Experiment Lab Server Architecture: a Web Services Approach to Supporting Interactive LabVIEW-based Remote Experiments under MIT's iLab Shared Architecture

Luciano A. Mendes
Production and Systems Engineering Graduate Program
Pontifical Catholic University of Parana - PUCPR
Curitiba, Brazil

Letitia Li, Philip H. Bailey, Kimberly R. DeLong and Jesus A. del Alamo
Center for Educational Computing Initiatives - CECI
Massachusetts Institute of Technology - MIT
Cambridge - MA, USA

***Abstract*— The MIT iLab Shared Architecture (ISA) is widely used to support remote experimentation, providing basic functionalities such as users management, scheduling, access control and results storage. This paper addresses the optimization of LabVIEW-based RE's creation under ISA, with the standardization of common lab server functionalities and communications. The Experiment Lab Server Architecture (ELSA), created as an ISA extension, fulfills the requirements of an ISA process agent and uses web services to connect the client with the lab server. ELSA automatically manages the initial definition and runtime traffic of experiment-specific input/output parameters. Once the experiment and client codes are created, RE configuration, startup and operation can be easily accomplished. In a pilot implementation using ELSA, the web services mediation of data transfer performed at rates high enough for a smooth user interaction experience. ELSA web services approach allows for a loose coupling in RE design and can effectively shorten its development cycle.**

Keywords*— remote laboratories; web services; middleware architecture; iLab shared architecture

I. INTRODUCTION

After several years since the use of remotely operated laboratories in education began, encouraging results of research efforts to evaluate its pedagogical effectiveness have been providing important arguments to support the modality [1], [2], [3], [4]. Particularly in STEM-related areas, real laboratory applications, in opposition to virtual simulations, produce a higher motivation on students, as interacting with real equipment resembles much more the reality found in professional life [5], [6]. Whilst the creation of simulations require basically software development efforts and can be run locally in any computer, remote experiments based on real equipment must often be modified, automated with instrumentation and actuation features, leading to multidisciplinary projects with an important component of information and communication technologies (ICT's). In remote experimentation, existing supportive technologies are relatively young and still evolving with internet technologies. Improving the performance of existent systems and the easiness of technical accomplishment is a permanent goal. New architectures proposals, variations, or even a localized advancement can be valuable to providing more effective, simpler means of remotely controlling experiments.

A. Context and Previous Related Works

The value remote laboratories add to engineering education has been extensively discussed in the literature, as synthesized by [7]. These authors have reviewed approaches and cases in multiple institutions worldwide, and in different areas, and have posed a relevant observation in their conclusions: many remote laboratories have a static structure with few configurable parameters for students to interact with, in predefined experiments with a rather static setup. Therefore, increasing the flexibility to dynamically interact with the functionalities of remotely operated systems could expand pedagogical possibilities.

Considering the educational objectives of remote experimentation, requirements for the development of remote laboratories (or just weblabs) should prioritize learning, pedagogical goals, which tend to be specific in each remote laboratory context. Systems complexity may vary from simple, fast execution of basic electronics circuits experiments to assemblies with intricate arrangements, long execution duration and varied interaction modes. The development process of a weblab can benefit from a systematic approach, as proposed in [8], for systems of this kind require the integration of a physical system, hardware and software components [9], in a multidisciplinary sense. As internet evolves, remotely operated systems underlying architectures are updated to explore the benefits of web technologies [10], and the new applicable solutions used to drive better projects.

Most remote experiments require an underlying set of common tasks, such as access control, scheduling and data management. As remarked in [1], "*there is an obvious need for*

978-1-4673-8245-8/16/$31.00 ©2016 IEEE 24-26 February 2016, UNED, Madrid, Spain
2016 13th International Conference on Remote Engineering and Virtual Instrumentation (REV)

a definition of models, frameworks and standardization of different aspects of distance learning laboratories". This has been the motivation of several previous projects towards the development of effective and reliable supportive infrastructure, aiming to manage technical, administrative and even pedagogical components, usually referred to as Remote Laboratories Management Systems (RLMS). Some representatives of such systems are MIT's iLab [11], [12], Weblab Deusto [13] and REALabs [14]. In [15], a comparison among different RLMS architectures is presented in detail. The low bandwidth available in many locations, that must be considered when designing these supportive systems, has been discussed in [16]. The access and scheduling issues are addressed by [17]. Reference architecture, scope and development stage may vary, but these projects' main objectives remain similar.

Peer-to-peer labs are discussed in [18], proposing a distributed remote control framework focusing in popular, low cost microcontrollers. It addresses three subsystems to handle user interface, instruction interpretation and execution, based upon the usage of a message-oriented middleware. In [19], an OPC server (OPC Foundation) solved the connectivity between physical equipment and a database, and a web server supported by a content management system (CMS) was used to serve the client interface displayed in an internet browser. The CMS and a proxy server replaced the service broker normally used to manage system usage in other platforms.

MIT's iLab Shared Architecture (ISA) is based on a web services infra-structure capable of supporting a broad range of remote laboratories providers' requirements, including scheduling and access control, in a scalable sense and independent of the software development environment used in a given experiment implementation. Currently, ISA is used in universities in all continents. WebLab Deusto also uses a services-oriented architecture SOLA to support remote operation, pointing to a desirable operational system independence on the client side, the firewall compatibility of HTTP protocol and no access to the user's computer hard disk. WebLab Deusto also explores the federation concept. In REALabs, issues such as security, quality of service, and federated operation are emphasized.

The client-experiment connectivity approach used in REALabs is based upon RESTful web services calls, for the technical advantages of this architecture: HTTP is firewall-friendly, XML is a neutral format, authentication and authorization can be made on-the-fly and both HTTP and XML favor the creation of light clients.

The usage of web services to support internet applications according to the W3C standard is a well-established reality, and their use in remote laboratories is naturally an option of interest. A typical example of a batch, queued application designed upon a web service architecture can be found in [20]. In this implementation, a LabVIEW-based experiment on electronic device characterization is operated from an AJAX-based client. In this concept-proof project, however, scheduling and access control services are not assessed.

Enabling remote operation over web services has been tested in [21], with the use of a service-oriented middleware to support management tasks and the end resources (instruments used in the experiment). Such middleware consists of an abstraction layer that implements a common set of web services (engine tier services), whilst laboratory resources are accessed through another specific set of services (instrument services).

Similarly, a three-tier architecture – client, resources manager, application - has been used in [22], with client inputs received in a web server and converted into files, which are queued in a local directory. Therefrom, they are read and processed (parsed into variables) in order of arrival by the LabVIEW code in charge of watching out for incoming requests. Despite the disk intensive files write/read (and its inherent overhead), the code modularization is a plus. Web services calls could do the job more efficiently in this approach.

In a move towards making RE's totally independent component modules, an interesting approach in which remote experiments are exposed as sets of pre-defined, consumable web services has been proposed in [23]. This architecture is also oriented to LabVIEW-based remote experiments and uses RESTful services and LabVIEW shared variables to enable the communication between the web services front end (the proxy) and the main application supporting an experiment. Context-specific services and variables are created and deployed for every RE design. As a result, a service description document is provided for every RE implementation, and can be used to drive the creation of custom clients.

B. Supporting Interactive LabVIEW-based Remote Experiments under ISA

The two classic formats in remote experimentation provide distinct outcomes. *Batch-type experiments* process a set of test parameters submitted by the user, and can return results immediately after a run, that the student can analyze off-line. Generally speaking, the time separation of Experiment execution and its results analysis allows for simpler experimental systems architectures, in terms of automation resources mainly. One elaborated example of flexible batch experiments can be seen in the VISIR project [24], which uses contact matrixes to interconnect electrical components and dynamically assemble different circuits. Nevertheless, *interactive experiments* that draw continuous attention, analysis and decision-making abilities, hold important potential to explore weblabs didactical opportunities. Naturally, an increase in the engineering complexity may be expected, with the introduction of additional design requirements. Also, longer experiment duration (with lower down time) and more supportive features suggest a higher overall system reliability must be present. Control engineering experiments usually fall in this category, with many citations in the literature [25], [26], [27].

In the creation of instrumented systems with remote operation purposes, National Instruments' LabVIEW platform is often the chosen platform, as it provides multiple useful resources and libraries for data acquisition, analysis and presentation. Its environment was designed to easily integrate real world analog and digital signals through varied forms of

I/O and connectivity interfaces. LabVIEW programs are created as interconnected virtual instruments, each containing a front panel, a block diagram and a connector panel. The graphical programming language and the run-time compiler makes coding and execution efficient.

Remote laboratory projects require some degree of automation, with the addition of I/O signals and complementary systems to a traditional laboratory experimental apparatus (actually making the remote operation feasible), a context in which the aforementioned LabVIEW capabilities are particularly valuable.

The use of LabVIEW front panels in the fast creation of user-friendly VI's interfaces is a relevant characteristic of interest from the point of view of remote operation, because a panel can be easily published in the form of a web page using the LabVIEW built-in web server, in a proprietary VI publishing scheme. This is the so-called Remote Panel technology that replicates the very VI's front panel into an html page, which is opened in a compatible web browser. User operation starts after the remote control is requested and granted. However, there are limitations with this solution regarding secure access through firewalls and deployment of the html client, because it is required that a heavy plugin be installed in the remote computer - the LabVIEW Run Time Engine (matching the version running in the server). Furthermore, the quality of the user experience highly relies on a fast network connection between the remote computer and the LabVIEW Web Server [28].

Typically, interactive LabVIEW based RE's created under ISA at the MIT have been built upon the embedded LabVIEW remote panel technology. A much better approach is to use modern web technologies in the creation of the user side application. Light clients, such as AJAX, HTML5 and mobile applications [29] [30] avoid the need for specific add-on's or plug-in's for compatible browsers, which may also require version matching, as is the case with the LabVIEW Remote Panel technology. Nevertheless, the so called "heavy clients" are also an option, including LabVIEW executables, [31] and JAVA Applets. Platform independent, web services provide the desired flexibility in the creation of light clients.

Applying the premise of a more flexible, modular approach for LabVIEW-based RE's under ISA suggests the lab server could be designed in the form of a compliant, generalized software module, configurable to attend the context of any LabVIEW-based experiment. To favor shorter development cycles, specific experiments codes should be initially created independently of the remote operation functionalities, and then seamlessly configured as a RE's, with a loose coupling between the reusable code and the experiment-specific code. The lab server should consist of a reusable code applicable to any experiment context, comply with ISA (by implementing a mandatory set of web services for an ISA module), and also intermediate the communication between client and the RE through a generalized set of web services.

In LabVIEW, the versatility of network shared variables, multi-type global variables and RESTful web services can be explored, along with powerful multithread programming techniques. SOAP web services, however, require another platform. JAVA, a popular SOAP-capable language, with services being deployed typically to a Glassfish or a Tomcat server, can provide the environment for the creation of the services necessary for compliance with ISA.

Given the preamble, this paper addresses the problem of optimizing the creation of LabVIEW-based RE's under ISA through a better design and implementation of an ISA lab server architecture. Considering the sense of software engineering research [32], the design problem of an ISA-compliant lab server architecture, able to support general interactive experiments, can be defined with the following features and/or requirements:

- capable of handling interactive remote experiments created in the LabVIEW language;
- dismisses the need for LabVIEW's remote panel technology;
- web services-based, allowing the use of multiple web technologies to build light clients;
- fully compliant with the ISA lab server specification;
- mediates communications independently of experiment context;
- frees remote experiments designers to focus in experiment related issues, rather than on supportive aspects, shortening development cycles;
- easy to configure and manage.

A lab server architecture with the aforementioned characteristics was designed, coded and tested. It has been developed in LabVIEW and JAVA languages, and baptized Experiment Lab Server Architecture (ELSA).

The rest of this paper is organized as follows: section II briefly reviews ISA architecture; section III depicts ELSA operational concept as a message oriented middleware, its integration with ISA and how lab servers and experiments are set up; section IV contains an application example and performance test results; section V addresses these results, safety aspects, the flexibility of the architecture and foreseen possibilities of innovative applications; finally, conclusions are presented in section VI.

II. THE MIT iLAB SHARED ARCHITECTURE - ISA

In the ISA architecture, a set of Process Agents (PA's) backed by Microsoft SQL databases interact with each other through web services, using authenticated access and a ticketing scheme. The interaction among multiple PA's relies on an initial registration process with exchange of credentials. In this sense, every PA is expected to expose a standard set of methods that allows for the initial exchange of some data-types containing unique identification information, that will be used in the authentication of further communication. Specific services are provided by each PA, in accordance with its role in ISA.

The Service Broker (SB) is a central-role PA in which users' accounts, access permissions and experiment's agendas are managed. Scheduling services were designed to be present at both the user side (User Scheduling Service, USS) and laboratory side (Laboratory Scheduling Service, LSS), since users and laboratories may be located in distinct institutions or networks. In that sense, ISA has been designed to support the

scenario of different universities sharing remotely operated laboratories, so that each institution can keep its own policies of student's usage and resources (experiments) availability. The Experiment Storage Service (ESS) provides storage for the results created during experiments execution sessions, so these can be accessed later for offline analysis and creation of reports by students. The aforementioned ISA modules were implemented in the Microsoft Visual Studio platform and run on Windows Server 2008 backed by SQL databases.

The Laboratory Servers (lab servers), also integrated as PA's, are responsible for all tasks related with experiment execution, including the means for direct communication with the client application. Once a lab server is linked to a SB, it can make requests to its services and to other PA's ones (e.g. the ESS). The independence of the lab server with respect to the rest of the ISA architecture, by establishing direct communication with a client after a successful authentication process, is key for the flexibility of ISA.

In order to run an experiment, the following typical sequence takes place: the registered user logs into the SB before the beginning of the time slot of a previously scheduled session. At his/her command, the SB launches the client, embedding in it the authentication information. Such information is packed in a specific data type, a 'Coupon', sent to the lab server, where it is used in a subsequent web service call to the SB as an argument to the 'redeem Ticket' method. Expectedly, the 'Ticket' was previously created at the SB and stored in the associated database. The SB will check the authenticity of the caller credentials – the lab server's in this case - closing the verification loop. Thereafter, information incoming from the client will be processed, in accordance with the experiment type: batch experiments input data is stored and queued for execution, since they run quickly and don't depend on further user activity (being automatically closed right after completeness); interactive experiments are kept active during the time slot, receiving and returning information to a client interface continuously. Experiment results are stored at ESS, to be retrieved by the user later in his/her user area in the SB.

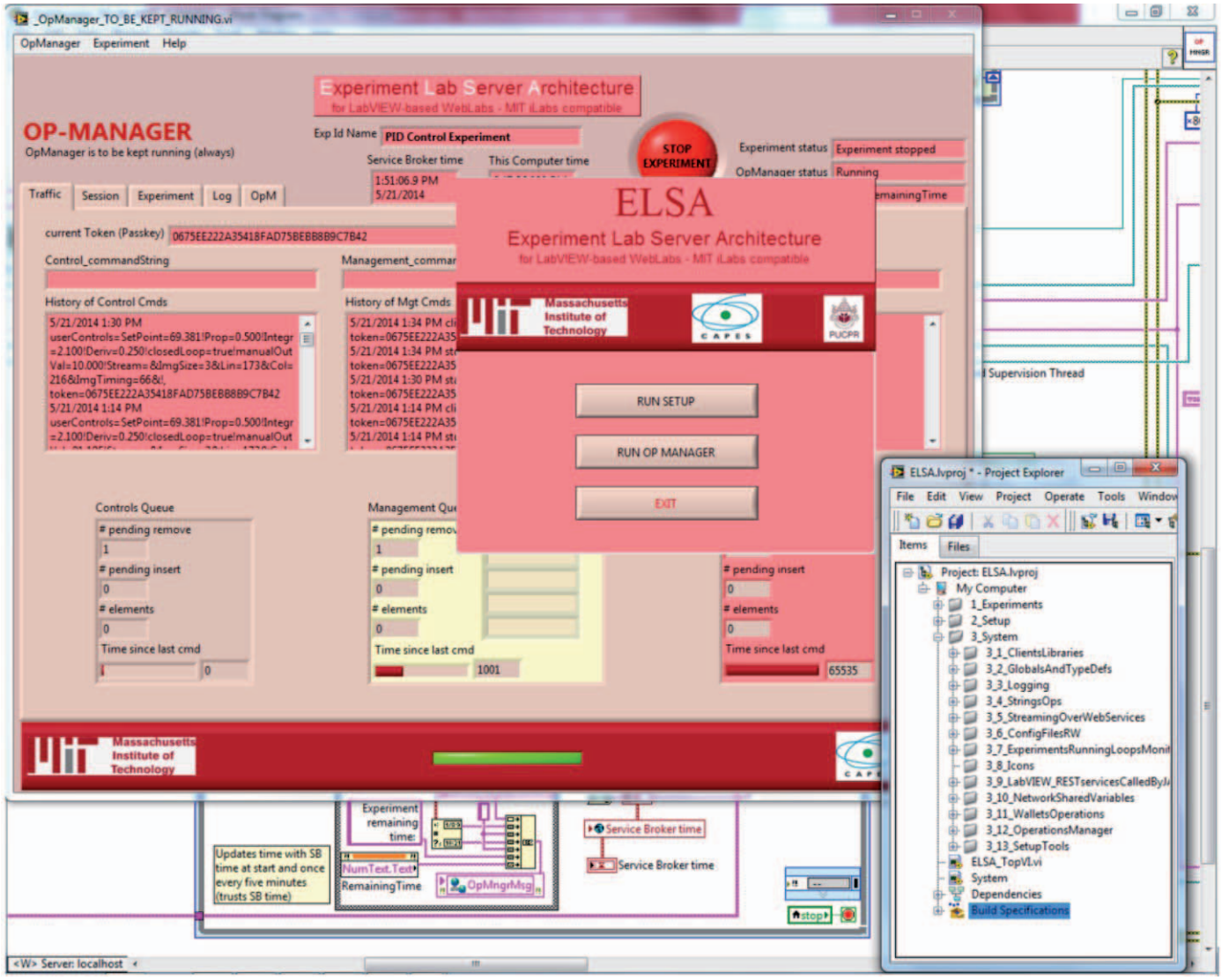

Fig. 1. ELSA environment in LabVIEW.

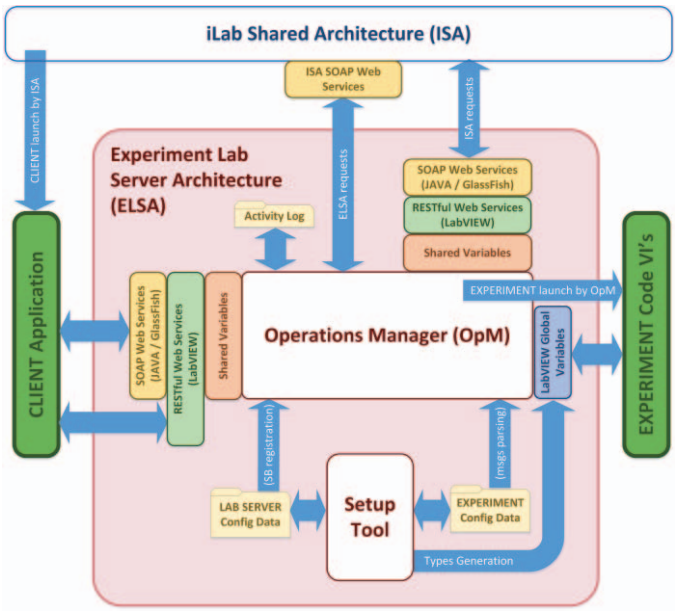

Fig. 2. ELSA components and its integration with ISA, client and experiment code.

III. ELSA IN DETAIL

ELSA was structured in three components: one dedicated to the general operations management, the Operations Manager (OpM), including experiment runtime communications; the second is used for the lab server configuration, including the experiment definition parameters (Setup Tool); and third, a front end of JAVA/SOAP web services running on a GlassFish web server. Experiment operation during runtime is carried out via calls from a client to a set of services, either the SOAP methods or their RESTful counterparts (both are supported in ELSA). ISA architecture supports SOAP web services, whilst LabVIEW exposes only RESTful ones. Both Glassfish and the LabVIEW web servers are set up in the same computer (the lab server's). Only one experiment configuration can be enabled to run at a time, but different experiment definitions and their respective codes can be created and stored locally. This makes alternating among different experiment configurations or variants a practical task, improving the flexibility of a lab server station (e. g, one hardware set may be shared by different experimental setups).

ELSA architecture relies on a set of front-end web services backed by one general management software module (OpM), which is in charge of a set of supportive tasks. Most of them are related to processing incoming messages, intermediating the information exchange with the experiment and formatting responses accordingly to be returned to the client. Its environment is shown in Figure 1. From a set of high level user interfaces, all configuration, management and experiment execution tasks of a LabVIEW-based lab server are handled, including its registration, with credentials exchange, in an ISA Service Broker. Once set up and run, the lab server is supposed to be kept in duty 24/7.

Figure 2 depicts the integration between ISA, ELSA, client and experiment. Messages containing incoming information, e. g. Experiment Control commands, go through a set of SOAP web services, which forward them to their REST counterparts (direct calls to the RESTful ones are possible as well). Incoming data is transferred to the OpM component via an associated Shared Variable (SV) overwritten at the REST service. This action is actually conditioned to an Authentication Token to match the expected value for the ongoing usage session. OpM polls these SV's for updates continuously, and uses the experiment configuration information, retrieved initially from a locally stored configuration file, to parse the incoming messages and populate a mixed-types data structure, namely a LabVIEW Cluster, set up as a Global Variable (GV). Outgoing information from Experiment output variables reverses the process: from the outputs GV, data is packed into SVs, where from the REST web services get their updated data at calls; this data is then passed to the corresponding front end SOAP web services (as they make the RESTful calls); finally, the SOAP services caller (the client) gets a response, as shown in Figure 3. The client application is designed to make intermittent calls to keep data updated at the end user interface. If the client calls directly the RESTful services, a slightly higher data transfer performance may be expected.

ELSA development started in LabVIEW version 2012, that keeps web services and main application variables in different namespaces, requiring the use of the SV's to establish the above described data flow. In subsequent versions of the platform, both spaces have been unified, so that SV's could just be replaced with GV's. Considering the similar data transfer performance of SV's and GV's, and the SV's flexibility for network connectivity that may be useful eventually, SV's have been kept in the architecture.

Information regarding an experiment configuration and the SB provided credentials necessary in any communication

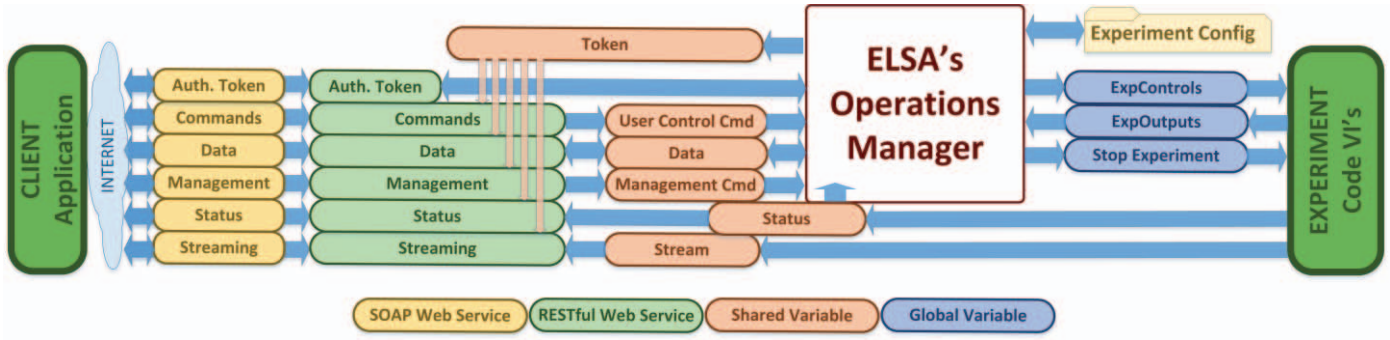

Fig. 3. Web services, shared variables and global variables connecting client and experiment (SOAP web services may be bypassed).

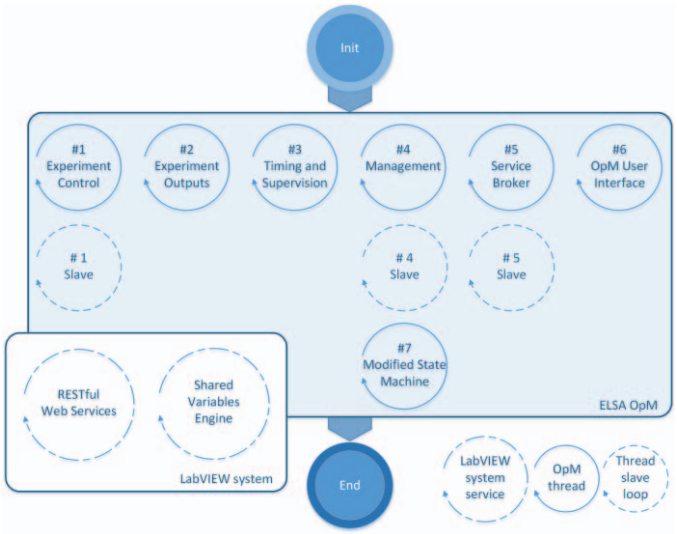

Fig. 4. Operations Manager parallel-running threads and LabVIEW system services.

exchange are stored in local section-key (.ini) files. This type of text file is used to store ELSA information locally (the low amount of data does not justify the use of a database). The heavier experimentation results data can be stored at the ISA ESS database, and be readily available in the user SB profile for retrieval at any time.

The loose coupling between OpM and experiment code is the key for a general, broad applicability of ELSA. The aforementioned OpM communication messages parsing feature frees the developer from implementing such lower level operations in every experiment code, so that the development team may keep focus on specific weblab related development issues.

The client application is supposed to include the Authentication Token argument along with every web service call to the lab server during an experiment run. Once the Experiment time elapses, the Token is automatically cancelled at ELSA. Attempts to pass commands to the experiment with an invalid Token causes incoming calling messages to be ignored at the RESTful web services level - except for the 'Authenticate Token' method call, when a candidate Authentication Token is submitted for validation.

A. Operations Manager

The core ELSA component is OpM, designed to run indefinitely in the LabVIEW main application. It relies directly on the front end of web services. LabVIEW G language supports RESTful web services only, the reason why counterpart SOAP versions of these services were set up using a popular SOAP-capable language, JAVA, and deployed into a Glassfish server, so that the SOAP services mediate the communication with the REST ones. SOAP web services are used in all information exchange among ISA Process Agents, and so does ELSA. The SOAP services are called using XML messages, which can be parsed and assembled with specific libraries in JAVA.

OpM's architecture beholds a set of parallel-running threads, responsible for independent tasks. Once launched, OpM initializes the system variables and starts a group of threads, as shown in Figure 4. The RESTful services run in the LabVIEW web server, whilst another independent LabVIEW process is in charge of the Shared Variables - the Shared Variables Engine. Threads #1, #4 and #5 master their own slave loops, programmed to remain on hold until an order from a respective master is received. The threads roles are explained in detail below:

Experiment Control: this thread's main (master) loop acts as a listener for incoming Experiment Control messages (commands used in the experiment control by the end user). The thread code is in charge of monitoring an associated string-format SV for changes, and runs on a 1 ms cycle time basis. This SV is updated on every call (started by the client) to its associated local RESTful web service method. When a SV's contents change is detected, such contents are submitted to a secondary processing loop, that remains idle unless a new processing task is assigned,

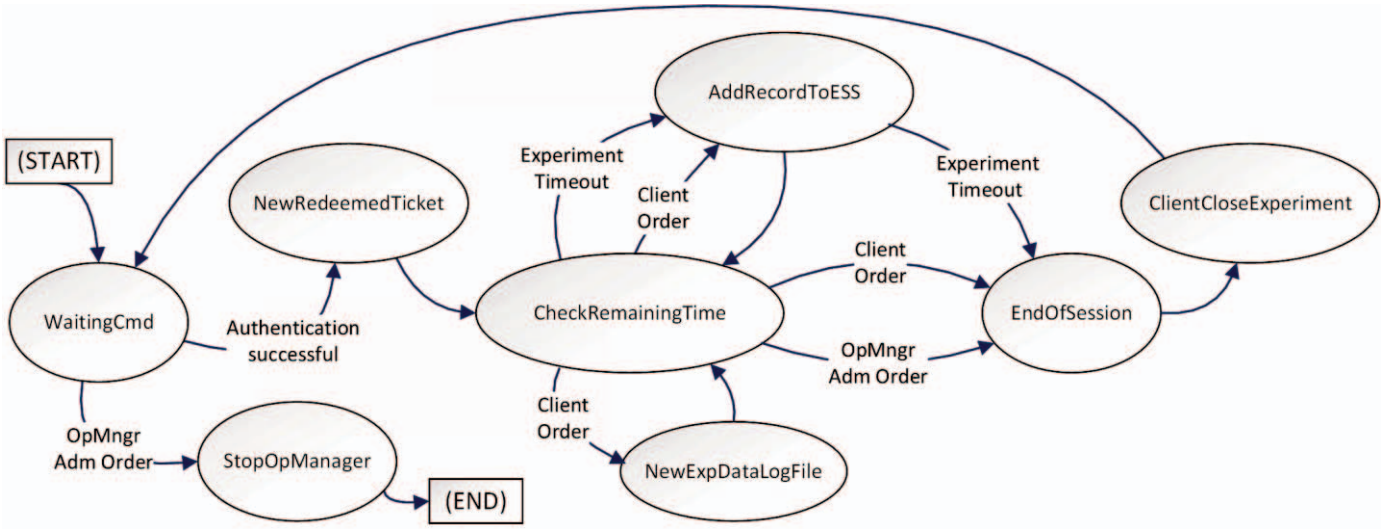

Fig. 5. Operations Manager states machine.

TABLE I. OPERATIONS MANAGER MODIFIED STATES MACHINE DETAILS

State	*Details*	*Details*
WaitingCmd	Neutral state; Operations Manager is waiting for a client authentication call.	On a successful authentication by a Client, switches to NewRedeemedTicket; On a user / administrator command to exit Operations Manager, switches to StopOpManager, that performs closing actions.
NewRedeemedTicket	Session initialization context.	Switches to CheckRemainingTime (unconditionally).
CheckRemainingTime	Checks for experiment finish time; updates experiment initial, current and finish times indicators and a Progress Bar (%).	Ongoing session context; If experiment time is over, switches to AddRecordToESS; External user / administrator orders may detour execution to other states (AddRecordToESS, NewExpDataLogFile, EnfOfSession).
AddRecordToESS	Makes a service call to ESS to send experiment data contained in the current log file.	If experiment time is over, switches to EndOfSession, otherwise returns to CheckRemainingTime.
NewExpDataLogFile	Commands the beginning of a new experiment data log file at ESS.	Returns to CheckRemainingTime (unconditionally).
EnfOfSession	Set a new random Token.	Switches to ClientCloseExperiment (unconditionally), closing records at the ESS.
ClientCloseExperiment	Makes a service call to ESS and orders experiment closing (no more saving operations).	Returns to WaitingCmd state (unconditionally).
StopOpManager	Executes stop sequence, clearing allocated resources.	(no next state, OpManager execution ends).

minimizing demand on the computer processor. The latter task is actually a parsing operation, carried out according to the Experiment configuration definitions initially taken from a local file, followed by the separation of the different values carried in the message and their conversion into the respective data types. The unpacked information is used to update a Global Variable (GV), in the form of a cluster of data types, containing the multiple Controls used in the experiment. Since the GV's are accessible from any point in the LabVIEW code running in the lab server computer, the data dispatched from the client becomes promptly available to the Experiment code.

Experiment Outputs: this thread continuously updates an outgoing SV with the contents of the experiment Outputs Cluster GV, also on a 1 ms update cycle time. The latest experiment data updates are properly formatted and forwarded to the client application every time the outgoing web service method is called.

Timing and Supervision: this thread keeps the clocks necessary to carry out experiment execution sessions updated with the Service Broker time. The synchronization is kept via periodic calls to a specific SB web service method. This allows for consistent verification of an experiment's remaining time in accordance with the scheduling timetables. In addition, an array of experiment loops states is updated (every parallel experiment loop is monitored continuously in a 'watch dog' sense), so that an administrator can check this array for problems with the experiment code during run time (e. g. the experiment code did not attend a regular command to stop execution). This feature, primarily added as a development-debugging tool rather useful in new experiment creation and start up test sequences, was aggregated as a handy administrative monitoring feature. Finally, this thread is also responsible for deleting old temporary files that have aged beyond the limit defined in the lab server configuration file.

Management: the management thread listens for incoming commands at its associated SV and submits them to a cascaded structure of slave loops. The first slave is the primary consumer of the orders and performs immediate actions directly. Orders related with ongoing experiment execution are forwarded to and executed by a secondary slave, the Modified State Machine thread. Client authentications for new sessions are submitted to this thread. An authorized client can start and stop the experiment any moment during the designated time block. Commands processed in this thread include orders to send locally stored experiment results to the ESS; to close a session at the SB; to start a new experiment data log file; and, processing a client termination message received from the client, notifying a user decision to do so. There are also local options at OpM for some commands - e. g. experiment start and stop, OpM reset and OpM stop. A 'brute force' command that aborts the execution of a running experiment code, in case it stops responding during execution, is available.

SB: this thread is in charge of attending incoming calls to the SB-called methods exposed by ELSA. Such calls are expected at specific moments – e.g., an order to finish an experiment due to a time out.

User Interface: this is the thread responsible for gathering OpM front panel inputs by the system administrator and executing the required actions internally.

Modified State Machine: this thread has the architecture of a non-standard finite states machine, and is in charge of controlling experiment execution sessions. Its design allows to dynamically interpose states in the execution flow on incoming orders. Interposing occasional sequence of states, (e.g. due to client calls or administrator actions) is important to keep the system responsiveness. The corresponding states diagram is shown in Figure 5 and explained in Table I. If necessary, future additions of functionalities to the system can be accomplished by expanding this logical structure to hold new states.

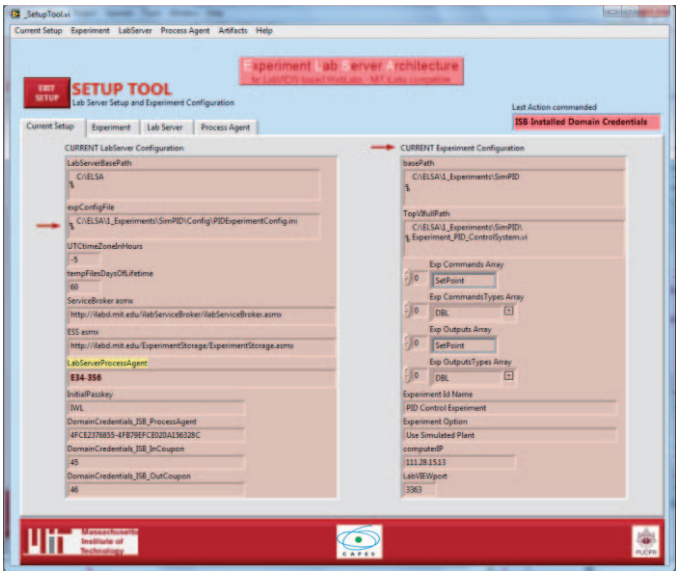

Fig. 6. Setup Tool to configure lab server and experiment.

ELSA communications traffic is monitored and logged to allow post-session analysis and eventual debugging. Messages related with experiment control and management, along with SB calls, feed history text fields at OpM that can be checked out by an administrator at any time. Log files store management and SB messages. Storing the traffic of specific experiment control messages is an option left to the experiment code, when relevant. Usually, the experiment variables storage at ESS is enough for most after-session practical purposes.

B. *Setup Tool*

The second ELSA component, the 'Setup Tool', shown in Figure 6, was designed for lab server configuration tasks. These include defining the registration information to be used in the linkage to an ISA SB (ELSA lab server Process Agent credentials), along with the ISA services end point URL's.

The configuration information of an ELSA lab server is managed by an administrator and stored in local files. The enabled experiment configuration, with its specific input / output parameters definitions, local paths in ELSA and experiment files are managed through this interface.

ELSA supports the following basic data types for parameters in an experiment definition: booleans, strings, double-precision floating point numbers, and integer numbers, both signed and unsigned, 32 or 64 bits. Any combination of these types can be used to design the sets of experiment commands (or controls) and outputs. Should any other composite or special type be needed, a string can be formatted and parsed by the experiment code. The creation of the specific clusters for the experiment commands and experiment outputs is carried out with the automated procedure using VI scripting techniques in LabVIEW. These Clusters definitions are then used to overwrite type definitions in .ctl LabVIEW files, which actually define the Global Variables contents.

C. *Experiment Integration*

Integrating a new experiment in ELSA requires few modifications in a ready-to-run LabVIEW application. Once the commands and outputs are defined and configuration files updated by using the Setup Tool, the clusters globals can be generated. Next, the experiment main VI input controls and output indicators must be manually attached to their respective instances in the clusters globals. This is easily accomplished employing ordinary unbundle and bundle operations, in a dedicated thread expected to be present in every experiment code. Such thread overwrites the original front panel controls with the experiment controls global cluster elements values - actually writing on them the information sent by the client, since the clusters globals are kept updated with respect to the respective SV's. On the way back, the local outputs indicators values are used to constantly update the experiment outputs global cluster, whose contents are packed within OpM into formatted strings used to keep the contents of each respective SV updated, wherefrom the information is took to feed answers to the client's polling web services calls. At a start experiment order, the experiment code launch takes place on an asynchronous call to its top VI file stored on the local hard disk. This call is executed at OpM and configured for "fire-and-forget". The call moves the top-level VI into computer's memory and runs it, in a procedure that explores LabVIEW's VI Server methods. At closing time, the VI is stopped and removed from memory dynamically. Its connecting pane must comply with a standard format designed to pass in two parameters: one used to enable remote operation mode, and a second for an arbitrary experiment option. Both are for custom (albeit non-mandatory) usage in the code, and are provided for increased flexibility.

Finally, a group of reserved Global variables used in the coordination between OpM and the experiment (e. g. communicating the effective experiment started / stopped state), and a Global 'Experiment stop' Boolean capable of effectively quitting all code threads, must be applied at the experiment code.

IV. The Switching Circuit Energy Balance iLab (IR Lab) in ISA/ELSA

In a pilot implementation, ELSA has been used to support a new remote experiment designed to teach students about energy flow and dissipation in a NMOS logic inverter. The temperature distribution in the various components of the

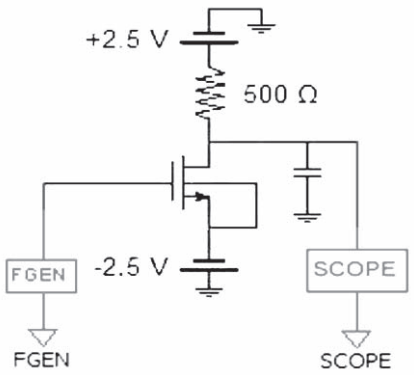

Fig. 7. NMOS 2N700 circuit layout.

circuit is visualized through an infra-red thermal camera. From a Java Applet client, students can adjust the parameters of a function generator, select a temperature measuring spot in the thermal image and watch how it behaves in a combined graph in response to waveforms inputs of varied shapes.

The experiment electric circuit in Figure 7 is described in an introduction panel at the JAVA client, so that the student can understand how the circuit is arranged, especially the location of the input and output in the circuit - the gate and drain of the transistor, respectively. By examining the thermal image of the circuit while the transistor is on, students should notice that the transistor and capacitor remain relatively cool while the resistor is heating up, helping them visualize the concept that most power dissipation occurs at the resistor. The transistor acts as a switch and not much heat dissipation should happen in it. The most significant voltage drop occurs across the resistor. Students may also calculate power dissipation and thermal time constants. By continually measuring the temperature, the student can see that the rise and drop of the temperature waveform is, approximately, exponentially rising on a rising input and exponentially decaying on a falling input - allowing for the calculation of the relevant time constants.

The Experiment has been designed with the commands and outputs reproduced in Table II, configured via ELSA Setup Tool and stored in a configuration '.ini' file. The experimentConfigFile parameter points to IR Lab Experiment configuration '.ini' file. This lab server configuration is shown in Table III. Also in Setup Tool, the Globals Clusters artifacts were generated, resulting in the elements shown in Figure 8.

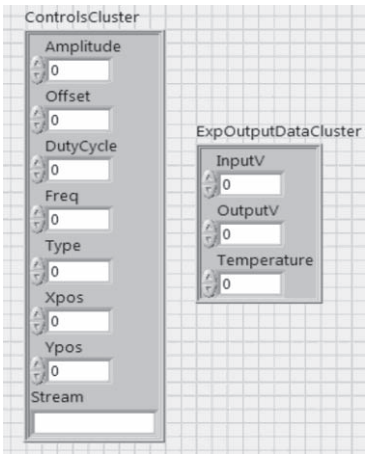

Fig. 8. Linkage globals clusters automatically generated after the provided experiment configuration information.

TABLE II. IR LAB EXPERIMENT CONFIGURATION INFORMATION

(ExperimentInstallInfo)	*(ExperimentVariables)*
expName = "IRLab Experiment" basePath = "C:\…\IRexp" TopVIfullPath = "C:\...\IRexp\IRlab_soap.vi" LabVIEWport = 3323 computerIP = "192.168.1.28"	expCommands = "Amplitude,Offset,DutyCycle,Freq,Type,Xpos,Ypos,Stream" expCommandsTypes = "DBL,DBL,DBL,DBL,STRING,DBL,DBL,STRING" numberOfCommands = "8" expOutputs = "InputV,OutputV,Temperature" numberOfOutputs = "3" expOutputsTypes = "DBL,DBL,DBL"

The remotely operated IR-Lab implementation using ELSA is schematically presented in Figure 9.

During the integration of the IR Lab infra-red camera, an unexpected technical issue came up: due to its internal architecture, it could not be configured as an IP camera. A web services-based data transfer solution, in the same scheme used for user commands and experiment outputs, was used to address the issue. In this alternative strategy to transmit raw, uncompressed image data, web services may not reach an usual image streaming performance, but can still be fast enough to capture a slowly changing phenomenon, as in the present case. Therefore, in order to circumvent the problem, the following approach was devised: the 8-bit, 320x256 grayscale image data is packed into an one-dimensional array, converted into a comma separated values string and transmitted to the client in answers to the streaming web service calls the client is programmed to do continuously. The image is promptly reassembled at the client by processing the transmitted string. The spot of interest in the picture for temperature follow up can be located by the user positioning a 'hot spot box'. Since viewing the hot spot and its surrounding area is enough for this experiment's purposes, it is possible to establish what sub part of the image is to be spared for transmission. The demanding streaming task favors an evaluation of the achievable web services-based throughput performance in transferring data.

The client interface is continuously updated through polling calls to the "Data" and "Stream" methods. The performance of the outgoing data transfer on the lab server side through these methods calls were measured during IRLab operation (incoming traffic used for Commands and Management is

TABLE III. ELSA LAB SERVER CONFIGURATION INFORMATION

(LabServer)	*(ISAasmxPages)*	*(Experiment)*
LabServerBasePath = "C:\\…\ELSA" UTCtimeZoneInHours = "-5" tempFilesDaysOfLifetime = "60" LabServerProcessAgent = "E34356…2013A4" InitialPasskey = "helloServiceBroker!" DomainCredentials_ISB_ProcessAgent = "1CC9…628C" DomainCredentials_ISB_InCoupon = "49" DomainCredentials_ISB_OutCoupon = "50"	ServiceBrokerAsmx = "http://ilabsproject.univ.edu/ilabServiceBroker/ilabServiceBroker.asmx" ExpStorageServiceAsmx = "http://ilabsproject.univ.edu/ExperimentStorage/ExperimentStorage.asmx"	experimentConfigFile = "C:\\ELSA\\1_Experiments\\IRLab\\Config\\IRLabExperimentConfig.ini"

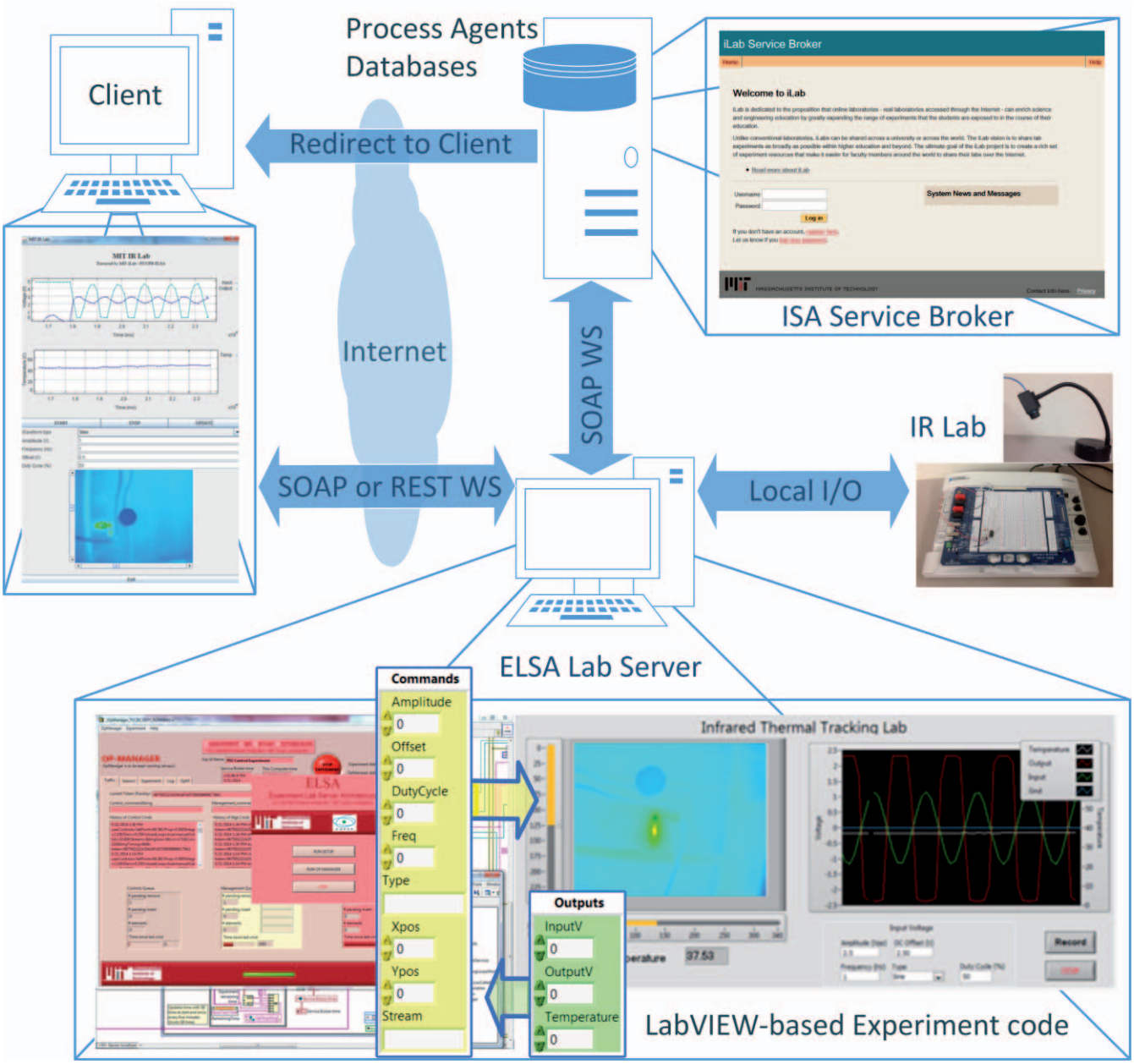

Fig. 9. IR Lab implementation in ISA / ELSA architecture.

negligible). Some observed experiment data and image-streaming transfer rates, considering only payload information, are presented in Table IV. In this pilot implementation, a JAVA Applet has been used for the client creation. Due to the Oracle's tightening on safety policies (which came up after this project, requiring code signing), HTML5 would now likely be a better option.

V. Discussion

In the tests with continuous data transfer to the client, the pilot implementation of the interactive IR Lab created under ISA / ELSA executed a responsive user interface. The rates of web services calls observed with client and lab server running on the same computer demonstrates the ability of the multi-thread design used in ELSA project to deal with continuous, non-stop calls efficiently. As expected, performance of this over-web-services communication scheme is rather limited by the speed and calls overhead in the network path between client and server then by the local processing capacity at the communicating ends.

The useful payload data transfer rates verified between a caller client located in South America and an ELSA lab server in North America sustained a minimum of 40kB/s, fast enough for a clear, natural user experience. As per the tests done, the present web services architecture supported small-size image streaming satisfactorily for the IR Lab purposes. If video streaming is left to specialized web servers devices (e. g. IP cameras), optimally designed for high throughputs, there would

TABLE IV. PAYLOAD DATA TRANSFER RATES FOR 600S (NORMALIZED) TEST TIME.

Client location	*Lab Server location*	*Experiment Data*			*Streaming*		
		traffic (MB)	*rate (kB/s)*	*web service calls*	*traffic (MB)*	*rate (kB/s)*	*web service calls*
CECI/MIT network[2]	Same computer	2.75	4.70	10464	1321.73	2255.75	4244
CECI/MIT network[2]	CECI/MIT network[2]	2.23	3.81	8473	494.99	844.77	1602
PUCPR 60Mbps (download) network[1]	CECI/MIT network[2]	0.64	1.09	2418	24.05	41.05	78
Private 15Mbps (download) network[1]	CECI/MIT network[2]	0.66	1.12	2498	23.45	40.02	86

be room for many additional experiment variables (in a comparable context), as is suggested by the ratio between the streaming and experiment data transmission rates observed.

The web services approach may find a limit in data transfer speeds, as calls overhead and packages routing over internet vary the throughput constantly. Overheads are higher with SOAP then with REST services, and that's why REST is preferred when speed is important, like in the traffic between client and experiment. This could be a limiting factor on the types of RE's that run with an acceptable quality of user interaction, despite network speed is more of a common problem in any remote experimentation architecture. Nevertheless, proper design of experiments codes can help to minimize update rates problems. Large amounts of data – e. g., data acquired at high sampling rates - can rather be processed locally and then sent to the client side in a simplified form. Anyways, even adopting best practices, web services approach will pose a limit to performance at some point.

Usually, one ISA Service Broker will manage several interactive experiments under the responsibility of an administrator. Considering that interactive remote experiments may run for a long time, safety of the real system is an issue (as much as in the design context of automated systems in general). As an additional safeguard, options for prompt, remote administrative actions, providing supervision tools to check running experiments status at any time are recommended. Such a supervision tool feature can be easily set up in ELSA by defining a private 'safety Token', that an administrator can enter in the supervisory application to check for the status or variables of an experiment and, there from, send master overriding commands if necessary. It's worth noting that implementing such feature in the context of the so called LabVIEW Remote Panel feature is not such a simple task, as the remote operation is granted by the LabVIEW Web Server to one computer at a time only.

The client design using the web services architecture is very flexible, allowing for custom clients. Importing web services and their methods into a library to be used in the client creation is currently an embedded feature in most software development platforms. In this sense, the student is no longer limited to interacting with an experiment using a fixed set of programmed client functions, but can actually develop his/her own clients. This is actually the highlighted advantage of LaaS architecture, designed with a high level of abstraction, allowing for the attachment to virtually any RLMS. However, either a middleware similar to ELSA will be created to support the integration with the services administration layer of each RLMS, or this point will have to be addressed in the code development of each experiment design.

Eventually, the template of an editable client can be provided, and then be incremented by students with routines to make arbitrary, dynamic use of data during remote experiment execution. As an illustrative example, several online feedback control experiments test the capacity of students to select an efficient set of values for the parameters of a control algorithm running on the experiment code. The task of calculating the controller output based on the process variable readouts (both over web services) can be transferred to the student, i.e., the design and coding of the control algorithm, with the architecture of his/her choice, expanding his/her role in the remote experimentation. Naturally, effects of the internet latency will have to be taken into account when determinism is relevant for the application. In this user-customizes-the-client scenario, practical implementation aspects under ISA / ELSA would require a minor adaptation: since a modified, user-built client would not be launched by the ISA Service Broker, the authentication that normally goes embedded in the client would be alternatively provided via an intermediary application (e. g. a pre-defined redirect web page, or "bridge" client), where from it can be transferred to the user's own client.

The degree of modularity and loose coupling provided by the present lab server architecture will favor its application in the creation of collaborative experiments. In such arrangements, students in different locations share the operation of a real system and do a concurrent effort to collectively pursue an objective. By calling the web services of an ELSA lab server, multiple clients can be enabled to act upon the same remote experiment, each taking care of an assigned sub-system, by using a subset of the defined Experiment Controls. ELSA can seamlessly manage different clients in one same session by checking the tokens submitted along with the methods calls in a list of valid ones for a multiple-client session, with the definition of sub-experiments in ISA. The sub-experiments and their respective clients can be set up in the SB to share common time blocks, and to point to the same ELSA lab server, which will just route the incoming commands and make them available to the experiment. After the first client has successfully authenticated and session information been stored, the following one(s) only need have the SB provided token(s) checked and stored locally in the control list.

VI. CONCLUSION

The ELSA web services–based solution for remote operation of laboratories under ISA was successfully devised,

implemented, tested, and fulfilled all design requirements / features initially proposed. According to tests done with a pilot implementation, the reached performance was observed to be well enough to qualify the architecture for the remote operation of a broad range of RE's applications under ISA.

Basically, ELSA benefits from the high degree of modularity provided by a concise set of web services. Its integration with the level of services administration in ISA completely detaches the LabVIEW experiment code from administrative issues. The standard and small set of web services used in the client-experiment conversation, taking care of management tasks and parameters traffic, favors code reuse in both client and experiment creation. Due to the loose coupling of the LabVIEW-based experiment code, this can be can be created as a regular stand-alone application without initial concern with the remote operation and supportive administrative features. Existent LabVIEW applications can be adapted for remote operability with minor modifications. Consequently, shorter and less expensive developments of remote experiments can be expected.

The addition of ELSA extension to the ISA platform provides a powerful set of resources to expand remote experimentation by combining ISA administrative capabilities with faster design, integration and deployment of LabVIEW-based laboratories. ELSA's convenient architectural features simplify the creation of RE's and make way to exploring user-defined clients and multi-user experiments. Overall, ELSA is a contribution to fostering the goals of remotely operated educational laboratories, i. e., to broaden learning experience opportunities by connecting students and remote equipment, and increase the usage of expensive laboratory equipment, often found idle, for long periods, in most engineering education institutions.

ACKNOWLEDGMENT

The authors wish to express their gratitude to the supporters of this research: PUCPR-Brazil, CAPES-Brazil Post-doctoral Program grant# BEX-17932-12-2, MIT Donner Endowed Chair, CECI, National Science Foundation under E3S STC grant #0939514, and MIT Class of 1960 Endowment Fund for Innovation in Education.

REFERENCES

[1] M. Stefanovic, "The objectives, architectures and effects of distance learning laboratories for industrial engineering education," *Comput. Educ.*, vol. 69, pp. 250–262, 2013.

[2] J. V. Nickerson, J. E. Corter, S. K. Esche, and C. Chassapis, "A model for evaluating the effectiveness of remote engineering laboratories and simulations in education," *Comput. Educ.*, vol. 49, no. 3, pp. 708–725, 2007.

[3] D. a Harris and C. Krousgrill, "Distance Education: New Technologies and New Directions," *Proc. IEEE*, vol. 96, no. 6, pp. 917–930, 2008.

[4] H. Wuttke, M. Hamann, and K. Henke, "Integration of Remote and Virtual Laboratories in the Educational Process," *Int. J. Online Eng.*, vol. 11, no. 3, pp. 62–67, 2015.

[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," *Comput. Educ.*, vol. 57, no. 3, pp. 2054–2067, 2011.

[6] E. Fabregas, G. Farias, S. Dormido-Canto, S. Dormido, and F. Esquembre, "Developing a remote laboratory for engineering education," *Comput. Educ.*, vol. 57, no. 2, pp. 1686–1697, 2011.

[7] L. Gomes and S. Bogosyan, "Current Trends in Remote Laboratories," *IEEE Trans. Ind. Electron.*, vol. 56, no. 12, pp. 4744–4756, 2009.

[8] L. A. Mendes, M. Debner, and M. T. C. de Siqueira, "Systematization of the WebLabs Development Process : Towards an Approach Proposal," in *International Conference on Engineering Education, ICEE-2010*, 2010, pp. 1–9.

[9] J. Garcia-Zubia, P. Orduna, D. Lopez-de-Ipina, G. R. G. R. Alves, J. García-zubia, P. Orduña, and D. López-de-ipiña, "Addressing Software Impact in the Design of Remote Laboratories," *IEEE Trans. Ind. Electron.*, vol. 56, no. 12, pp. 4757–4767, 2009.

[10] D. Lowe, S. Murray, E. Lindsay, and D. Liu, "Evolving remote laboratory architectures to leverage emerging internet technologies," *IEEE Trans. Learn. Technol.*, vol. 2, no. 4, pp. 289–294, 2009.

[11] V. J. Harward, J. a. Del Alamo, S. R. Lerman, P. H. Bailey, J. Carpenter, K. DeLong, C. Felknor, J. Hardison, B. Harrison, I. Jabbour, P. D. Long, T. Mao, L. Naamani, J. Northridge, M. Schulz, D. Talavera, C. D. Varadharajan, S. Wang, K. Yehia, R. Zbib, and D. Zych, "The iLab shared architecture: A web services infrastructure to build communities of internet accessible laboratories," *Proc. IEEE*, vol. 96, no. 6, pp. 931–950, 2008.

[12] J. L. Hardison, K. DeLong, P. H. Bailey, and V. J. Harward, "Deploying interactive remote labs using the iLab Shared Architecture," *Proc. - Front. Educ. Conf. FIE*, pp. 1–6, 2008.

[13] J. Garcia-Zubia and P. Orduña, "Towards a distributed architecture for remote laboratories," *iJOE*, vol. 4, no. 1, pp. 11–14, 2008.

[14] E. G. Guimaraes, E. Cardozo, D. H. Moraes, and P. R. Coelho, "Design and Implementation Issues for Modern Remote Laboratories," *IEEE Trans. Learn. Technol.*, vol. 4, no. 2, pp. 149–161, 2011.

[15] A. Maiti, A. D. Maxwell, and A. a. Kist, "An overview of system architectures for Remote Laboratories," *Proc. 2013 IEEE Int. Conf. Teaching, Assess. Learn. Eng. TALE 2013*, no. August, pp. 661–666, 2013.

[16] A. Agrawal and S. Srivastava, "WebLab: A Generic Architecture for Remote Laboratories," *15th Int. Conf. Adv. Comput. Commun. (ADCOM 2007)*, pp. 301–306, 2007.

[17] D. Lowe, "Integrating reservations and queuing in remote laboratory scheduling," *IEEE Trans. Learn. Technol.*, vol. 6, no. 1, pp. 73–84, 2013.

[18] A. Maiti, A. A. Kist, and A. D. Maxwell, "Real-Time Remote Access Laboratory With Distributed and Modular Design," vol. 62, no. 6, pp. 3607–3618, 2015.

[19] M. a Prada, J. J. Fuertes, S. Alonso, S. García, and M. Domínguez, "Computers & Education Challenges and solutions in remote laboratories . Application to a remote laboratory of an electro-pneumatic classi fi cation cell," *Comput. Educ.*, vol. 85, pp. 180–190, 2015.

[20] S. Dutta, S. Prakash, D. Estrada, and E. Pop, "A web service and interface for remote electronic device characterization," *IEEE Trans. Educ.*, vol. 54, no. 4, pp. 646–651, 2011.

[21] A. Bagnasco, A. Boccardo, P. Buschiazzo, A. Poggi, and A. M. Scapolla, "A service-oriented educational laboratory for electronics," *IEEE Trans. Ind. Electron.*, vol. 56, no. 12, pp. 4768–4775, 2009.

[22] E.-S. Aziz, Z. Wang, S. K. Esche, and C. Chassapis, "Development of a modularized architecture for remote-access laboratories," *ASEE Annu. Conf. Expo. Conf. Proc.*, pp. 1–16, 2011.

[23] M. Tawfik, C. Salzmann, D. Gillet, D. Lowe, H. Saliah-Hassane, E. Sancristobal, and M. Castro, "Laboratory as a service (LaaS): A novel paradigm for developing and implementing modular remote laboratories," *Int. J. Online Eng.*, vol. 10, no. 4, pp. 13–21, 2014.

[24] M. Tawfik, E. Sancristobal, S. Martin, R. Gil, G. Diaz, A. Colmenar, J. Peire, M. Castro, K. Nilsson, J. Zackrisson, L. Hakansson, and I. Gustavsson, "Virtual instrument systems in reality (VISIR) for remote wiring and measurement of electronic circuits on breadboard," *IEEE Trans. Learn. Technol.*, vol. 6, no. 1, pp. 60–72, 2013.

[25] M. Stefanovic, V. Cvijetkovic, M. Matijevic, and V. Simic, "A LabVIEW-based remote laboratory experiments for control engineering education," *Comput. Appl. Eng. Educ.*, vol. 19, no. 3, pp. 539–549, 2011.

[26] K. Bauer and L. A. Mendes, "Weblab of a Control Experiment in a Newborn Baby Incubator," *REV 2015 12th Int. Conf. Remote Eng. Virtual Instrum.*, no. February, pp. 163–171, 2015.

[27] K. Yeung and J. Huang, "Development of a remote-access laboratory: A dc motor control experiment," *Comput. Ind.*, vol. 52, no. 3, pp. 305–311, 2003.

[28] P. Orduña, J. García-Zubia, L. Rodriguez-Gil, J. Irurzun, D. López-De-Ipina, and F. Gazzola, "Using LabVIEW remote panel in remote laboratories: Advantages and disadvantages," *IEEE Glob. Eng. Educ. Conf. EDUCON*, 2012.

[29] J. Garcia-Zubia, D. López-de-Ipiña, and P. Orduña, "Mobile devices and remote labs in engineering education," *Proc. - 8th IEEE Int. Conf. Adv. Learn. Technol. ICALT 2008*, pp. 620–622, 2008.

[30] A. A. Cruz, F. a L. Gomes, F. a C. M. Cardoso, E. B. Martin, and D. S. Arantes, "Development of a robust and flexible Weblab framework based on AJAX and design patterns," *Proc. - Seventh IEEE Int. Symp. Clust. Comput. Grid, CCGrid 2007*, pp. 811–816, 2007.

[31] V. Kumar, "ECG Acquisition Under Incorrect Electrode Positions," *REV 2015 12th Int. Conf. Remote Eng. Virtual Instrum.*, pp. 63–68, 2015.

[32] M. Shaw, "What Makes Good Research in Software Engineering ?," vol. 4, no. 1, pp. 1–7, 2002.