2000, and the Ph.D. degree in software engineering and computer systems from UNED, in 2007. His research and teaching interests include software engineering, computational logic, and e-learning. On these topics, Ruben Heradio has (i) published more than forty papers in top-ranked journals and conferences, (ii) advised six doctoral theses, and (iii) participated in 13 research projects, being the Principal Investigator of three of them.

Rajarathnam Chandramouli (Mouli) is a Principal Scientist at Galois, Inc. and CEO of Spectron, Inc. He brings more than 20 years of RD experience in mobile networking, edge computing, and machine learning in both academic and startup environments. He designs and analyzes machine learning algorithms for wireless network systems, cybersecurity, and space applications. He has led the RD for DoD, first responders, and commercial end-user products. Mouli frequently gives keynote and plenary talks at major conferences. His professional activities include the following: IEEE COMSOC Distinguished Lecturer, Editor of IEEE Journal on Selected Areas in Communications (JSAC)–Cognitive Radio Series, Associate Editor of IEEE Transactions of Circuits and Systems for Video Technology, Founding Chair of the IEEE COMSOC Technical Committee on Cognitive Networks (TCCN), Member of the IEEE COMSOC Standards Board, Advisory Board member of several journals and conferences, and organizing Chair of several international conferences.