CoderLabs: A Cloud based platform for Real Time Online Labs with User Collaboration

Abdellah Touhafi[1], An Braeken[1], Abderrahim Tahiri[2], Mostapha Zbakh[3]
(1)INDI - Industrial Engineering Sciences Department
Vrije Universiteit Brussel - VUB
Brussels, Belgium
(2) Computer Science Engineering Department
National School of Applied Sciences
Abdelmalek Essaädi University
(3) ENSIAS
Mohammed V University
Rabat, Morocco

Abdellah.touhafi@vub.ac.be, an.braeken@vub.ac.be, abderahim.tahiri@gmail.com ,m.zbakh@um5s.net.ma,

Abstract—**In this paper we describe the architecture of a real time environment for numerous remote experiments. The environment is created with web standards as HTML5 such that no plug-in needs to be installed by the user. Users are able to use the remote lab simultaneously and in collaboration. This collaboration between users is made feasible by adopting a message broker. Finally, by using Google Coder, developers can easily change or create the user interface of their remote experiments and share experiments in the cloud.**

Keywords-component; Real Time Online Labs, Google Coder, REDIS, Lab Server, Relay Server, Node.js

I. INTRODUCTION

Experience in laboratory work is crucial for an engineer. It allows translating the theoretical concepts in practice, leading to the competence of solving practical problems. Although, it is not always feasible to practically organize these lab sessions, due to limited availability of expensive equipment or due to flexible student programs for working students. Online laboratories can offer a solution in these situations. In fact, online laboratories are composed of virtual or remote experiments. Virtual laboratories offer a simulation environment, while remote laboratories execute experiments with real equipment and infrastructure. Remote experiments can be further split in batch, sensor and interactive experiments. Batch experiments are experiments without any type of interaction. Sensor experiments monitor the real time sensor without influence of the user. The focus in this work is on remote interactive laboratories and its implementations. Although, it would be perfectly possible to use our environment also for batched and sensor experiments. Moreover, we note that the purpose of the paper is not the study of the pedagogical impact like in [1], [2], [3]. Instead, the technical aspects are described to offer a solution using the latest standard web technologies.

A large number of online laboratories already exist in literature [8],[9],[10],[14]. However, none of them answered our predefined goals of collaboration between users and simplicity of design for developers. Moreover, we opted for a real time environment, where real time means that answers are guaranteed in a reasonable time frame. Our developed online lab environment is called CoderLabs. In the whole design of CoderLabs, existing standards are used as much as possible instead of own developed software. By using open source software, the online laboratory is able to evolve together with the tools.

The outline of the paper is as follows. Section 2 gives some background on the architecture of CoderLabs and compares it with other online labs. Section 3 explains the experimental setting, used to show the operation of CoderLabs. In Section 4, the lab server is described. Section 5 deals with Redis, the link between the separated relay server and web server. Sections 6 and 7 explain the relay server and web server respectively. In Section 8, the client is described. Finally, Section 9 ends with conclusions and possible extensions.

II. BACKGROUND

In online laboratories, methods or procedures are called on a remote computer. Remote calls can be implemented in different ways. In the architecture of the first online laboratories, a client has direct access to a server, located in the laboratory. Through this server, the client is connected with the equipment that can be remotely used. This architecture implies several problems. Firstly, the server in the laboratory is often a simple desktop computer, not configured to operate as server for a large number of clients. As we have made collaboration with a large number of users in a real time environment as a crucial condition, this is clearly not a

978-1-4673-8894-8/16/$31.00 ©2016 IEEE

satisfactory setting. Secondly, the server is mostly positioned in the internal LAN. Since the internal LAN is blocked by a firewall, the server is not directly available from outside. Moreover, this situation can lead to several security risks. If the server is compromised, the attacker has direct access to the whole internal network. Thirdly, this server is mostly not equipped with functionalities for authorization and authentication since the goal of the server is to deal with the major functionalities, being the import of data, the realization of the experiment and the export of data.

An architecture based on a relay service outside the local network is a better solution. The relay service can be placed in a Demilitarized Zone (DMZ) or hosted on a Cloud server. The relay service functions as a mediator between the laboratory environment and the client. If the relay service is hosted in the cloud, only the traffic between the relay service and the laboratory will use Internet connection of the institute. If for instance a video feed should be sent from the laboratory to the clients, it only needs to go from lab server to relay service and not from the lab server to each individual client. Thanks to a loose coupling of the lab servers, the relay service and the web server, this architecture can evolve to a Service Oriented Architecture (SOA). However, the initial condition of keeping a simple structure will not stand since in a SOA architecture the different parts of the architecture are split in several services with each a clearly defined task.

A relay server can be implemented as part of the web server or stand-alone. The most well-known existing online laboratories like MIT iLab [4], Weblab Deusto [5], Sahara labs [6], Virtual Instrument Systems in Reality (VISIR) [2] and the Library of Labs (Lila) [3] are built following the first way. On the other hand, in CoderLabs, web server and relay server are split. As a consequence, both can focus on their main task. The web server handles the requests for the web pages sent by the client, the relay server relays the messages between the client and the laboratory. It is important to mention that web server and relay server work in parallel in order to avoid an extra network hop. The architecture of CoderLabs is depicted in Figure 1.

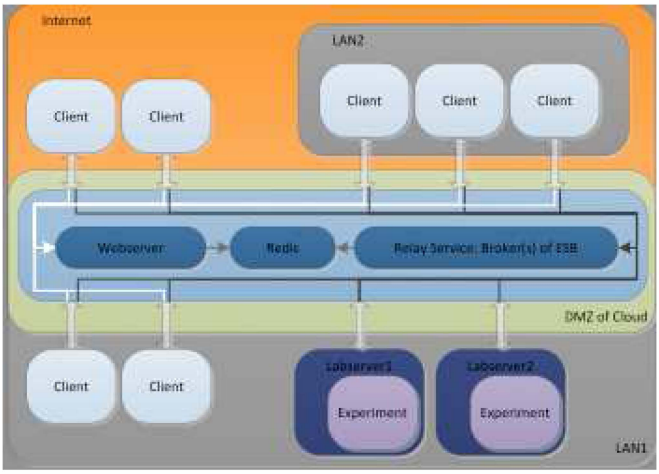

Figure 1: Coderlabs Architecture

The effective implementation of a relay service can be done by means of a broker or several brokers on an Enterprise Service Bus (ESB). Moreover, this architecture avoids most problems with firewalls and the lab server only needs to handle a few connections. The message broker Apollo is used in CoderLabs.

Two requirements were also crucial for CoderLabs: sharing of labs and collaboration of users. With respect to the first requirement, only the online lab environments of MIT iLab, WebLab Deusto and Lila offer that functionality. These labs use a federated architecture and corresponding federated authentication mechanisms. On the other hand, we found only 1 online lab, Sahara, of those who were examined by us which allows collaboration of users. In this context, Sahara defines active and passive slave users. The other online labs mostly work with a ticketing service, allowing users to reserve a time slot to the lab.

III. EXPERIMENT

In order to demonstrate the online lab environment, we created a remote experiment, allowing remote control of a logic analyzer, connected with a Field Programmable Gate Array (FPGA) that can be remotely reconfigured. The goal of the experiment is to program or reconfigure the FPGA, and to monitor the behavior of the FPGA with the logic analyzer. In this setting, the Spartan 6 Atlys FPGA is coupled with an Agilent Logic Analyzer 16804A. The Atlys board allows communication with a computer through a USB2 or a gigabit Ethernet port. The logic analyzer is based on a Windows XP operating system, contains an Ethernet port and supports a Component Object Model (COM) automation server. Consequently, the logic analyzer can be operated by means of programming code on the LAN network by calling and manipulating objects through COM or DCOM. The logic analyzer is connected through USB2 with the Atlys Adept USB port of the FPGA. The probe of the logic analyzer is connected on the 68 pins VHDC connector of the FPGA board for the measurements. Figure 2 shows the experimental setting.

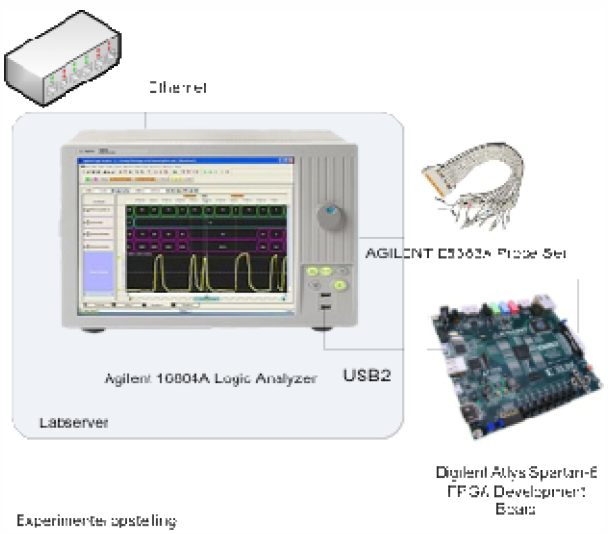

Figure 2. Experimental setting in online lab of CoderLabs.

IV. LAB SERVER

The lab server can communicate with the lab equipment on the one hand and the web server and relay server on the other hand. We now explain both types of communications.

A. Communication between lab server and lab equipment

For the physical communication between computer and lab equipment the General Purpose Interface Bus (GPIB) and USB are most often used. If higher bandwidth is required, VME eXtensions for Instrumentation (VXI) or PCI eXtensions for Instrumentation (PXI) are preferred. Moreover, nowadays much lab equipment has an Ethernet port for communication through the network. In our remote lab environment, we have a different situation for communication with the logic analyzer and the FPGA board.

Logic analyzer. For the specific experimental setting of the logic analyzer, the COM interface is used, since there are no Interchangeable Virtual Instruments (IVI) drivers available for our analyzer. It would be possible to develop a COM wrapper that offers an IVI-COM interface, but that would require unnecessarily much effort.

PythonCom, consisting of win32com and pythonCom extensions module, is used to call the COM objects. For usage in client application, only the client module, that calls the COM interfaces, is required. A COM object can implement several interfaces, each having an interface ID (IID), which represents a Globally Unique Identifier. To connect through COM a certain COM object, a program ID or ProgID is used. The logic analyzer offers its interface as a COM automation object. In such a way, the methods and properties of a COM object can be determined at runtime. For the logic analyzer, methods are available for selecting the right module, defining triggers, running the analyzer,... Simple triggers can be added by means of the SimpleTrigger method. More advanced triggers are defined through an XML file. In order to execute this trigger, the Trigger property of the AnalyzerModule object is completed with a string value that satisfies the XML trigger specification. Another way is by loading a locally available trigger XML file through the RecallTriggerByFile methode. Using the Run method of the instrument object, all analyzer modules are started. The WaitComplete method waits until all data acquisition modules have finished their measurements or if a certain specified time-out is reached.

Note that Labview, software package of National Instruments, is another solution for the driver problem. Labview does not offer a common driver model but contains a lot of drivers for communication of a computer with lab infrastructure.

Moreover, Labview offers the ability to quickly build graphical interfaces for sending commands to the infrastructure and visualize data. Its modular structure and its dataflow oriented programming language G, makes Labview [6] a standard for online experiments, especially in industrial environments. However, if the Labview server is directly used, this architecture does not support collaboration of the users. Moreover, installation of a Labview runtime browser plug-in is required. Also the authentication options in Labview are very restricted and at the time of writing no external authentication provider can be used. Finally, a considerable license cost needs to be taken into account. Consequently, there are enough reasons to question the use of Labview as labserver in our setting [7].

FPGA board. To handle the communication between the lab server and the FPGA board, FPGALink is used. This package allows the communication of a computer with a whole range of popular FPGA boards, which are using FX2LP or Atmel AVR. Through TCL commands it is possible to send commands to the Xilinx tools to load program code into the FPGA. FPGALink has a modular structure. The NeroJTAG module is used to load a csvf file in the FPGA. The CommFGPA module handles the communication between the FPGA and the host computer. Both modules use the Libusb C library for their USB communication. This library facilitates the implementation of drivers for synchronous and asynchronous communication over the USB bus.

B. Communication between lab server and webserver/relay server

In the experimental setting, the logic analyzer is used as lab server since it runs Windows OS and is powerful enough. The lab server runs two Python programs. One program handles the communication with the logic analyzer through COM, the other takes care of the communication of the FPGA board through FPGALink. Both programs work independently and are connected with the relay service through the Simple/Streaming Text Oriented Messaging Protocol (STOMP) protocol via a Python STOMP client library, called stomp.py. They both connect to the relay service with the same credentials and subscribe on the same topic.

By using the websocket-client module that offers an interface, similar to the to Javascript available websocket API, a websocket transport (WS) mechanism can be added. However, the usage of STOMP has the advantage that a message broker

can be freely chosen and the communication with an ESB can be developed without special message bridge.

V. REDIS

Splitting the web server and the relay server has consequences for the authentication process. When a user logs in on the web server, the relay server also needs to be informed from the fact that the user is authorized to send messages through the relay server to the lab server and vice versa. In CoderLabs, Redis is used for authorization purposes. Redis in CoderLabs, is put in the same secured internal network as the relay server and webserver. Preferably, the machine running Redis, allows only traffic from the lab server and web server to the Redis port 6379.

Redis is not a classical Relational Database Systems (RDBS). It represents a NoSQL solution, belonging to the group of key value stores. In key-value stores, simple key-value combinations are stored, which is for this setting more than enough. Since NoSQL solutions do not contain schemes, they allow faster write operations than RDBS. Redis can store besides key-value pairs, also data structures like lists, hash sets and sorted sets. A main restriction of Redis is that all data should fit in RAM, allowing very fast write operations. Data can be partitioned over several instances of Redis, called sharding.

Redis is used in CoderLabs for four purposes. Firstly, the key data type of Redis is used for the login procedure. There exists a very practical method in Redis to expire keys after a self defined period, using the commands setex, expire and expireat. A second purpose is to store the rights. Several groups can be coupled to a remote lab and users can be member of several groups. Although Redis does not contain schemes, it is still possible to build relations in Redis with the Redis set datatype. Thirdly, Redis is used as session store for Express, a web application framework for Node.js, which will be used as web server as explained later. Here, the Node.js connect-redis module is used. As a consequence, seperated Node.js processes can still be aware of all session data.

VI. RELAY SERVICE

Several methods exist for relaying messages. CoderLabs uses a message broker architecture for this purpose. Since this architecture already exists for quite a long time, several open source message brokers are available and have been tested with the STOMP protocol. We compared Apache ActiveMQ, Apache Apollo and RabbitMQ. Apache ActiveMQ is the most popular open source message broker, written in JAVA. This broker works standard with a blocking threaded model, making a thread for each connection. Apache Apollo is a newer open source broker of Apache, written in Scala. Apollo uses a non-blocking reactor based model with a thread pool where the number of threads used for connection handling is proportional with the number of CPU cores. RabbitMQ is another popular open source broker, written in Erlang. RabbitMQ uses asynchronous I/O with a small thread pool, running many lightweight Erlang processes. We compared the speed of these three brokers in two tests.

In the first test, the time was measured to echo a fixed "Hello World" message. The message is sent from the browser via the relay server to a TCP connected lab server, implemented in Python. The lab server then echoes the received message back to the browser via the relay server. The average response time and the confidence intervals were computed by repeating this action thousand times at arrival of this echo message. Mozilla Firefox 15.0.1 is used as browser, Python 2.7.3 32 bit for the lab server, and a 64 bit variant for each relay server. The tests were executed on an Intel i5-2450M 2.5GHz with 4GB RAM and 64bit Windows 7 Pro as OS. From Figure 3, it can be concluded that Apollo and Rabbitmq clearly outperform ActiveMQ. In a second test, ten thousand "Hello World from Python" messages are sent to a queue and repeated hundred times by a publisher implemented in Python. Afterwards, a listener starts and subscribes on this queue. Finally, for each of this 100 repeats, the time is measured between the reception of the first message and the last message out of these ten thousand messages. Listener and publisher use Stomp.py over TCP. In both tests, no persistent messages are used. Here, message broker Apollo clearly outperforms RabbitMQ and ActiveMQ, as can be seen in Figure 3.

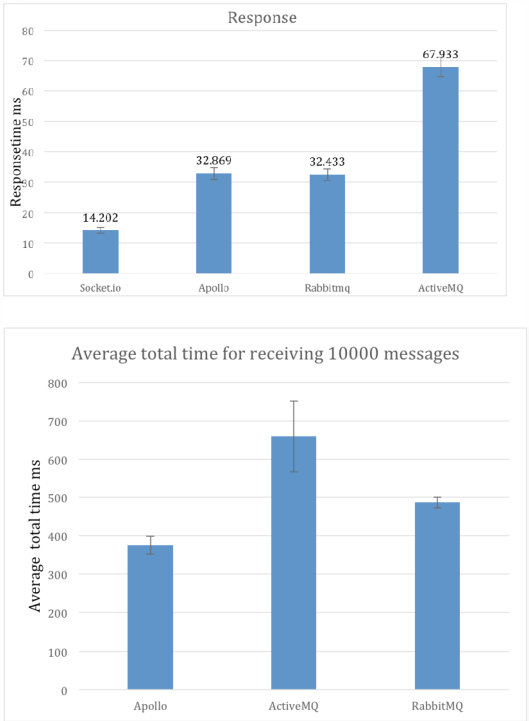

Figure 3: Comparison of response time for different brokers

ActiveMQ offers a management interface, addressable by Java Management Extensions. The management interface of Apollo and RabbitMQ is addressed with REST. Since most programming languages contain a REST client library, many other programming languages instead of only Java can

perform the configuration of Apollo. Moreover, Apollo also automatically reloads in case of changes at the configuration.
ActiveMQ, Apollo and RabbitMQ support the MQTT and STOMP protocols. The openWire protocol is not supported by RabbitMQ, but well by the other two.
It should be noted that the different supported protocols are plugins. This allows, in the future, a more easy support for new protocols or new versions of existing protocols.
Taking into account the speed tests, the REST management interface and the larger number of supported protocols, Apollo has been chosen as message broker in CoderLabs. Apollo enables message swapping, determining which messages from the queue are written to the hard disk. As a consequence, this leads to an optimization of the memory usage of the server, hosting Apollo.
Java Authentication and Authorization Service Modules (JAAS) support the authentication in Apollo. A standard JAAS module in Apollo uses file-based authentication, entered in user and group property files. A standard JAAS Lightweight Directory Access Protocol (LDAP) module was already available. By implementing javas.security.auth.spi.LoginModule interface, a new JAAS login module, specific for CoderLabs, is developed. Jedis, a Java library for communication with Redis, takes care of the communication between the in Java written login module and Redis.
Authorization in Apollo is obtained by Access Control Lists (ACL). These lists contain access rules specifying which sources (brokers, connectors, virtual hosts, queues, topics or dsubs) can execute which group of actions.

VII. WEB SERVER

Google Coder is selected as tool to develop the web interface by remote lab designers, since it is an online open source editor with no extra server software requirements. Google coder is developed by Google to create websites running on the Raspberry Pi. Google Coder uses Node.js as server software and is based on web standards like HTML, CSS and Javascript. Since it is developed to run on a Raspberry Pi, it uses only one Node.js process. Google Coder has a modular structure, where each App is a separate Node.js module. We call an App related to an online lab, a labapp. Coder uses the Express web application framework with Mustache as templating engine. For the connection to the broker, the labapp designer needs to add a reference to a client side javascript library like Stomp.js.
The web server is a crucial part of the online lab. Consequently, the web server should be stable, scalable and secure. Node.js is a software platform to develop scalable network applications and provides modules so that Node.js can be used as webserver. Node.js was previously used as real time server in Orchestra as part of the Chrome Weblab. Orchestra provided an environment where users could simultaneously play music instruments from the browser. Node.js uses javascript as server side script language and works with one event loop, running in one single thread. The disadvantage is that multi-core processors are not by default optimally used by Node.js. The advantage is that no concurrency problems occur since all operations run sequentially. Node.js uses an event loop to process asynchronous requests, in contrast with blocking synchronous servers that make a new process or thread for each new client request. Also many other webservers, like Nginx, Netty, Vert.x, Twisted,... are built following the asynchronous model. Note that using a webserver that follows an asynchronous model is not enough. If own programming code or a library spawns several threads, all advantages of an asynchronous web server are lost. That makes Node.js interesting because Node.js is developed with the idea that all IO should be non blocking and thus most of the libraries are. We now describe its architecture into more detail.

A. Nodes.js architecture

Google's V8 javascript is used to interpret the javascript code on Node.js. Before (until version 0.9), Node.js was based on the libev eventloop library that was only available on the Unix operating system. Therefore, these days, libuv is used. Node.js is relatively new and that is why several measurements should be taken in order to guarantee our requirements of stability, scalability and security.

Stability. Node.js runs by default as one single process, not as a service or deamon. If the Node.js process fails, the webserver does not recover. Consequently, it is necessary to use an external package, like eg. *forever*, to ensure that the webserver remains online.

Scalability. There are two types of scalability, vertically and horizontally. Vertical scalability means the addition of extra sources to a certain node. Horizontal scalability corresponds to the addition of a new node. Vertical scalability can be obtained by using Node.js cluster. There are different methods to organize horizontal scalability. One can use a layer three load balancer in front of the nodes that sends all IP packages of a certain client IP, port combination to that node and divides the different clients well-balanced over the nodes. It is also possible that each node is aware of all client sessions by exchanging session data through Redis. In this last setting, a reverse proxy server is required, like for instance the node-http-proxy module.

Security. Here we discuss the authentication and authorization aspects. For authentication, distinction should be made between authentication of user and authentication of lab. Labs are directly connected to the relay service, while users first need to login through the web server. Since the webserver sends the login page with a session cookie, the user is uniquely identified during the lifetime of the session. If the login of the user is successful, the server side session of the user is set to authenticated. The unique session-id is added to the lab name and stored as key in Redis with as value the unique username. The email address is often used for unique username, since it even represents unique usernames for cross

domain scenarios. The rest of the authentication process is described in Figure 5. The login page is secure since CoderLabs utilizes the Secure HTTP (HTTPS) module of Node.js. It is relatively easy to change the authentication strategy in Node.js. When a shared remote lab architecture needs to be developed, it is necessary to choose an appropriate authentication strategy. For these purposes, SSO is very often used by means of the Jasig Central Authentication Service (CAS) protocol. Besides central authentication mechanisms like CAS, also decentralized mechanisms exist like OpenID (used in Weblab Deusto) and BrowserID (implemented in Persona). In CoderLabs, the authentication module of Google Coder has been adapted in order to use Passport.js, an authentication middleware for Node.js that supports several authentication strategies. Since each authentication strategy represents a module, it is easy to change strategy and to add a new one. An authentication module for authentication of users against Redis was developed for demonstration purposes. Collaboration with independent labs is provided in CoderLabs with FSSO. Another lab can be addressed by a redirect lab, which contains only the call of a Redirect method. This adds an HTTP redirect header in the response of the client, redirecting the browser of the client to the CoderLab of the other institute.

Authorization for lab designers is important since they can write server side code with Google Coder node.js. Server side code can create large damage to the system if not well designed. A solution is to run the Node.js process under a user with less rights, but still the required rights for starting a socket connection.

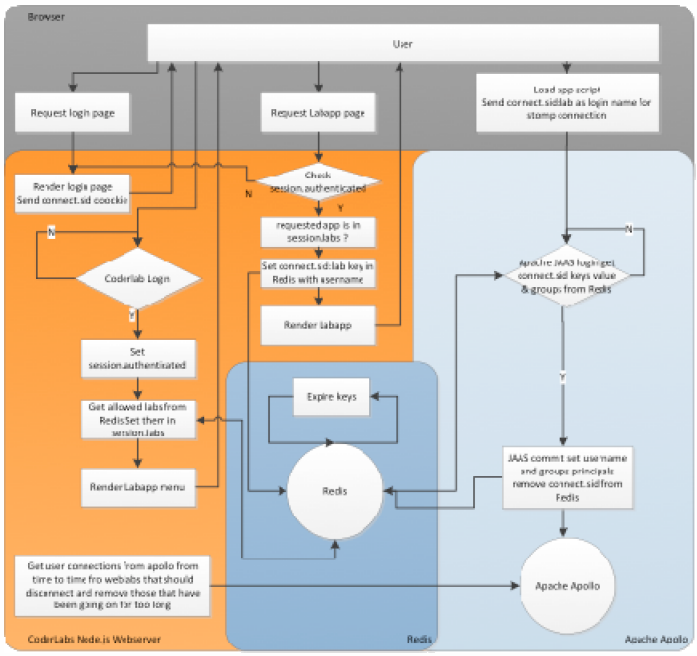

Figure 4. Steps from the authentication procedure

VIII. CLIENT

A webinterface without installation of extra plugins is a crucial requirement for CoderLabs. However, such demand implies many restrictions. Fortunately, HTML5 has nowadays several functionalities that could only be obtained by browser plugins before. There are several possibilities to develop a user interface without plugins. One can use server side scripting to generate pages with dynamic content. With a minimum of client side scripting, it is possible to retrieve the dynamic generated content. This can be made possible by means of a STOMP library on Node.js. However, this way of working would unnecessarily overload the web server. Another way, as applied in CoderLabs, is to render only the basic pages server side and to generate the dynamic content client side. Apollo is used to transport the data of the lab servers to be visualized to the client. Users' actions can then be sent to the lab servers through the client script via Apollo. The only programming language supporting cross browser client side scripting is Javascript, or eventually some related dialects of the ECMA Script standard. The Stomp.js client library is used for the communication of the client with the relay server. An event emitter working on destinations is added to the library in order to register on events.

Creating the user interface that visualizes the data of the logic analyzer is a real challenge. It needs to show digital timing diagrams representing the state of millions of bits. When using the Document Object Model (DOM) of the browser to add thousands of DOM elements, the reaction speed op the browser dramatically decreases. Another option is to work with Scalable Vector Graphics (SVG), a popular format to represent graphical data and interfaces. Javascript can draw vector images to an SVG browser element, which can be added to the DOM. Wavedrom for instance is a web application that uses SVG to represent digital timing diagrams in the browser. However, the application can handle only a limited number of bits. When extending Wavedrom, SVG turned out too slow to show that much data in the browser. To conclude, to represent large number of data, each technique using DOM is discouraged since DOM interaction is inherently slow.

The solution we adopted is the canvas element of HTML5. The canvas element offers a drawing surface, which can be drawn to by the javascript code. The canvas element is added to the DOM of the browser, but no DOM nodes are made from the content. SVG uses a retained mode, while the canvas element an immediate mode. This mode, together with the fact that the browser uses GPU acceleration for the rendering of the canvas element, allows fast drawing of elements to the canvas. The only disadvantages are that the canvas element is only supported by the latest generation of browsers (starting from Chrome 18, Internet Explorer 9, Firefox 4, Opera 11, Safari 5) and that the canvas element only supports a limited number of primitive elements. The canvas element offers a low level drawing API with methods for drawing lines, elementary forms, text and images. However, when updating the canvas, these additional drawn elements are not saved, which is required to develop GUI elements. Scene graphs

offer a solution for this. In javascript, several libraries for scene graphs are available. The most popular being Paper.js, Kinetic.js, Fabric.js and Easel.js. We selected the library Kinetic.js, which is extendable and offers high performance. The widget representing the Atlys FPGA board is developed with this library while for the digital timing diagram widget a pure HTML5 Canvas approach was chosen because of speed considerations. Also other widgets like buttons, knobs, LED displays, meters, and a large range of graphs are developed. Figure 5 shows the user interface of the remote lab for controlling and analyzing an FPGA.

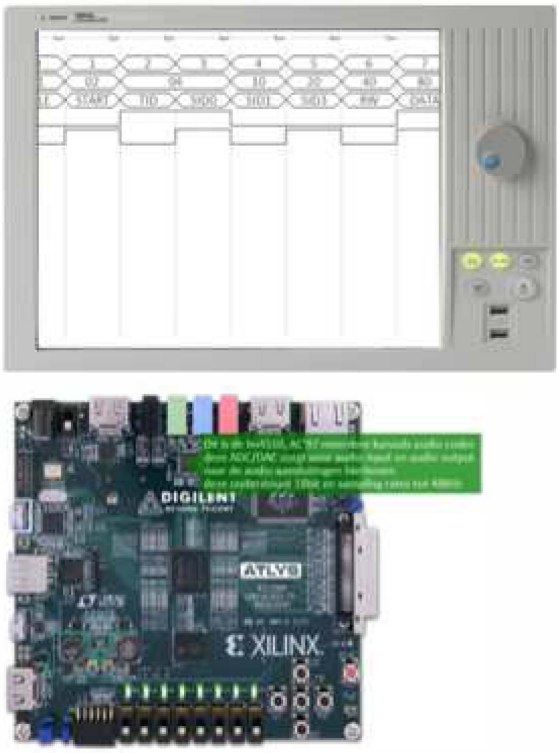

Figure 5: User interface of the remote lab for FPGA study

In many online labs, data needs to be visualized. An interesting javascript library for visualizing data in the browser is D3, data driven documents. D3 can be used for instance to map sensor data with geographic coordinates to an equidistant cylinder projection.

IX. Conclusions and Future Work

An online lab or remote experiment offers the opportunity to obtain at distance lab experience. CoderLabs provides an environment for adding online labs and building user interfaces. CoderLabs mainly focuses on exchanging real data between clients and labs via a relay service, using websockets as transport mechanism. Thanks to the *publish-subscribe* functionality in CoderLabs, users can collaborate in the same online lab. CoderLabs uses technology that is also often used in industry. Consequently, students developing labapps come in contact with message brokers and ESB. Finally, CoderLabs is very user friendly since no browser plugins are required. Also for the developer it is reasonable easy to make server side changes for the lab environment.

A universal environment for remote labs can be created by replacing the message broker with an ESB, by using the message broker as gateway to an ESB or by combining the message broker with an integration platform. The message broker can only communicate via a limited number of wire-level protocols. ESBs and integration platforms on the other hand are made to communicate with a large variety of protocols. Moreover, they enable the transformation of protocols, with some extra configuration and programming. Another possibility is to keep the central message broker infrastructure, but to give the lab servers the responsibility to handle the transformations by using an integration framework on the lab server itself. Apache Camel, for instance, can be coupled with an already existing lab server that uses web services like Labview.

Some extra features as a common file system, streaming video server, and ticketing system should also be added to CoderLabs.

References

[1] R. Krneta, D. Damnjanovic, M. Milosevic, D. Milosevic en M. Topalovic, „Blended Learning of DSP Trough the Integration of On-Site and Remote Experiments," *TEM Journal,* vol. 1, nr. 3, 2012.

[2] M. Tawfik, E. Sancristobal, M. Sergio, R. Gil, G. Diaz, A. Colmenar, K. Nilsson, J. Zackrisson, L. Håkansson en I. Gustafsson, „Virtual Instrument Systems in Reality (VISIR) for Remote Wiring and Measurement of Electronic Circuits on Breadboard," *IEEE Transactions on Learning Technologies,* vol. 6, nr. 1, pp. 60-72, 2013.

[3] T. Richter, Y. Tetour en D. Boehringer, „Library of Labs A European Project on the Dissemination of Remote Experiments and Virtual Laboratories," in *Multimedia (ISM), 2011 IEEE International Symposium on*, Dana Point CA , 2011.

[4] V. J. Harward, J. A. d. Alamo, S. R. Lerman, P. H. Bailey, J. Carpenter, K. DeLong, C. Felknor, J. Hardison, B. Harrison, I. J. P. D. Long, T. Mao, L. Naamani, J. Northridge en D. Mark Schulz, „The iLab Shared Architecture: A Web Services Infrastructure to Build Communities of Internet Accessible Laboratories," *Proceedings of the IEEE,* vol. 06, nr. 96, pp. 931-950, 2008.

[5] P. Orduña, D. López-de-Ipiña, L. Rodriguez-Gil en J. García-Zubia, „Sharing the remote laboratories among different institutions: A practical case," in *Remote Engineering and Virtual Instrumentation (REV), 9th International Conference on*, Bilbao, 2012.

[6] M. Drutarovský, J. Šaliga en I. Hroncová, „Hardware Infrastructure Of Remote Laboratory For Experimental Testing Of FPGA Based Complex Reconfigurable Systems," *Acta Electrotechnica et Informatica,* vol. 9, nr. 1, pp. 44-50, 2009.

[7] P. Orduña, J. García-Zubia, L. Rodriguez-Gil, J. Irurzun

en D. López-de-Ipiña, „Using LabVIEW Remote Panel in Remote Laboratories advantages and disadvantages", *Global Engineering Education Conference (EDUCON), 2012 IEEE*, pp 1-7.

[8] Braeken An, Sterckx Lucas, Verbelen Yannick, Touhafi Abdellah, "E-Learning Platform with SPICE web service", World Academy of Science, Engineering and Technology, Issue: 65, pp: 454 - 458, ISBN-ISSN: 2010-376X,2012

[9] Selmer, A., Kraft, M., Moros, R., and Colton, C. (2007). "Weblabs in chemical engineering education." *Education for Chemical Engineers*, 2(1), 38–45.

[10] Pastor, R., Martín, C., Sánchez, J., and Dormido, S. (2005). "Development of an xmlbased lab for remote control experiments on a servo motor." *International Journal of Electrical Engineering Education*, 42(2), 173–184.

[11] Hasnim, H. and Abdullah, M. Z. (2007). "Remote lab generator (RLGen): A software tool using auto-generating technique to develop a remote lab." *International Journal of Online Engineering*, 3(4), 49–51.

[12] E G Guimaraes, E Cardozo, D H Moraes, P R Coelho, "Design and Implementation Issues for Modern Remote Laboratories", *IEEE Transactions on Learning Technologies*, vol.4, no. 2, pp. 149-161, April-June 2011, doi:10.1109/TLT.2010.22

[13] C. Salzmann, D. Gillet, and P. Mullhaupt, "End-to-End Adaptation Scheme for Ubiquitous Remote Experimentation," *Personal and Ubiquitous Computing,* vol. 13, no. 3, pp. 181-196, Mar. 2009.

[14] C. Buiu and N. Moanta, "Using Web Services for Designing a Remote Laboratory for Motion Control of Mobile Robots," *Proc. AECE World Conf. Educational Multimedia, Hypermedia and Telecomm.,* June 2008.

[15] Verbelen Yannick, Taelman Pieter, Braeken An, Touhafi Abdellah, Reconfigurable and Modular Mobile Robotics Platform for Remote Experiments", International Journal of Online Engineering, Issue: 3, Volume: 9, pp: 19 - 26, ISBN-ISSN: 1861-2121,2013

[16] Pablo Orduna, Javier Garcia-Zubia, Diego Lopez-de-Ipina, Philip H. Bailey, James L. Hardison, Kimberly DeLong, V. Judson Harward, "Sharing Laboratories across Different Remote Laboratory Systems", *ICALT*, 2012, Advanced Learning Technologies, IEEE International Conference on, Advanced Learning Technologies, IEEE International Conference on 2012, pp. 493-494, doi:10.1109/ICALT.2012.137