

INSTRUMENT AUTOMATION AND MEASUREMENT DATA CURATION PLATFORM  
FOR ENHANCING RESEARCH REPRODUCIBILITY AND KNOWLEDGE DISCOVERY

By

Yousef Gtat

A THESIS

Submitted to  
Michigan State University  
in partial fulfillment of the requirements  
for the degree of

Electrical Engineering – Master of Science

2019

## **ABSTRACT**

### **INSTRUMENT AUTOMATION AND MEASUREMENT DATA CURATION PLATFORM FOR ENHANCING RESEARCH REPRODUCIBILITY AND KNOWLEDGE DISCOVERY**

By

Yousef Gtat

Many applications demand the continued development of sensing systems that employ smart sensors, instrumentation circuits, and signal processing techniques to extract relevant information from real-world environments. In the engineering efforts to develop new sensors, tasks such as instrument automation, measurement process curation, real-time data acquisition, data analysis, and long-term tracking of inter-related datasets generate a significant volume and variety of information that is challenging to organize, record, and analyze. Sensor development and characterization experiments can be laborious, prone to human error, difficult to repeat precisely, and can produce data that are challenging to interpret. Such issues highlight a need for a structured, automated approach to curate measurement processes and data acquisition. This thesis presents the first software platform for i) digitally designing measurement recipes, ii) remotely scheduling and monitoring experiment execution, iii) automatic data acquisition, iv) analyzing and storing results datasets, and v) linking the datasets with their prospective meta-datasets for deeper analysis and inspection. The proposed platform is flexible and capable of managing a large set of diverse instruments, measurement recipes and sensor datasets. By employing several design abstractions, it allows users to remotely design, schedule, monitor and execute measurement-based experiments while archiving results along with their information-rich metadata therefore preserving the provenance of the datasets. The platform enable precise timing control of instruments and stimulus signals along with long-term tracking of datasets eliminating manual errors and human omissions thus enhancing research reproducibility and promoting knowledge discovery methodologies.

## ACKNOWLEDGMENTS

I am most grateful to my parents and siblings for their tremendous support throughout my life; I owe it all to them! I also extend my gratitude to my relatives, the Huwio family, for lending me their support throughout college and graduate school.

My special thanks to all of the developer who made the presented platform proposed in this thesis possible. Specifically, I am thankful to the software developers: Ian Bacus, Luke Wiseman, Ben Buscarino, and Trevor Sabo for lending me their expertise and helping with the code base.

Finally, there are no proper words to convey my deep gratitude and respect for my thesis and research advisor, Professor Andrew J. Mason. My sincere thanks must also be extended to my colleagues at the AMSaC group and the members of my thesis committee: Prof. Zhang, Prof. Biswas, and Prof. Tan.

This work is supported by the National Institutes of Health (NIH) grant R01ES022302 and grant R01AI113257.

## TABLE OF CONTENTS

<b>LIST OF TABLES .....</b>	<b>vi</b>
<b>LIST OF FIGURES .....</b>	<b>vii</b>
<b>CHAPTER 1 .....</b>	<b>1</b>
<b>MOTIVATION .....</b>	<b>1</b>
<b>1.1 Significance .....</b>	<b>1</b>
1.1.1 Research Reproducibility .....	1
1.1.2 Knowledge Discovery .....	2
1.1.3 Institutional Memory Loss .....	4
<b>1.2 Requirements and Challenges .....</b>	<b>5</b>
<b>1.4 Approach and Goals .....</b>	<b>7</b>
<b>1.5 Summary .....</b>	<b>8</b>
<b>CHAPTER 2 .....</b>	<b>9</b>
<b>BACKGROUND .....</b>	<b>9</b>
<b>2.1 Experimental Research and Scientific Method .....</b>	<b>9</b>
<b>2.2 Research Objects (ROs) .....</b>	<b>11</b>
<b>2.3 Cloud Computing and Services .....</b>	<b>12</b>
<b>2.4 Backend Architectures: Monolithic and Microservices .....</b>	<b>14</b>
<b>2.5 Databases: Relational and Non-relational .....</b>	<b>16</b>
<b>2.6 Review of Existing Solutions .....</b>	<b>19</b>
2.6.1. LabVIEW .....	20
2.6.2. Scientific Workflow Tools .....	22
2.6.3. Electronic Lab Notebooks (ELN) .....	26
2.6.4. Previous Work .....	27
<b>2.7 Summary .....</b>	<b>28</b>
<b>CHAPTER 3 .....</b>	<b>29</b>
<b>DESIGN AND IMPLEMENTATION .....</b>	<b>29</b>
<b>3.1 Requirements and Terminology .....</b>	<b>29</b>
<b>3.2 Instrument Control and Device Drivers .....</b>	<b>30</b>
<b>3.3 Web-based Graphical User Interface .....</b>	<b>36</b>
<b>3.4 Data Management, Models, and Provenance .....</b>	<b>39</b>
<b>3.5 Microservices and Messaging Mechanisms .....</b>	<b>41</b>
<b>3.6 Collaboration, Security, and Networking .....</b>	<b>43</b>
<b>3.7 Platform Versatility and Adaptability to Existing Tools .....</b>	<b>44</b>
<b>3.8 Platform Dataflow .....</b>	<b>45</b>
<b>3.9 Software Development Environment .....</b>	<b>46</b>
<b>3.10 Summary .....</b>	<b>47</b>
<b>CHAPTER 4 .....</b>	<b>48</b>
<b>RESULTS and DISCUSSION .....</b>	<b>48</b>
<b>4.1 User Experience Design .....</b>	<b>48</b>

<b>4.2</b>	<b>Instrument Automation and Measurement Data Curation .....</b>	<b>50</b>
4.2.1	Experiment Setup.....	50
4.2.2	Automated Measurement Design using the Designer Tool .....	51
4.2.3	Measurement Execution and Monitoring using the Executer Tool .....	59
4.2.4	Data Visualization and Analysis using the Analyzer Tool .....	65
<b>4.3</b>	<b>Platform Deployment in the Cloud.....</b>	<b>69</b>
<b>4.4</b>	<b>Table of the Platform Features vs Existing Tools .....</b>	<b>71</b>
<b>4.5</b>	<b>Summary .....</b>	<b>73</b>
<b>CHAPTER 5 .....</b>	<b>75</b>	
<b>CONCLUSION .....</b>	<b>75</b>	
<b>5.1</b>	<b>Summary .....</b>	<b>75</b>
<b>5.2</b>	<b>Contributions.....</b>	<b>76</b>
<b>5.3</b>	<b>Current Status and Future Work.....</b>	<b>77</b>
<b>BIBLIOGRAPHY .....</b>	<b>79</b>	

## LIST OF TABLES

Table 1 Proposed Platform Features vs. Existing Software Tools.....	72
---	----

## LIST OF FIGURES

Fig. 2. 1 Scientific method flow diagram. ....	10
Fig. 2. 2 Cloud computing services architecture showing the hierarchy of these services starting with lower layer IaaS, PaaS, and then SaaS.....	12
Fig. 2. 3 Shows the difference between two common backend architecture, microservices vs. traditional monolithic.....	16
Fig. 2. 4 Document based non-relational database model (left) vs. table based relational database model (right) .....	17
Fig. 2. 5 Relational database example consists of three tables linked together using foreign keys. ....	18
Fig. 2. 6 A simple LabVIEW diagram creating a Virtual Instrument (left) and the user interface provided for the functional diagram (right) .....	21
Fig. 2. 7 An example of LabVIEW user interface tools available for data visualization. ....	22
Fig. 2. 8 An example workflow designed using Apache Taverna desktop software.....	24
Fig. 2. 9 An example of workflow designed using Kepler's graphical interface and components. ....	25
Fig. 2. 10 An example of Jupyter Notebook showing live code containing the algorithm, libraries used, and results plots. ....	27
Fig. 2. 11 Shows the state of the previously eGor software tool communicating with a device and collecting data in real time. ....	28
Fig. 3. 1 A diagram of the platform consisting of three bundles of tools: browser interfaces, instrument manager (IM), and database.....	30
Fig. 3. 2 The Instrument Manager has an aggregation of device drivers, customized by user-plugins for communicating with a wide variety of commercial and custom lab instruments. ....	31
Fig. 3. 3 A physical experiment workbench composed of a local PC connected to multiple instruments to examine a device under test (DUT). The Instrument Manager is an aggregation of device drivers, customized by user-plugins for communicating with wide variety of lab instruments. ....	33
Fig. 3. 4 shows an example of how pySerial library can be used to communicate with a device connected to a USB port. ....	34
Fig. 3. 5 shows an example code of a device driver of a commercial sensor. ....	35

Fig. 3. 6 (left) Model of measurement space as instrument setup and procedures over time. (right) Simplified description of the information rich Research Object containing recipe, measurement data and more. ....	36
Fig. 3. 7 Database architecture depicting digital curation between user account and research objects that integrate data, metadata, and processed data in one digital sharable identity. ....	40
Fig. 3. 8 An example schema and instance a blog post using Mongoose Library.....	41
Fig. 3. 9 Publish-Subscribe (Pub-Sub) messaging, used for services to interact but function independently from each other's.....	43
Fig. 3. 10 Networking model of the deployed platform and its inter-communication. ....	44
Fig. 3. 11 Functional schematic of the proposed platform for managing user experience from the experiment definition and hardware execution stage to digital curation throughout the life cycle of experiment data.....	46
Fig. 3. 12 Agile software development environment consisting of seven steps. ....	47
Fig. 4. 1 shows the welcome screen of the designed user interface.....	48
Fig. 4. 2 shows the login and signup screen where the user could input their credential to login or register to the platform to start using it. ....	49
Fig. 4. 3 hardware and software to configure a test environment and then use eGor:AutoX to digitally define their test configuration and execute the tests.....	50
Fig. 4. 4 Snapshot of the Designer browser tool with a sample setup with two instruments to be used during execution of the designed recipe. ....	51
Fig. 4. 5 Snapshot of the setup shelves used to organize different types of instruments based on user preference. ....	53
Fig. 4. 6 Snapshot of the setup indicating which instruments are used and how many of them are needed to conduct the designed experiment. ....	54
Fig. 4. 7 Snapshot of the tabular bar showing the recipe name and all three tabs available in the Designer tool. ....	55
Fig. 4. 8 Snapshot of the Designer browser tool with a sample recipe with two instruments and multiple methods (i.e. CV, CA, etc.) to be executed over a pre-defined time period.....	55
Fig. 4. 9 Snapshot of the <i>Designer</i> browser tool with a short sample recipe with two commercial instruments.....	56
Fig. 4. 10 Snapshot of the method popup dialog box, all methods has global parameters such as start time, end time, etc. However, each method has it is own unique parameters/inputs to define the method such as sampling rate, initial violate, etc. ....	57



Fig. 4. 11 Snapshot of the Meta tab showing user tags, keywords, and notes.....	58
Fig. 4. 12 Snapshot of the Designer tool indicating that the submitted experiment was compiled successfully.....	58
Fig. 4. 13 Snapshot of the Executer browser tool indicating the running, ready, and completed status of example measurement recip. ....	59
Fig. 4. 14 Snapshot of the Executer browser tool indicating the running, ready, and completed status of example measurement recip. ....	60
Fig. 4. 15 Snapshot of the pop-up screen used in the Scheduler tab to run an experiment immediately or schedule it for later execution time.....	62
Fig. 4. 16 Snapshot of the Executer browser tool indicating a running experiment scheduled in the scheduler tab. ....	63
Fig. 4. 17 Snapshot of the Executer browser tool indicating two running experiments simultaneously. ....	64
Fig. 4. 18 Snapshot of the Executer browser tool indicating the completed experiments and run-time errors. ....	65
Fig. 4. 19 Snapshot of the Analyzer browser tool displaying run time error and debug logs, timestamped raw data logs, and plots of cyclic voltammetry measurement data along with an input stimulus signal. ....	66
Fig. 4. 20 Cyclic voltammetry measurements, data outputted from the Analyzer tool as a simple csv file (left) and plotted using a MATLAB script (right).....	67
Fig. 4. 21 Snapshot of the Analyzer browser tool displaying run time error and debug logs, timestamped raw data logs, and plots of temperature, humidity, and digital multimeter measurement data.....	68
Fig. 4. 22 Snapshot of the Analyzer browser tool displaying ROs residing in the database with filtering options. ....	69
Fig. 4. 23 Local deployment of the platform for debugging and privacy. ....	70
Fig. 4. 24 Snapshot of the mLab cloud platform deploying the proposed eGor:AutoX platform database.....	71

# CHAPTER 1

## MOTIVATION

### 1.1 Significance

A growing community of researchers have expressed concerns regarding three main elements in the field of sensor development and characterization. Researchers have often faced the issue of non-reproducible research due to the lack of documentation and human omission [1]. Furthermore, the need to implement knowledge discovery methodologies on a complete information-rich database has come to a halt due to the unavailability of such a resource [2]. Finally, the transitioned of such knowledge as turnover occurs between group members is often handled poorly which leads to institution memory loss [3].

#### 1.1.1 Research Reproducibility

Many applications require precise documentation of the complex scientific process from experiment design and execution through data recording and analysis, where such documentation is a key factor in research reproducibility. The inability to replicate the studies of others could potentially result in consequences such as significant theories being grounded by experimental work that are questionable due to inability to repeat the studies. In fact, upon subsequent investigation, reproducibility has scored as low as 10% in a span of scientific fields [4] such as medicine and social psychology.

Reproducible research is defined as a published study with replicable findings and results based on the reported scientific methods [5]. For instance, an experimentally obtained value is said to be reproducible if there is a high degree of agreement between measurements or observations conducted on replicated devices in different locations by different people. Researchers rely on the

documentations published by other researchers in order to replicate the findings and results. However, the methods recorded are often missing important information resulting in non-reproducible data. For instance, researchers often do not record the experiment setup and all of the instruments used and how they are connected. Moreover, publications tend to neglect documentation of the stimuli used, instead focusing on presenting the test measurements and results. Furthermore, the experiment flow and the precise timeline of the stimulus applied with respect to the recorded measurement timestamp is often not tracked due to manual experimental execution.

Unfortunately, poor reproducibility of scientific publications has grown in many research areas, and the need for reproducibility is increasing dramatically as scientific methods and data analyses have become more complex. Many research communities are calling for improving research methods, standardizing documentation processes, data organization, and meta-data recording, thus highlighting the need for a structured and automated approach to i) design and conduct experiments, ii) execute and monitor experimental trials, and iii) analyze and store result datasets. In other words, the need for a standardized platform with history tracking capabilities to curate experiments from the design stage to execution and analysis stages is highly desirable in pursuit of achieving reproducible research. Ideally, researchers should be able to use one platform to i) describe the experimental setup, ii) define the experiment methods over time, iii) automate the experimental execution, iii) automatically collect and store test measurements, and iv) link the datasets with their prospective meta-datasets for deeper analysis and inspection.

### 1.1.2 Knowledge Discovery

In addition to reproducible research, the ongoing rapid growth of research data due to the plethora of devices connected to the internet [6] and the widespread use of scientific databases [7]

have created an immense need for knowledge discovery methodologies [8]. Knowledge discovery is an interdisciplinary area focusing upon methodologies for extracting useful knowledge from inter-related datasets; a knowledge that may never be observed or learned by researchers due to the large volume of information [9]. The resulting knowledge needs to be in a machine-readable and machine-interpretable format. In general, knowledge discovery methodologies consist of the following stages [10]: i) selection stage: selecting the target data set upon which discovery is to be performed, ii) preprocessing stage: data cleansing and preprocessing for removing noise and outliers, iii) transformation stage: data reduction and projection for finding useful features representing the datasets, iv) data mining stage: searching for patterns and building classification, regression, clustering, etc. models, and v) evaluation stage: interpreting the mined patterns and consolidating discovered knowledge

The process of discovering knowledge in databases is highly dependent on the datasets and the targeted application. The above stages assume that the datasets are well constructed for the targeted application and available for applying new data mining techniques. The missing, most critical stage is the creation of relevant information from structured and well-defined datasets and unstructured meta-data sources such as text files, documents, images, timelines, etc. In the realm of developing new sensors and recording test measurements, the challenge of extracting knowledge from research data lies in the creation of an information-rich database that not only contains result datasets, but also all associated metadata and ontologies such as the experimental procedures, input stimuli, devices and instruments, post processing algorithm, keywords, researcher's notes and annotations, etc. This would only be possible through using one general-purpose platform that can design, execute, and analyze experiments all under one hat. Such a platform will generate the desired research database that could pioneer new statistical models for

significant knowledge discovery impact on many scientific disciplines. Thus, the main criteria for knowledge discovery methodologies should go beyond the reuse of existing formal datasets and employing new classification models, extending to focus on the creation of information-rich datasets with meta-data integration on which knowledge discovery methods are to be performed.

### 1.1.3 Institutional Memory Loss

Lastly, institutional memory loss is another growing concern in the industry and research community. Institutional memory is the accumulated body of data, information, and knowledge created over the course of an individual organization's existence [11]. The collective knowledge and learned experiences of a group must be transitioned as turnover occurs between group members. Today's employers face higher turnover rates than at any other time in history [12]. Employers lose the institutional knowledge or history which the leaving group members take with them, and many organizations lack sufficient transfer programs to stem the loss. Without a plan or program to transfer business processes, institutional policies and practices, and historical knowledge to the new group members, organizations may be faced with severe business continuity and knowledge issues as older employees leave them. This issue places a burden on organizations and research groups which makes it difficult to build on prior work and advancing knowledge at a faster pace, thus leading to an institutional memory loss.

Researchers have called for knowledge management tools that aim to capture and preserve these memories without having too much overhead. Knowledge management is the process of creating, sharing, using, and managing the knowledge and information of an organization [13]. It is a multidisciplinary approach to achieving organizational objectives by making the best use of knowledge. In the field of sensor instrumentation and test measurement automation, institutional memory loss could be mitigated through automation platforms that automatically produce

executable processes and store all associated meta-data with results and research findings. This would allow new group members to search through executable experimental processes and study everything related to the experiment without the need for mentorship from the previous group. Moreover, if experimental processes are reproducible, the new group members would be able to repeat the experiment and learn from the replicated results with addition to the previous group's notes and documentation. This can be achieved by having one universal platform for designing, executing and analyzing research experiments all in one hardware-software flexible environment. The history tracking of experimental methods, instrument inventory, and results data could be used as a knowledge management tool to facilitate new groups to quickly adapt to the institutional knowledge, build on prior work, and ultimately mitigate institutional memory loss

## **1.2 Requirements and Challenges**

Creating a platform for enhancing reproducible research, facilitating knowledge discovery, and mitigating institutional memory loss requires many elements to be crafted in parallel. In the field of sensor development and characterization, such a platform must implement novel ways for conducting instrumentation and test measurement automation. Reproducible sensor data and information-rich databases are the main two missing elements needed to facilitate ways to create novel sensor structures and expedite sensor characterization. Developing new sensors demands tasks such as i) designing multiple sensor characterization experiments, ii) executing multiple experimental trials and collecting result datasets in real time, iii) analyzing result datasets and applying signal processing models, and iv) storing and long-term tracking of result datasets and their prospective meta-data.

Such tasks are often performed by ad-hoc means along with manual approaches which lead to human errors and omissions. For instance, researchers often design and draw the experiments

on personal notebooks and lab operators manually adjust instruments (e.g. turn a knob) during an ongoing experiment. Moreover, the input stimuli and analysis algorithms are often not recorded along with the result datasets, especially in collaborative multi-location research environments. Furthermore, characterization data is often collected and stored manually on a personal computer which could lead to institutional memory loss if not backed up and annotated properly. Such tasks can make it challenging for researchers to i) run long experiments for days or even months, ii) track connections between sensors under test and their result datasets, iii) draw honest conclusions between experimental and analytical errors, iv) record and organize significant volumes and variety of information.

These challenges above can be articulated in three keywords: repeatability, integrity, and productivity. Although these terms are rather generic, in scientific research, repeatability is the ability for someone else to repeat the exact experimental process based on the documented scientific research methods. In other words, it is the ability to repeat the experiment precisely as before and the ability to cross check datasets from multiple trials for reproducible results. Repeatability requires detailed precision for an ongoing experiment to have high temporal resolution, meaning a precise execution of experimental process, precise timing control, and input-output signal synchronization. The second keyword is data integrity, which mainly concerns the correctness and truthfulness of drawing conclusions by validating event driven results and eliminating human errors and omission such as forgetting to perform an experimental step or wrongfully recording events as much as possible. This is hard to assess based on the research findings, and that is why automation can play a big role in enhancing data truthfulness and integrity. Finally, research productivity could slow down findings and hinder an ongoing experiment. Many researchers spend consecutive days conducting experiments leading to fatigue

and extreme tiredness. This often causes human errors in the measurement processes and human omission of documentation. These issues hinder the ability to generate information-rich databases that could subsequently be used in developing knowledge discovery models and techniques.

## **1.4 Approach and Goals**

In the era of Internet of Things (IoT) [14], the respectable state of the hardware permits cloud computing [15] and Software as a Service (SaaS) [16] to address and solve the challenges facing research reproducibility and knowledge discovery. Cloud computing refers to running workloads remotely over the internet in a commercial provider's data center such as Amazon Web Services (AWS) [17] and Microsoft Azure [18]. SaaS is a software distribution model in which a third-party provider hosts application on the cloud and makes them available to customers over the Internet, wherein users can access using a thin client such as a web browser. Google's G Suite [19] and Salesforce [20] are great examples of SaaS. This thesis takes the approach of combining all three technologies: IoT, cloud computing, and SaaS to resolve the issues obstructing instrumentation and test measurement automation. For example, most modern instruments and lab equipment have digital interfaces to receive computer permitting real-time automation and report back the resulting measurements. The approach is to connect these devices to internet and implement a SaaS to enable users to remotely control instruments and log their measurement data in realtime while using cloud computing resources to process and store datasets for long-term tracking.

This thesis explores the potential of software solutions to address the challenges outlined above through advanced automation of instruments and engineered curation of the measurement process. To this end, the goals of this thesis are:



- 1- Develop a software platform for instrumentation and measurement that will enable experimental research reproducibility by i) enhancing measurement process precision and data integrity, ii) constructing a digital record that permits automated replication of measurement process, and iii) achieving accessibility to broad range of disciplines through intuitive user interfaces that do not require programming skills.
- 2- Design the measurement platform to promote knowledge discovery by i) facilitating collaboration across scholarly disciplines and over multiple locations, and ii) providing curated results to overcome challenges of institutional memory and permit deep analysis.

## **1.5 Summary**

The presented approach of tackling the issues facing experimental research reproducibility and knowledge discovery provides high-level capabilities for remotely controlling lab equipment and routing captured sensor data through a vision of connecting research labs to enable IoT applications. Chapter 2 will cover the literature review of relevant theories as well as existing technologies and software solutions. In Chapter 3, the proposed platform architecture design is described in detail along with an overview of the developers' implementation choices. Chapter 4 presents experimental result datasets and discuss the proposed platform and its capabilities. Finally, Chapter 5 concludes with the summary of the proposed work, author's contributions, and the platform status and future work.

## **CHAPTER 2**

### **BACKGROUND**

#### **2.1 Experimental Research and Scientific Method**

Research comprises of “creative and systematic work undertaken to increase the stock of knowledge, including knowledge of humans, culture and society, and the use of this stock of knowledge to devise new applications” [21]. The process of research is the "steps used to collect and analyze information to increase our understanding of a topic or issue" [22], which consists of three research stages: pose a research question, collect data to answer the research question, and present and publish an answer to the research question.

In the field of experimental research, scientists use the scientific method approach to conduct experimental research. The scientific method is the process for experimentation that is used to explore observations and answer questions. The fundamental goal of this process is: to discover cause and effect relationships by asking questions, carefully gathering and examining the evidence, and seeing if all the available information can be combined in to a logical answer. Specifically, the scientific method consist of six main phases: ask a question, do background research, construct a hypothesis, conduct experiments, analyze data and draw conclusions, and report results [23]. However, the scientific method is not merely a series of sequential steps, some of these steps generates new information that might causes a scientist to back up and repeat the steps at any point during the process. A process like the scientific method that involves such backing up and repeating is called an iterative process [24] as shown in the Fig. 2.1.

The scientific method shown starts when a research question is asked about something that researchers have observed. The next step is to do a background research, rather than starting from scratch in putting together a plan for answering the research question to ensure that mistakes from the past are not repeated. Subsequently, the researcher constructs a hypothesis, which is an attempt to answer the research question with an explanation

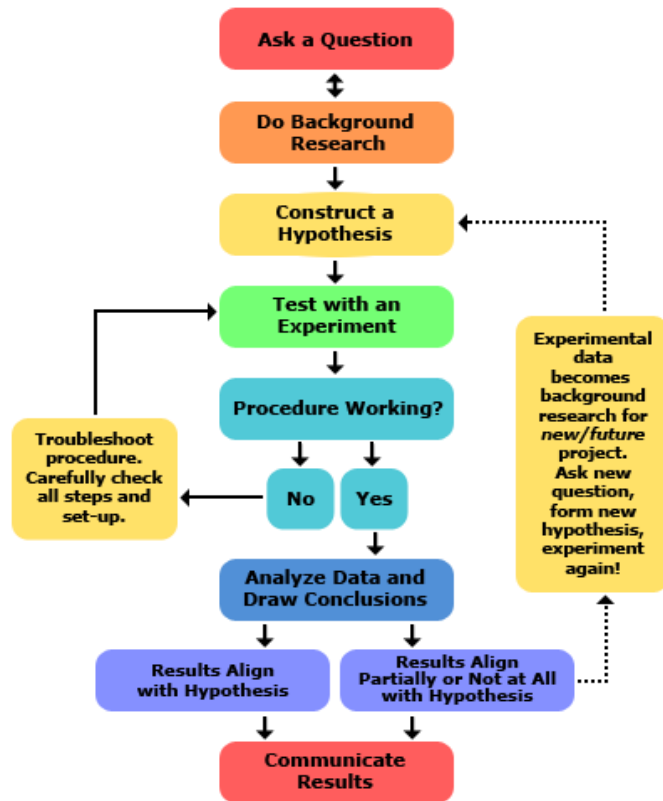


Fig. 2. 1 Scientific method flow diagram.

that can be tested. Researchers often express both the hypothesis and the resulting prediction that will be tested, which then they start conducting the necessary experiments, which test whether the prediction is accurate and thus the hypothesis is supported or not. Each experiment should conduct a fair test by making sure that only one factor at a time is changing while keeping all other conditions the same. Each experiment should also be repeated several times to make sure that the first results were not just an accident. Scientists often find that their predictions were not accurate, and their hypothesis was not supported, and in such case, they will communicate the results of their experiment and then go back and construct a new hypothesis and prediction based on the information they learned during their initial experiments. Lastly, researchers analyze their data and

draw a conclusion to see if they support the hypothesis or not. The research finding is often published in a journal to share the results with the scientific community.

## **2.2 Research Objects (ROs)**

Reproducibility and replicability together are among the main beliefs of the scientific methods, with the concrete expressions of the ideal of such a method varying considerably across research disciplines and fields of study [25]. The reproduced measurement may be based on the raw data and computer programs provided by researchers. The values obtained from distinct experimental trials are said to be commensurate if they are obtained according to the same reproducible experimental description and procedure. An experimentally obtained value is said to be reproducible if there is a high degree of agreement between measurements or observations conducted on replicate specimens in different locations by different people, that is, if the experimental value is found to have a high precision.

A Research Object (RO) is an advanced form of enhanced publication and a method for the identification, aggregation and exchange of scholarly information [26]. The primary goal of the research object approach is to provide a mechanism to associate together related resources about a scientific investigation so that they can be shared together using a single identifier, including supporting data, software executables, source code, presentation slides, presentation videos, etc. [27]. This approach is primarily motivated by a desire to improve reproducibility of scientific investigations. ROs are not one specific technology but are instead guided by a set of principles. Specifically ROs are guided by three principles: I) digital identity: use unique identifiers such as DOIs for publications or data [28] and ORCID for researchers [29], II) data aggregation: use some form of aggregation to associated related things together that are part of the broader study so that others may more readily discover those related resources, and III) annotation:

provide additional metadata about those things, how they relate to each other, their provenance, and how they were produced.

## 2.3 Cloud Computing and Services

Cloud computing is a general term for anything that involves delivering hosted services over the Internet. These services are broadly divided into three categories: Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS) and Software-as-a-Service (SaaS) [30]. Fig. 2.2 shows the architecture of cloud computing services and the interactions between the clients and the service.

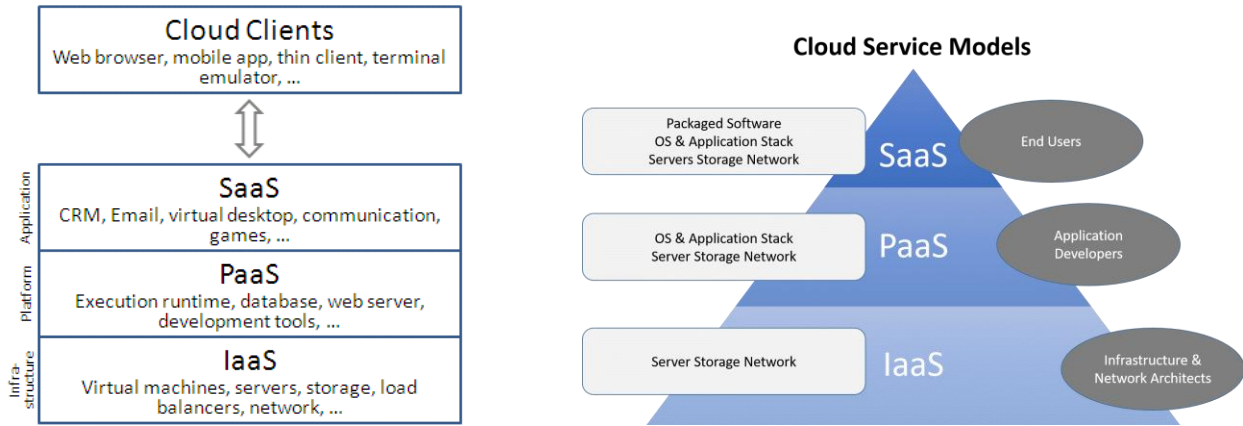


Fig. 2. 2 Cloud computing services architecture showing the hierarchy of these services starting with lower layer IaaS, PaaS, and then SaaS.

IaaS refers to online services that provide high-level APIs used to dereference various low-level details of underlying network infrastructure like physical computing resources, location, data partitioning, scaling, security, backup etc. [31]. IaaS providers, such as AWS, supply a virtual server instance and storage, as well as APIs that enable users to migrate workloads to a VM. Users have an allocated storage capacity and can start, stop, access and configure the VM and storage as

desired. IaaS providers offer small, medium, large, extra-large and memory- or compute-optimized instances, in addition to customized instances, for various workload needs.

New emerging cloud technologies and services has led public IaaS providers to offer far more than common compute and storage instances. For example, serverless, or event-driven computing is a cloud service that executes specific functions, such as image processing and database updates [32]. Traditional cloud deployments require users to establish a compute instance and load code into that instance. Then, the user decides how long to run -- and pay for -- that instance. With serverless computing, developers simply create code, and the cloud provider loads and executes that code in response to real-world events, so users don't have to worry about the server or instance aspect of the cloud deployment. Users only pay for the number of transactions that the function executes. AWS Lambda, Google Cloud Functions and Azure Functions are examples of serverless computing services.

PaaS is the capability provided to the consumer is to deploy onto the cloud infrastructure consumer-created or acquired applications created using programming languages, libraries, services, and tools supported by the provider [33]. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, or storage, but has control over the deployed applications and possibly configuration settings for the application-hosting environment. In the PaaS model, cloud providers host development tools on their infrastructures. Users access these tools over the internet using APIs, web portals or gateway software. PaaS is used for general software development, and many PaaS providers host the software after it's developed. Common PaaS providers include Salesforce's Force.com, AWS Elastic Beanstalk and Google App Engine.

Finally, SaaS is the capability provided to the consumer is to use the provider's applications running on a cloud infrastructure [34]. The applications are accessible from various client devices through either a thin client interface, such as a web browser (e.g., web-based email), or a program interface. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, storage, or even individual application capabilities, except for limited user-specific application configuration settings. SaaS is a distribution model that delivers software applications over the internet; these applications are often called web services. Users can access SaaS applications and services from any location using a computer or mobile device that has internet access. One common example of a SaaS application is Microsoft Office 365 for productivity and email services.

## **2.4 Backend Architectures: Monolithic and Microservices**

In software engineering, the terms frontend and backend refer to the separation of concerns between the presentation layer (frontend), and the data access layer (backend) of a piece of software. In the client–server model [35], the client is usually considered the frontend and the server is usually considered the backend, even when some presentation work is done on the server itself. While frontend have their own styles and architecture, this section focuses on the backend architectures because it has higher impact on the software application services in terms of scalability, availability and speed.

One type of backend architecture is a monolithic application, which is built as a single unit. Enterprise applications are often built in three main parts: a client-side user interface (consisting of HTML pages and JavaScript running in a browser on the user's machine) a database (consisting of many tables inserted into a common, and usually relational, database management system), and a server-side application. The server-side application handles HTTP requests, execute domain

logic, retrieve and update data from the database, and select and populate HTML views to be sent to the browser. This server-side application is a monolith, a single logical executable. Any changes to the system involve building and deploying a new version of the server-side application. Such a monolithic server is a natural way to approach building such a system. Monolithic applications can be successful, but increasingly people are feeling frustrations with them, especially as more applications are being deployed to the cloud. Change cycles are tied together, a change made to a small part of the application, requires the entire monolith to be rebuilt and deployed. Over time it is often hard to keep a good modular structure, making it harder to keep changes that ought to only affect one module within that module. Scaling requires scaling of the entire application rather than parts of it that require greater resource.

Microservices are a software development technique that structures an application as a collection of loosely coupled services [36]. In a microservices architecture, services are fine-grained, and the protocols are lightweight. The benefit of decomposing an application into different smaller services is that it improves modularity. This makes the application easier to understand, develop, test, and become more resilient to architecture erosion. It also parallelizes development by enabling small autonomous teams to develop, deploy and scale their respective services independently. Microservices architecture is highly maintainable and testable, loosely coupled independently deployable, and organized around business capabilities. Fig. 2.3 below shows the difference between the traditional monolithic architecture as opposed to microservices architecture.



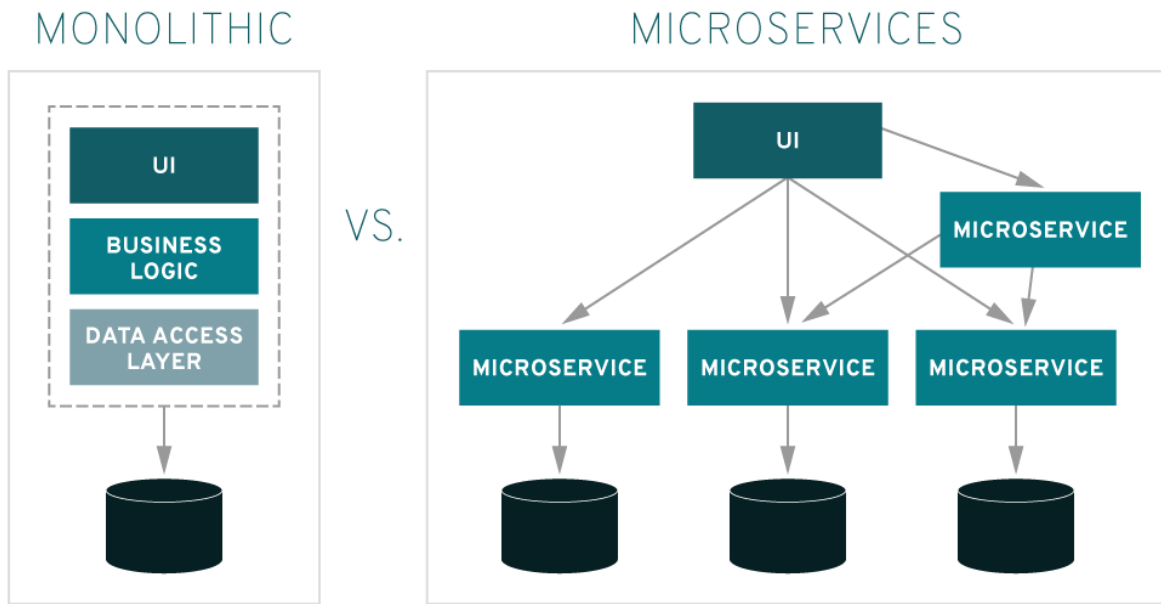


Fig. 2. 3 Shows the difference between two common backend architecture, microservices vs. traditional monolithic.

## 2.5 Databases: Relational and Non-relational

Relational databases like MySQL, PostgreSQL and SQLite3 represent and store data in tables and rows [37], meanwhile, non-relational databases like MongoDB represent data commonly in collections of JavaScript Object Notation (JSON) documents [38]. Fig. 2.4 below shows an example of both database models containing typical user information.



Fig. 2. 4 Document based non-relational database model (left) vs. table based relational database model (right)

MongoDB query data are technically represented as binary JASON (BSON), whereas relational databases use Structured Querying Language (SQL). SQL is often a good choice for applications that involve the management of several transactions due to its inherent strict architecture. The structure of a relational database allows you to link information from different tables using foreign keys (or indexes), which are used to uniquely identify any atomic piece of data within that table as shown in Fig. 2.5. Other tables may refer to that foreign key, to create a link between their data pieces and the piece pointed to by the foreign key. This comes in handy for applications that are heavy into data analysis. If the application need to handle a lot of complicated querying, database transactions and routine analysis of data, then relational database is a suitable option.

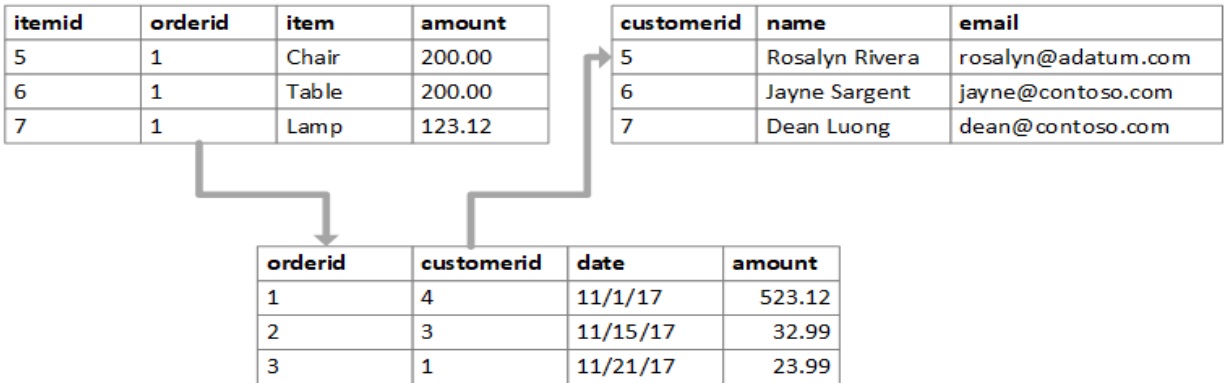


Fig. 2. 5 Relational database example consists of three tables linked together using foreign keys.

While relational databases are great option for well-defined data structures, they do come with trade-offs. One of those is ORM Impedance Mismatching [39], because relational databases were not initially created with the OOP languages in mind. The best way to avoid this issue is to create your database schema with referential integrity at its core. Thus, when using a relational database with an OOP language (like Ruby), it is crucial to think about how to set up your primary and foreign keys and the use of constraints (including the cascade delete and update). However, if the application is dealing with a phenomenally huge amount of data, this process can be tedious and the probability of error increases. In that situation, non-relational database. A non-relational database are usually a better option for just storing the data without explicit and structured mechanisms to link data from different tables (or buckets) to one another. MongoDB is the most popular non-relational database for MEAN stack developers [40] because it is basically written in JavaScript and JSON, which is a lightweight data interchange format. If the applications data model turns out to be very complex, or if de-normalize process is necessary to the application database schema, then non-relational databases is much suitable option. Other reasons for choosing a non-relational database include:

- The need to store serialized arrays in JSON objects

- The nature of the application data are arbitrary and may not be easy to organize
- Storing records in the same collection that have different fields or attributes
- Finding de-normalizing of database schema or coding around performance and horizontal scalability issues

In non-relational databases like MongoDB, there are no joins like there would be in relational databases. This means developers need to perform multiple queries and join the data manually within the application code, which can get very messy [41]. Since MongoDB does not automatically treat operations as transactions the way a relational database does, the application API must manually choose to create a transaction and then manually verify it. Some operations will succeed while others fail, which emphasize back on knowing how to effectively whiteboard the application data model. It is this key step that will allow any software application to determine the best route for choosing the use of one database over another.

## **2.6 Review of Existing Solutions**

Among existing tools that can assist in experiment automation is LabVIEW made by National Instruments, which permits automatically performing and monitoring panels of instrument while choreographing measurement and actuation processes executed by computer-controlled devices [42]. Other existing tools, such as Apache Taverna, and Kepler, are a specialized form of software named scientific workflow systems. These software designed specifically to compose and execute a series of computational or data manipulation steps, or workflows, in a scientific application [43]. Finally, the last existing software tool worth mentioning is the Electronic Lab Notebooks (ELN) which can organize and archive all laboratory data safely, manage a research lab, and review students' lab work [44]. It is important to note that most of these

software solutions are outdated or not maintained as well as they should be, therefore, many researchers do not know of these software tools and use them regularly.

### 2.6.1. LabVIEW

LabVIEW is a graphical programming language developed by National Instruments in the mid to late 80's. Creating a program in LabVIEW is called a VI, which stands for Virtual Instrument. To create a VI, the programmer uses the LabVIEW programming environment to make the user interface by dragging and dropping objects and arranging them as desired. To add functionality to the interface, the diagram, which resembles a flow chart is wired with the various structures and functions. In most LabVIEW programs, no lines of code are actually written, the functionality of the program is provided by the diagram. For this reason, LabVIEW is called a graphical programming language as shown in Fig. 2.6 below.

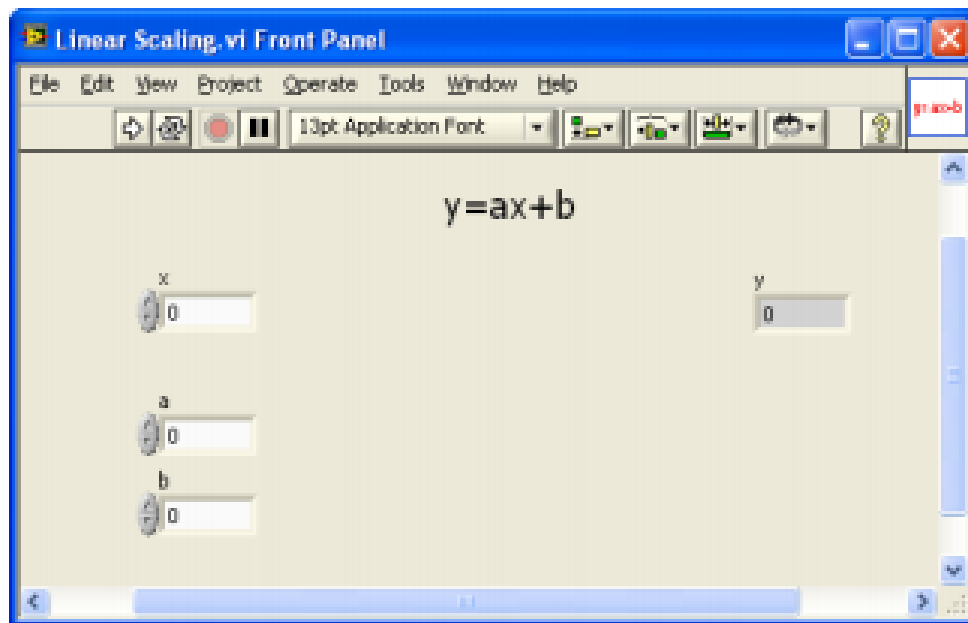
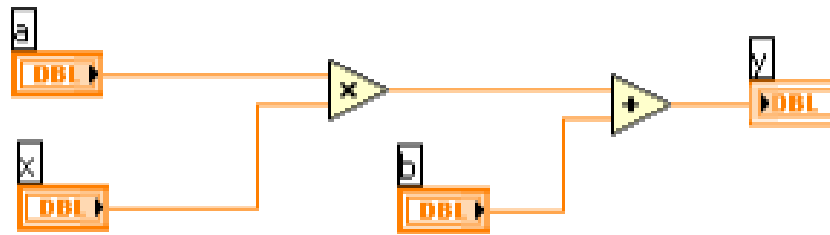


Fig. 2. 6 A simple LabVIEW diagram creating a Virtual Instrument (left) and the user interface provided for the functional diagram (right)

Another important feature of LabVIEW is that it closely supports a multitude of processing cards available from National Instruments, and other vendors also build cards that are LabVIEW compatible. The cards are so tightly coupled to the LabVIEW system that it is common to be collecting data within a few hours of receiving the data collection cards. For these reasons, LabVIEW has become one of the most popular data collection systems in recent years. Within the framework of LabVIEW, user interfaces can be created, data can be collected, signals can be

generated and transmitted from LabVIEW cards, and data can be plotted in realtime. An example of the user interface tools are shown in Fig. 2.7 below. LabVIEW, however, does not permit the setup and protocols of instruments (e.g. sampling rate, wiring, etc.) to be automatically recorded and linked to results datasets, yielding an incomplete record that may inhibit experiment reproducibility. When LabVIEW does not provide what is needed, C code or MATLAB programs usually are developed to be tied to it to provide the required functionality, which results in more scattered software tools and results.

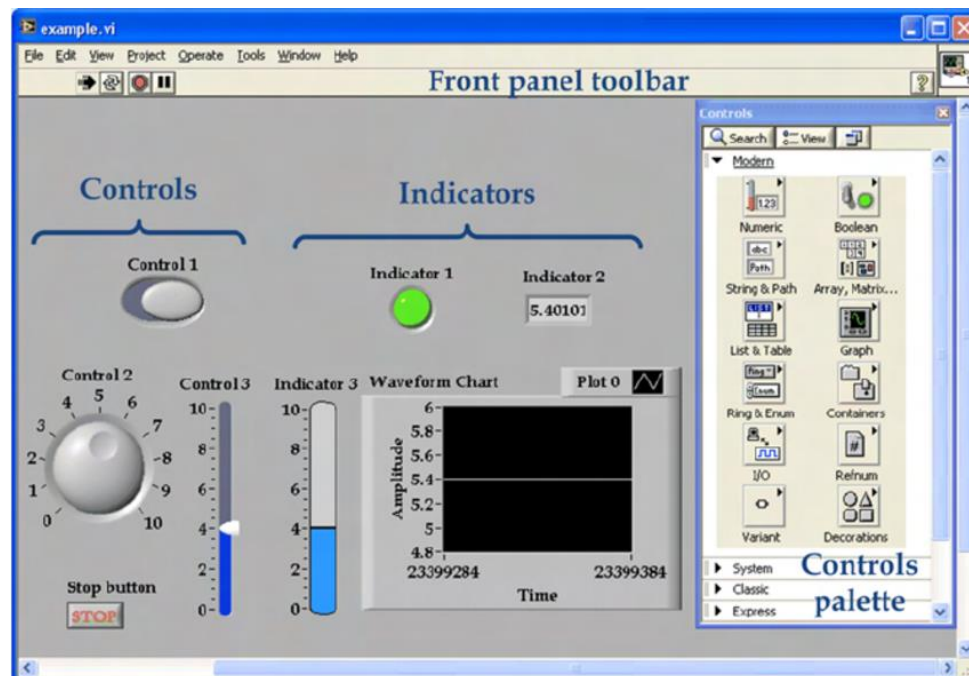


Fig. 2. 7 An example of LabVIEW user interface tools available for data visualization.

### 2.6.2. Scientific Workflow Tools

The typical automated scientific workflows are scripts that call in data, programs, and other inputs and produce outputs that might include visualizations and analytical results. These may be implemented in programs such as R or MATLAB or using a scripting language such as Python or Perl with a command-line interface. However, a significant number of scientists are not

programmers and therefore they cannot spend many hours into writing code to process their data and analyze it for results. Thus, scientific workflow tools and software were developed to equip scientist with graphical user interfaces to create scripts and programs that consumes their data and analyze it for any research findings. A key assumption underlying all scientific workflow systems is that the scientists themselves will be able to use a workflow system to develop their applications based on visual flowcharting, logic diagramming, or, as a last resort, writing code to describe the workflow logic. Powerful workflow systems make it easy for non-programmers to first sketch out workflow steps using simple flowcharting tools, and then hook in various data acquisition, analysis, and reporting tools. Details of the underlying programming code should normally be hidden from the user to increase productivity.

Scientific workflows are now recognized as a crucial element of the cyberinfrastructure [45], facilitating e-Science [46]. Typically sitting on top of a middleware layer, scientific workflows are a means by which scientists can model, design, execute, debug, re-configure and re-run their analysis and visualization pipelines. Part of the established scientific method is to create a record of the origins of a result, how it was obtained, experimental methods used, machine calibrations and parameters, etc. It is the same in e-Science, except provenance data are a record of the workflow activities invoked, services and databases accessed, data sets used, and so forth. Such information is useful for a scientist to interpret their workflow results and for other scientists to establish trust in the experimental result.

There are wide range of scientific workflow systems that are used in various fields of science. The most notable mentions are Apache Taverna, Kepler, and Galaxy. Starting with Apache Taverna, it is used by users in many domains, such as bioinformatics, cheminformatics, medicine, astronomy, etc. Taverna allows users to integrate many different software components,



including WSDL SOAP or REST Web services provided by third parties. Taverna enables a scientist who has a limited background in computing, limited technical resources and support, to construct highly complex analyses over data and computational resources that are both public and private. Fig. 2.8 shown below shows the Taverna Workbench during the design of a workflow.

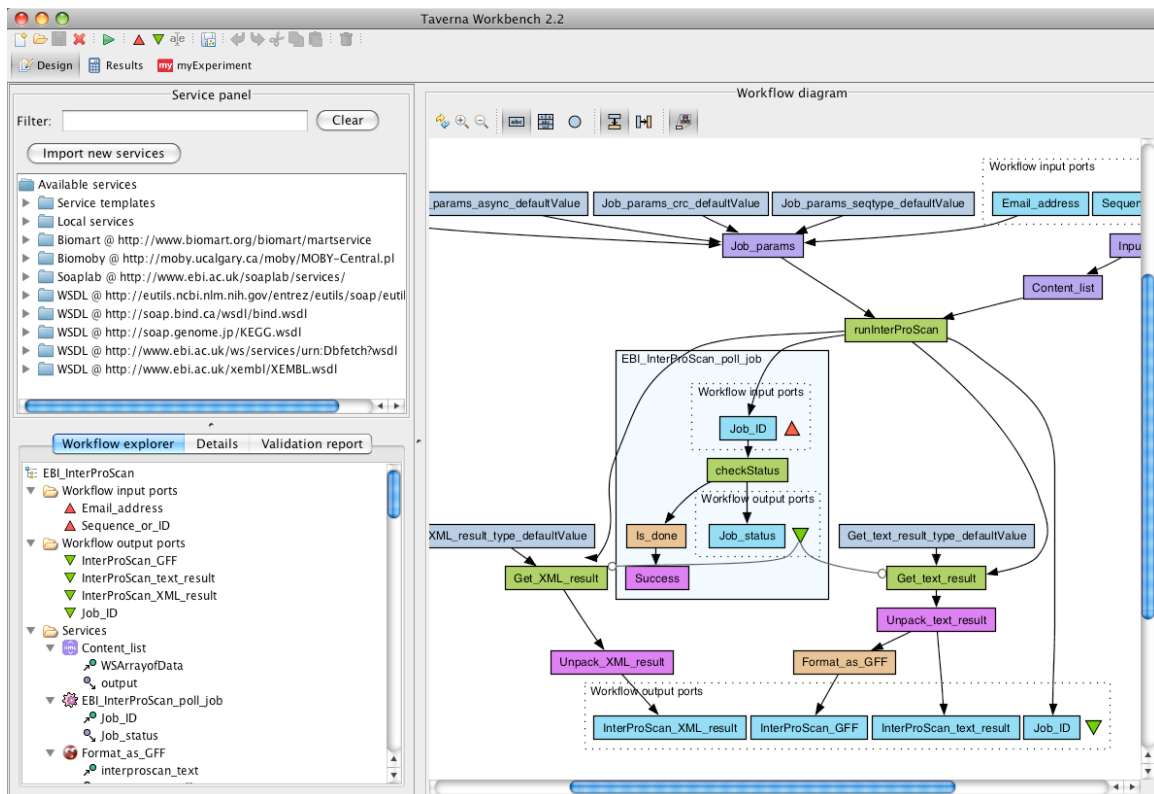


Fig. 2. 8 An example workflow designed using Apache Taverna desktop software.

Another scientific workflow system is Kepler, which is a software application for the analysis and modeling of scientific data. Scientists with little background in computer science can create executable scientific workflows, which are flexible tools for accessing scientific data (streaming sensor data, medical and satellite images, simulation output, observational data, etc.) and executing complex analysis on the retrieved data as shown in the Fig. 2.9 below.

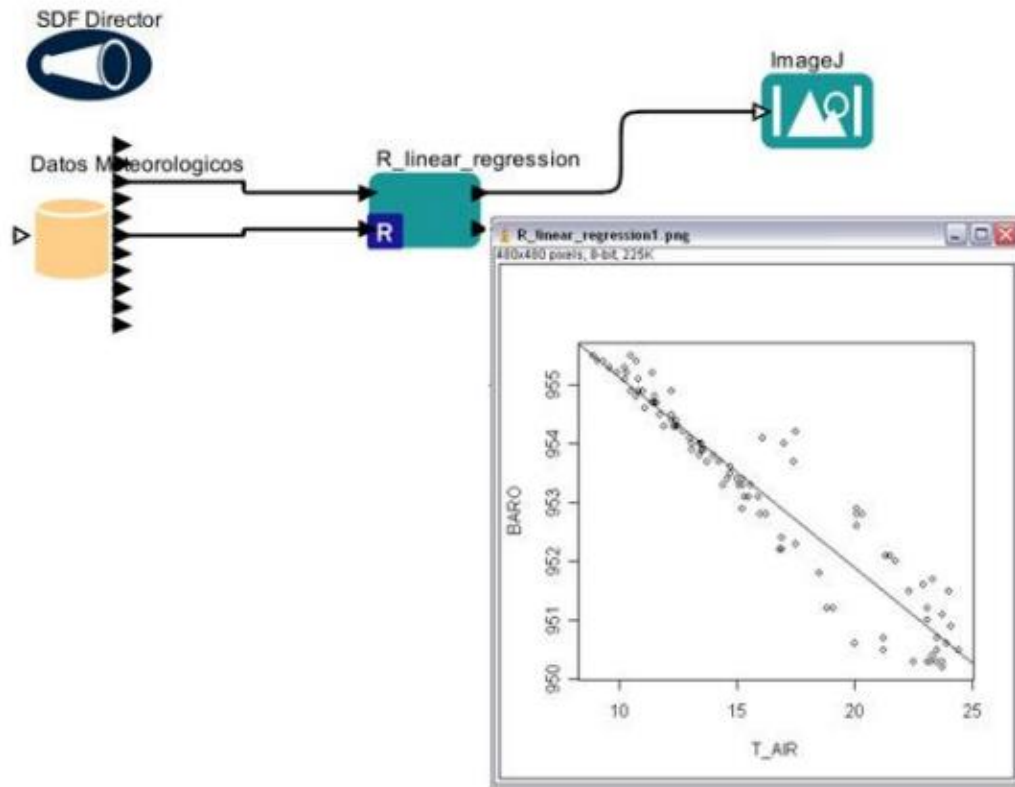


Fig. 2. 9 An example of workflow designed using Kepler's graphical interface and components.

The last scientific workflow system to mention is Galaxy, which is a web-based platform for data integration and publishing that aims to make computational biology accessible to research scientists that do not have computer programming experience [47]. It supports data uploads from the user's computer, by URL, and directly from many online resources. Galaxy supports a range of widely used biological data formats, and translation between those formats. Galaxy provides a web interface to many text manipulation utilities, enabling researchers to do their own custom reformatting and manipulation without having to do any programming. The project goal is to create a platform for performing accessible, reproducible, and transparent genomic science

### 2.6.3. Electronic Lab Notebooks (ELN)

ELN is a concept of a computer program designed to replace paper laboratory notebooks. Lab notebooks in general are used by scientists, engineers, and technicians to document research, experiments, and procedures performed in a laboratory [48]. A lab notebook could also be maintained to be a legal document and may be used in a court of law as evidence. The lab notebook is also often referred to in patent prosecution and intellectual property litigation. There are many ELNs, such as Lab Archives, which is designed to safely organize all laboratory data, manage a research lab as a principal investigator, or review students' lab work as an instructor.

Recently, ELNs starting to expand in the field of computer science which resulted in the birth of one of the most popular ELNs, Jupyter Notebook. The Jupyter Notebook is an open-source web application that allows the user to create and share documents that contain live code, equations, visualizations and narrative text. Uses include: data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, etc. as show in Fig. 2.10 below.

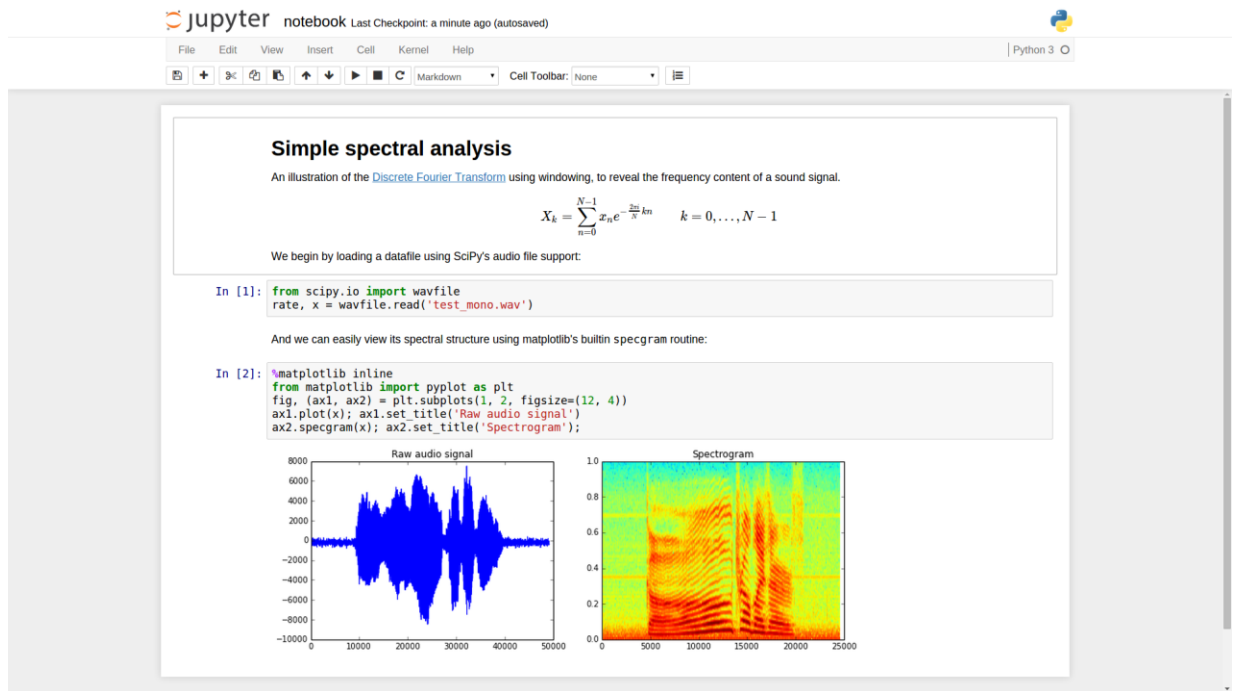


Fig. 2. 10 An example of Jupyter Notebook showing live code containing the algorithm, libraries used, and results plots.

#### 2.6.4. Previous Work

A concept architecture has been previously implemented for a collaborative software toolchain for automatic collection and comparative analysis of sensor characterization data [49]. This work has dubbed the proposed software platform “eGor” which focuses on modularity and runtime extensibility of the software for researchers to build upon to meet specific application need. The short coming of the outcomes of this work, however, lies in the lack of a user graphical interface and overall a functioning software platform for users to design and automate measurement processes and monitor multiple experiment in real time. A previous demonstration of the most recent implementation status is shown in Fig. 2.11.

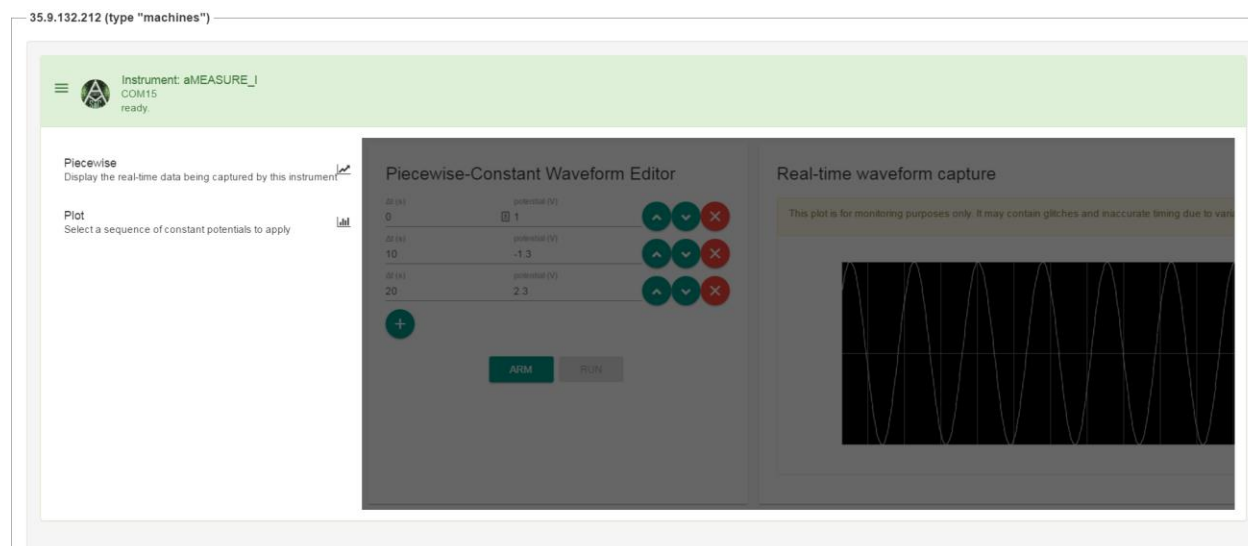


Fig. 2. 11 Shows the state of the previously eGor software tool communicating with a device and collecting data in real time.

## 2.7 Summary

Several software tools for data acquisition, workflow automation, data sets analysis and visualization, and interdisciplinary scientific collaboration have emerged in recent years. Many of these packages provide much needed informatics capabilities that are currently being leveraged by both research labs and the industry. However, addressing the full set of challenges posed by interdisciplinary high-throughput sensor research and development will require the integration of many key features from functionality to intuitive interfacing for experiment automation into a cloud-based software framework that currently does not exist. The remainder of the thesis chapters will describe the proposed design and implementation of a software platform as well as validate the proposed software solution by conducting multiple experiments in real time.

## CHAPTER 3

### DESIGN AND IMPLEMENTATION

#### 3.1 Requirements and Terminology

Given the complexity and variety of sensor characterization experiments, a flexible digital lab assistant platform must possess many general-purpose capabilities. Ideally, the platform should be:

- Automated, employing computer-controlled instruments to collect raw data in real time under user programmable protocols
- Modular and extendable, promoting adoption of new instruments, test devices, experimental methods, and data analysis techniques
- Provenance-aware, tracking the history of executed measurements along with their recipes and any subsequently applied data analysis algorithms
- User friendly, providing researchers with an intuitive graphic user interface to define, schedule, and execute autonomous sensor characterization events
- Collaborative, allowing results and proposed experiments to be shared, annotated, and reviewed at many levels of detail

Before describing, throughout this chapter, the extensive design efforts to simultaneously meet these requirements, it is useful to overview the resulting platform architecture. As shown in Fig. 3.1, the proposed digital lab assistant platform consists of three bundles of software tools linking users to their instruments and measurement datasets. The Instrument Manager (IM) is a tool that runs on a local experiment workbench and communicates with physical instruments for real-time automation and data acquisition. The cloud database is a backend tool that integrates and

curates data and metadata as digital identity research objects (ROs). To provide a friendly user interface to the IM tool and cloud databases, three separate web-based tools are available: the Designer tool for digitally defining measurement recipes (instruments used and their methods and parameters); the Executor tool for scheduled execution and real-time monitoring of events; and the Analyzer tool for producing, browsing and analyzing information rich datasets.

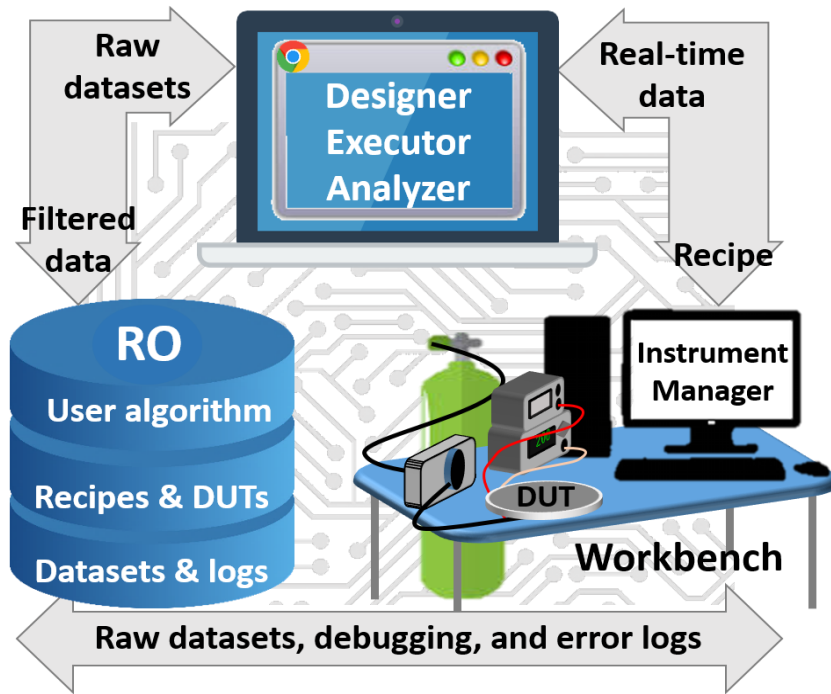


Fig. 3. 1 A diagram of the platform consisting of three bundles of tools: browser interfaces, instrument manager (IM), and database.

### 3.2 Instrument Control and Device Drivers

Programmable instrument control is a requirement for automating measurement recipes to choreograph executable events using a variety of commercial and custom instruments that employ many different communication interfaces with poorly standardized data formats. While some commercial lab equipment conforms to the Standard Commands for Programmable Instruments (SCPI) [50], many other commercial instruments have different standards and data formats.

To accommodate this need, the first tool developed for this platform was the Instrument Manager (IM). The IM tool detects all instruments connected on a physical experiment workbench and identifies their communication interface, data format, and functionality allowing the user to digitally automate the instruments with only abstract knowledge of their operation. These abstractions are manifested in local programs called device drivers. Each apparatus has a unique pre-defined device driver that consist of local data storage, instrument operation API, and communication protocol as shown in the Fig. 3.2 below. The local data storage is used to act as a buffer during real-time data acquisition and thus sending data in chunks to the backend to avoid network delays. The operation API is created by developers to identify the functionality of each instrument so that the user interface can show a list of methods to the user later on. Lastly, the communication protocols are the specific binary commands that the instrument can understand, and the data formats returned by the device itself. The IM tool has a library of device drivers at its disposal to match them with their perspective instrument at run time.

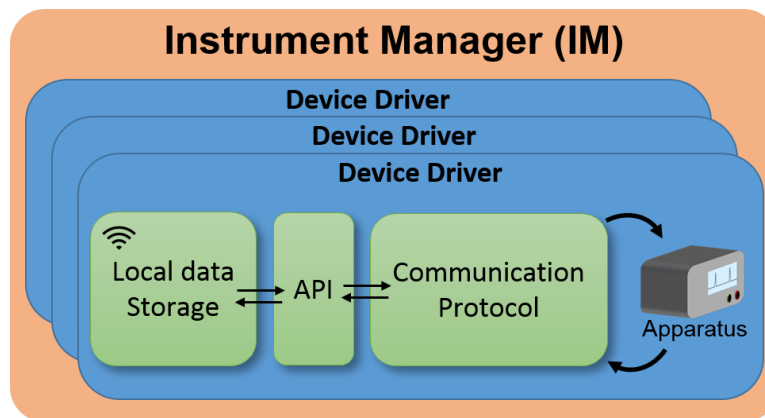


Fig. 3. 3 The Instrument Manager has an aggregation of device drivers, customized by user-plugins for communicating with a wide variety of commercial and custom lab instruments.



The IM design is modular and adaptive so it can easily be expanded to recognize new instruments and their protocols. One have to write a device driver in Python3 to a new instrument by studying the specific instrument datasheet. The developer have to understand how the instrument operate and should create an API that match the device operations. Within each operation, the developer write a function that sends a series of commands that achieves the described behavior. This is a onetime process required to incorporate new equipment to the IM so that it could be integrated with the rest of the proposed software platform.

The IM tool is connected to the backend and waits for an automation job (or a workflow). One a job is received, the IM tool scans for all connected instruments to the PC and match each port with a virtual device driver as shown in Fig. 3.3. The matching process is proprietary algorithm used to query several common commands and analyze the device response by performing pattern recognition using regex. If all of the connected instruments are successfully matched with their perspective device driver, then the IM proceeds by comparing the connected devices to the received automation job to ensure that all of the instruments required by the job is currently available and connected. Otherwise, the IM will not execute the automation job and send it back to the backend with an error status reflecting this process.

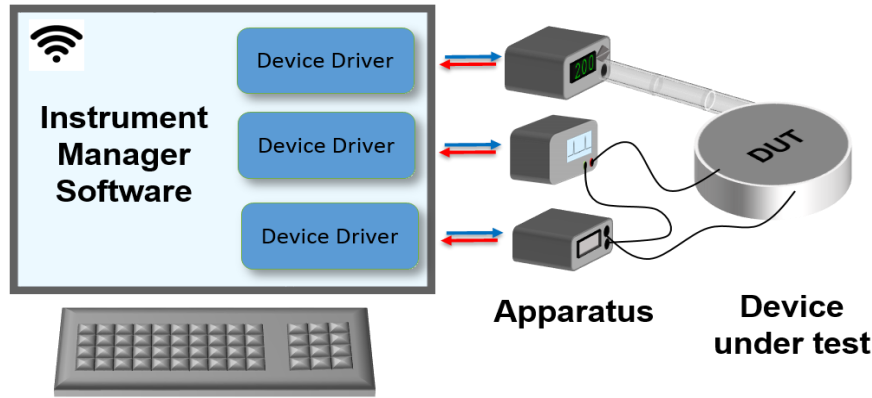


Fig. 3. 5 A physical experiment workbench composed of a local PC connected to multiple instruments to examine a device under test (DUT). The Instrument Manager is an aggregation of device drivers, customized by user-plugins for communicating with wide variety of lab instruments.

However, if there are no errors, then the IM tool starts the automation job by spawning a thread for each port with a connected device. Each thread executes a script of operation that each instrument is supposed to do. During execution, each thread also records the datasets in real-time and annotates results with metadata such as sampling rate, timestamps, code versions, and error messages, allowing users to subsequently deploy deeper analysis on the datasets. The IM dynamically spawns other separate threads during run time in case if one instrument operation thread relies on another instrument. These separate threads contain some common information such as global variables and flags to communicate with each other's.

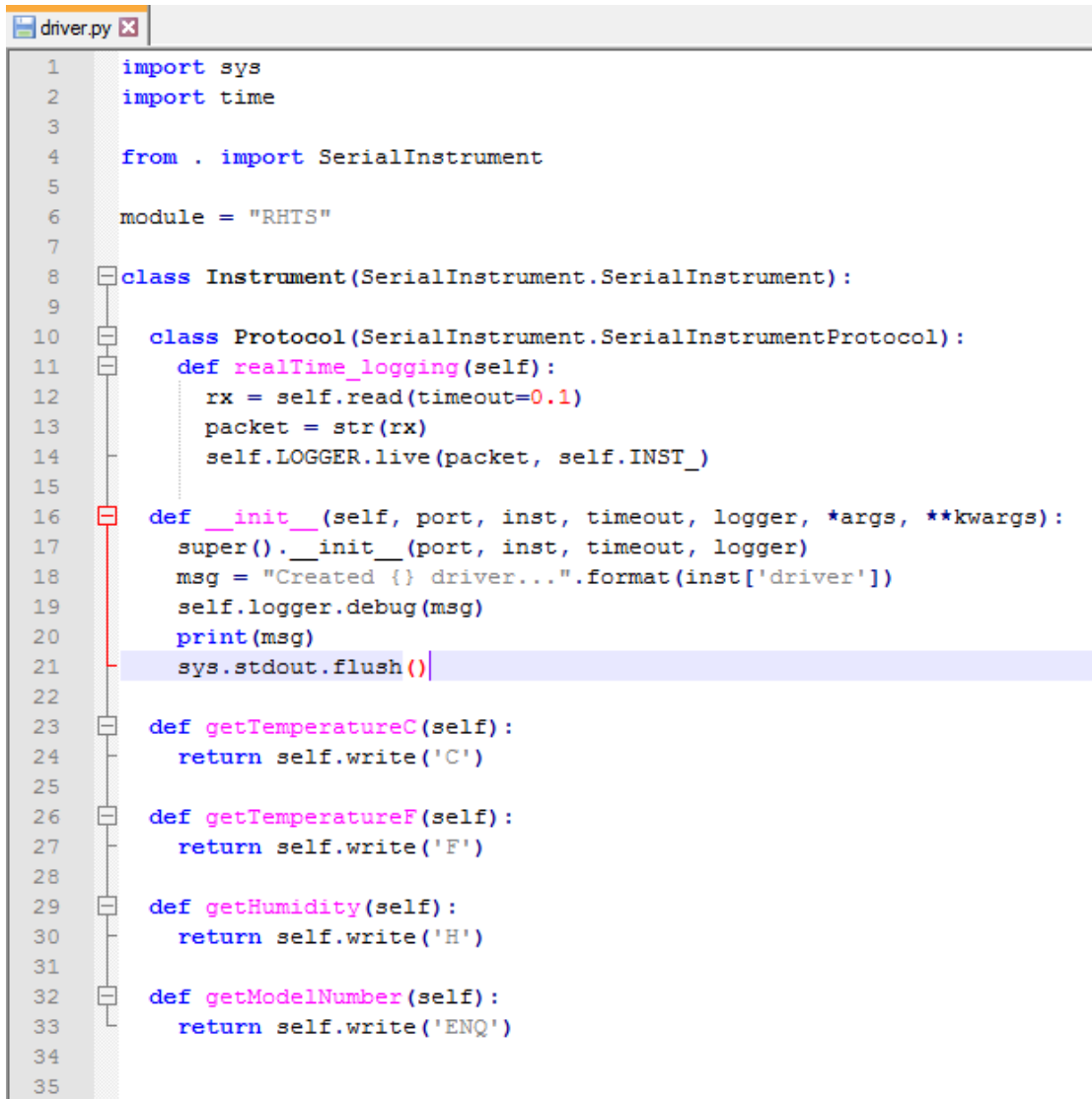
The IM software can be installed as an executable to run on Windows, Linux, and MacOS operating systems and was written in Python3, which was chosen for its code organization and introspection facilities as well as its large set of libraries. The real-time automation tasks were managed using Python's Multiprocessing and pySerial libraries. The pySerial module encapsulates the access for the serial port. It provides backends for Python running on Windows, OSX, Linux, and BSD. It supports for different byte sizes, stop bits, parity and flow control with RTS/CTS

and/or Xon/Xoff. It also provides a file-like API with “read”, “write”, “readline”, etc. methods. Fig. 3.4 shows an example of how pySerial module can be used to open a USB port with default baud rate of 9600 and write a hello message to the device connected on the other end of the USB port.

```
>>> import serial
>>> ser = serial.Serial('/dev/ttyUSB0') # open serial port
>>> print(ser.name)                    # check which port was really used
>>> ser.write(b'hello')                # write a string
>>> ser.close()                       # close port
```

Fig. 3. 7 shows an example of how pySerial library can be used to communicate with a device connected to a USB port.

An example code of a device driver of a simple commercial temperature and humidity sensor called RHTS is shown in Fig. 3.5 below. This driver has four API methods that can be used to read the temperature in two units, get the humidity measured, as well as query device internal information such as model number and firmware version.



```

1  import sys
2  import time
3
4  from . import SerialInstrument
5
6  module = "RHTS"
7
8  class Instrument(SerialInstrument.SerialInstrument):
9
10     class Protocol(SerialInstrument.SerialInstrumentProtocol):
11         def realTime_logging(self):
12             rx = self.read(timeout=0.1)
13             packet = str(rx)
14             self.LOGGER.live(packet, self.INST_)
15
16         def __init__(self, port, inst, timeout, logger, *args, **kwargs):
17             super().__init__(port, inst, timeout, logger)
18             msg = "Created {} driver...".format(inst['driver'])
19             self.logger.debug(msg)
20             print(msg)
21             sys.stdout.flush()
22
23         def getTemperatureC(self):
24             return self.write('C')
25
26         def getTemperatureF(self):
27             return self.write('F')
28
29         def getHumidity(self):
30             return self.write('H')
31
32         def getModelNumber(self):
33             return self.write('ENQ')
34
35

```

Fig. 3. 9 shows an example code of a device driver of a commercial sensor.

This driver code seems to be simple due to the modular design of the IM, which has a higher generic class called `SerialInstrument` that takes care of most of the underlying transmission communication protocols which details specific information such as read and write terminators, encoding mechanism, Unicode handling, logging modules, default sampling rate, data storing, and network connectivity.

### 3.3 Web-based Graphical User Interface

One of the core challenges of designing the targeted platform was to provide the user with an intuitive graphical user interface (GUI) for defining, executing, and analyzing measurement events with limited understanding of programming constructs and without being present at the physical workbench. To achieve this goal, three frontend tools were developed using web technologies that permit a comfortable user experience with remote capability.

Starting with the Designer tool, it was important to develop a simple, flexible, task model that accurately capture realistic experiment parameters and procedures. Fig. 3.6 defines our model of a generic measurement recipe, which exhibits a spatial dimension in the form of Setup (instruments used and their connectivity) and a temporal dimension in the form of Procedures containing the methods (i.e. type of signals) and parameters employed by each instrument over time. This model permits a measurement recipe that users can intuitively define and edit using the Designer graphical tool and ultimately store as a component of the larger RO document for reusability and collaboration purposes. Screenshots of the Designer tool is provided in Chapter 4 of this thesis, which shows that the graphical interface has a timeline for the procedure tab.

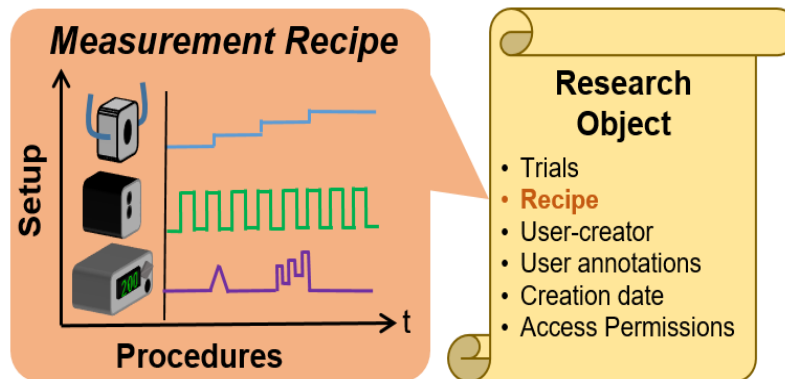


Fig. 3. 11 (left) Model of measurement space as instrument setup and procedures over time. (right) Simplified description of the information rich Research Object containing recipe, measurement data and more.

The Executer tool was designed as a user control panel to remotely execute or schedule (for later execution) recipes with “ready” status, monitor “running” measurement events in real-time, and quickly view and archive “completed” trials. When a design of an experiment is submitted by the user, it gets processed and compiled into a job (or workflow) by the backend then sent back to the Executer graphical tool to show the users their submitted experiments. The Executer tool allows the user to perform few operations on “ready” designs such as execute right now or schedule for later execution. Once the execution time is set, the compiled job is sent to the IM tool to automate the experiment. The automation process is described in the previous section of this Chapter. During experiment execution, the Executer is constantly listening on custom socket for realtime datasets. The Executer graphical tool provides several interfacing tools to plot the data points or view the raw datasets integrated with information-rich metadata constructed by the device driver itself. The realtime data streaming is not stored in the database, the Executer tool was designed to quickly investigate and view any “running” experiment to ensure that it is being executed properly. Moreover, this tool was designed to be independent of the Designer tool, allowing the user to design as many experiments as needed and then switch to the Executer tool to remotely schedule, execute, and monitor experiments.

The last interface element is the Analyzer tool that was designed to allow the user to navigate through stored ROs, inspect logs and view/plot measurement data with respect to input signals and their parameters. The Analyzer tool was designed to complement the previous two graphical tools. The Analyzer tool receives any finished experiments from the Executer tool so that the user can employ further data analysis and inspection. Due to the wide variety of datasets in each field, the Analyzer tool provides a portal for processing data through built-in common filters or user-defined algorithms (e.g. MATLAB code), and for statistical comparison between

multiple datasets. This tool also should give the user an option to download the data in various data formats (text files, csv, etc.) if needed. However, this feature is not commended because the goal of this platform is to encompass every step of scientific workflow and record it along with the raw datasets.

The three frontend GUI tools were implemented using web technologies including Vanilla JavaScript, jQuery, HTML, and Bootstrap CSS. Hypertext Markup Language (HTML) is the standard markup language for creating static web pages and web applications. With Cascading Style Sheets (CSS) and JavaScript, developers can make responsive web design by making static HTML web pages into dynamic pages. Responsive web design is the practice of building a website suitable to work on every device and every screen size, no matter how large or small, mobile or desktop. It is focused around providing an intuitive and gratifying experience for everyone. Desktop computer and cell phone users alike all benefit from responsive websites. The main component of a responsive web design is flexible layouts, which is the practice of building the layout of a website with a flexible grid, capable of dynamically resizing to any width. Flexible grids are built using relative length units, most commonly percentages or “em” units. These relative lengths are then used to declare common grid property values such as width, margin, or padding. Fortunately, Bootstrap CSS framework [51] was developed for this exact reason and therefore the three web-based tools designed in this section are implemented with Bootstrap CSS.

The most notable libraries used to implement the three web-based GUI tools are vis.js, chart.js, and JQueryUI. Starting with vis.js [52], the Designer tool uses this library to construct a timeline component for the user to describe the experiment procedures. The timeline is an interactive visualization chart to visualize data in time. The data items can take place on a single date or have a start and end date (a range). It can be freely moved and zoomed by dragging and

scrolling in the Timeline. Items can be created, edited, and deleted in the timeline. The time scale on the axis is adjusted automatically, and supports scales ranging from milliseconds to years. Secondly, chart.js [53] library is used by the Executer and Analyzer tool to visualize and plot realtime data. This library provides a wide variety of charts such as line, bar, radar, pie, scatter, area, mixed, etc. The last library is JQueryUI, which is a curated set of user interface interactions, effects, widgets, and themes built on top of the jQuery JavaScript Library. This library can be used whether for building highly interactive web applications for adding a date picker to a form control. JQueryUI has many useful common components that are utilized throughout the three web-based tools designed in this section.

### **3.4 Data Management, Models, and Provenance**

The backend database management tools were designed to seamlessly integrate data and metadata into RO records that enable users to repeat measurements and share recipes and meta-rich results. The ROs also enable data curation, searching through records and tracing the history and connections of any RO component.

Fig. 3.7 shows the database design and relationships between user accounts and their measurement recipes, raw experimental results, and processed results. This architecture enables processed datasets to be saved along with the analysis algorithm without losing the original raw data. These features enhance collaborative data sharing and permit deep data mining of information-rich datasets. The database was designed using the non-relational database mechanism, NoSQL, which provides better scalability than SQL databases for large quantities of data and more flexibility for rapidly changing structures. Specifically, the MongoDB platform was utilized and optimized to manage sensor characterization datasets. Choosing this type of database



is the best option for this platform due to the various types of datasets generated from experiments and thus flexibility is a key design component.

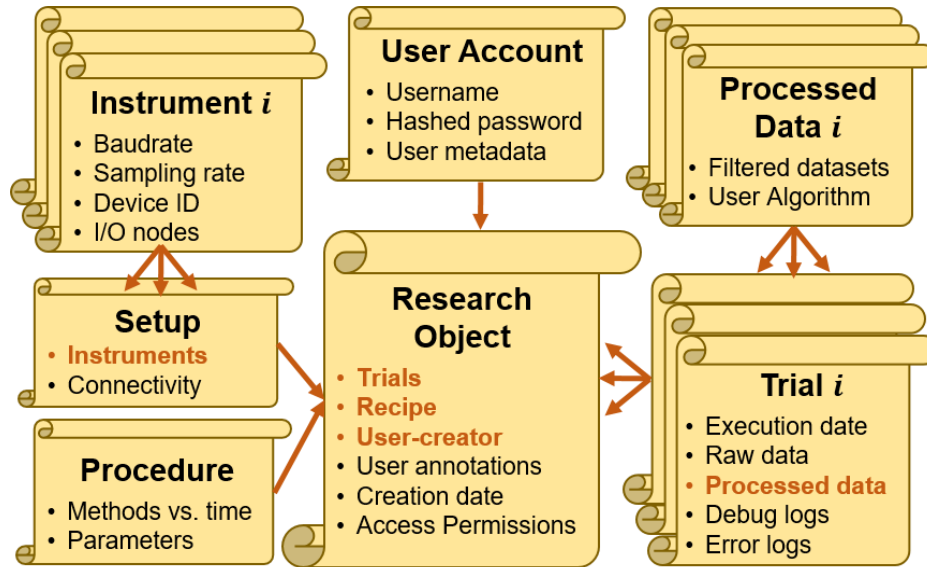


Fig. 3. 13 Database architecture depicting digital curation between user account and research objects that integrate data, metadata, and processed data in one digital sharable identity.

MongoDB is a complex piece of software used by many large enterprises in the industry. However, this platform was implemented to interface with a MongoDB server using a Node.js library called Mongoose [54]. Mongoose is a MongoDB object modeling tool designed to work in an asynchronous environment. It provides a straight-forward, schema-based solution to model the application data. It includes built-in type casting, validation, query building, business logic hooks and more, out of the box. Everything in Mongoose starts with a schema. Each schema maps to a MongoDB collection and defines the shape of the documents within that collection. The example in Fig. 3.8 shows a `blogSchema` created for saving user blogs in MongoDB. Each key in the code `blogSchema` defines a property in the documents which will be cast to its associated `SchemaType`. For example, this example defines a property `title` which will be cast to the `String SchemaType`.

and property date which will be cast to a Date SchemaType. Keys may also be assigned nested objects containing further key/type definitions like the meta property below.

```
var mongoose = require('mongoose');
var Schema = mongoose.Schema;

var blogSchema = new Schema({
  title: String,
  author: String,
  body: String,
  comments: [{ body: String, date: Date }],
  date: { type: Date, default: Date.now },
  hidden: Boolean,
  meta: {
    votes: Number,
    favs: Number
  }
});
```

Fig. 3. 15 An example schema and instance a blog post using Mongoose Library.

### 3.5 Microservices and Messaging Mechanisms

The microservices architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API. These services are built around business capabilities and independently deployable by fully automated deployment machinery. There is a bare minimum of centralized management of these services, which may be written in different programming languages and use different data storage technologies. The design of this platform utilizes the microservices architecture to implement three main services: compiler, DB and scheduler.

These services are employed to perform specific tasks in the background making the platform functional. The compiler service is used by the Designer tool to translate the user defined

recipe into executable instrument commands (or workflow). The compiler was written in C++ using Lex, Yacc, Flex, and Bison tools. The scheduler service is used by the Executor tool to allocate a date to schedule future execution of multiple experiment trials. The scheduler tool was written in Python3 using the Advanced Python Scheduler library. The DB service is used to handle specific database CRUD operations: create, read, update and delete. All of these software services are wrapped using the Docker [55] container to guarantee that these services will always run the same, regardless of the server environment.

Microservices must communicate with one another to exchange necessary information for each service. Therefore, the publish-subscribe messaging mechanism is utilized to handle these intercommunications responsibly. Fig. 3.9 illustrates how the messaging scheme is implemented in this platform. Starting with the user/client, when the user uses the Designer GUI tool to create a new experiment, it would get published with a unique message specifier. Other services are designed to listen to a set of specifiers to react to other services and exchange data. For instance, once the new experiment is published, the DB service is subscribed to that specifier and will take the new experiment in save it in the database. Moreover, the DB service disassemble the new experiment and publishes two new messages denoted by “time” and “recipe”. The recipe message is heard by the compiler, which gets processed into a workflow and published into a new message. The instrument manager tool is listening to the workflow message, which gets automated and results into publishing a new message containing the raw datasets. Finally, the datasets are subscribed by the DB service and gets published back as a RO message. The DB service is critical because it subscribes to most of the messaging specifier to eventually create one research object.

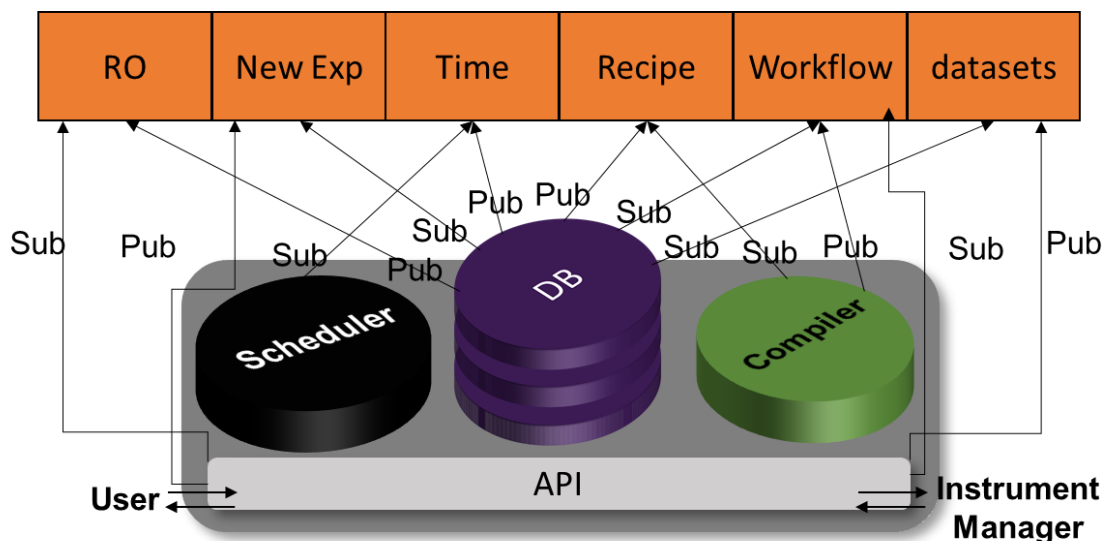


Fig. 3. 17 Publish-Subscribe (Pub-Sub) messaging, used for services to interact but function independently from each other's.

### 3.6 Collaboration, Security, and Networking

Modern research labs are increasingly interdisciplinary and rely on remote sharing of techniques, data, and publications. Software designed for assisting researchers with performing and documenting their work should reflect these realities, ideally offering native support for sharing and collaboratively reviewing resources over the Internet. Software systems with distribution in mind are also well equipped to enforce policies about data usage and to maintain end-to-end provenance information about ROs by managing records in a server-side database. This platform was designed to provide each user with an account that protects their data and provides permission control for sharing results. User accounts are password protected preventing other users from unsupervised access. This platform was designed to be deployed in the cloud as a Software as a Service (SaaS) [56], permitting network access to be handled by the network's firewall and security layer. Fig 3.10 illustrates the networking model of the deployed platform and how communication between two servers are navigated. The platform was designed to be lightweight

in terms of network connectivity; for instance, instrument drivers have local storage to handle chunks of data before sent to the server periodically depending on the sampling rate. Instrument drivers also down sample data for real-time plotting and logging, but do not drop packets.

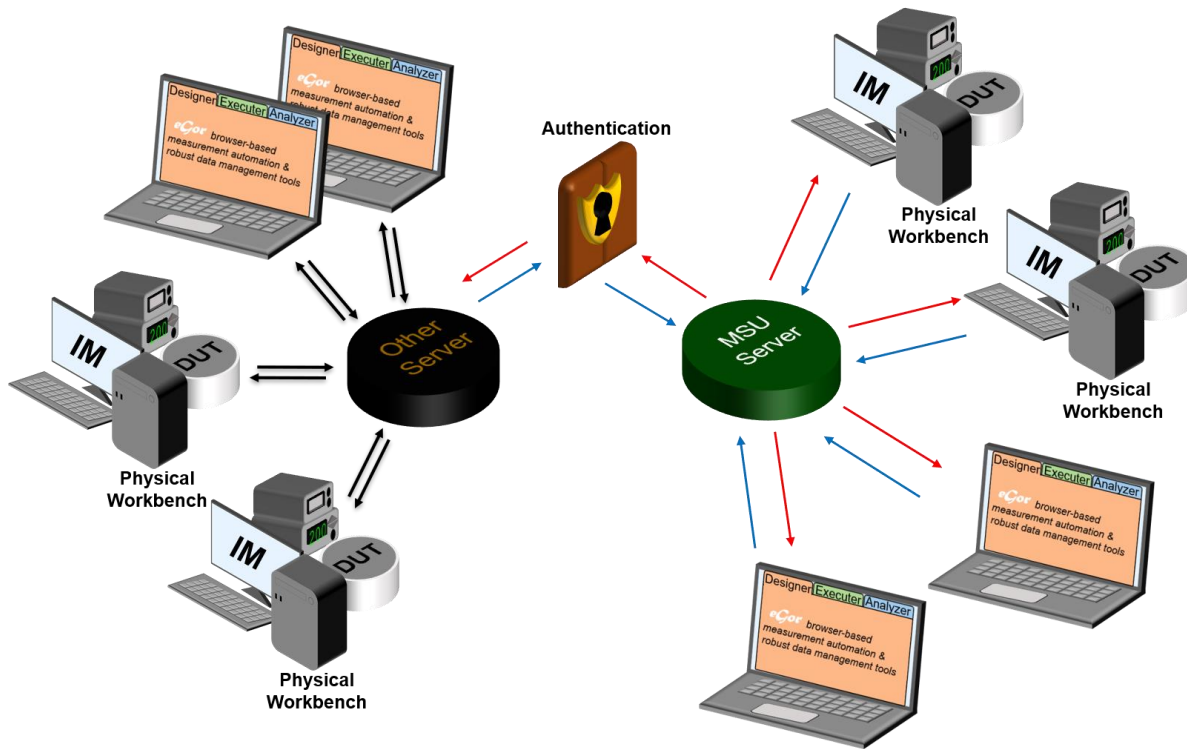


Fig. 3. 19 Networking model of the deployed platform and its inter-communication.

### 3.7 Platform Versatility and Adaptability to Existing Tools

Commercial software with proprietary code is often overly rigid for adapting to the rapidly changing needs of users. To overcome this challenge, the platform should be extendible and modular to meet the users need. The proposed platform was designed to achieve seamless integration of many features while being adaptable to third party user plugs-ins and existing tools. First, the IM tool can easily be expanded to include new instruments and/or add new methods and parameters. The Executor tool could readily be interfaced with LabVIEW for real-time monitoring

and archiving purposes using the LabVIEW WebSocket API [57], allowing users to run the platform in parallel with LabVIEW while preserving the platform's RO database. Moreover, the Analyzer tool has the capabilities to incorporate third party workflow management systems such as Jupyter or user defined MATLAB and Python algorithms. Together these things enable the proposed platform to versatility, expandability, extendibility, and useful to be used in many labs.

### **3.8 Platform Dataflow**

Section 3.1 described the design requirement and architecture from a functional point of view. However, the implementation of such design greatly affects how the platform operates and how the data flows from the IM to the backend and from the database to the three GUI tools on the client's browser. The three GUI tools empowers the users with a virtual lab that enables them to remotely control laboratory equipment and automatically collect stream data to their laptop. To illustrates the implementation of such platform, Fig. 3.11 below shows the dataflow from the user GUI on their personal laptop to the physical experiment workbench and everything in between.

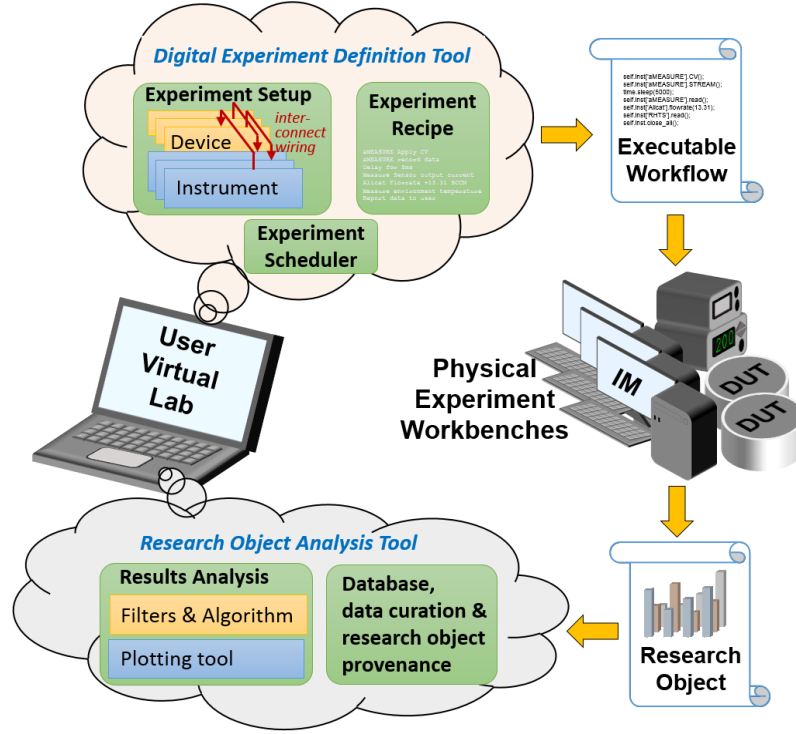


Fig. 3. 21 Functional schematic of the proposed platform for managing user experience from the experiment definition and hardware execution stage to digital curation throughout the life cycle of experiment data.

### 3.9 Software Development Environment

All code development process was tracked and maintained using the Git [58] version control system, which is a standard for software team environment. Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later. The development process of the proposed software followed a popular approach called Agile development process [59], which consist of seven steps to be performed in a specific period of time (often two weeks). This period is called a “sprint” in which developers and teams plan a two-week sprint, design and develop the plan, test and deploy the code, then review the final product and launch it to the marker. Fig. 3.12 shows the Agile software development process for a typical two-week sprint.



Fig. 3. 23 Agile software development environment consisting of seven steps.

### 3.10 Summary

This chapter outlined the system architecture and implementation choices for a set of interacting software components. This approach provides a reasonable flexibility to realize the list of desirable features to meet the platform goals. In particular, the design patterns and implementation choices employed emphasis on modularity benefits any future platform development in terms of scalability, extensibility, and user customization that are not seen in existing software solutions. In the next chapter, the platform is realized and shown with screenshots and experimental results will be discussed and compared with other existing solutions.



## CHAPTER 4

### RESULTS and DISCUSSION

#### 4.1 User Experience Design

The interface is designed to with a dark mode style showing a night sky with stars in the background throughout the entire web-based graphical user interface as shown in the Fig. 4.1 below. The welcome screen shows the platform logo, which is a night owl representing wisdom and emphasizing that this platform’s goal is to run experiments throughout the night to increase research scientist’s productivity.

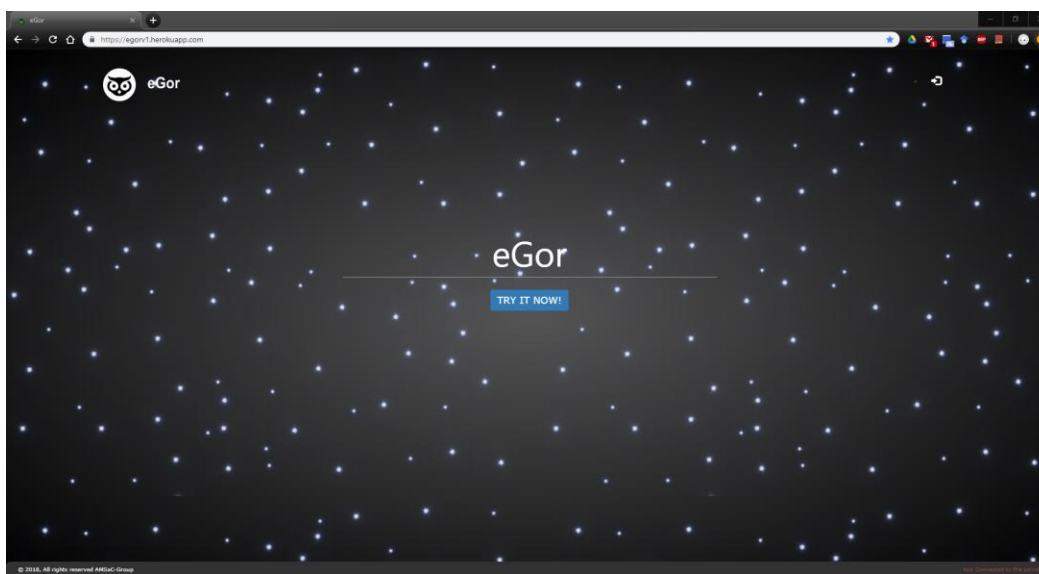


Fig. 4. 1 shows the welcome screen of the designed user interface.

The platform is called “AutoX”, which a release of the digital lab assistant “eGor” previously discussed in Chapter 2. Combing the two names together, this work is branding this new platform as “eGor:AutoX”.

The user interface experience begins its functionality with the shown buttons on the welcome screen. Once the user clicks on the “TRY IT NOW” button or click on the login icon on the top right corner, then the user will be navigated to the login/signup screen as shown in the Fig. 4.2 below.

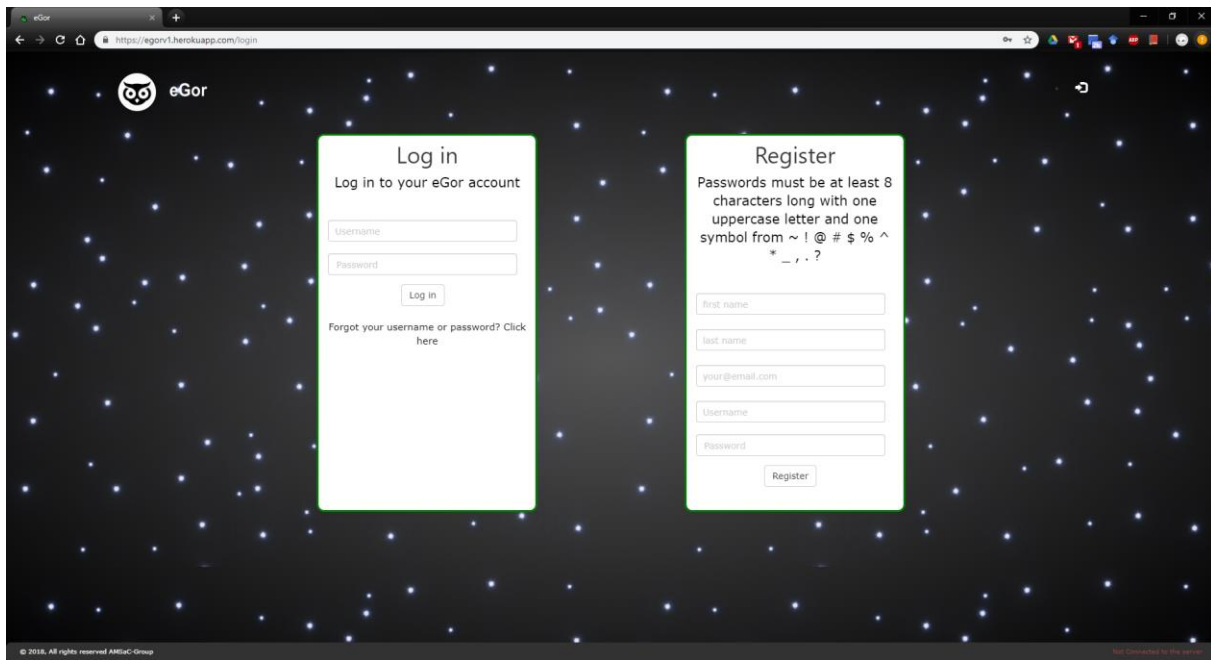


Fig. 4. 2 shows the login and signup screen where the user could input their credential to login or register to the platform to start using it.

The default behavior of the platform is to navigate to the Executer tool as soon as the users logs in so that they can execute “ready” experiments or monitor “running” experiments. We anticipate that many users will design several experiments using the Designer tool then spend most of their time executing the same experiment for multiple trials, and finally use the Analyzer tool to quickly view results and apply data analysis techniques. The three web-based GUI tools are shown and discussed in the following section.

## 4.2 Instrument Automation and Measurement Data Curation

### 4.2.1 Experiment Setup

To show functionality of web-based tools, different lab bench setups were prepared using a combination of different instruments as shown in Fig. 4.3. To verify functionality, the platform was used to automate few experiments including a characterization of an experimental electrochemical gas sensor [60] using a custom miniaturized electrochemical readout system [61] [62]. To record environmental conditions, a commercial relative humidity and temperature sensor (RHTS) was used. The platform has been tested to automate other commercial instruments such as Fluke 289 digital multimeters, Alicat mass flow controllers, and a custom Badge [63] packed with several sensors for social and environmental monitoring. The platform also was live demonstrated in professional peer-reviewed IEEE conferences [63] [64] [65].



Fig. 4. 3 hardware and software to configure a test environment and then use eGor:AutoX to digitally define their test configuration and execute the tests.

### 4.2.2 Automated Measurement Design using the Designer Tool

The Designer tool like the rest of the other web-based GUI tools is implemented with a tabular format. Specifically, the Designer tool has three main tabs: Setup, Procedure, and Meta as shown in the Fig. 4.4 below.

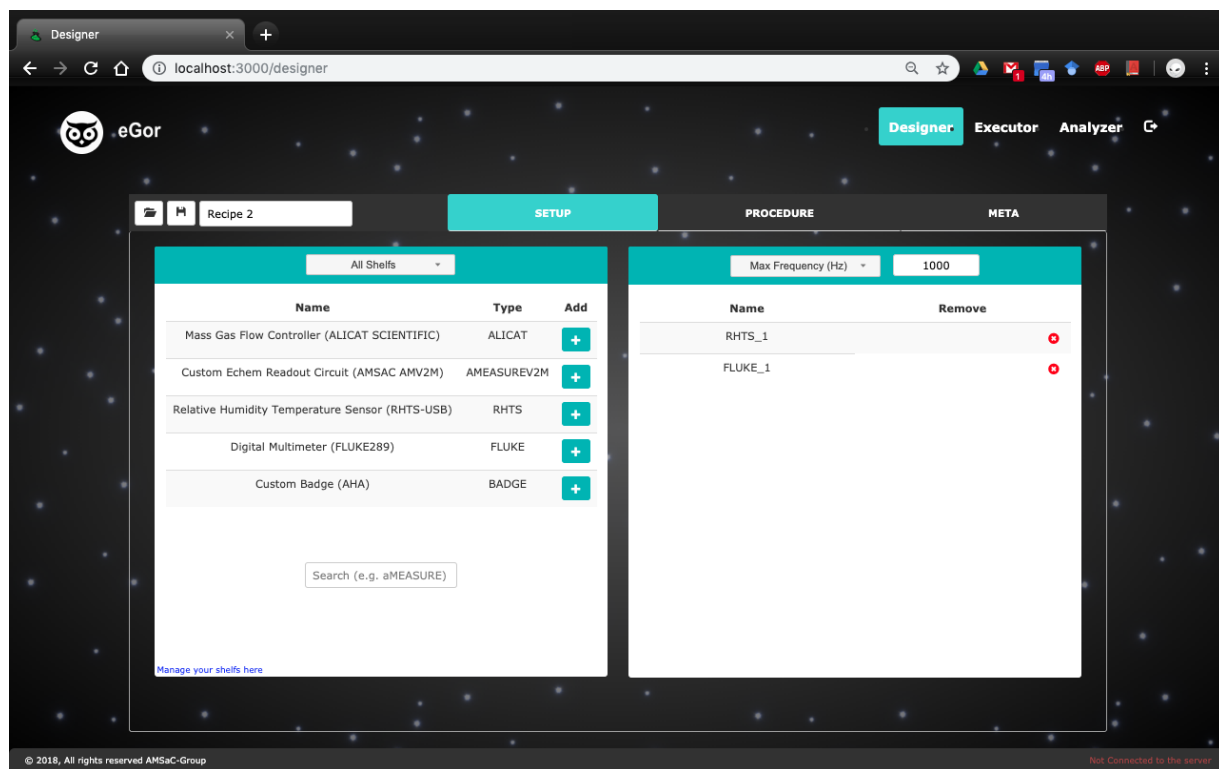


Fig. 4. 4 Snapshot of the Designer browser tool with a sample setup with two instruments to be used during execution of the designed recipe.

The Setup tab allows the user to search the inventory and add instruments to the measurement recipe. The setup tab has two main panels, the left window displays all the instrument that the logged-in user can use for defining an experiment. The platform itself controls permission accesses to these instruments using a user identification unique identifier (UID) generated by the MongoDB database. These instruments represent the device drivers that each user can use during

experiment execution. No other instrument would show up in this panel unless a pre-defined device driver is developed for the IM tool to use.

The user has the ability to organize these instruments in a form of “shelves” that contain a subset of these instruments as shown in the two examples below in the Fig. 4.5. The first window shows the “echem” shelf which contains instruments that are often used in conducting electrochemical experiments, and the “commercial” shelf contains all commercial instruments that the user has access to. One instrument may show up in more than one shelf, however, the backend and the database only have one copy of the instrument driver thus reducing data redundancy and decreasing the database initial deployment size.

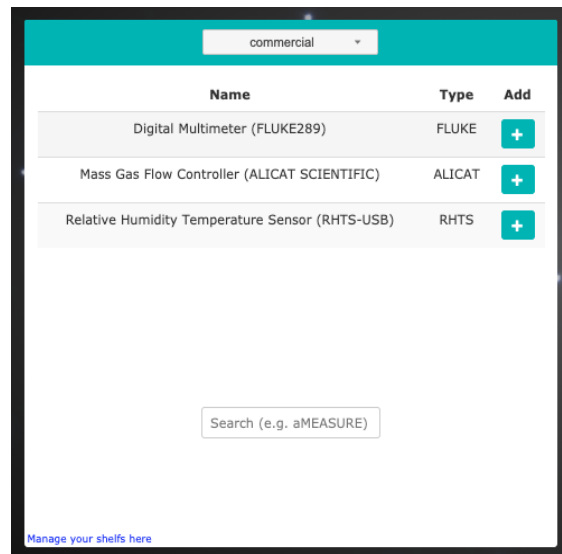
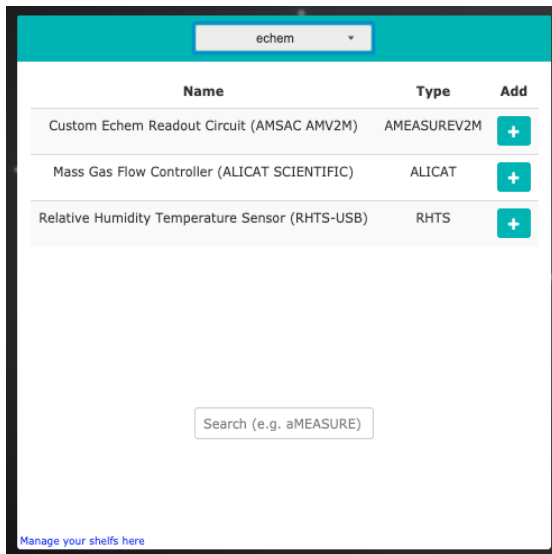
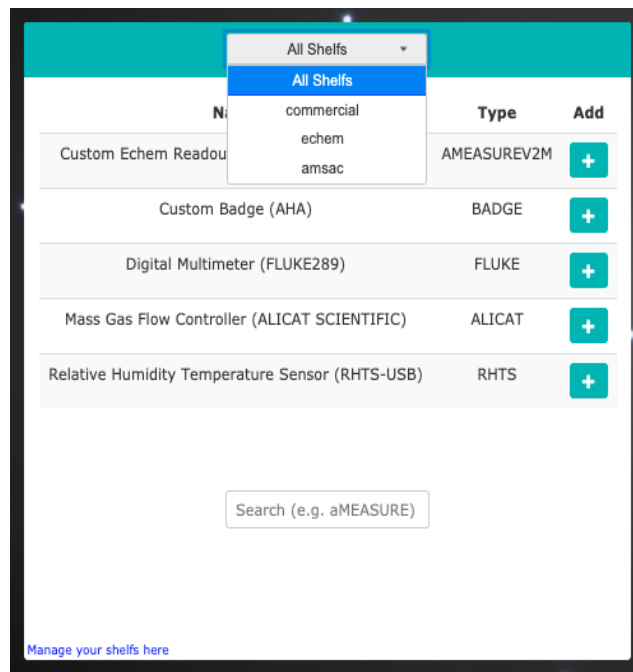
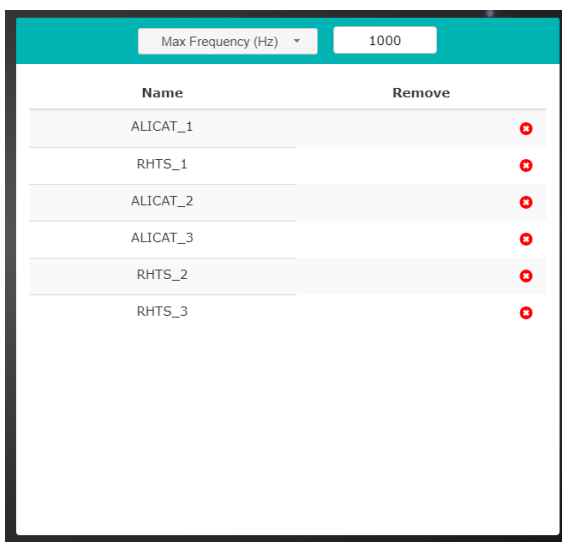


Fig. 4. 5 Snapshot of the setup shelves used to organize different types of instruments based on user preference.

The user can add an instrument to the experiment design by clicking on the plus button residing right next to each instrument. Subsequently, the instrument will pop up in the right panel as shown in Fig. 4.6 with the device name and a suffix indicating how many of each instrument is

going to be used. The number tagged to each device name as the user adds the same instrument multiple times as show below. The user can subsequently delete any instrument added by clicking on the red “x” icon. This can be done as long as the experiment is not submitted yet, in which case, the experiment recipe (i.e. setup and procedures) are immutable.









Name	Remove
ALICAT_1	
RHTS_1	
ALICAT_2	
ALICAT_3	
RHTS_2	
RHTS_3	

Fig. 4. 6 Snapshot of the setup indicating which instruments are used and how many of them are needed to conduct the designed experiment.

Before moving to the procedure tab, the user is recommended to name the designed recipe in the top left corner of tabular bar as shown in Fig. 4.7. This name should be used by the user to identify experiment from each other’s. The “save” and “folder” icons can be used to save this design as a template and subsequently load this template for future experiment design. These templates will not show up in the Executor tool because their purpose is meant for expediting the design stage by starting from a previously saved template. Users also have the ability to open previously submitted experiment recipes and use them as a start, however, once submitted again, a new experiment will be created in the database with a new RO document. The platform can track if any recipe has originated from a previously submitted design or template.

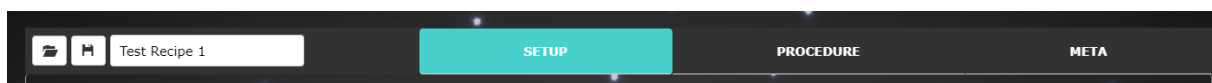


Fig. 4. 7 Snapshot of the tabular bar showing the recipe name and all three tabs available in the Designer tool.

The Designer tool includes a second tab, a Procedure tab, that allows users to define the sequence of methods and parameters for each instrument in an intuitive and user-friendly manner. Once the user is finished with the setup tab, the selected instrument will show up in the Procedure tab timeline as shown in Fig. 4.8. In this experiment, a recipe was prepared to continuously record temperature and humidity using the environmental sensor (RHTS) and run a cyclic voltammetry (CV) method followed by a constant-potential amperometry (CA) method using our custom electrochemical instrument, aMEASURE. The timeline indicates when and how long each method is going to be executed and can be scaled to seconds, minutes, or hours.

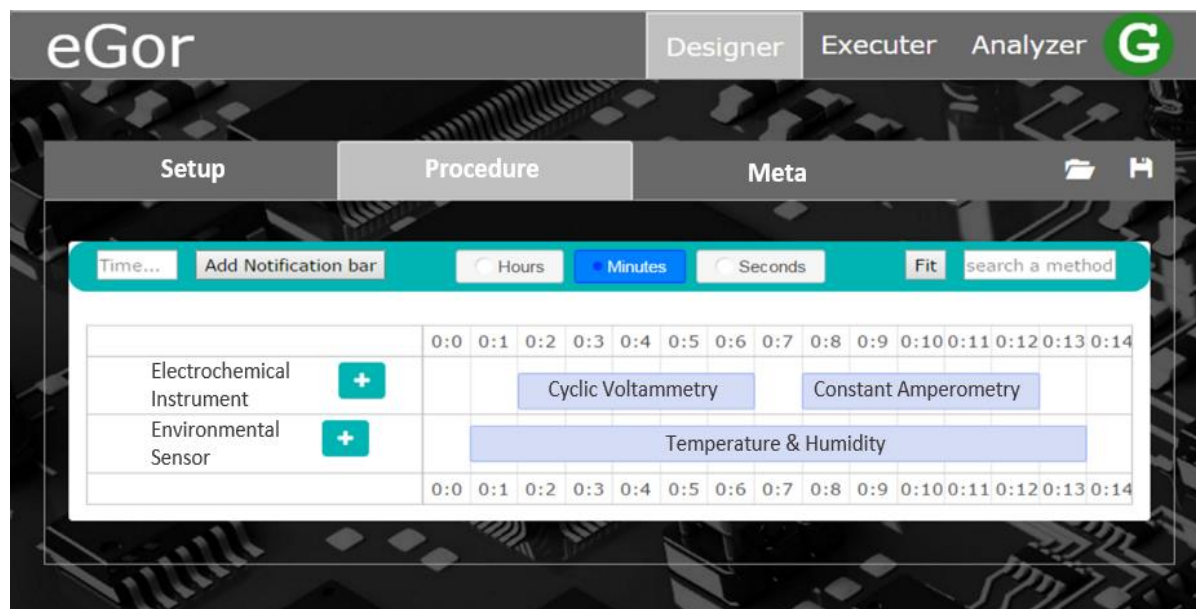


Fig. 4. 8 Snapshot of the Designer browser tool with a sample recipe with two instruments and multiple methods (i.e. CV, CA, etc.) to be executed over a pre-defined time period.



Another example of a recipe was prepared to continuously record humidity using the environmental sensor (RHTS) and read measurement data from the FLUKE digital multimeter as shown in Fig. 4.9 below. The method “Read Data” of the digital multimeter is flexible and automatically determines what the digital multimeter is measuring, whether it is voltage, current, resistance, etc. measurement.

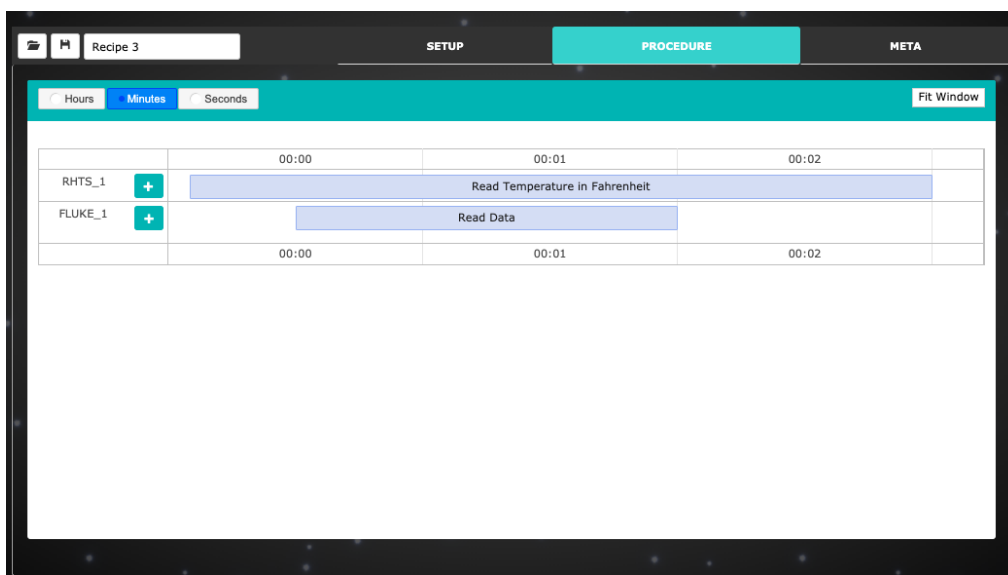


Fig. 4. 9 Snapshot of the *Designer* browser tool with a short sample recipe with two commercial instruments

The general parameters for each method (e.g. start time, end time, method type) were defined in a pop-up dialog box upon clicking on the method (i.e. the blue bar) as shown in Fig. 4.10. These general parameters are common among all methods and are necessary for each method to be included as part of the experiment timeline. Furthermore, each method has its own unique sets of inputs such as sampling rate, initial voltage, etc. Different inputs for two different methods are depicted below.

The figure displays two instances of the 'Method Selection/Editing Popup' dialog box. The left instance shows the 'Cyclic Voltammetry' method selected, with parameters: range (radio buttons), point (radio button), start time (00:00:05), end time (00:01:00), channel\_number (0), dynamic\_gain (1), initial\_voltage (1), High\_voltage (4), low\_voltage (1), and scan rate (1000). The right instance shows the 'Read Temperature in Celsius' method selected, with parameters: range (radio buttons), start time (00:00:05), and end time (00:01:00). Both dialogs have an 'Add' button and a 'Cancel' button at the bottom right.

Fig. 4. 10 Snapshot of the method popup dialog box, all methods has global parameters such as start time, end time, etc. However, each method has it is own unique parameters/inputs to define the method such as sampling rate, initial violate, etc.

In the Meta tab, the user can add personal notes and submit the recipe for error checking as shown in Fig. 4.11 below. When the recipe was complete, it was archived in the database and compiled into an executable form (or workflow) that can be automated by the IM tool at execution time. The GUI does not navigate to a new window but rather start a new design sheet for the user to work on. The idea is that the platform anticipates that the user would want to design multiple experiment then move on to the Executer tool to start running experimental trials. The user gets a notification, however, regarding their previous experiment and whether it was compiled successfully or not as shown in Fig 4.12.

Test Recipe 1

SETUP PROCEDURE **META**

Enter Metadata

Enter semicolon-separated tags

signal processing; electrochemistry; sensors;

Enter notes

This experiment was conducted to demo the designed platform and its wide range of features.

Save & Check

Fig. 4. 11 Snapshot of the Meta tab showing user tags, keywords, and notes.

Designer

localhost:3000/designer

Designer Executor Analyzer

[API\_LOG] saved new exp with id5ca52592db1d3b356e071d51 and sent it to compiler

compiled successfully!

Type Recipe Name

SETUP PROCEDURE META

All Shelves

Name	Type	Add
Mass Gas Flow Controller (ALICAT SCIENTIFIC)	ALICAT	+
Custom Echem Readout Circuit (AMSAC AMV2M)	AMEASUREV2M	+
Relative Humidity Temperature Sensor (RHTS-USB)	RHTS	+
Digital Multimeter (FLUKE289)	FLUKE	+
Custom Badge (AHA)	BADGE	+

Search (e.g. aMEASURE)

© 2018, All rights reserved AMSAC-Group

Not Connected to the server

Fig. 4. 12 Snapshot of the Designer tool indicating that the submitted experiment was compiled successfully.

### 4.2.3 Measurement Execution and Monitoring using the Executer Tool

Fig. 4.13 shows the Executer tool without any experiments designed yet. The tool itself has a similar look to the previous Designer tool. The Executer tool has three main tabs: Monitor, Scheduler, and Notification. The monitor tab has two panels, one for currently running experiments and the other is for scheduling (or pending) experiments. This is the default tab that the user sees as soon as logged in. We anticipate that the users will run many experiments for such long period of time and thus the default monitoring tab will aid the user to periodically and quickly check the experiment execution to ensure that the experiment is running accordingly.

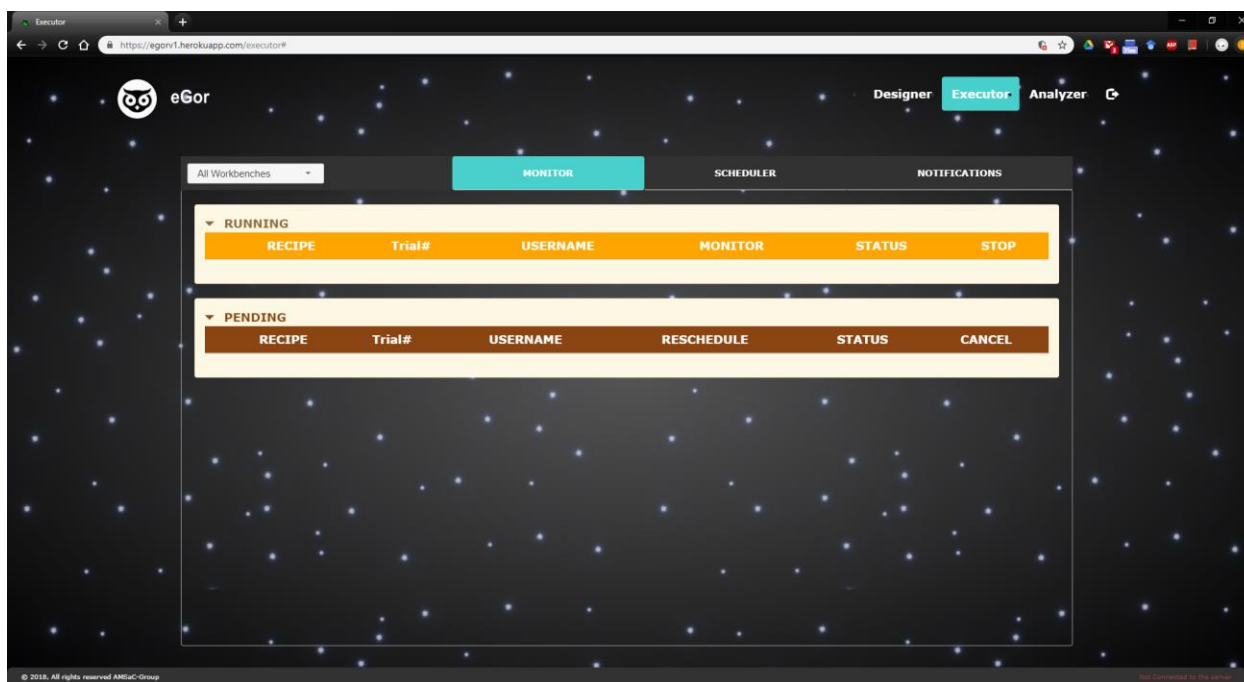


Fig. 4. 13 Snapshot of the Executer browser tool indicating the running, ready, and completed status of example measurement recip.

Once an experiment design is submitted using the Designer tool, the compiled recipe shows up in the second tab of the Executer tool, the scheduler tab. Fig. 4.14 shows these recipes along with other meta information such as the date and time of creation, user creator, status of the recipe,

etc. Each recipe has a distinct set of action icons for user interaction; for example, users can schedule one or more “compiled” recipes for future execution. Other actions include deleting the recipe or re-editing it by making a separate copy of the recipe and opening it using the Designer tool. The Scheduler tab shown in below also includes a section for filtering the compiled recipes through usernames, user tags, data ranges, instruments used, etc. This section enables the user to query the database of compiled recipes in case the number of compiled recipes becomes overwhelming to the user.

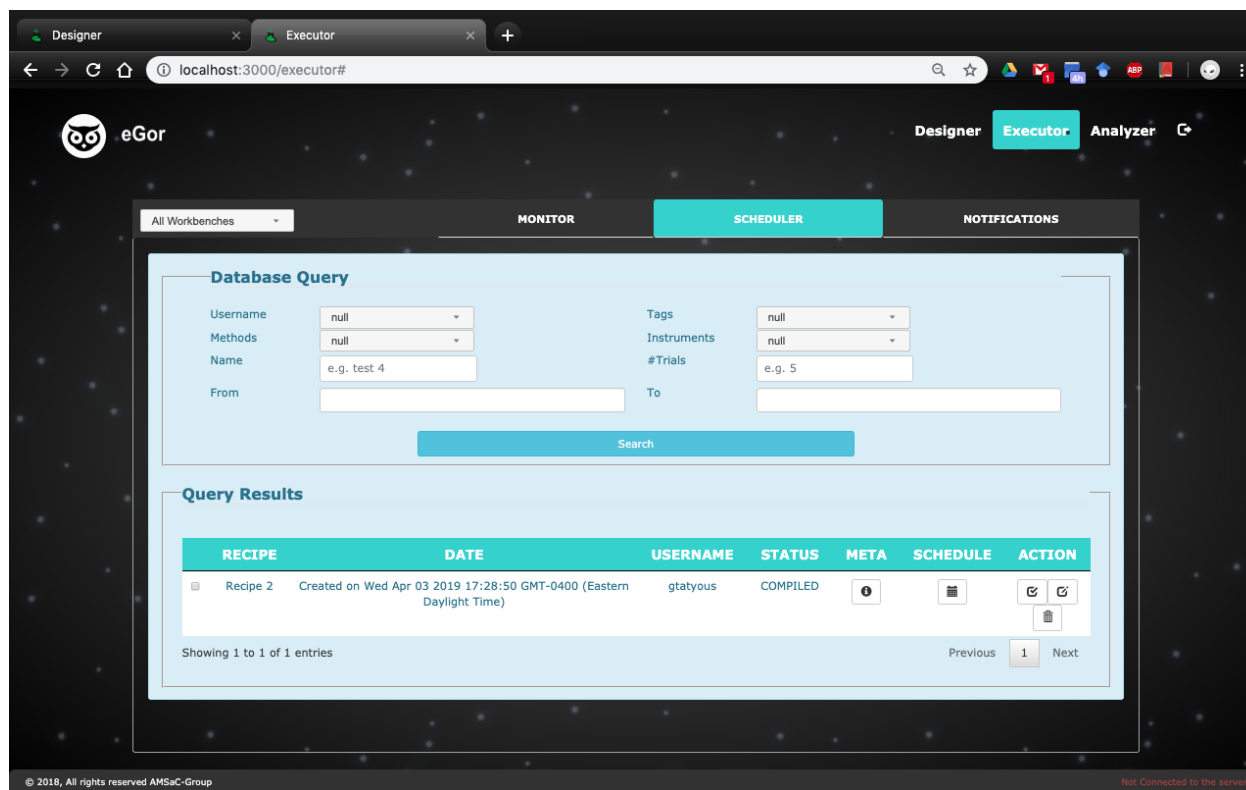


Fig. 4. 14 Snapshot of the Executor browser tool indicating the running, ready, and completed status of example measurement recip.

Once the user clicks on the schedule button, a pop-up screen shows up to let the user either run the experiment immediately or schedule it for later time as shown in Fig 4.15. Other options

for executing the experiment includes choosing which workbench to run the experiment on in case the IM tool was running on multiple desktops, and how many iterations should the platform execute the experiment. The pop-up screens for all of these options are shown below.

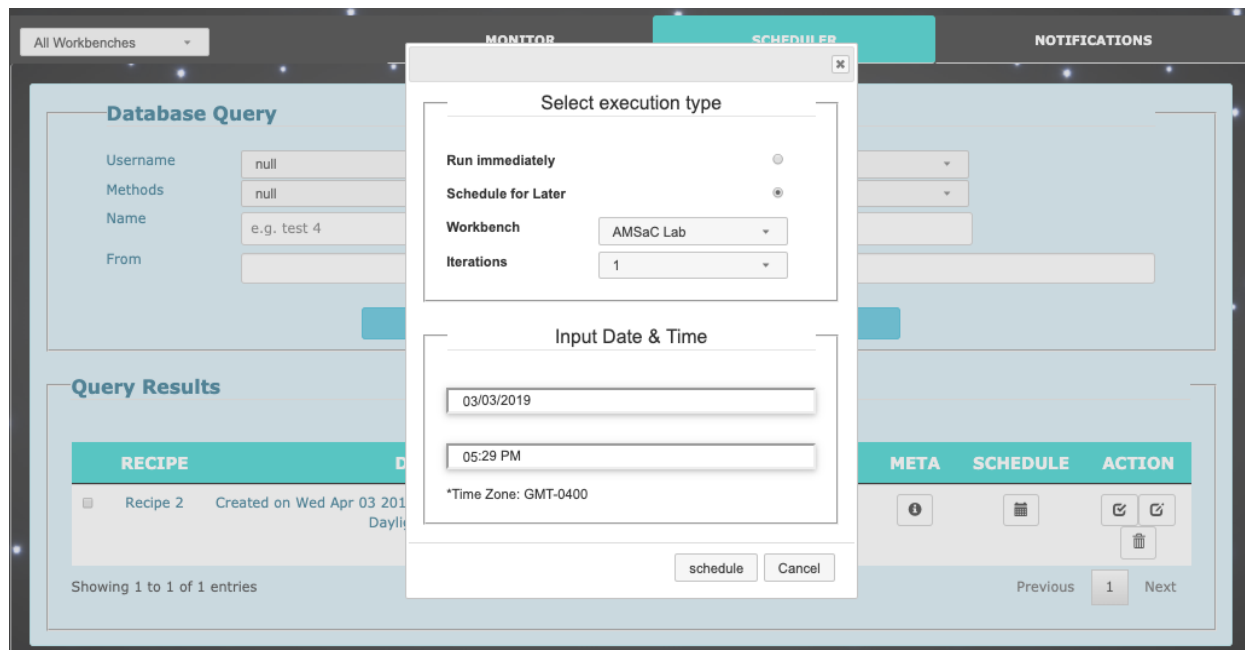
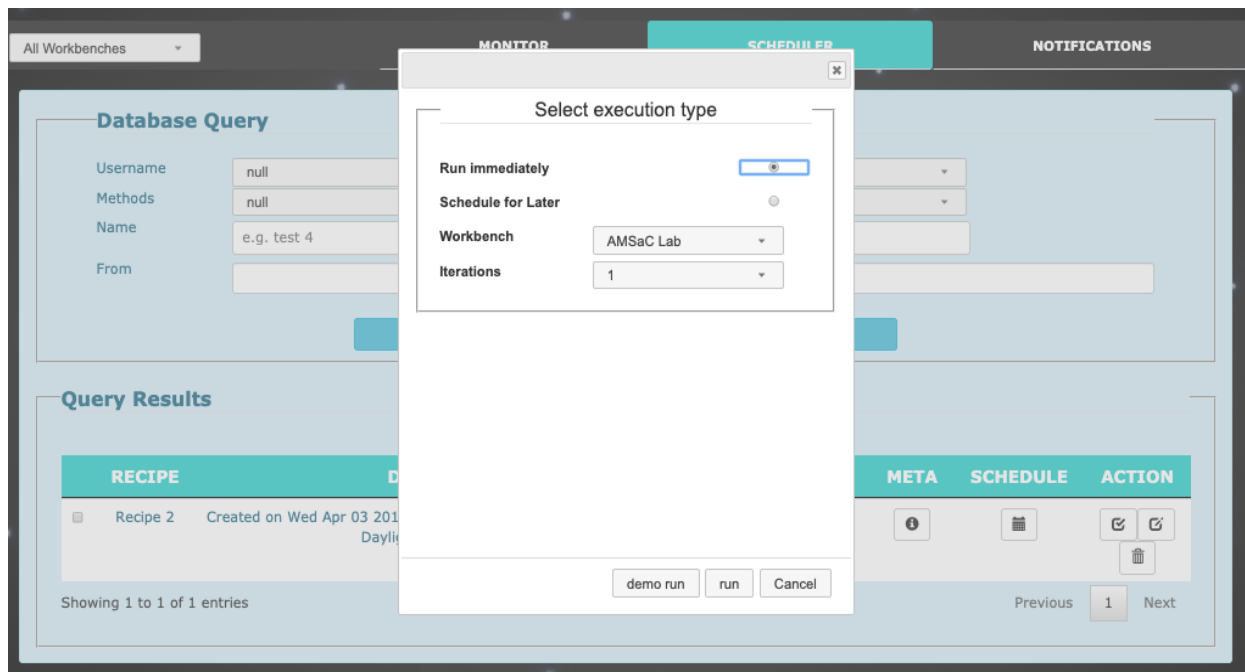


Fig. 4. 15 Snapshot of the pop-up screen used in the Scheduler tab to run an experiment immediately or schedule it for later execution time.

Once the experiment is scheduled, the platform shows a pop-up notification informing the user whether the experiment started executing or not as shown in Fig. 4.16 below. The platform

also navigates from the scheduler tab to the monitor tab, which now show the experiment running in the according panel. The user can use the monitor tab to quickly view real-time data and other meta information regarding the running experiment. The user is also able to perform a set of actions on the running experiment such as stopping the experiment or plotting real time data as shown below.

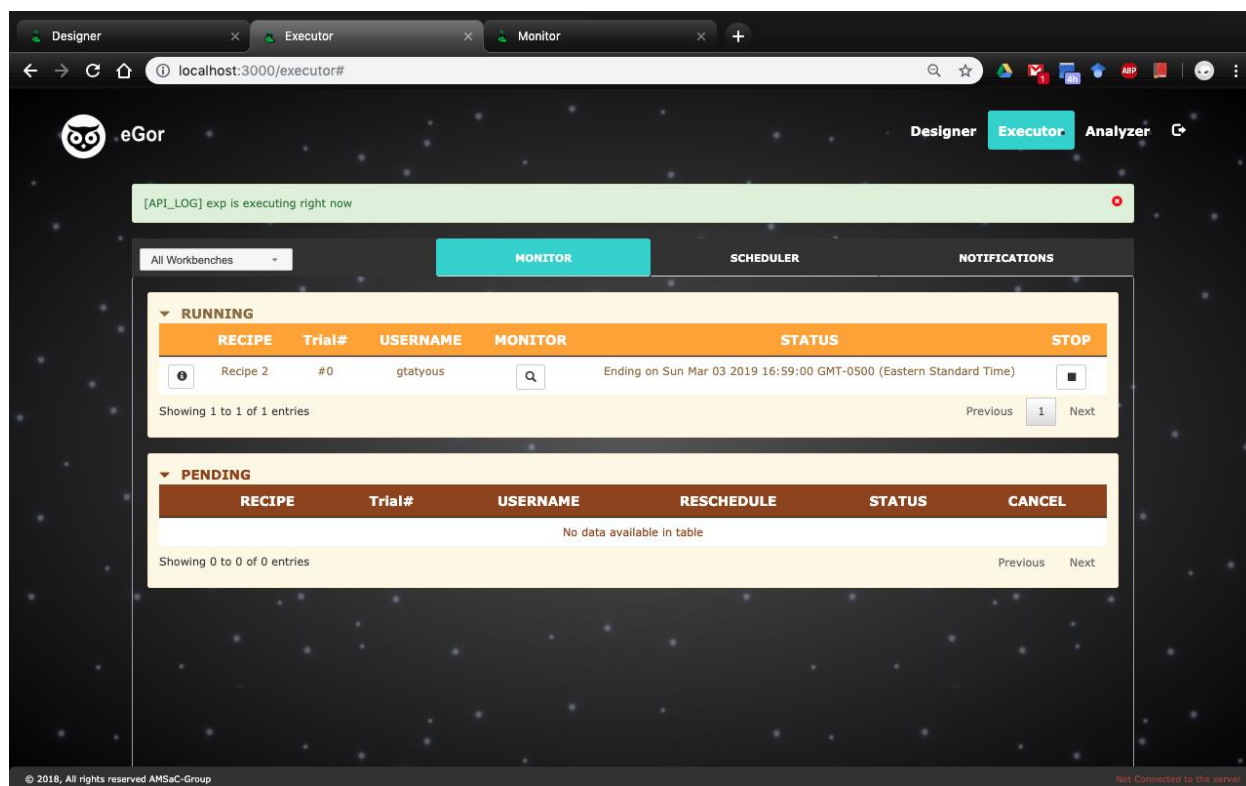


Fig. 4. 16 Snapshot of the Executer browser tool indicating a running experiment scheduled in the scheduler tab.

The second recipe shown in the previous section was also executed simultaneously with the first one. Fig. 4.17 shows the monitor tab indicating that two different recipes or experiments are currently running in the according panel.



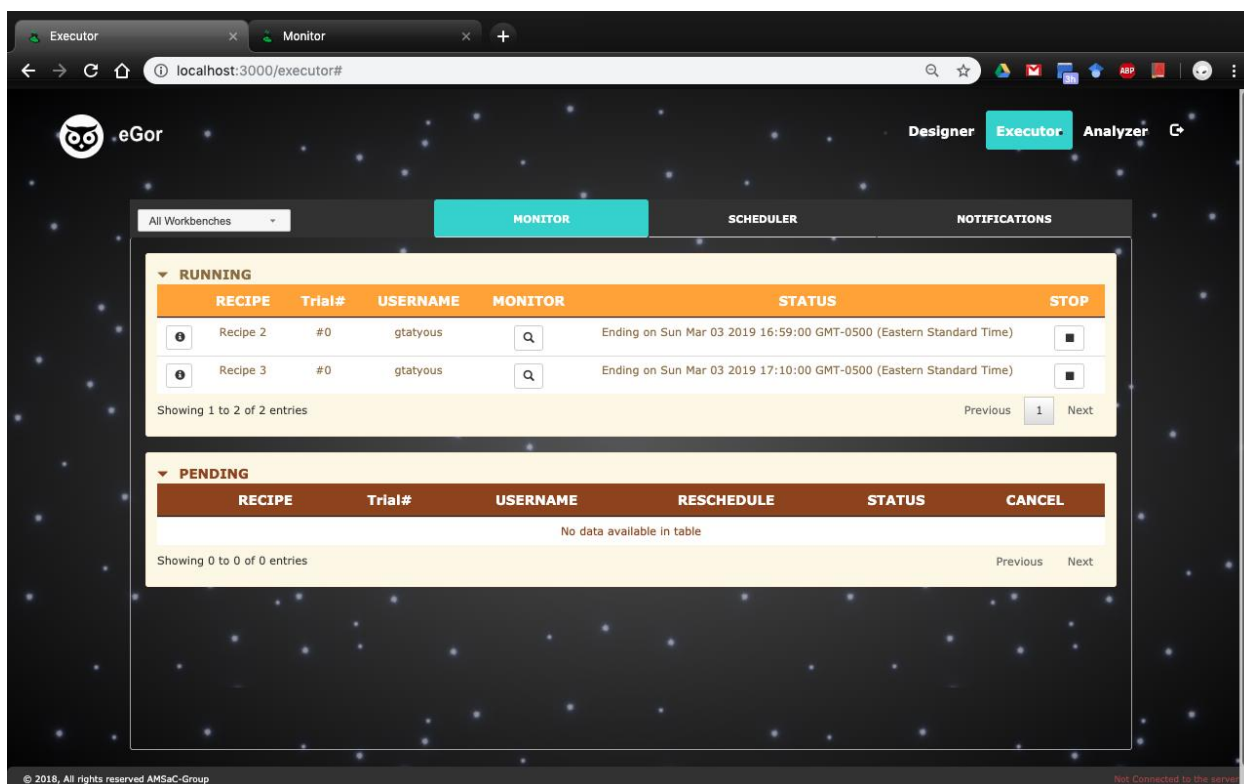


Fig. 4. 17 Snapshot of the Executer browser tool indicating two running experiments simultaneously.

Once an experiment is done executing, the user will be navigated to the notification tab as shown in Fig. 4.18 below. This tab has two panels: Completed and Error panels. Each panel displays the executed recipes and whether it was successfully executed, or a runtime error has occurred. The idea behind this tab is to allow the user to check the results if the recipe was successfully executed and decide whether the trial should be archived or deleted. This allows the user to save space in the cloud in case the experimental results were not desired. The user is also able to reschedule or re-execute the experiment to conduct more trials. The Error panel allows the user to view the run-time errors such as an instrument was unplugged during execution and then quickly be able to either reschedule the another trial or redesign the experiment recipe to correct the errors. Finally, both panels allow the user to delete the data if not desired to save memory

space. In such case, only the data are deleted, all the associated information will still be saved in the databases to preserve provenance of each recipe.

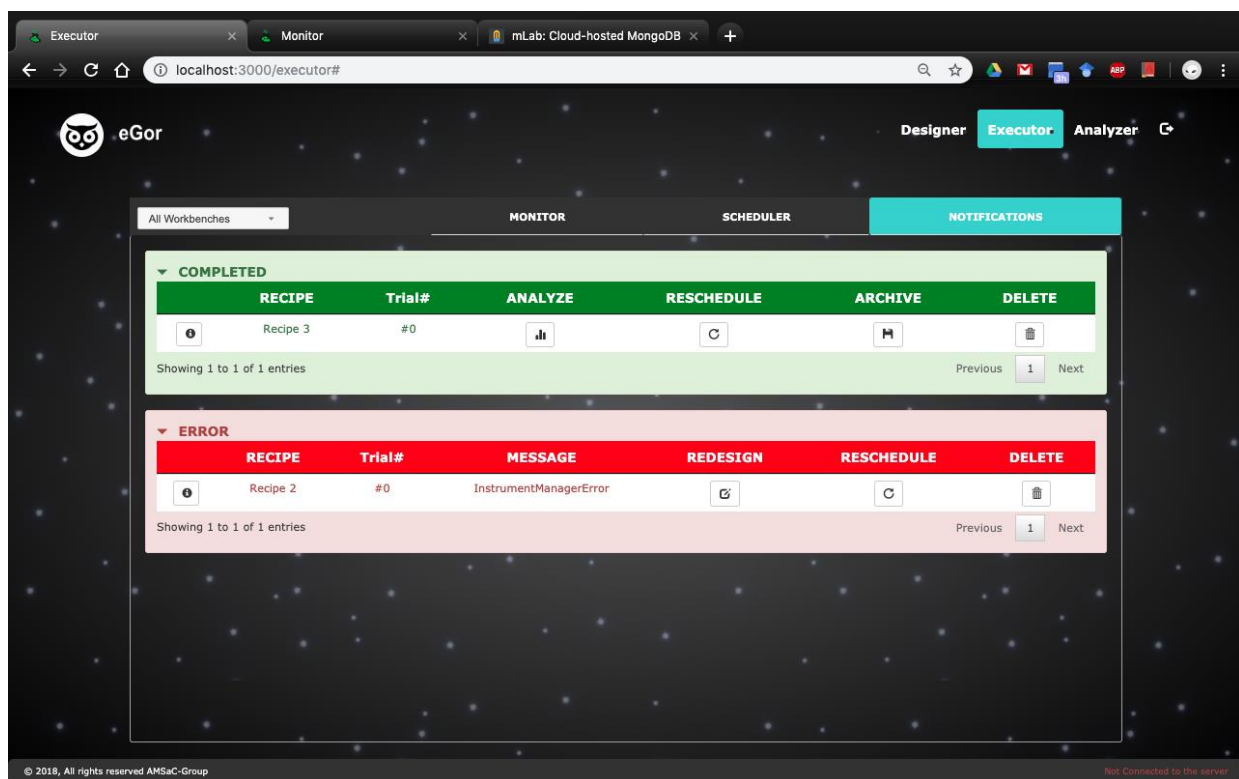


Fig. 4. 18 Snapshot of the Executer browser tool indicating the completed experiments and run-time errors.

#### 4.2.4 Data Visualization and Analysis using the Analyzer Tool

Finally, the Analyzer tool, shown in Fig. 4.19, was used to investigate error and debug logs and view environmental and electrochemical raw data, all shown here as collapsed windows. The Analyzer tool was also used to plot the recorded input stimulus signal and the CV results curve measured during electrochemical gas sensor test. The Analyzer tool does not tamper with the raw data but associates new processed datasets to the source data, thus providing the user with a provenance-aware database. The Analyzer tool was also used to download a .csv record of the RO to further inspect and analyze instruments, methods and parameters and their associated datasets

using MATLAB. All the Analyzer tools can be used after the experiment has finished and all the datasets have been collected and saved in one RO. This simple experiment demonstrates the basics operations of the platform.

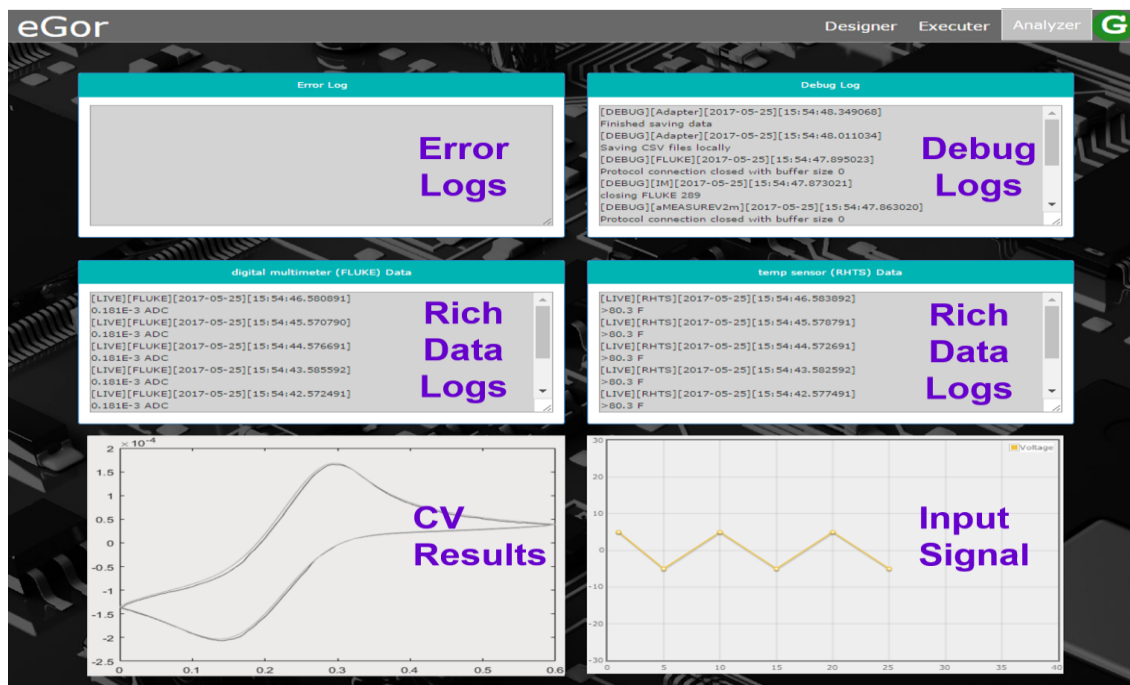


Fig. 4. 19 Snapshot of the Analyzer browser tool displaying run time error and debug logs, timestamped raw data logs, and plots of cyclic voltammetry measurement data along with an input stimulus signal.

This experiment recipe was prepared for performing electrochemical cyclic voltammetry measurements, of a custom screen-printed electrochemical sensor. The scripting language used to define the experiment recipe was given five domain-specific keywords: “IM” to instruct the local automation software, “aMEASURE” to automate with the custom readout circuit, “RHTS” to monitor the surrounding environment, “delay” to halt the automation process for a set amount of time, and “Alicat” to control the air flow directed towards the device under test. To aid the experiment recipe design, extra information was displayed for the user, such as defined instruments and API functions. Every scheduled experiment is pushed to a temporary stack, waiting for the

results to be reported back. The results can be reported back as a csv data recorder along with experiment metadata and logging information, all combined within a single research object. The RO analysis tool provides researchers with data curation features to track the history of previous experiments, apparatus, devices, and their relationship. An external plotting tool was used to transcribe the measured electrochemical sensor data from the csv record into a cyclic voltammetry plot, both shown in Fig. 4.20 below.

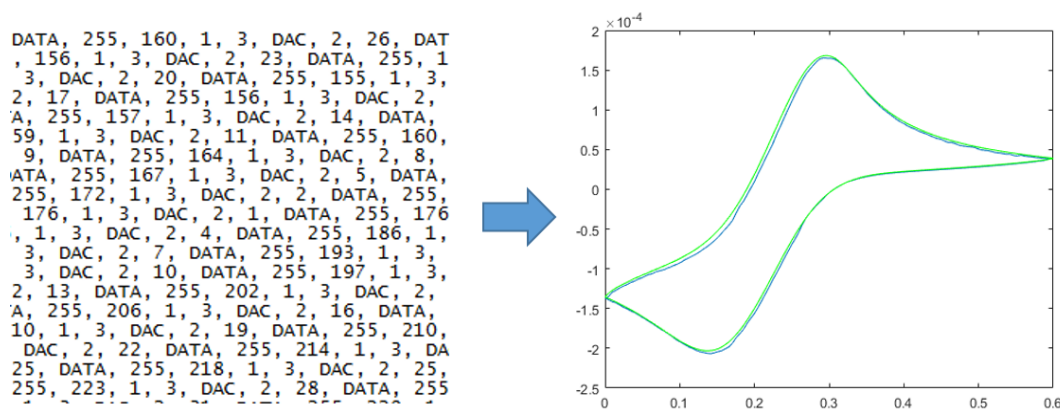


Fig. 4. 20 Cyclic voltammetry measurements, data outputted from the Analyzer tool as a simple csv file (left) and plotted using a MATLAB script (right).

A second example of a recipe was prepared to continuously record humidity using the environmental sensor (RHTS) and read measurement data from the FLUKE digital multimeter.

The results of this experiments and real-time dataset are shown in Fig. 4.21 below.



Fig. 4. 21 Snapshot of the Analyzer browser tool displaying run time error and debug logs, timestamped raw data logs, and plots of temperature, humidity, and digital multimeter measurement data.

Finally, the analyze tool provides a querying interface to search for previously executed experiments for viewing, annotating, and copying as a template for a new design and investigating ROs as shown in Fig. 4.22 below.

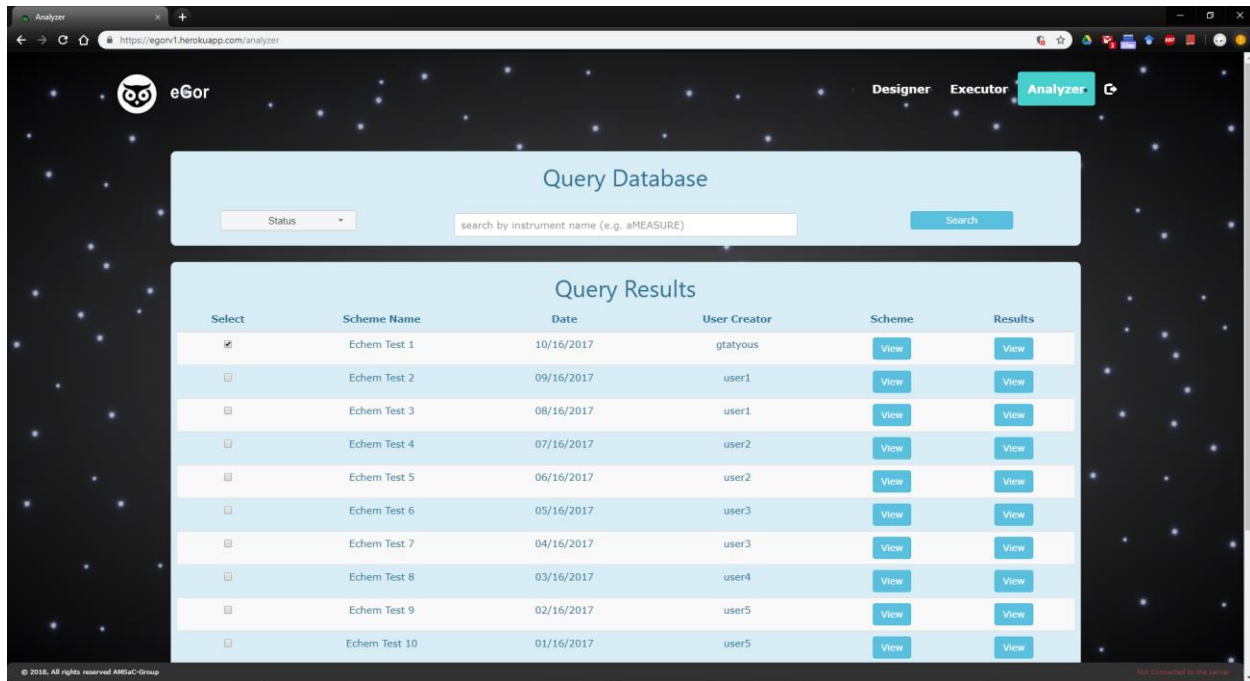


Fig. 4. 22 Snapshot of the Analyzer browser tool displaying ROs residing in the database with filtering options.

### 4.3 Platform Deployment in the Cloud

The platform was deployed using open source cloud providers such as Heroku [66] and mLab [67]. Heroku is a cloud platform as a service (PaaS) supporting several programming languages. Heroku, one of the first cloud platforms, supports Java, Node.js, Scala, Clojure, Python, PHP, and Go. This platform backend and REST API is implemented in Node.js and Heroku provides simple virtual buckets to run Node applications in the cloud. As of this writing, the platform is currently deployed on the domain <https://egorv1.herokuapp.com/>. The platform can also be deployed locally and privately as illustrated in Fig. 4.23. The user can access the locally deployed platform by navigating to the URL <https://localhost:3000>.

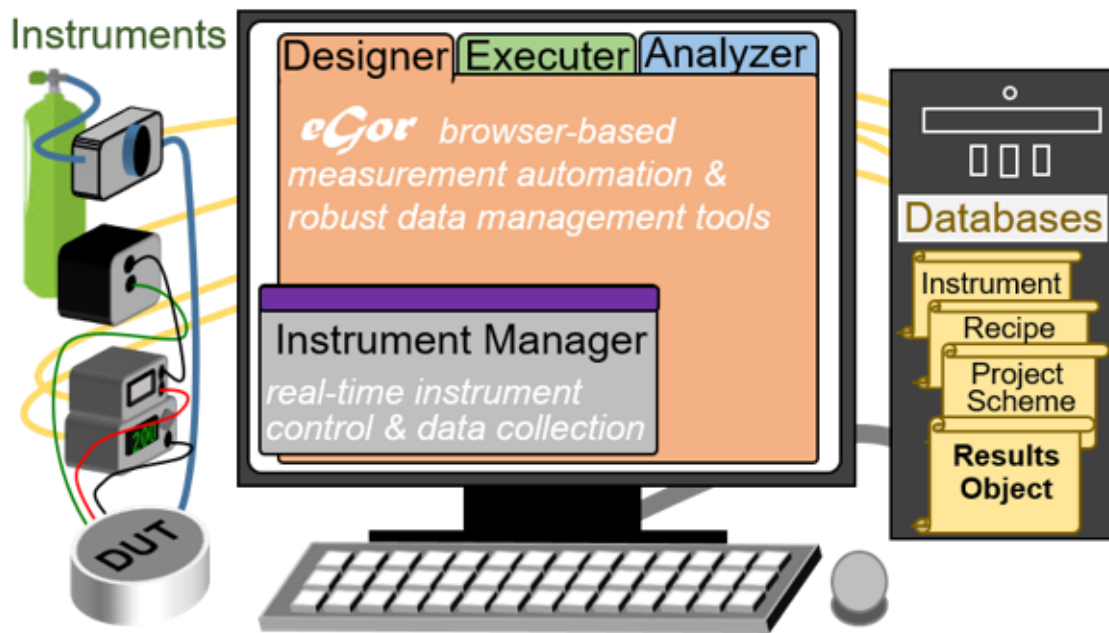


Fig. 4. 23 Local deployment of the platform for debugging and privacy.

mLab is a fully managed cloud database service that hosts MongoDB databases. mLab runs on cloud providers Amazon, Google, and Microsoft Azure, and has partnered with platform-as-a-service providers. Fig. 4.24 shows the MongoDB database deployed using mLab server. The collections tab shows the platform database architecture design described in Chapter 3. Each collection is a bundle of multiple document-based JSON objects.

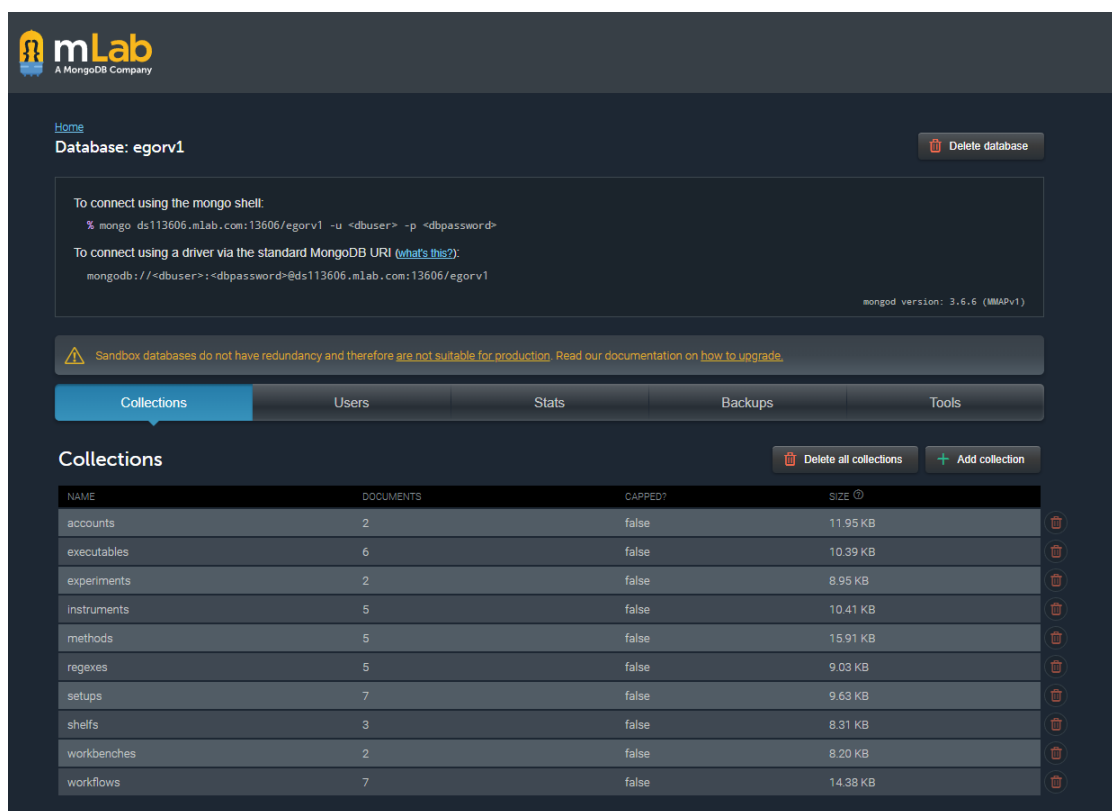


Fig. 4. 24 Snapshot of the mLab cloud platform deploying the proposed eGor:AutoX platform database.

#### 4.4 Table of the Platform Features vs Existing Tools

It is important to compare the features of the designed eGor:AutoX platform with the existing software solutions discussed in Chapter 2. Below is a table that includes all the essential components needed to create such a versatile platform for enhancing research reproducibility and knowledge discover. The designed platform is compared against LabVIEW, ELNS (such as Jupiter Notebook), and scientific workflow tools (e.g. Apache Taverna). The first two columns list the most important features necessary for each stage of scientific method from design and execution to instrumentation, data analysis and archiving. The X marker in each column indicates that the corresponding software tool has the listed features (or in some cases it indicates that it is achievable by obtaining another third-party plugin and extensions).



Table 1 Proposed Platform Features vs. Existing Software Tools

Features		LabVIEW	ELNs	Scientific Workflow Tools	This Platform
Designer	Capture experimental setup	X			X
	Capture experimental temporal procedures (e.g. stimulus signals and their parameters)				X
	Digital procedures creation	X			X
	Remote digital procedures creation				X
	Sharable repeatable measurement plan	X			X
	Traceable procedure provenance				X
	Worthwhile time domain procedure interface that eliminates the need for programming				X
Executer	Digital experiment execution	X			
	Remote digital experiment execution				X
	Future scheduling of experiment execution				X
	Remote real-time data monitoring	with an add-on LabVIEW web UI			X

Table 1 (cont'd)

Instrument Manager	Workflow automation and data acquisition	X			X
	User customizable instrument drivers				X
Analyzer	Data visualization and filtering	X	X	X	X
	Data processing and mining			X	X
	User annotations, tags and metadata		With an add-on Electronic lab notebook	X	X
	Sharable repeatable analysis workflow			X	X
Database	Data storage	with an add-on LabVIEW Cloud Toolkit for AWS	X		X
	Automatic data/metadata curation and integration				X
	Data and measurement plan provenance		X		X
	Sharable repeatable RO				X

## 4.5 Summary

This chapter presented the eGor:AutoX platform implementation results in the form of screenshots for each stage of the experimental process. The platform user experience overviews

and shows the welcome screen and the login/signup screen that the user first sees. Subsequently, the experiment setup is discussed and the three web-based GUI tools: Designer, Executer, and Analyzer tools were presented to show the user experience from defining an experiment recipe and setup to executing an experiment using and monitoring realtime datasets. The analyzer tool shows a table of successfully complete experiments in the form of ROs. This table also provides the user with filtering options to search among a large amount of data. The presented digital lab assistant platform was compared to existing solutions to show the advantages of the proposed eGor:AutoX platform. The following Chapter will summarize the thesis work and explain the contributions made by this work.

## CHAPTER 5

### CONCLUSION

#### 5.1 Summary

This thesis described the first digital assistant platform for facilitating research reproducibility and promoting knowledge discovery. The presented approach of tackling the issues facing experimental research reproducibility and knowledge discovery methodologies provides high-level capabilities for remotely controlling lab equipment and routing captured sensor data through a vision of connecting research labs to enable IoT applications. In addition, the proposed eGor:AutoX platform has capabilities of archiving information rich datasets to track measurement results, recipes, input parameters, instruments, and processed datasets. The design architecture and implementation choices of the tools comprising the digital assistant platform were described, and the literature review of relevant theories as well as existing technologies and software solutions was extensively covered. The proposed digital lab assistant platform was validated by constructing a dynamic recipe for characterization of an electrochemical gas sensor in the Designer tool, then running the recipe through the Executer tool, which initiated real-time autonomous data acquisition using the IM tool, and finally viewing resulting logs, data, and plots in the Analyzer tool. The platform was also validated through running other measurements to monitor the temperature and humidity of the test environment using commercial sensors and instruments. The eGor:AutoX platform provides accessibility across disciplines through its intuitive simple web-based graphical user interface and can dramatically improve multi-disciplinary collaboration through generation and sharing of a reproducible digital research objects (ROs) that associate all the related resources involved in any test measurement projects.

## 5.2 Contributions

The proposed platform was designed and implemented due to the lack of essential features and concepts with the existing software solutions. This thesis takes the approach of combining all three technologies of IoT, cloud computing, and SaaS to resolve the issues obstructing instrumentation and test measurement automation to achieve reproducible research and promote knowledge discovery methodologies. Specifically, the contributions of this work are described in the following:

1. The **first available platform to completely automate and curate the entire measurement process**, allowing users to remotely interface with equipment and the data that they generate and enabling collaborate environment with the proper security measures
2. The design of a **new measurement model that facilitates digitalization of the entire measurement recipe**, enabling experimental processes to be easily repeated, test procedures to be precisely replicated, and relationships between input parameters and output datasets to be thoroughly explored
3. The **formation of a novel portable digital identity structure of the research object (RO)** that eliminates obstacles facing research reproducibility by automatically generating datasets that are provenance-aware and trackable throughout their entire lifecycle promoting methodologies for checking systematic errors and finding unprecedented connections between datasets
4. The **development of a new intuitive user-friendly interface** that incorporates state of the art cloud-based software tools providing accessibility to other disciplines that may not have a programming background to use complex tools

### 5.3 Current Status and Future Work

The proposed software approach should allow the user to define test procedures and components through a user-friendly access point. Specific experimental definitions can be saved for later use for reproducibility. The user should then be able to remotely execute multiple tests on any connected physical experiment workbench in a real lab environment. Once the test is complete, the proposed platform would capture an organized and metadata-rich model that includes the raw test data, detailed definition of the test setup, and all procedural elements of the executed test such as timings, test successions, device conditions, etc. thus creating the database necessary to employ any knowledge discovery techniques. The generated model is stored for subsequent curating or data analysis, and any access or treatment of the results is automatically recorded to maintain data provenance. The proposed approach would also allow stored models to be shared with collaborators or provided to any institution that would be interested in reproducing the same results. The proposed approach requires an autonomous mechanism for designing, executing, and analyzing experiments to be automatic employing computer control over experiment parameters, provenance-aware, tracking the history of datasets and user algorithm, user-friendly, providing researchers with an intuitive graphical user interface, and collaborative, allowing research communities to securely share resources and results to enhance research reproducibility, facilitate knowledge discovery methodologies, and mitigate institutional memory loss.

The vision elaborated in this thesis targets major implementation goals that has not yet been completely realized. At the time of this writing, the platform can be usable for automating measurements and collecting real-time data using several custom and commercial instruments. We realize that in order for this platform to be useful over multiple disciplines, more instruments must be added to become compatible with the proposed platform. This goal could be realized by

collecting possibly a hundred most common instruments used and developing device drivers so that the Instrument Manager (IM) tool could use to automate the instrument.

The web-based graphical user interface has the essential tools implemented and fully functional, this is especially true for the Designer and Executer tool. However, we recognize that some usability features have yet to be implemented especially with the Analyzer tool. To achieve this implementation goal, the platform's backend API and database needs to be further expanded to handle large amount of information and structure them well. This allows the Analyzer tool to query the database microservice for retrieving relevant user information and their experimental datasets. The infrastructure for dynamically connecting to microservices and communicating between them is completed and functional. Real-time remote interaction with devices via the web-based interface is available through the Executer tool.

## **BIBLIOGRAPHY**



## BIBLIOGRAPHY

- [1] Leonard P. Freedman, Iain M. Cockburn, and Timothy S. Simcoe, The economics of reproducibility in preclinical research, *PLOS Biology* 13 (2015), no. 6, 1–9.
- [2] Fayyad U. (1997) Knowledge discovery in databases: An overview. In: Lavrač N., Džeroski S. (eds) *Inductive Logic Programming. ILP 1997. Lecture Notes in Computer Science (Lecture Notes in Artificial Intelligence)*, vol 1297. Springer, Berlin, Heidelberg.
- [3] John W. Coffey and Robert R. Hoffman, Knowledge modeling for the preservation of institutional memory, *Journal of Knowledge Management* 7 (2003), no. 3, 38–52.
- [4] T. Oinn, et al. “Taverna: lessons in creating a workflow environment for the life sciences: Research Articles,” *Concurr. Comput. Pract. Exp.*, vol. 18, no. 10, pp. 1067–1100, Aug. 2006.
- [5] Roger D. Peng, Reproducible research in computational science, *Science* 334 (2011), no. 6060, 1226–1227.
- [6] J. Bohli, A. Skarmeta, M. Victoria Moreno, D. García and P. Langendörfer, "SMARTIE project: Secure IoT data management for smart cities," 2015 International Conference on Recent Advances in Internet of Things (RIoT), Singapore, 2015, pp. 1-6.
- [7] ME Williams, Electronic databases, *Science* 228 (1985), no. 4698, 445–450.
- [8] Mariscal, G., Marbán, Ó, & Fernández, C. (2010). A survey of data mining and knowledge discovery process models and methodologies. *The Knowledge Engineering Review*, 25(2), 137-166.
- [9] Gregory Piatetski and William Frawley, *Knowledge discovery in databases*, MIT Press, Cambridge, MA, USA, 1991.
- [10] Fayyad, Piatetsky-Shapiro, Smyth, "From Data Mining to Knowledge Discovery: An Overview", in Fayyad, Piatetsky-Shapiro, Smyth, Uthurusamy, *Advances in Knowledge Discovery and Data Mining*, AAAI Press / The MIT Press, Menlo Park, CA, 1996, pp.1-34.
- [11] Omar A. El Sawy, Glenn M. Gomes, and Manolete V. Gonzalez, Preserving institutional memory: The management of history as an organizational resource, *Academy of Management Proceedings* 1986 (1986), no. 1, 118–122.
- [12] Arie C. Glebbeek and Erik H. Bax, Is high employee turnover really harmful? an empirical test using company records, *Academy of Management Journal* 47 (2004), no. 2, 277–286.
- [13] Amrit Tiwana, *The knowledge management toolkit: Practical techniques for building a knowledge management system*, Prentice Hall PTR, Upper Saddle River, NJ, USA, 2000.
- [14] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, Marimuthu Palaniswami, Internet of Things (IoT): A vision, architectural elements, and future directions, *Future Generation Computer Systems*, Volume 29, Issue 7, 2013, Pages 1645-1660.

- [15] R. Buyya, C. S. Yeo and S. Venugopal, "Market-Oriented Cloud Computing: Vision, Hype, and Reality for Delivering IT Services as Computing Utilities," 2008 10th IEEE International Conference on High Performance Computing and Communications, Dalian, 2008, pp. 5-13.
- [16] M. Godse and S. Mulik, "An Approach for Selecting Software-as-a-Service (SaaS) Product," 2009 IEEE International Conference on Cloud Computing, Bangalore, 2009, pp. 155-158.
- [17] K. R. Jackson et al., "Performance Analysis of High Performance Computing Applications on the Amazon Web Services Cloud," 2010 IEEE Second International Conference on Cloud Computing Technology and Science, Indianapolis, IN, 2010, pp. 159-168.
- [18] C. Kotas, T. Naughton and N. Imam, "A comparison of Amazon Web Services and Microsoft Azure cloud platforms for high performance computing," 2018 IEEE International Conference on Consumer Electronics (ICCE), Las Vegas, NV, 2018, pp. 1-4.
- [19] Puneet Agarwal, Continuous scrum: Agile management of saas products, Proceedings of the 4th India Software Engineering Conference (New York, NY, USA), ISEC '11, ACM, 2011, pp. 51–60.
- [20] Charles H. Schwepker Jr.. (2015) Influencing the salesforce through perceived ethical leadership: the role of salesforce socialization and person–organization fit on salesperson ethics and performance. *Journal of Personal Selling & Sales Management* 35:4, pages 292-313.
- [21] Oecd (2015). Frascati Manual 2015. The Measurement of Scientific, Technological and Innovation Activities.
- [22] Creswell, J. W. (2008). *Educational Research: Planning, conducting, and evaluating quantitative and qualitative research* (3rd ed.). Upper Saddle River: Pearson.
- [23] Skeptic, T. (2019). Critical Attributes Which Distinguish the Scientific Method. [online] The Ethical Skeptic. Available at: <https://theethicalskeptic.com/2018/03/31/the-scientific-method-contrasted-with-the-experimental-method/> [Accessed 8 Mar. 2019].
- [24] Eric von Hippel, The dominant role of users in the scientific instrument innovation process, *Research Policy* 5 (1976), no. 3, 212 – 239.
- [25] Bechhofer, S.; Bechhofer, S.; De Roure, D.; Gamble, M.; Goble, C.; Buchan, I. (2010). "Research Objects: Towards Exchange and Reuse of Digital Knowledge". *Nature Preceding*
- [26] A. Lefebvre, M. Spruit and W. Omta, "Towards reusability of computational experiments: Capturing and sharing Research Objects from knowledge discovery processes," 2015 7th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K), Lisbon, 2015, pp. 456-462.
- [27] Belhajjame, Khalid; Zhao, Jun; Garijo, Daniel; Gamble, Matthew; Hettne, Kristina; Palma, Raul; Mina, Eleni; Corcho, Oscar; Gómez-Pérez, José Manuel; Bechhofer, Sean; Klyne, Graham; Goble, Carole (2015). "Using a suite of ontologies for preserving workflow-centric research objects". *Journal of Web Semantics*. 32: 16–42.
- [28] Juan Gorraiz, David Melero-Fuentes, Christian Gumpenberger, and Juan-Carlos ValderramaZurin, Availability of digital object identifiers (dois) in web of science and scopus, *Journal of Informetrics* 10 (2016), no. 1, 98 – 109.

- [29] Laurel L. HAAK, Martin FENNER, Laura PAGLIONE, Ed PENTZ, and Howard RATNER, Orcid: a system to uniquely identify researchers, *Learned Publishing* 25, no. 4, 259–264.
- [30] S. Subashini and V. Kavitha, A survey on security issues in service delivery models of cloud computing, *Journal of Network and Computer Applications* 34 (2011), no. 1, 1 – 11.
- [31] R. Moreno-Vozmediano, R. S. Montero and I. M. Llorente, "IaaS Cloud Architecture: From Virtualized Datacenters to Federated Cloud Infrastructures," in *Computer*, vol. 45, no. 12, pp. 65-72, Dec. 2012.
- [32] Baldini I. et al. (2017) Serverless Computing: Current Trends and Open Problems. In: Chaudhary S., Somani G., Buyya R. (eds) *Research Advances in Cloud Computing*. Springer, Singapore.
- [33] C. Pahl, "Containerization and the PaaS Cloud," in *IEEE Cloud Computing*, vol. 2, no. 3, pp. 24-31, May-June 2015.
- [34] La H.J., Kim S.D. (2009) A Systematic Process for Developing High Quality SaaS Cloud Services. In: Jaatun M.G., Zhao G., Rong C. (eds) *Cloud Computing. CloudCom 2009. Lecture Notes in Computer Science*, vol 5931. Springer, Berlin, Heidelberg.
- [35] M. Bertocco, F. Ferraris, C. Offelli and M. Parvis, "A client-server architecture for distributed measurement systems," in *IEEE Transactions on Instrumentation and Measurement*, vol. 47, no. 5, pp. 1143-1148, Oct. 1998.
- [36] A. Balalaie, A. Heydarnoori and P. Jamshidi, "Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture," in *IEEE Software*, vol. 33, no. 3, pp. 42-52, May-June 2016.
- [37] P. Griffiths Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price, Access path selection in a relational database management system, *Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data (New York, NY, USA)*, SIGMOD '79, ACM, 1979, pp. 23–34.
- [38] Y. Li and S. Manoharan, "A performance comparison of SQL and NoSQL databases," 2013 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM), Victoria, BC, 2013, pp. 15-19.
- [39] C. Ireland, D. Bowers, M. Newton and K. Waugh, "A Classification of Object-Relational Impedance Mismatch," 2009 First International Conference on Advances in Databases, Knowledge, and Data Applications, Gosier, 2009, pp. 36-43.
- [40] Simon Holmes, *Getting mean with mongo, express, angular, and node*, 1st ed., Manning Publications Co., Greenwich, CT, USA, 2015.
- [41] N. Leavitt, "Will NoSQL Databases Live Up to Their Promise?," in *Computer*, vol. 43, no. 2, pp. 12-14, Feb. 2010.
- [42] C. Elliott, V. Vijayakumar, W. Zink, and R. Hansen, "National Instruments LabVIEW: A Programming Environment for Laboratory Automation and Measurement," *J. Assoc. Lab. Autom.*, vol. 12, no. 1, pp. 17–24, Feb. 2007.

- [43] V. Curcin and M. Ghanem, "Scientific workflow systems - can one size fit all?," 2008 Cairo International Biomedical Engineering Conference, Cairo, 2008, pp. 1-9.
- [44] Lysakowski, Rich, and Leslie Doyle. "Electronic Lab Notebooks: Paving the Way of the Future in R&D." *ARMA Records Management Quarterly* 32.2 (1998): 23-30. ProQuest. Web. 16 Apr. 2019.
- [45] Winter, H.H. & Mours, M. *Rheol Acta* (2006) 45: 331. <https://doi.org/10.1007/s00397-005-0041-7>.
- [46] Ewa Deelman, Dennis Gannon, Matthew Shields, and Ian Taylor, Workflows and e-science: An overview of workflow system features and capabilities, *Future Generation Computer Systems* 25 (2009), no. 5, 528 – 540.
- [47] Deelman E. et al. (2004) Pegasus: Mapping Scientific Workflows onto the Grid. In: Dikaiakos M.D. (eds) *Grid Computing. AxGrids 2004. Lecture Notes in Computer Science*, vol 3165. Springer, Berlin, Heidelberg.
- [48] Hai Nguyen, David A Case, Alexander S Rose, NGLview—interactive molecular graphics for Jupyter notebooks, *Bioinformatics*, Volume 34, Issue 7, 01 April 2018, Pages 1241–1242, <https://doi.org/10.1093/bioinformatics/btx789>
- [49] Boling, Charles Samuel, Mason, Andrew J., A collaborative software toolchain for automatic collection and comparative analysis of sensor characterization data, 2016, <https://doi.org/10.25335/M5VB08>
- [50] Interchangeable Virtual Instruments (IVI) Foundation, "Standard Commands for Programmable Instruments (SCPI)," May. 1999.
- [51] Zainab Aljazzaf, Bootstrapping quality of web services, *Journal of King Saud University - Computer and Information Sciences* 27 (2015), no. 3, 323 – 333.
- [52] <http://visjs.org/>
- [53] Lekha Nair, Sujala Shetty, and Siddhant Shetty, Interactive visual analytics on big data: Tableau vs d3.js, *Journal of e-Learning and Knowledge Society* 12 (2016), no. 4.
- [54] Mardan A. (2014) Boosting Your Node.js Data with the Mongoose ORM Library. In: *Practical Node.js*. Apress, Berkeley, CA.
- [55] C. Anderson, "Docker [Software engineering]," in *IEEE Software*, vol. 32, no. 3, pp. 102-c3, May-June 2015.
- [56] M. Godse and S. Mulik, "An Approach for Selecting Software-as-a-Service (SaaS) Product," 2009 IEEE International Conference on Cloud Computing, Bangalore, 2009, pp. 155-158.
- [57] V. Gašpar and R. Andoga, "Remote real-time monitoring of a small turbojet engine," 2016 IEEE 17th International Symposium on Computational Intelligence and Informatics (CINTI), Budapest, 2016, pp. 000359-000362.
- [58] John D. Blischak, Emily R. Davenport, and Greg Wilson, A quick introduction to version control with git and github, *PLOS Computational Biology* 12 (2016), no. 1, 1–18.

- [59] J. Highsmith and A. Cockburn, "Agile software development: the business of innovation," in *Computer*, vol. 34, no. 9, pp. 120-127, Sept. 2001.
- [60] H. Li, X. Mu, Y. Yang, and A. J. Mason, "Low Power Multimode Electrochemical Gas Sensor Array System for Wearable Health and Safety Monitoring," *IEEE Sens. J.*, vol. 14, no. 10, pp. 3391–3399, Oct. 2014.
- [61] Sina Parsnejad, Yaoxing Hu, Hao Wan, Ehsan Ashoori and Andrew J. Mason, "Wide Dynamic Range Multi-Channel Electrochemical Instrument for In-Field Measurements," in *IEEE Sensors*, 2016.
- [62] S. Parsnejad, Y. Gtat, T. Lin, X. Liu, P. B. Lillehoj and A. J. Mason, "Self-Ranging Thumb-sized Multichannel Electrochemical Instrument for Global Wearable Point-of-Care Sensing," 2018 IEEE 61st International Midwest Symposium on Circuits and Systems (MWSCAS), Windsor, ON, Canada, 2018, pp. 57-60.
- [63] Y. Gtat, S. Dávila-Montero and A. J. Mason, "Live Demonstration: Sensor Automation Platform and Multi-sensor Badge for the Sensory Impaired," 2018 IEEE Biomedical Circuits and Systems Conference (BioCAS), Cleveland, OH, 2018, pp. 1-1.
- [64] Y. Gtat, S. Parsnejad and A. J. Mason, "Live demonstration: Automated data acquisition and digital curation platform for enhancing research precision, productivity and reproducibility," 2017 IEEE International Symposium on Circuits and Systems (ISCAS), Baltimore, MD, 2017, pp. 1-1.
- [65] Y. Gtat and A. J. Mason, "Live demonstration: Enhancing biomedical research precision, productivity and reproducibility via autonomous data acquisition and robust data curation," 2017 IEEE Biomedical Circuits and Systems Conference (BioCAS), Torino, Italy, 2017, pp. 1-1.
- [66] B. Lee, E. K. Dewi and M. F. Wajdi, "Data security in cloud computing using AES under HEROKU cloud," 2018 27th Wireless and Optical Communication Conference (WOCC), Hualien, 2018, pp. 1-5.
- [67] Agarwal, S. & Rajan, K.S. *Spat. Inf. Res.* (2016) 24: 671. <https://doi.org/10.1007/s41324-016-0059-1>.