Enabling Autonomous Electron Microscopy for Networked Computation and Steering

Anees Al-Najjar, Nageswara S. V. Rao, Ramanan Sankaran
Maxim Ziatdinov, Debangshu Mukherjee, Olga Ovchinnikova
Computational Sciences and Engineering Division
Oak Ridge National Laboratory
Oak Ridge, TN, USA
{alnajjaram,raons,sankaranr,ziatdinovma,mukherjeed,ovchinnikovo}@ornl.gov

Kevin Roccapriore,
Andrew R. Lupini, Sergei V. Kalinin
Center for Nanophase Materials Science
Oak Ridge National Laboratory
Oak Ridge, TN, USA
{roccapriorkm,arl1000,kalininsv}@ornl.gov

***Abstract*—Advanced electron microscopy workflows require an ecosystem of microscope instruments and computing systems possibly located at different sites to conduct remotely steered and automated experiments. Current workflow executions involve manual operations for steering and measurement tasks, which are typically performed from control workstations co-located with microscopes; consequently, their operational tempo and effectiveness are limited. We propose an approach based on separate data and control channels for such an ecosystem of Scanning Transmission Electron Microscopes (STEM) and computing systems, for which no general solutions presently exist, unlike the neutron and light source instruments. We demonstrate automated measurement transfers and remote steering of Nion STEM physical instruments over site networks. We propose a Virtual Infrastructure Twin (VIT) of this ecosystem, which is used to develop and test our steering software modules without requiring access to the physical instrument infrastructure. Additionally, we develop a VIT for a multiple laboratory scenario, which illustrates the applicability of this approach to ecosystems connected over wide-area networks, for the development and testing of software modules and their later field deployment.**

***Index Terms*—science workflows, scanning transmission electron microscope, virtual infrastructure twin, science instrument ecosystem.**

I. INTRODUCTION

There is an increasing interest in scientific workflows that incorporate remotely controlled, automated experiments over collections of physical instruments and computing systems. Often, these resources are located at geographically dispersed sites, and they need to be federated over wide-area networks to form *ecosystems* that seamlessly support these workflows [1], [2]. Recent workflow developments enable the use of Artificial Intelligence (AI) codes both as a part of scientific computations and data analyses, and for orchestrating automated experiments at potentially remote physical instruments. In particular, the latter tasks may involve configuring instruments, collecting and transferring measurements and analyzing them to extract parameters for the next set of remote experiments. Effective execution of such workflows requires the application codes to be customized for remote computing and storage resources connected over networks. Currently, science users manually orchestrate several of these tasks, which may have to be repeated with different parameters based on analyses results. These human-driven, time-consuming processes limit the scalability and execution tempo of scientific workflows, and often lead to inefficient idling of expensive resources.

The electron microscopy workflows are expected to significantly benefit from the computing and storage capabilities provided by these ecosystems [3], [4]; general versions of such science ecosystems may utilize diverse instruments, for example, light sources [5]. We consider an ecosystem of Scanning Transmission Electron Microscopes (STEM) that are extensively used in science workflows [6], for example, ptychography using atomic imaging for novel materials synthesis. More generally, the transmission electron microscope with electron imaging, electron diffraction, and spectroscopy capabilities is aptly called "A Synchrotron in a Microscope" [7] and has extensive uses in physical and life sciences [8]. Currently, no general software frameworks exist to build these microscopy ecosystems, unlike others such as the Experimental Physics and Industrial Control System (EPICS) [9] widely deployed at neutron and light source facilities. The microscopes are typically controlled by local control computers using custom Windows software, and the computing systems are typically Linux platforms located in different networks separated by firewalls. To fully realize the potential of STEM ecosystems, *eSolutions* for software design, testing and implementation, are needed for seamlessly collecting and transferring measurements and steering the microscopes from remote computing systems, as illustrated in Figure 1.

We consider Nion STEM systems at Oak Ridge National Laboratory (ORNL) and remote Linux computing systems

This research is sponsored in part by the INTERSECT Initiative as part of the Laboratory Directed Research and Development Program and in part by RAMSES project of Advanced Scientific Computing Research program, U.S. Department of Energy, and in part by the Office of Basic Energy Sciences, Division of Materials Sciences and Engineering, U.S. Department of Energy, and is performed at Oak Ridge National Laboratory managed by UT-Battelle, LLC for U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

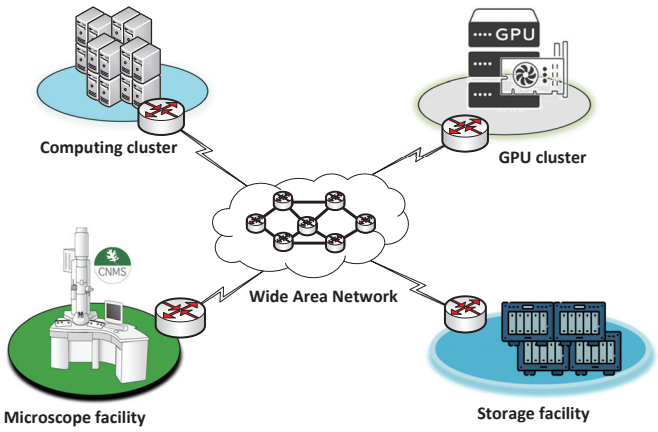

Fig. 1: An ecosystem of microscope, computing and storage sites connected over a wide-area network is needed to seamlessly support science workflows that require automated and remotely controlled experiments.

with GPUs to support measurements and steering, in addition to computations that use the measurements. Our contributions are two-fold: (a) demonstration of remote steering and automated measurement transfers over the ORNL STEM ecosystem (more than two decades after some of these microscopes were installed), and (b) Virtual Infrastructure Twins (VIT) that support the development and testing of ecosystem software prior to and in preparation for deployment. In prior work, the measurement transfers required manual steps, and the steering was only available from the control computer co-located with the instrument. For part (a), the measurements are made available at remote Linux workstations using a Network Attached Storage (NAS) at the instrument site, and remote mounting its file system on the servers. We develop Pyro server and client codes to enable remote microscope commands to be sent and executed on the microscope using the Nion Swift microscope software. For part (b), we developed VITs of ORNL and multi-site ecosystems with network control channels between remote computing systems and microscope systems, and tested initial Pyro codes prior to deployment(in part (a)).

The development and testing of the ecosystem software modules (such as Pyro server client codes) typically takes several days or longer. It is not cost-effective to require physical access to the microscope for the entire duration, and indeed may not be necessary. Instead, we develop a VIT of the ORNL ecosystem that emulates the network and computing systems, and incorporates the Nion Swift simulator. It provides the software environment nearly identical to the deployed system, and is used to develop and test Pyro server and client codes, without tying up an expensive microscope and its human operator for several days. We also develop another VIT of four laboratory sites connected over a wide-area network, and demonstrate the remote steering capability across the sites. It illustrates the broader applicability of the VIT approach for (initial) development of ecosystem software components without requiring access to the physical infrastructure.

The organization of this paper is as follows. A brief account of STEM and the associated scientific workflows and ORNL infrastructure are presented in Section II. Design and implementation of data and control channels are presented in Section III. Experimental results over ORNL STEM ecosystem are presented in IV. VITs of ORNL site and four-sites scenario are described in Section V. Conclusions and directions for future research are presented in Section VI.

II. STEM Workflows and Infrastructure

Scientific STEM workflows are supported by an infrastructure of instruments and their control computers typically connected over local networks.

A. STEM Systems, Workflows and Software

Scientific workflows that utilize STEM are quite varied and extensive.

1) Principles and applications: The STEM images are generated by scanning a focused electron beam across a thin sample. The distribution of transmitted (and/or scattered) electrons in the detector plane depends on the sample composition and structure. Hence, the variation in detected intensity across the formed image can provide valuable insights into a material's local properties. The most common STEM measurement is annular dark-field imaging where the image intensity varies as approximately $Z^{1.7}$, with Z being the atomic number. The generated images contain a wealth of information about the material structure and, if atomically-resolved, allow among other things, for mapping local polarization fields, identifying topological defects, and studying charge-density wave formation. By using configurations with parallel detectors, the Z-contrast structure imaging can be combined with spectroscopic measurement – such as electron energy loss spectroscopy (EELS) or energy-dispersive X-ray spectroscopy - to probe materials' electronic functionality. Particularly, EELS can be used for probing the physics of collective excitations in nanoscale systems, as well as precise chemistry characterization. For example, STEM-EELS enables studies of local effects in plasmonic systems which are critical to the design of nanostructures with desired optical properties. Finally, the recent advances in pixelated and multi-segmented detectors enable the acquisition of a diffraction pattern at each probe position, which constitutes the collection of techniques known as 4D-STEM. As a result, insights can be gained into the structure of electric and magnetic fields at the atomic scale, which in turn enable the study of chemistry of individual atomic defects, and mapping of interlayer spacing in quasi-2D materials (to name a few examples).

2) Microscopy Workflows: A typical experimental study in STEM-EELS and 4D-STEM proceeds as follows. First, an annular dark-field scan over a relatively large field of view is acquired. Then, the regions for spectral or diffraction imaging, either single-point spectroscopy or a grid of points, are manually selected based on operator's intuition. The acquired data is usually stored at local resources or commercial cloud (e.g., Dropbox, Google Drive, etc.) and

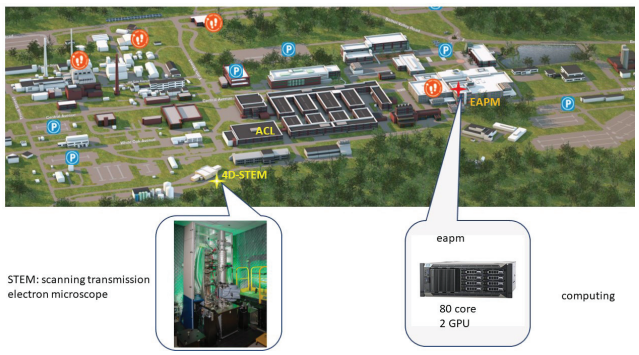

Fig. 2: The STEMs and computing workstations are in two ORNL facilities at separate physical sites and are connected to different site networks separated by firewalls. They include U100 and U200 Nion Microscopes and NAS systems at AML of CNMS, and eapm computing system with 80 CPU cores and two GPUs at K200 facility.

analyzed after the experiment is completed. The standard post-acquisition analysis of the hyper-spectral EELS data is performed via linear unmixing/decomposition techniques such as non-negative matrix factorization. For 4D-STEM data, the processing is based on advanced analyses involving physics-based inversion. Recently, authors in [10], [11] demonstrated an approach for an 'intelligent' probing of dissimilar structural elements to discover in the automated fashion a desired physical functionality in STEM-EELS and 4D-STEM experiments. This approach utilizes deep kernel learning (DKL) to inform the next measurement by continuously learning a relationship between the local structure visualized via the dark-field STEM image and EEL spectra or 4D-STEM diffraction patterns. However, it is currently limited to relatively small data volumes as the DKL model training and inference are performed using local or on-board computational resources. The remote computations and steering of the microscope based on analyses of measurements, by automated codes or manual operations, will contribute to the effectiveness of these workflows, and eSolutions that enable them are our main goal of this paper.

3) Nion Swift Software: Here we focus on the Nion Swift software, which is open source, can be run on multiple OS platforms, and provides access to almost every aspect of the microscope. Note that other software platforms exist for different electron microscopes. The STEM microscopes considered here are controlled via Nion Swift software that provides a Graphical User Interface (GUI) and a python-based API. It is installed and executed inside a Python virtual environment on the control computer, which is typically a Windows workstation co-located with the instrument. Its API provides instrument commands executed in a python console for measurement collection, microscope positioning and other tasks [12]. A STEM instrument simulator, called *nionswift-tool* [13], is also provided as an open source software package that provides an off-line Swift environment identical to the physical installation.

B. ORNL STEM and Servers

The Nion STEM microscopes U100 and U200 of ORNL's Center for Nanophase Materials Science (CNMS) are located at the Advanced Microscopy Laboratory (AML) site. The computing systems, including a server with 80 CPU cores and 2 GPUs, called **eapm**, are located in a data center facility at a different site, as shown in Figure 2. These facilities are serviced by different site networks which are separated by firewalls. A Nion microscope with an attached AS2 controller is controlled by the instrument control computer running the Swift software; they are connected over a local hub network which is not routed to other site networks. The scientists conduct the microscopy experiments typically using the Swift GUI accessible via the control computer which is dual-homed to connect to a hub and site networks under strict firewall configurations. Once the measurements are collected, they may be transferred to a NAS system connected to the microscopes and the control computer over the local hub network. In current workflows, the measurement collection and positioning commands are manually executed using Swift software, and data may be directly transferred to a local NAS.

C. Related Microscope and Instrument Control System

SerialEM [14] is an electron microscope controller software enriched with complement applications for image acquisition, (pre)processing, display, buffering and file saving. It provides tools for supporting the essential microscopy operations, like tilt series, 3D reconstruction and single-particle reconstructions. SerialEM is equipped with GUI, Python APIs as well as built-in script commands to support the data acquisition and processing. The controller supports a wide range of electron microscopes and Complementary Metal Oxide Semiconductor/Charge Coupled Device (CMOS/CCD)-based cameras. The Nion microscopes considered in this paper are not supported by SerialEM, but our Pyro-based solutions are in principle implementable using Python APIs.

EPICS [9] is an open-source toolkit used for distributed control of scientific instruments at experimental facilities, for example, Spallation Neutron Source (SNS) at ORNL and Advanced Photon Source (APS) at Argonne National Laboratory (ANL). It supports application interfaces and networking protocols for hardware components of instruments, such as sensors, motors, detectors, and magnets. It provides interfaces to Input/Output Controllers (IOCs) and Process Variables (PVs) using command lines (e. g., caget and caput to read and write PV values) or graphical interfaces. It also provides networking and interfaces access, known by Channel Access (CA) and Client CA (CAC), allowing science users to access, collect measurements and supply configuration parameters for targets used in science workflows.

Tango Controls [15] is an open-source framework that manages a variety of systems and hardware types. It is applicable to different systems, including Distributed Control Systems (DCS), Integrated Control Systems (ICS) and Supervisory Control And Data Acquisition (SCADA) systems, for instance,

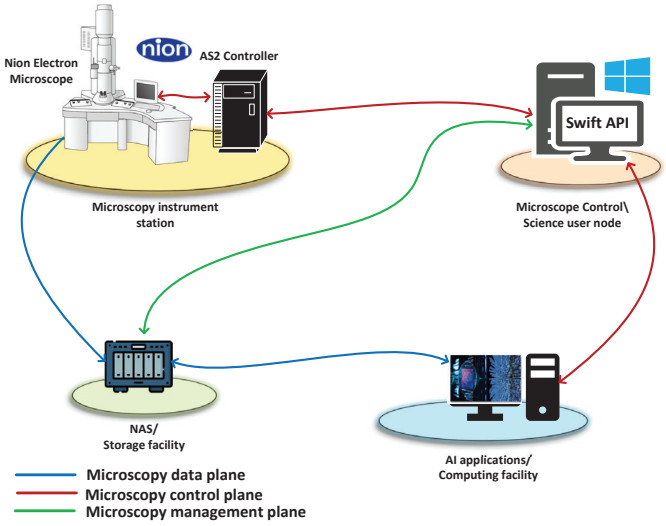

Fig. 3: Separate end-to-end control and data channels utilize Pyro client-servers and NAS, respectively, to support microscope steering and measurements collection needed for remote, automated experiments.

Machine to Machine (M2M), Internet of Things (IoT), Industrial IoT (IIoT) applications, as well as national experimental facilities like synchrotrons. Tango provides classes for different instrument hardware, called the hardware classes, and it is scalable to build and integrate new hardware classes. Tango also supports a range of programming languages, such as C++, Java and Python, for developing, controlling, and building hardware classes and framework software modules. Furthermore, various deployment modes are implemented with Tango for achieving autonomous and scalable control, for example, in-situ and distributed deployments, as well as remote steering functionalities via client-server or web client deployments. The Nion microscopes at ORNL are not supported by EPICS or Tango controls, and consequently our solutions are primarily designed for their Swift software.

III. STEM Ecosystem Design and Implementation

The implementation of a STEM ecosystem over an infrastructure, such as ORNL, requires the development, integration, and/or implementation of network and system configurations as well as new and available software modules.

A. System Challenges and Solution Approach

A STEM ecosystem needed to support remote autonomous experiments must address the following challenges:

- *Local and remote access:* Microscopes are manually operated using custom software installed on control computers co-located with instruments, which are typically connected only to local hub networks. The access is needed to both their hardware and software over networks that are protected by access controls and firewalls.
- *OS and software:* Microscope software is typically proprietary and runs on Windows OS, and the measurements are stored on the local computer, and the storage systems often use custom mechanisms and formats. Both control and data access from remote Linux servers may be required, which entails interfacing different OSs, including programming environments, file and data formats, and host firewalls and access mechanisms.
- *Networking:* Network connections to control computers carry both measurements and control traffic, typically, over the same IP path and network interfaces. Over long network connections, this non-separation can potentially lead to the loss of instrument control when large measurement transfers occupy the entire available bandwidth. Suitable end-to-end network channels and mechanisms are needed between the microscope control computers and remote servers.

We propose a design based on separate end-to-end channels for control and data that are serviced by software modules that communicate across Windows and Linux OS. The control channels enable the scientists and automated codes to remotely access the microscope control computer and execute steering commands (Section III-B). The microscope measurements are collected on the NAS and made available on remote computing systems. Figure 3 shows this design for ORNL ecosystem for the infrastructure described in previous section. We configure the NAS to export its file system (Section III-C), thereby making it available for analyses codes that utilize powerful remote computing systems such as servers with multiple GPUs.

B. Control Channel

A control channel is used to remotely access the instrument control computer for sending control commands and parameters to steer the microscope experiments. Upon command execution, the control computer may send back the results or other data. We developed Pyro client-server codes to support remote steering of microscope experiments across the ecosystem. Pyro provides a Python API for network access [16], and we installed it on the control computer's Swift virtual environment and on the remote computing systems. Figure 4 illustrates the developed solution of Pyro client-server communication across the ecosystem.

Our Pyro server embeds python objects corresponding to STEM experiment tasks and makes them accessible and executable over control channels that span the ecosystem. These task codes are developed by us using Swift instrument commands [12] by utilizing APIs executable only under the control computer's Swift python environment. As shown in Figure 4, the developed python object *Embedded_Swift_Server* is a python class with multiple functions that encapsulate the STEM tasks. The Pyro daemon turns the python object into a Pyro object (*Swift_Server*) and publishes it via the control node's IP address and a TCP port to be accessible over the control channel. The Pyro server application is called from the *File→Scripts* option in Swift GUI, and initiates as a background process which listens for incoming requests to execute microscope tasks using API commands.

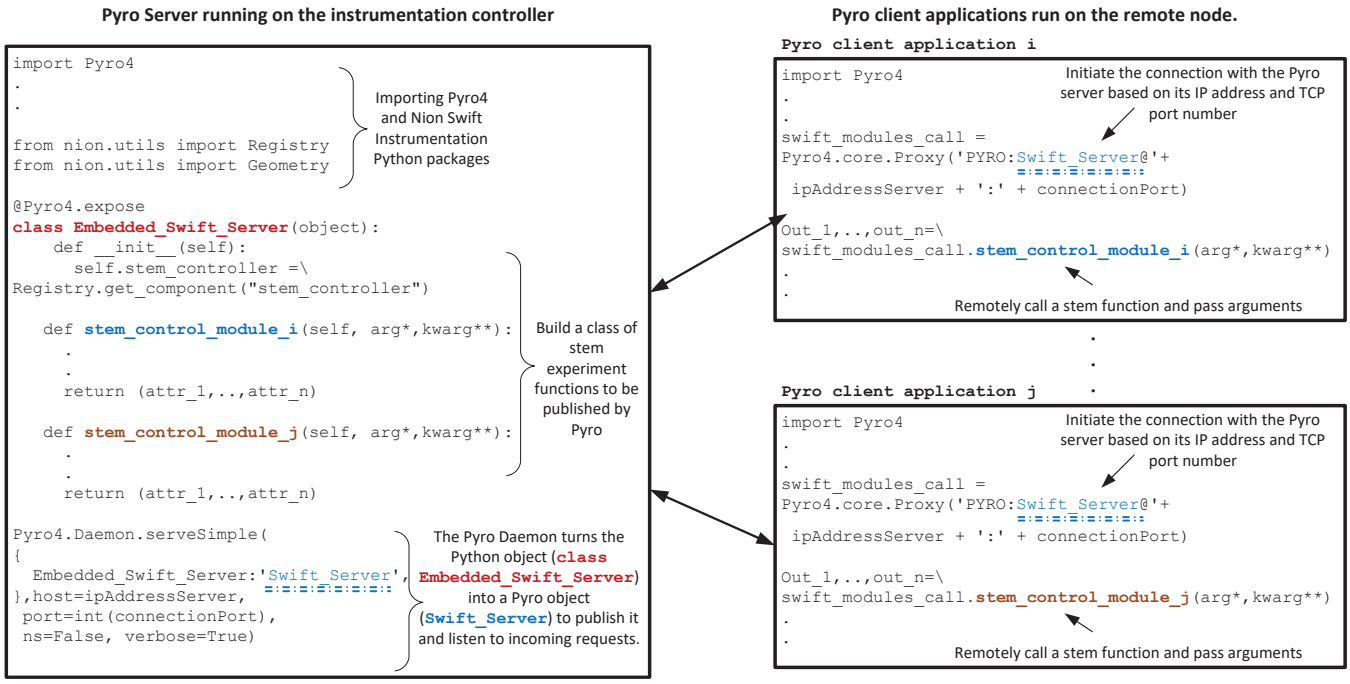

Fig. 4: Pyro client-server communication applied between the instrument control node and a compute system.

Pyro client applications can be executed concurrently on multiple remote computing systems across the ecosystem. The client applications communicate with the Pyro server to execute the exposed functions on the control computer, as shown in Figure 4. A client initiates a connection with the server using a Pyro object URI, which is a resource identifier in the format: **PYRO:objectid@IP:TCP port**. The **objectid** refers to the Pyro object published by Pyro daemon running on the control computer, which is *swift_Server* in our implementation shown in Figure 4, and **IP** is the network address of the control computer. The **TCP port** specifies the communication port between the Pyro client and server applications that allows them to communicate and exchange control messages. In our implementation, we developed a client application for each STEM experiment task exposed by the Pyro server. The client applications are called from python console or embedded in automated scripts, for example, using Jupyter notebooks, by passing the IP address of the control node and control commands and parameters required to steer the microscopy experiments. Overall, the approach of wrapping the microscope Swift APIs using Pyro client-servers enables this solution to scale and be portable across multiple STEMs and computing systems across the ecosystem.

We developed several microscope task modules, namely *scan_status* to get the current scan status, *scan_channel* to obtain measurements from a particular microscope channel, and *probe_position* to position the beam at specified coordinates. These modules are implemented as functions of the Pyro server (details in Appendix A). They are paired with the corresponding Pyro client modules, *check_scan.py*, *scan_channel.py*, and *probe_position.py*, respectively. These client modules pass the user or machine-driven control commands to the Pyro server to execute microscope tasks.

The concept and codes for Pyro servers and clients are developed without requiring access to the physical infrastructure by using the VIT of ORNL ecosystem with Nion Swift simulator, as described in Section V. These steps took several days, and once matured, the codes were tested and demonstrated over ORNL STEM ecosystem which required the physical access and an operator for the microscope to ensure safety, as described in Section IV.

C. Data Channel

The data channel makes the measurements available at the NAS connected to the control computer and also at the remote servers of the ecosystem, up on the execution of commands either locally or remotely over the control channel. The Swift software is configured to store the measurements on the Windows-based NAS as files. We implement the data channel by remote mounting the NAS data files using Samba and Common Internet File System (CIFS) file sharing, which provides access across different operating systems, in particular, Linux servers. The Windows-based NAS files are natively available on the Linux-based servers to perform scientific analysis and computations. The access privileges to the computing nodes and Samba/CIFS file sharing are configured to allow the users to access the NAS files across the ecosystem. This access via one-time setup makes it persistent across the ecosystem. Indeed, the cross-facility mounting of microscopy data automates large-scale data transfer and makes it available for the computations across the ecosystem. File transfer tools such as GridFTP [17] or Globus [18] applications

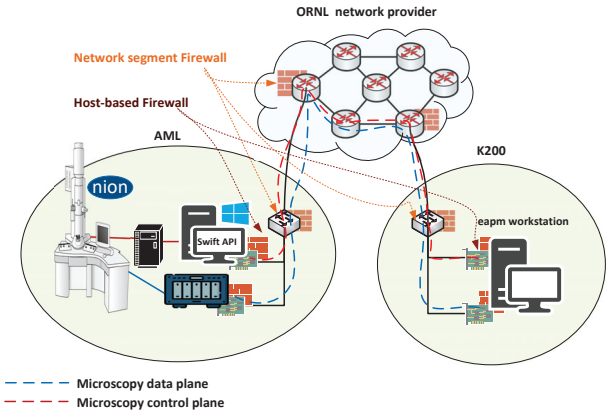

Fig. 5: Control and data channels are enabled by configuring the network and host firewalls, and are separated by two NICs on computing server.

require additional hardware and/or software, including licenses and credentials, which may become difficult to manage by the scientists. The Pyro communication is not used for large measurement transfers since they are limited by memory size and they can also generate cross traffic that can potentially limit the control traffic.

The data channel transparently transfers high-volumes of microscopy measurements over network connections that are separate from control channel connections, thereby mitigating the impact of large measurement transfers on the steering operations, as described in next section.

D. Networks, Access and Firewalls

The microscope computers and remote servers are located at different networks separated by network firewalls and Windows and Linux host firewalls, as shown in Figure 5. Both the data and control channels are enabled by inserting various firewall rules to support file mounting between the NAS and computing servers over the data channel, and open communication ports for Pyro servers and clients over the control channel. Their rules are inserted both at the firewalls separating these two networks, and also on the Windows and Linux hosts. Two physical interfaces are configured on the compute servers with different IP addresses to separate the control channel traffic between Pyro server and clients, and remote mounting the NAS CIFS file system. The end points of both control and data channels are two separate NICs on the same compute servers but their other ends points are on separate systems, namely, the control computer and NAS.

IV. Experimental Setup and Demonstration

The ecosystem capabilities described in the previous section are implemented on U100 and U200 microscopes at the AML science facility, and on the eapm workstation at the K200 computing facility, which are both parts of ORNL physical infrastructure. The experimental setup shown in Figure 6 is used for remote steering and measurement transfer operations

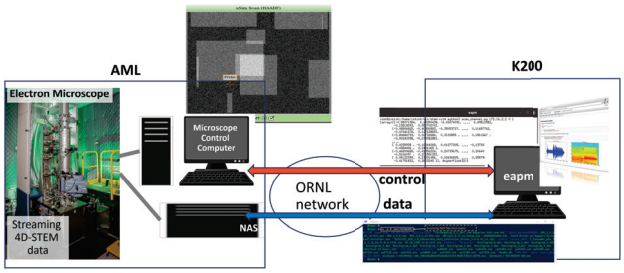

Fig. 6: Data and control channels between STEMs at AML and computing servers in K200 in ORNL infrastructure.

carried out between the microscope and its NAS system and on the eapm at K200.

A. Steering experiments over ORNL ecosystem

We have successfully tested the microscopy control channel between the U200 microscope and eapm computing system by steering the microscope experiments over the physical infrastructure of ORNL ecosystem (Figure 2).

We integrated a number of Python functions for STEM APIs to obtain the beam status and other microscope parameters as well as to position the beam. In particular, the functions *scan_status* and *probe_position* (explained in Appendix A-A and A-C) are the corresponding Pyro server objects on the U200 control computer. The microscope is steered by *check_scan.py* and *probe_position.py* Pyro client applications running on eapm.

On the U200 control computer, the response to cross-facility communication with eapm system is shown in the screen shot in Figure 7. The Pyro server's response to the scan_status commands from Pyro client on eapm is shown in the console, as two True and False responses which are also sent back to eapm. The response to probe_position command is shown in console as the previous probe position ($x = 0.5$ and $y = 0.5$) and the new probing position at $x = 0.2$ and $y = 0.8$ from Pyro client. The resultant new probe position is depicted on the right side window of Figure 7.

On the eapm computing system, the output of Pyro client applications for executing remote microscope tasks is shown in Figure 8. The scan status of the U200 microscope, denoted by the IP address (160.91.156.73), is frequently checked using *check_scan.py* application. The scan status is True while running a physical scan on the control node via Swift GUI, and when the scan is completed its status becomes False. Also, a new scan position is sent to the microscope using *probe_position.py* application with coordinates ($x = 0.2$ and $y = 0.8$) to which the Pyro server responded by positioning the microscope.

These Pyro codes are initially developed without requiring physical access using VIT as described in Section V, and are readily executed on ORNL infrastructure.

B. Automated Data Transfers

We deployed and tested the proposed data channel over ORNL physical infrastructure by remote mounting the mi-

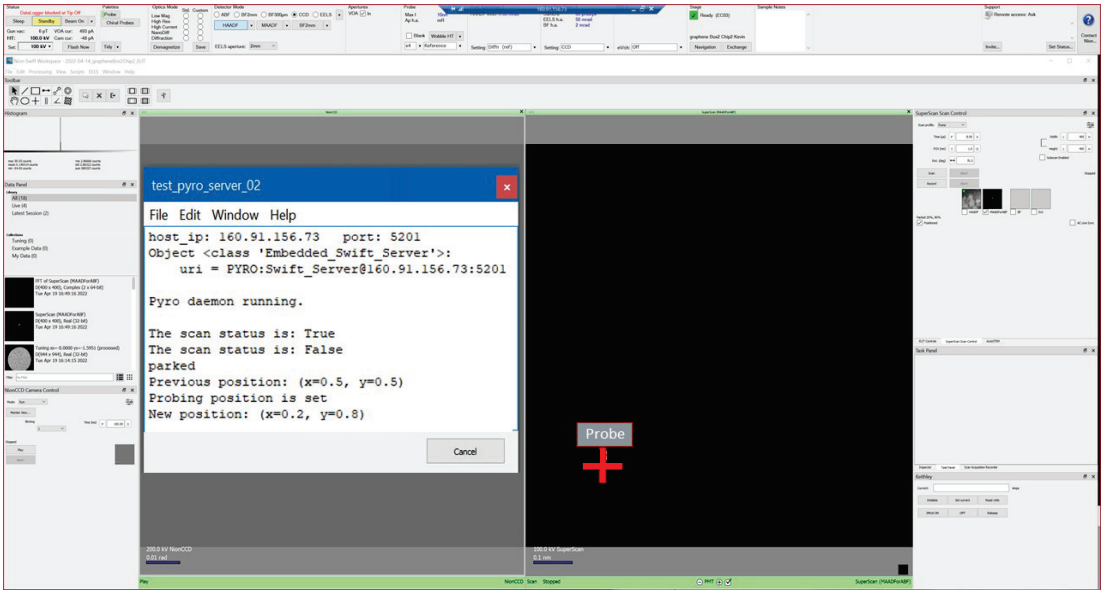

Fig. 7: Screenshot of Swift GUI running at U200 microscope control compute at AML, ORNL. The Pyro server output corresponds to the execution of Pyro client commands from eapm computing system located at K200 and the resultant positioning of microscope probe.

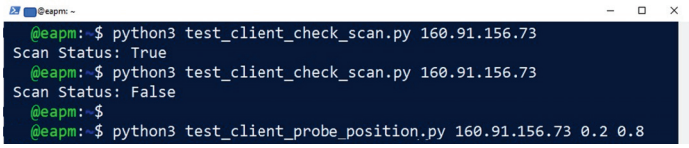

Fig. 8: Output of Pyro client on eapm in checking scan status and probing position on U200 over ORNL physical infrastructure.

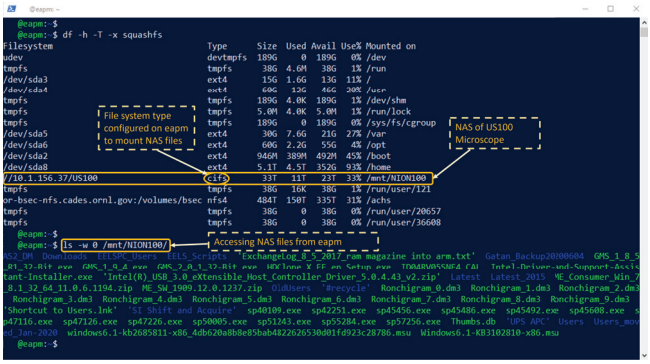

Fig. 9: NAS mounted on eapm for automatic data transfer.

croscope measurements directory of NAS devices on eapm. For example, the US100 NAS system, accessed by IP address 10.1.156.37, is mounted to eapm using CIFS file system. The authorized microscopists and automated codes seamlessly access NAS files at the mounted directory **/mnt/NION100** and utilize them in computations on the computing system. A screenshot in Figure 9 shows these U100 measurement files on NAS being available on eapm.

V. STEM Virtual Infrastructure Twins

The development and implementation of a STEM ecosystem requires various software components and network and system configurations to be designed and tested. These workflows that utilize networked computations and instruments present challenges that are not typically faced in (pure) computing ecosystems. It is too expensive and potentially disruptive for the whole STEM ecosystem to be available during the entire development and testing period, particularly, in early stages. More generally, the ecosystems, such as a multi-site STEM complex, may not be available while they are being designed and developed, and indeed, may not always be needed, for example, instrument time is not usually necessary to debug network code. In this context, we propose employing a STEM VIT (S-VIT) as an enabling software tool to be used prior to the field deployment. In particular, ORNL S-VIT (OS-VIT) is used to develop the Pyro steering codes described in previous sections, and an additional Multi-site S-VIT (MS-VIT) is developed in this section to show their applicability to ecosystems that span multiple sites connected over wide-area networks.

A. VIT: Concept and Design

VIT emulates the network and computing components of the ecosystem, and incorporates instrument software simulators, such as Nion Swift simulator in S-VIT. It provides a software environment nearly identical to the physical ecosystem to support early and continual development, testing and design space explorations. It is implemented using mininet [19] by using virtual hosts to execute and test scientific applications (including microscopy applications and simulations), and to emulate the network infrastructure using virtual switches and

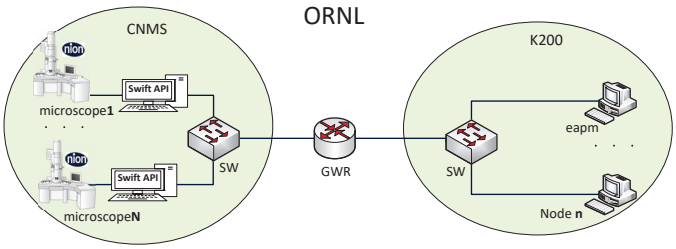

Fig. 10: OS-VIT: Emulation of ecosystem of ORNL STEM infrastructure.

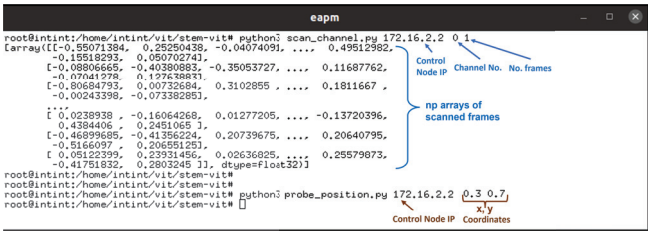

Fig. 12: Running Pyro client applications on eapm node at K200. The applications represent scanning a microscope channel and probe certain position after the scan is complete.

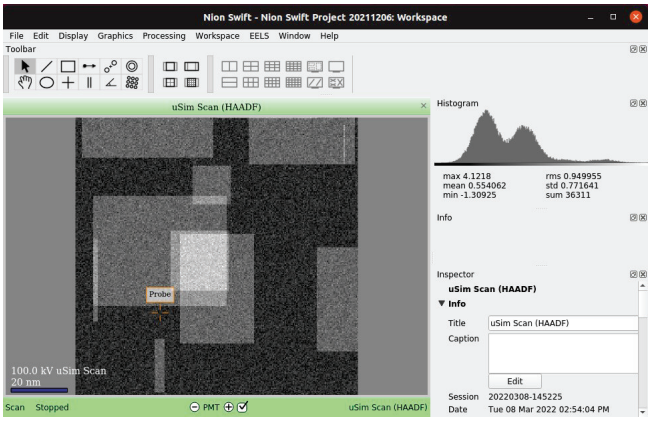

Fig. 11: Swift GUI runs on microscope1 node at CNMS.

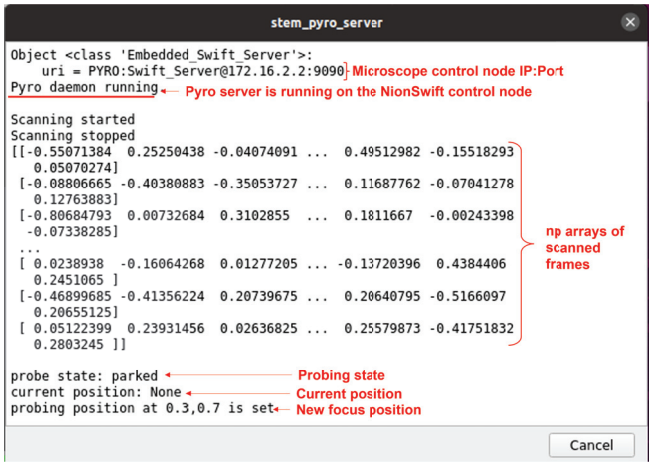

Fig. 13: Script window at nionswift-tools runs at microscope1/CNMS shows running Pyro server exposing STEM experiments across the emulated ORNL infrastructure and logs the status of the execution.

routers. The emulated components communicate over virtual links that reflect the physical network infrastructure links. VIT is packaged as a portable virtual machine to facilitate the development process among scientific communities. Once the VIT-based solutions are suitably tested, they are ready to be field-deployed and integrated into the physical infrastructure. Previous VIT implementations address other scenarios including software defined networking [20], EPICS [21] and federation software stack [22].

B. ORNL STEM Ecosystem: OS-VIT

The OS-VIT for developing and testing STEM workflows across the ORNL infrastructure is shown in Figure 10. It consists of two facilities, K200 computing facility and CNMS microscope facility. It subsumes the physical infrastructure used in experiments in Section IV, and provides additional systems. The K200 facility emulation provides multiple virtual hosts (including one for eapm) that are used for computations and remote access. The simulation of the CNMS facility includes multiple microscope control computers (related to U100 and U200 at AML) that support simulated STEM experiments. The control computers are emulated using virtual hosts that run *nionswift-tool* simulator that includes Swift GUI. The facilities' devices are connected via virtual switches, which in turn, connect the two facilities via a Gateway Router (GWR).

The OS-VIT provides remote access from eapm at K200 to microscope1 control computer at CNMS to steer STEM experiments, send back the data for potential use in computations and change the microscope focus position (using *scan_channel* and *probe_position* Pyro STEM tasks). The workflow tasks described in Section IV are first carried out on OS-VIT.

First, the Swift API is run on the microscope1 node to load the Swift GUI, as shown in Figure 11. Next, the Pyro server module is loaded on Swift GUI and run as a daemon that waits for Pyro client communications across the ecosystem. Figure 12 shows the output of Pyro client applications on eapm: *scan_channel.py* is executed to scan channel number zero and gather data related to one frame at the control node, which is sent as a NumPy array to eapm for computations.

Once the channel scan finishes, the microscopist would be able to reconstruct and analyze the scanned data, including for particular positions. For example, The microscopists at eapm can change the focus position and collect new measurements. We demonstrated changing the probe position via executing the *probe_position.py* client application with new coordinates $x = 0.3$ and $y = 0.7$ that are sent as parameters of *probe_position* to microscope1. The effect of changing the focus position is depicted on the updated image in Figure 11. The STEM experiments' status is interactively shown on the Script window at Swift API shown in Figure 13, including the generated data from executing channel_scanning and

probe_position tasks.

C. Multi-site STEM Ecosystem: MS-VIT

In addition to reflecting a specific infrastructure (as by OS-VIT), VITs can be developed for more general purposes such as assessing new designs, building and testing more complex ecosystems that support more sophisticated scientific workflows. In particular, we consider ecosystems of multiple sites, each with several computing facilities and scientific instruments. MS-VIT is an emulation of an infrastructure of multiple Department of Energy (DOE) sites shown in Figure 14. This infrastructure consists of four DOE labs, namely ORNL, Brookhaven National Laboratory (BNL), Argonne National Laboratory (ANL), and The National Energy Research Scientific Computing Center (NERSC). These sites consist of computing systems and may include scientific facilities, such as CNMS at ORNL. Virtual hosts (denoted by COMP) are incorporated as part of the ecosystem to execute scientific computations and STEM applications. For example, the Swift simulator (nionswift-tool) is run on a virtual host at CNMS representing the instrument control node that simulates Nion STEM experiments responses. The ecosystem's site networks are connected to gateway routers, which are in turn, connected to edge routers of the wide-area network, namely, ESnet [23]; the latter is emulated as a set of virtual routers with dedicated site-to-site connections. Details of VITs used for multi-site federation implementation and a science use case using EPICS simulator, are described in [24] and [25], respectively.

We utilize MS-VIT to explore the feasibility of coordinating concurrent operations of an electron microscope by multiple STEM workflows across the multi-site ecosystem. In this scenario, we demonstrate having two microscopy workflows, namely BNL-ORNL WF and NERSC-ORNL WF, that are running concurrently. These workflows remotely access and execute the STEM experiments that are part of the Pyro object running on **stem_control_node** at CNMS/ORNL. Such concurrent workflows are feasible independent of which (user's) sample is currently loaded because the Nion Microscope is equipped with a magazine system capable of carrying multiple samples that can be dynamically loaded during the experiments. The BNL-ORNL WF remotely steers the microscope experiments from the **bnlcomp** compute node at the BNL while the other workflow steers the microscope from the **nersccomp** compute node at NERSC. The STEM tasks incorporated in this scenario are *scan_channel* and *scan_status*.

Different science users initialize these workflows at NERSC and BNL sites and both communicate with **stem_control_node**. The scenario works as follows. A science user at the NERSC site initiates NERSC-ORNL WF that includes executing *scan_channel.py* on the **nersccomp** node, as shown Figure 15. Concurrently, another science user at BNL is checking the scanning status of the Nion microscope, whether it is available to launch the BNL-ORNL WF. Figure 16 shows output of *check_scan.py* on the **bnlcomp** node. The results show **True** scanning status while NERSC-ORNL WF is running, and upon its completion scan status is **False** which is an indication of the availability of microscope at CNMS/ORNL.

The MS-VIT reflects the current facilities, and can be extended to include those that are currently being built or being designed for possible future deployments. Overall, the motivation for developing such multi-site ecosystem VITs range from developing solutions for certain current scientific instruments without requiring their physical access to future ecosystem designs.

VI. Conclusions and Future Work

We presented eSolutions, both field-deployments and enabling virtual twins, to support the development and testing of remote experiment capabilities for science workflows over microscope ecosystems. Together, they enabled the development of control and data channel implementations for a production STEM ecosystem, and also proof-of-principle demonstrations of a wide applicability of this VIT approach to multi-site microscope ecosystems. The overall approach is applicable to other science scenarios, such as automated, remotely controlled chemistry and materials experiments. In these cases, the custom software for instruments such as chromotographs, potentiostats, flow reactors, and others, may be leveraged to form ecosystems by wrapping their APIs using Pyro codes.

Future extensions of this work include GUIs that provide a comprehensive dashboard of the entire ecosystem for scientists and facility operators; performance assessment of the ecosystem including the data and control channels and workflows; and the development of production quality software stacks for establishing and operating ecosystems by building up on our experimental codes. It would be of future interest to develop similar solutions to other classes of microscopes by exposing their API or callable functions for network communications using Pyro modules, in particular, those supported by serialEM. Other future work areas include integration of machine-driven instrument control based on AI/ML methods into microscope ecosystems, and frameworks that support integrated workflows, for example, Jupyter notebooks that integrate instrument operations and computations based on measurements. It would of future interest to explore the applicability of the proposed solution based on parallel data and control channels to other instruments such as those supported by EPICS and Tango controls.

Appendix A
Pyro Modules for STEM Tasks Using API

This Appendix describes Nion microscope tasks developed as functions based on Swift API commands and exposed by Pyro server modules on the instrument control computer.

A. Scan status

This STEM task checks the scan status on the microscope. It is built as a function that returns *True* to the client as a Boolean variable if the microscope is occupied with scanning and *False* otherwise. The function is listed below.

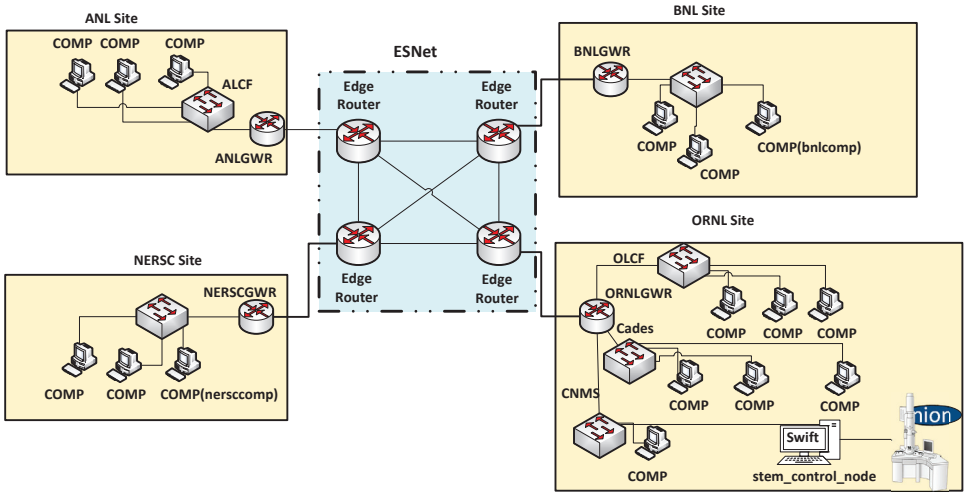

Fig. 14: MS-VIT: Emulation of STEM ecosystem of four DOE lab sites connected over a wide-area network.

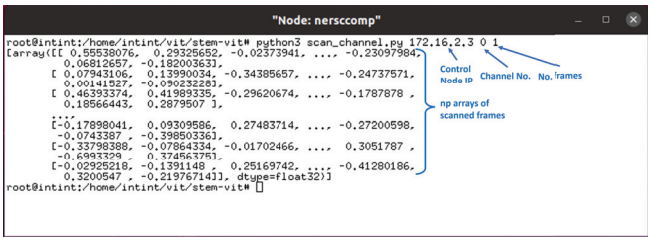

Fig. 15: Run a Pyro client module at a COMP node at NERSC site to scan a microscope channel.

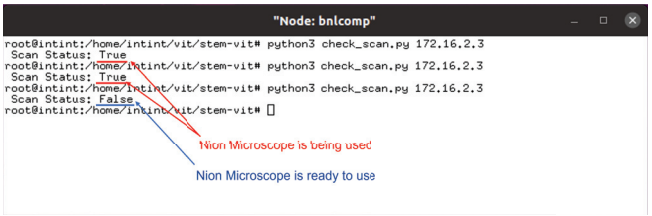

Fig. 16: Run a Pyro client module at a COMP node at BNL site to probe the scan status.

```
def scan_status(self):
1     scan = self.stem_controller.scan_controller
2     return scan.is_playing
```

B. Scan_channel

This STEM task function scans a microscope channel and transfers a number of frames of the data matrices corresponding to the scan. The function receives control parameters related to the channel number and number of frames from its peer Pyro client application, and returns arrays of the scanned frames in that channel. The code for scan_channel function is shown below.

```
def scan_channel(self, ch, num_frames):
1     ct=1 # constant time
2     scan = self.stem_controller.scan_controller
3     scan.set_enabled_channels([ch])
4     frame_parameters =\
      scan.get_current_frame_parameters()
5     frame_time =\
      scan.calculate_frame_time(frame_parameters)
6     scan.start_playing(frame_parameters)
7     time.sleep(frame_time * num_frames + ct)
8     frames_list = scan.grab_buffer(num_frames)
9     scan.stop_playing()
10    data_lst=\
      [frame[0].data for frame in frames_list]
11    return pickle.dumps(data_lst)
```

First, the scan is initialized and enabled to scan a channel **ch** (lines 2 and 3). Then the scan is triggered (line 6) and waited for a specific time to gather the frames (line 7). This time is calculated by multiplying the number of frames **num_frames** with the **frame time**. Here we added a constant time **ct** to ensure a proper scanning time is applied to the frames. After that, the frames are stored in **frames_list** (line 8), and the scan is stopped (line 9). Finally, the data of the scanned frames are extracted (line 10) and serialized to be wired back to the client (line 11).

When the client module receives the data, it is deserialized and stored in another data object to be ready for analysis. The serializing and deserializing processes are performed using the pickle python package [26].

C. Probe_position

This task sets the probing position for the scanned data using the coordinates provided by the associated Pyro client application. The function for this task is explained below.

```
def probe_position(self, x_coor, y_coor):
1     print(f'probe state:\
       {self.stem_controller.probe_state}')
2     print(f'current position:\
       {self.stem_controller.probe_position}')
3     if x_coor == 0.0 == y_coor == 0.0:
4        value = None
5     else:
6        value = Geometry.FloatPoint\
         (y=y_coor,x=x_coor)
7     self.stem_controller.probe_position = value
```

The task provides the status of the current probing state and the position (lines 1 and 2) before changing the focus to the new position (lines 3-7).

REFERENCES

[1] Y. Lu, M. Yang, F. Jiang, Z. Fu, Z. He, and G. Liu, *The Applications of the e-Science in National R&D Program for Major Research Instruments*, pp. 367–383. Springer Singapore, 2020.

[2] B. Enders, D. Bard, C. Snavely, L. Gerhardt, J. Lee, B. Totzke, K. Antypas, *et al.*, "Cross-facility science with the superfacility project at lbnl," in *2020 IEEE/ACM 2nd Annual Workshop on Extreme-scale Experiment-in-the-Loop Computing (XLOOP)*, pp. 1–7, 2020.

[3] S. R. Spurgeon, C. Ophus, L. Jones, A. Petford-Long, S. V. Kalinin, M. J. Olszta, *et al.*, "Towards data-driven next-generation transmission electron microscopy," *Nature Materials, vol.20, no.3, pp.274–379,2021*.

[4] M. Levental, R. Chard, K. Chard, I. Foster, and G. A. Wildenberg, "Ultrafast focus detection for automated microscopy," in *2021 IEEE 17th International Conference on eScience (eScience)*, pp. 237–238, 2021.

[5] T. Bicer, D. Gursoy, R. Kettimuthu, I. T. Foster, B. Ren, V. De Andrede, and F. De Carlo, "Real-time data analysis and autonomous steering of synchrotron light source experiments," in *2017 IEEE 13th International Conference on e-Science (e-Science)*, pp. 59–68, IEEE, 2017.

[6] C. Ophus, "Four-dimensional scanning transmission electron microscopy (4D-STEM): From scanning nanodiffraction to ptychography and beyond," *Microscopy and Microanalysis*, vol. 25, no. 3, pp. 563–582, 2019.

[7] L. M. Brown, "A synchrotron in a microscope," in *Electron Microscopy and Analysis 1997*, pp. 17–22, CRC Press, Jan 1997.

[8] E. J. Kirkland, *Advanced computing in electron microscopy*, vol. 12. Springer, 1998.

[9] "Experimental physics and industrial control system." epics.anl.gov.

[10] K. M. Roccapriore, S. V. Kalinin, and M. Ziatdinov, "Physics discovery in nanoplasmonic systems via autonomous experiments in scanning transmission electron microscopy," *arXiv:2108.03290*, 2021.

[11] K. M. Roccapriore, O. Dyck, M. P. Oxley, M. Ziatdinov, and S. V. Kalinin, "Automated experiment in 4d-stem: Exploring emergent physics and structural behaviors," *ACS Nano,vol.16,no.5*, pp. 7605–7614, 2022.

[12] "Nion Swift Instrumentation Guide." Available from: https://nionswift-instrumentation.readthedocs.io/_/downloads/en/latest/pdf/.

[13] "nionswift-tool." Available from: https://nionswift.readthedocs.io/en/stable/installation.html#installing-nion-swift-from-pypi-or-conda-forge.

[14] D. N. Mastronarde, "Automated electron microscope tomography using robust prediction of specimen movements," *Journal of structural biology*, vol. 152, no. 1, pp. 36–51, 2005.

[15] "Tango Controls." Available from: https://www.tango-controls.org/.

[16] "Pyro: Python Remote Objects." Available from: https://pyro4.readthedocs.io/en/stable/.

[17] "GT 4.0 GridFTP." http://www.globus.org.

[18] I. Foster and C. Kesselman, "Globus: A metacomputing infrastructure toolkit," *Intl J. Supercomputer Applications*, vol. 11, no. 2, pp. 115–128, 1997.

[19] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, p. 19, 2010.

[20] Q. Liu, N. S. V. Rao, S. Sen, B. W. Settlemyer, H. B. Chen, J. M. Boley, R. Kettimuthu, and D. Katramatos, "Virtual environment for testing software-defined networking solutions for scientific workflows," in *AI-Science'18: Proceedings of the 1st International Workshop on Autonomous Infrastructure for Science*, pp. 1–8, Jun. 2018.

[21] N. Rao, A. Al Najjar, I. Foster, Z. Liu, and R. Kettimuthu, "Virtual framework for science federations with instruments access and control," in *Workshop on Autonomous Discovery in Science and Engineering - Berkeley (Virtual), California, United States*.

[22] A. Al-Najjar, N. S. V. Rao, N. Imam, T. Naughton, S. Hitefield, L. Sorrillo, *et al.*, "VFSIE-Development and testing framework for federated science instruments," *arXiv preprint arXiv:2101.02184*, 2021.

[23] "Energy Sciences Network." http://www.es.net.

[24] A. Al-Najjar, N. S. Rao, N. Imam, T. Naughton, S. Hitefield, L. Sorrillo, *et al.*, "Virtual framework for development and testing of federation software stack," in *2021 IEEE 46th Conference on Local Computer Networks (LCN)*, pp. 323–326, IEEE, 2021.

[25] A. Al-Najjar, N. S. V. Rao, S. Hitefield, and T. Naughton, "Science federation emulation testbed: Demonstration of vfsie functionalities," in *Demonstrations of the 46th IEEE Conference on Local Computer Networks (LCN)*, IEEE, 2021.

[26] "pickle: Python object serialization." Available from: https://docs.python.org/3/library/pickle.html#module-pickle.