Towards a Software Development Framework for Interconnected Science Ecosystems

Seth Hitefield[1], Marshall McDonnell[1], Matthew Wolf[1], Lance Drane[1], Kevin Roccapriore[1], Maxim Ziatdinov[1], Jesse McGaha[1], Mark Abraham[1], Richard Archibald[1], Addi Malviya Thakur[1], and

[1] Oak Ridge National Laboratory, Oak Ridge, TN
{malviyaa, hitefieldsd, mcdonnellmt, wolfmd, archibaldrk, dranelt, roccapriorkm, ziatdinovma, mcgahajr, smithrw, hetrickjm, abrahamme, yakubovs, watsongr, chancebr, nguyencv, mintzbj}@ornl.gov
[2] Voltron Data, Mountain View, CA
matthew@voltrondata.com
[3] Roche Sequencing Solutions, Willmington, MA
jrobert.michael@gmail.com
[4] Pacific Northwest National Laboratory, Richland, WA
elke.arenholz@pnnl.gov

Abstract. The innovative science of the future must be multi-domain and interconnected to usher in the next generation of "self-driving" laboratories enabling consequential discoveries and transformative inventions. Such a disparate and interconnected ecosystem of scientific instruments will need to evolve using a system-of-systems (SoS) approach. The key to enabling application integration with such an SoS will be the use of Software Development Kits (SDKs). Currently, SDKs facilitate scientific research breakthroughs via algorithmic automation, databases and storage, optimization and structure, pervasive environmental monitoring, among others. However, existing SDKs lack instrument-interoperability and reusability capabilities, do not effectively work in an open federated architectural environment, and are largely isolated within silos of the respective scientific disciplines. Inspired by the scalable SoS framework, this work proposes the development of INTERSECT-SDK to provide a coherent environment for multi-domain scientific applications to benefit from the open federated architecture in an interconnected ecosystem of instruments. This approach will decompose functionality into loosely coupled software services for interoperability among several solutions that do not scale beyond a single domain and/or application. Furthermore, the proposed environment will allow operational and managerial inter-dependence while providing opportunities for the researchers to reuse software components from other domains and build universal solution libraries. We demonstrate this research for microscopy use-case, where we show how INTERSECT-SDK is developing the tools necessary to enable advanced scanning methods and accelerate scientific discovery.

Keywords: Autonomous Experiments, SDK, DevSecOps, Interconnected Science, Edge Computing, Research Infrastructure, Scientific Software, Scientific Workflows, Federated Instruments, Digital Twins

1 Introduction

The future of science is a highly connected ecosystem of instruments, sensors, devices, and computing resources at the edge combined with cloud computing and centralized high-performance-computing systems that execute machine learning infused workflows to enable self-driven, automated scientific experiments. DNA sequencing is a great example of how enabling automation in a workflow has had a significant effect in science. The initial sequencing of the human genome completed in 2001 was estimated to have cost over $100M and took over 10 years. With improvements to sequencing techniques and applying automation, the cost of sequencing a whole genome is now less than $10K and takes a few days or less. The impact this has had on scientific discovery and the medical community is profound.

The potential importance of interconnected science infrastructures has caused increased interest from many areas, and systems can go by many names such as: federated systems, cognitive digital twins, or (our preferred) interconnected science ecosystems. In this push toward interconnected science systems, much of the attention is focused on the runtime and deployment aspects. However, progress must be built on transitioning from ad hoc demonstrations to a reliable, reusable, verifiable, and scientifically reproducible infrastructure that addresses the hardware, software, and wetware components.

In this paper, we will present our designs and early progress for an integrated software development kit (SDK) geared specifically for the interconnected science ecosystem. The larger initiative at Oak Ridge National Laboratory (ORNL) for interconnected ecosystems goes by the name INTERSECT, and correspondingly we call our work the INTERSECT-SDK. The development process for INTERSECT-style applications and use-cases requires many iterations between hardware, software, and human process engineering, but the core data connections and processing all live within the software space, as does this work.

Scientists are limited in many ways by the current disjointed state of experimental resources. Scientific instruments produce massive amounts of data and managing this data is not trivial. Many systems are not directly connected with each other which further complicates data movement. This results in scientists collecting data from multiple runs using pre-determined experimental conditions, and spending weeks (if not months) analyzing collected data. If those experiments do not yield desirable results, the entire process may need to be restarted causing a long time-to-result. An interconnected ecosystem will enable more efficient scientific experimentation by integrating access to computing resources, allowing resource sharing, adopting new artificial intelligence (AI) control capabilities, and improving overall system utilization. This will enable self-driven, automated experiments which can significantly reduce the overall time-to-result.

The key contributions of this paper are three-fold: (a) the design and initial implementation of a user-centric SDK for interconnected science development; (b) an evaluation of how best practices for research software engineering are adapted to such development, and; (c) a concrete demonstration of using proposed SDK in a real-world application.

In the remainder of the paper, we start by grounding the discussion using this experience with interconnecting Scanning Transmission Electron Microscopy (STEM) instruments and edge computing to enable concurrent computational analysis (§ 2). We then discuss our development approach (§ 3) and the design (§ 4) of our SDK. The INTERSECT-SDK design, approach, and overall infrastructure build on many examples and lessons learned from some prior work we further detail in (§ 5) before we conclude with our analysis in (§ 6).

2 Motivating Use-case: Scanning Transmission Electron Microscope

Scanning Transmission Electron Microscopy (STEM) has long been an important technique for experimental science domains from material science to biology [26]. With the growth in computing power and data acquisition rates, microscopy has begun to study the data from Convergent Beam Electron Diffraction patterns (CBED) to greatly enhance the material properties that can be measured [25]. The combination of 2D CBED patterns, measured at the 2D spatial location of STEM produce what is known ad 4D–STEM data. The analysis of the data that comes out of the microscopes has long depended on computational analysis of the signals, but the goals for newer experimental design call for much more aggressive couplings between advanced simulation and artificial intelligence (AI) algorithms and the experimental plan. Recently, part of this team has used AI methods to develop algorithms for automatic experimental 4D–STEM acquisition [30]. A team at ORNL's Center for Nanophase Materials Sciences (CNMS), working with the larger INTERSECT community, has been working to develop a coupled platform between the microscopes and AI-guided experimental control systems to make scientific discovery faster and more repeatable.

Put simply, the goal of this project was to allow a user to put in a high-level goal for using the microscope to study a material sample, and the AI-guided process would automatically learn and guide adjustment of parameters to achieve those goals, accelerating analysis with GPU based "edge" systems. For example, a sample with a number of defects in it would be loaded in, and the experiment would be directed to find and characterize the material. The scientists would need to help define AI algorithms to identify the correct locations within a coarse-grained scan of the surface, target fine-grained examination of both 'normal' areas and the defects, and then prioritize the remaining list of potential targets against a heuristic metric of 'characterized enough'. This leverages previous work in both the electronic control of microscopes [33] and the use of AI characterization methods [30]. Incorporating this project into the interconnected INTERSECT ecosystem and distributing various components to accelerators allows the experiment operate in real-time and introduces new opportunities and deep complexities (Figure 1).

Beyond the science results, this 4D–STEM project also highlights some of the key difficulties in developing, sharing, and maintaining the infrastructures needed for science. As mentioned in the introduction, there are a laundry list of concerns

when trying to interconnect experimental and simulation or AI components. Understanding the data flow between them so that you capture not only the correct data interface types but also the frequencies and sizes is crucial during the development process. However, this can't generally be done by taking a facility off-line for months at a time so that the hardware and software links can be co-developed. Instead, the software development program needs to progress based on 'good enough' digital twins of the instrument that can respond to controls in typical ways and time frames, and that can generate data of appropriate sizes and complexity.

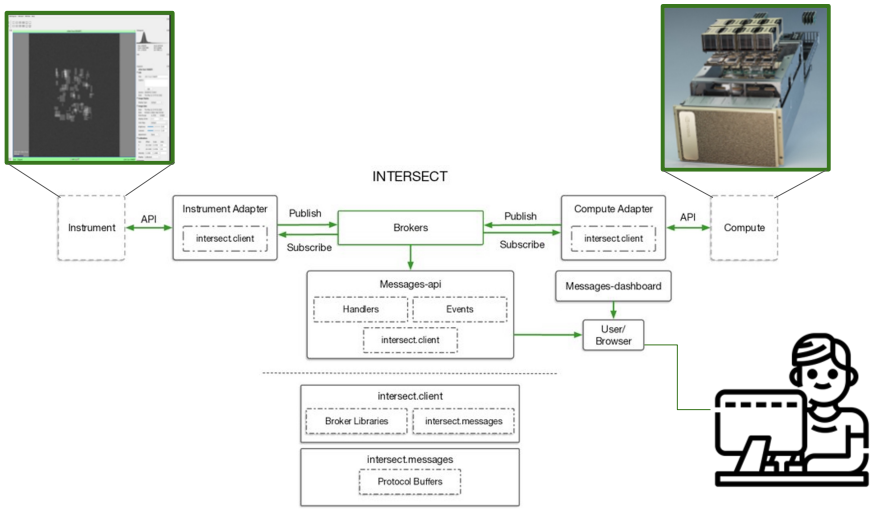


Fig. 1. SDK depiction of 4D–STEM use case in INTERSECT. The instrument control and analysis is distributed throughout the interconnected ecosystem allowing users to execute analysis remotely and take advantage of distributed accelerators.

Here, the INTERSECT 4D–STEM team helped the development process by providing expert knowledge about the Nion Swift control software and data processing pipeline for the STEM instruments [19, 24]. The synthetic data generation capabilities allow a developer to send control message to the digital twin using the exact same interfaces and response time scales as when running on a real microscope. So development can be tested, go through continuous integration, engage in test-driven development, and other similar strong research software engineering practices without the need for concurrent facility reservations. At the end of sprints, a much more limited reservation time could be utilized to assess any drift between the virtual and physical hardware behaviors.

In addition to cementing the central role of digital twins in the process of reliable development for interconnected infrastructures, the process of working on the 4D–STEM use case also highlighted the critical need for tested, stable, and reusable data management buses. It is easy enough to do a one-off demonstration, but stretching from that to a new capability is a software engineering

challenge. SDK needs to be able to connect highly heterogeneous devices and needs to be able to operate seamlessly. Next we describe in detail the software engineering practices needed to make autonomous 4D-STEM experiments accessible to domain scientists.

3 Approach

As systems become more interconnected, interoperability among scientific applications and control software packages becomes increasingly complex and difficult to achieve. Because these solutions are sometimes developed by domain scientists rather than software engineers, they can lack a foundation developed on strong software engineering principles making it difficult to integrate solutions with other packages or scale them beyond a certain limit. Such software environments can also fail to efficiently use the rich hardware platform where they are deployed, and security is often an afterthought rather than a requirement during the initial development. Many scientific software applications are also monolithic and developed under the assumption it will run in a single system context. Combined, these factors can make developing scientific applications for interconnected environments a challenging task that can impact on the quality of scientific outcomes. There is a growing need for better software development toolkits (SDKs) and environments that are specifically designed to enable interconnected science ecosystems. A multi domain aware, easy-to-use, reusable, scalable, and secure framework built on strong, industry standard software engineering principles is key to fully realizing this goal.

3.1 Designing for Interconnected Science Ecosystems

Our goal is developing an SDK that seamlessly integrates instruments, computational resources, applications, tools, data, and other components into a common, connected ecosystem to simplify and accelerate scientific applications. But, we first need to have a sound understanding of example scientific use-cases and how we intend to distribute applications before we can effectively design the SDK and integrate it with those use-cases. We are using an iterative approach to design and develop the SDK which allows us to easily and quickly pivot priorities, adapt to new use-cases, develop new features, and improve existing applications.

Distributed Workflow: The STEM workflow discussed in § 2 is our initial use-case that is driving the SDK design. The original workflow is a large script that includes the instrument control logic, data analysis, and the overall experiment decision logic and assumes direct access to NionSwift [34] to interface with the instrument. However, moving this fully integrated application into a distributed ecosystem where it can benefit from GPU acceleration for the AI models is a non-trivial task. The application was built assuming direct access to the control software application-programming-interface (API), so it cannot simply be moved to the edge compute system.

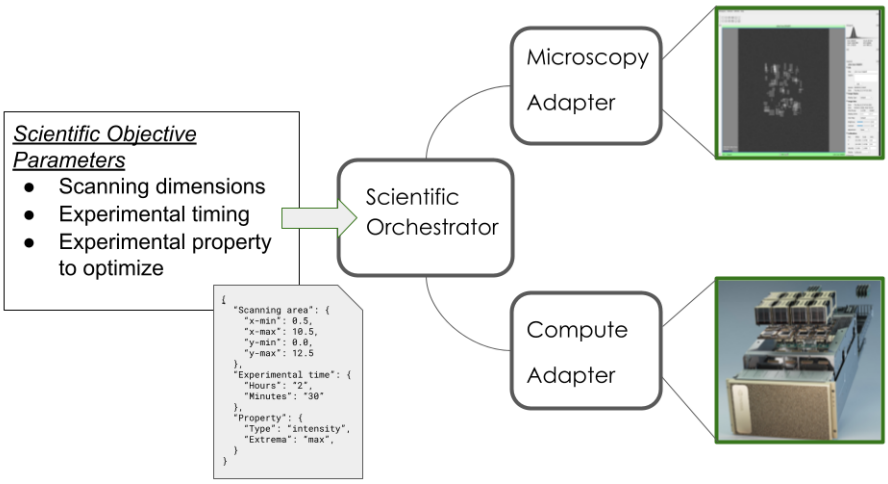


Fig. 2. Distributed version of the STEM use-case serving as the initial driver for SDK design. The original application is a single script executing in the NionSwift control application and here has been split into three main components: Microscopy adapter, compute adapter, and the scientific orchestrator.

We worked closely with the science team to understand their workflow and identified pieces that could be broken down into manageable components that could be automated and deployed. The overall approach is shown in Figure 2 and includes three main micro-services that together comprise the workflow functionality: the Microscopy adapter, the Compute adapter, and the Scientific Orchestrator. The Microscopy adapter micro-service is responsible for handling any interaction with the instrument itself (such as moving a probe or acquiring data), the Compute adapter micro-service will analyze data using AI models, and the orchestrator will manage data movement and determine when the overall experiment is complete. All of these micro-services use a common messaging subsystem and libraries that we are developing in the SDK which is further discussed in § 4.

Rather than tackling this workflow in its entirety, we are using a staged approach to develop each of the individual micro-services of the workflow. We first focused on developing the microscopy adapter and interfacing it with Nion Swift [34]. Once we identified patterns for interfacing with the microscope, these sections were moved to a separate adapter and the original code was modified to send and receive messages to a remote adapter. The next step was separating the higher-level experiment logic from the AI based analysis. The analysis service trains machine learning models and predicts the next coordinates to probe and the overall experiment orchestrator makes final decisions determining if the experiment goal was reached. Separating the higher-level logic and analysis allowed these micro-services to also be distributed at the edge. In this case, the "edge" system may not be typical of what is normally considered the edge; it is a large NVIDIA DGX-2 [13] distributed to instrument sites (compared to other central-

ized HPC resources) and used for accelerating the AI applications. The analysis micro-service needed to run persistently on the edge DGX accelerator, but the orchestrator could theoretically run anywhere. Once these micro-services are fully independent, they can be connected using the SDK and distributed throughout the ecosystem. The orchestrator can then manage the overall experiment where data is acquired by the microscopy adapter, passed to and analyzed by the edge compute adapter, and the next predicted probe locations are passed back to the microscopy adapter to start the next iteration.

SDK Design Goals: Initially, the micro-services that are split out from the original workflow are still very specific to the driver use-case, but stopping here leads to ad-hoc implementations that are not re-useable. As we continue to break down the STEM workflow, we are gradually improving and generalizing these components to support other science applications in the future, and we expect the components we develop now will need to change to support other scenarios. Good examples of this are the instrument adapter and the orchestrator. Our initial instrument adapter only implements the required functions to support this specific AI-infused STEM workflow, but we can continuously re-use those existing capabilities add new patterns based on other STEM workflows that use the same instruments. Similarly, the initial orchestrator is a simplified version of the original script which passes messages to the other micro-services in place of the original API calls. As we integrate other workflows, we will develop this into an intelligent orchestration engine which takes domain-specific configurations as input that include the experiment parameters such as scanning dimensions, timing, and property to optimize. This iterative approach will help us develop an SDK that enables complex systems that are easier to distribute, replicate, extend, reuse, and integrate. We can better understand what capabilities need to be fully implemented within the SDK and this allows for components to be gradually generalized and will result in re-useable and inter-operable components for future use-cases.

Our SDK will expose application programming interfaces (APIs) for services, scientific libraries, and components that can work with varied instruments across multiple sites (Figure 4). It will support reusable, open-standards-based components to enable the interconnected science ecosystem within the Department of Energy's (DOE) science facilities so workflows are not developed independently for each new use-case. Our standards-based, secure, and reusable framework will enable the laboratory of the future where instruments and resources are interconnected through an open, hardware/software architecture that science experiments to execute across systems and process data in a coherent fashion.

3.2 Digital Twins

Our approach includes supporting digital twin environments based on science proposals supported by the INTERSECT initiative. Digital twins are virtual environments that imitate a certain scientific instrument or setting as closely as

feasible and provide the user with the same software environment as the actual system. Some of the replicated components of the twin layer's workflow will be general, while others will be instrument- and physics-specific to the selected science use-case. The digital twin environments will provide the API and SDK for users to develop scientific workflows in the virtual environments and will include the expected micro-services and access methods. Digital twins will shift how physical world experiments and processes are realized by allowing users to first develop using a virtual environment rather than the actual instrument which improves safety and repeatability and accelerates discoveries. Digital twins include: (a) Instrument models and synthetic data sources: The basis of the digital twin are models of scientific instruments from specific use-cases. These virtual instruments will model a real system and provide the same control and monitoring interface expected of a real system. (b) Computational and data resources: The digital twin systems will provide computational resources so the virtual federated environment can execute actual workflows. These can be implemented using existing virtualization capabilities such as virtual machines and containers. (c) Network resources: Networking is the last main component of the digital twin environment. This provides the simulated connectivity between disparate resources to mimic the actual environment.

3.3 DevSecOps, Process and Software Excellence

DevOps: INTERSECT-SDK seeks to shorten development cycles and enable quick response and resolution to scientific software issues, boost productivity, test new methods and algorithms, and keep domain scientists engaged by reducing complexities of development and deployment. To address these challenges, we use a full development, security, and operations (DevSecOps) environment for end-to-end testing and demonstration. It supports the full OODA (Observe, Orient, Decide, Act) loop throughout development and testing [36].

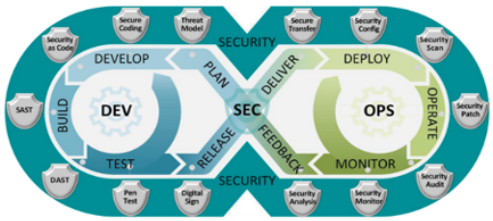


Fig. 3. DevSecsOps Workflow

As part of our DevSecOps (Figure-3), we plan to take the following approaches: (a) Developers implement low level (unit) tests as part of the code (b) Stand up CI automation and trigger builds for every pull request to run automated low-level testing (i.e., unit tests) as well as do static code analysis to provide fast feedback for rejected changes. (c) Ensure all high-level testing (i.e., integration, system, regression, performance, etc.) is executed. (d) Implement CI practices. All developers branch from the trunk, make changes in feature branches, and submit merge requests (MRs) back to the trunk. We will use Infrastructure-as-Code (IaC) practices to capture necessary infrastructure and

configuration management in version-controlled code to promote operational reproducibility and reliability.

Security: Another critical component of our SDK is the end-to-end security layer. Security in the past was isolated and added in the later stages of software development as an "after-thought" which leads to "retro-fitting" applications. Addressing vulnerabilities becomes an ad-hoc operation leading to poor accountability of critical components in production software. Thus, integrating security requirements as part of the overall workflow and system architecture is essential to build a robust foundation for the framework itself. The SDK puts security at the forefront and we are and designing end-to-end security into the core of the frameworks. This includes mechanisms such as encrypting messages for all data, application, and core services of the frameworks with end-to-end security [12]. Additionally, there needs to be a consistent method of managing authentication and user identities between multiple science domains.

Process: Developing any type of software, including scientific software, requires employing strong software engineering principles which lead to quality results. Adopting best practices helps scientific teams improve the sustainability, quality, and adaptability of their software, and ensures that the software that underpins ORNL scientific research is engineered to be reproducible and replaceable. The industry has recently shifted heavily towards using agile practices in order to rapidly develop products and meet user needs and requirements. Versus other traditional management processes, an agile approach allows software teams to quickly pivot towards new priorities as users, requirements, or available tools change. The SDK's teams development processes heavily borrow from the agile methods but is tailored to best fit our research and development and scientific software environment. Each iteration includes stages for core software engineering disciplines where teams plan for the sprint, define requirements, design tests, implement code, test and review changes, and finally release updated software. Other key features our process incorporates are user stories and issues, backlogs, and test-driven development. We also adopt other best practices into the software development life-cycle as needed.

Software Excellence: Software metrics are not like metrics in other professions in that they require context for proper interpretation. Scientific software metrics can become even more complex due to the fact the software is typically for very specific workflows which may not exist anywhere else in the world. However, both qualitative and quantitative metrics are key health indicators within the context of a project. Since our goal is enabling excellence in scientific software, we are developing a set of metrics to measure the quality and impact of the software we produce and provide a way to improve these over time. This also helps establish a road map, vision, and strategy for Scientific Software (SS) excellence across the lab by leading awareness and resources about software engineering processes and quality metrics [18].

4 INTERSECT-SDK

The INTERSECT-SDK includes several core parts, (shown in Figure-4) including: libraries, services and APIs, digital twins, and integrated DevSecOps runtime environments and testing stacks. It is underpinned by strong software practices inspired by industry standard processes.

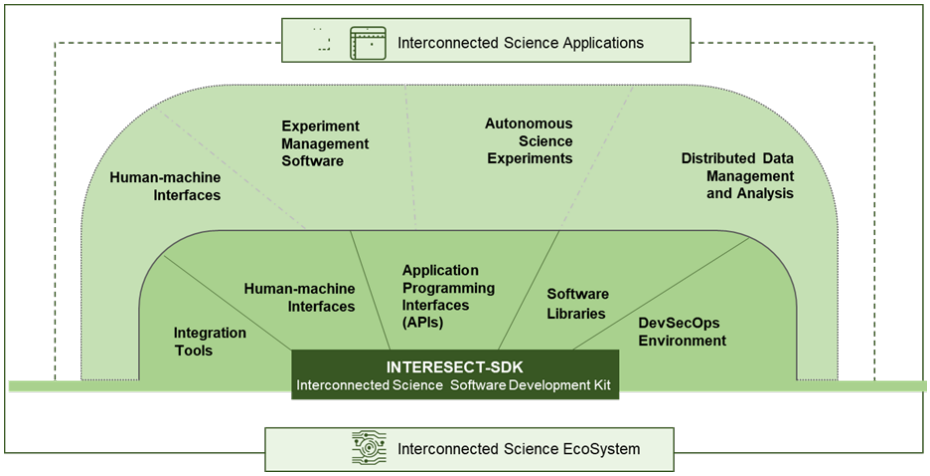


Fig. 4. INTERSECT SDK

4.1 Dev: Libraries, Services, and Adapters

Libraries: The INTERSECT-SDK provides a comprehensive set of libraries that abstract the integrated ecosystem architecture and design patterns away from the user to provide a unified method for accessing resources and instruments. They are designed to address the complexities of developing scientific applications and are cross-platform, cross-language, and reuseable to best support the unique environments that exist for different science use-cases, workflow patterns, instruments, and computing resources. The overall goal of our software stack is to be extremely flexible to easily adapt to future use-cases.

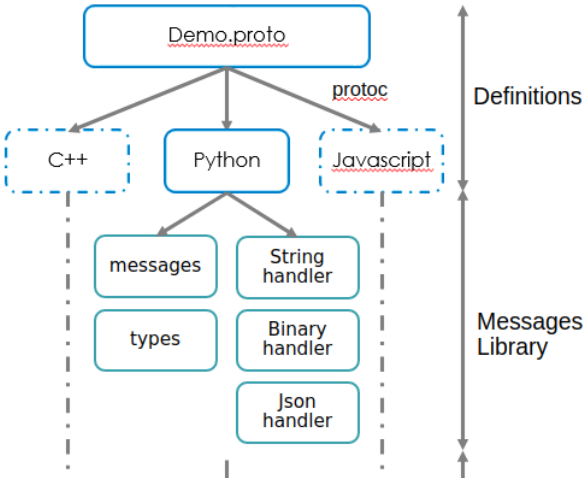
The core INTERSECT architecture is a hierarchical system-of-systems structure based on a publish-subscribe messaging architecture that allows communication between various micro-services. Two of the core libraries, *messages* and *client*, abstract the specific messaging implementations away from the user and provide the functionality to send and receive messages across the system bus. Both of these libraries implement plugins using the dependency inversion principle to be flexible and support different messaging engines and brokers. The dependency structure for the *messages* and *client* libraries is shown in Figure 5.

The *messages* library is the message abstraction layer for all of INTERSECT and is responsible for compiling message definitions to language-specific stubs. The messages are defined by the Architectures team and can be compiled and packaged to support multiple languages and platforms. These are currently written using Google's ProtoBuf library [10] to take advantage of code generation, but this is hidden from the user and can be easily changed if needed. This library also serializes and de-serializes message objects to over the wire formats and different handlers can be used based on requirements. However, the same serialization method must be used on a single message bus in order for microservices to communicate properly. We implemented a single base message type that encapsulates all messages sent over the message bus. Several core message types exist that map to specific messaging patterns defined by the architecture. Application and use-case specific messages inherit from a core message type and are defined separately and versioned.

These specific messages can be published in separate packages which lets users only install the specific messages required for an application. This design first ensures that any service that is on a message bus can receive and understand all of the base INTERSECT messages. The application messages can be unwrapped and deserialized if the proper message version and package are installed and the messages are applicable for the current microservice; otherwise, they are simply ignored.

The *client* library handles communication with the message brokers and provides a localized view of the ecosystem from a service's perspective. It uses plugins which allows it to support several different messaging brokers based on the use-case requirements; new backends can also be easily added if needed. Messages can be transmitted using *publish* and callbacks can be set with *subscribe* for handling received messages. The *client* internally depends on the *messages* library which handles all of the message serialization and de-serialization functionality. Published messages are first passed to *messages* and serialized before sending to the broker; received messages are first passed to the de-serializer for validation before calling the application's callback handler.

Core Services: Our INTERSECT reference implementation uses component-based micro-services to support reusability and abstract functionality to reduce overhead when developing new micro-services and components and extending functionality. Our software implements some functionality of traditional workflows, but it is focused more on managing the system-of-systems environment. Being flexible in design is a high priority to our team because this is a prerequisite for wide-spread adoption; if the system design is completely inflexible and not com-



patible with existing workflows in any way, it would not be used.

The main two services we have implemented are the messaging broker and the discovery service. We used the RabbitMQ broker [29] as the message bus primarily because of its support for multiple protocols and since we are developing the *client* library to support different brokers. We have currently implemented both the *MQTT* and *AMQP* protocols which can be chosen based on system requirements. The *discovery* service is an important requirement for implementing a flexible messaging system. Separate INTERSECT deployments may use different messaging brokers, but the same INTERSECT-SDK software stack. The *discovery* service implements a API the client calls when first connecting to an INTERSECT deployment to determine the system messaging configuration. The client can then use this to connect to the INTERSECT messaging bus.

Adapters: Adapters within the INTERSECT context are specialized microservices that handle interaction between the INTERSECT ecosystem and scientific systems and resources. These will implement specific capabilities that need to be exposed into INTERSECT through the *client* library and will be unique per system and/or use-case. Typically, this service will act as a bridge between a control system's API and the INTERSECT messaging bus and in some use-cases an adapter may be directly integrated into the scientific software application.

For our driver use case, we have identified two primary adapters in the workflow: the NionSwift adapter and the Edge DGX adapter. The NionSwift adapter uses a custom exposed API to communicate with the control software and exposes the higher level functions such as starting a scan, acquiring data, or moving the probe on the microscope. The Edge-DGX adapter handles the analysis steps of the workflow and executes AI models on the DGX accelerators. Both adapters communicate through the orchestrator microservice which manages data flow between the two and determines when the experiment goal has been achieved.

4.2 DevSecOps

The heterogeneity of scientific applications in the INTERESECT ecosystem provides unique challenges which are simplified by integrating DevSecOps processes. Releasing many smaller, iterative improvements over a period of time enables faster feature implementation and quicker issue resolution. Additionally, implementing security practices into the development pipeline ensures that security is addressed throughout the workflow, rather than something to be addressed by operations experts during deployment. While the initial transition to DevSecOps practices can be time-consuming for teams to adopt, these practices are extensible across other projects, streamline development, and help ensure software excellence and deliver strong scientific research software.

Containerization: The complexity of scientific applications demands consistent, segregated environments. For example, some applications (like NionSwift) require a graphical environment to successfully execute; others may demand a precise version of a plotting library due to incompatibilities with later versions or may require connections to a database or a message broker on startup. Scientific instruments also often invoke unique interactions which cannot be replicated within a generalized system; for example, a materials scientist may require working with a physical sample. To test the INTERSECT ecosystem, we require the ability to replicate a deployment of the software components of an instrument, known as a digital twin. A single scientific workflow could consist of multiple digital twins, each of which will make certain assumptions about its host system. Containerizing each application allows us to efficiently replicate its software environment without having to painstakingly recreate its environment each time we test minor modifications. As we grow the digital twin capabilities of instruments within INTERSECT, we can more easily deploy and scale an entire virtual national laboratory level deployment with container benefits. Given that we have containerized a digital twin of an electron microscope, one can envision now deploying many of these containers to resemble a microscopy User Facility. The benefits of containerization directly correlate to digital twin capabilities INTERSECT hopes to accomplish.

Infrastructure-as-Code : "Infrastructure-as-Code" (IaC) is a core DevOps practice that has emerged where teams capture system requirements for applications as code and automate creating and deploying infrastructure resources. There are many benefits of adopting IaC, including: easily reconfiguring changes across all managed infrastructure, easily scaling the managed infrastructure, and reliably deploying the infrastructure to a good, known state. Terraform [5] is used as the IaC tool to manage cloud infrastructure for hosting current INTERSECT services. IaC benefits have already affected scientific research. One such benefit is the ability to share IaC openly with other researchers to enable starting with

[5] HashiCorp's Terraform: https://www.terraform.io/

properly configured infrastructure to run complex scientific sofware. An example is Google publishing open-source IaC scripts to provision infrastructure on Google Cloud Platform to run Folding@Home to support COVID-19 research. [6]

Continuous Integration: Continuous integration (CI) is accomplished by multiple levels of integrating new code changes, and ensuring high quality of software determined by the project standards. We use libraries which install pre-commit hooks on developers' local systems to enforce good practices before code can even be committed to the system. Once it is committed, it runs through additional stages of automated integration enabled by CI pipelines, which perform linting, testing, formatting, build, code coverage, etc. Using these standard software engineering practices help build reliable, high-quality scientific software.

Continuous Delivery and Continuous Deployment: Continuous delivery is a mostly automated deployment pipeline, but one that does have a manual stage for human-in-the-loop confirmation the deployment should be carried out. Continuous deployment is a fully automated deployment pipeline without any manual intervention. Continous deployment is the desired deployment approach, but sometimes the complexity necessitates using continuous delivery or even manual approaches. We deploy infrastructure to our own Kubernetes cluster in the CADES OpenStack cloud [5], to the Slate OpenShift cluster [32] inside Oak Ridge Leadership Computing Facility for HPC resources, and to ORNL's DGX Edge compute resources.

Security: For the initial implementation of authentication, tokens issued via available identity providers will be used across services. Based on existing infrastructure, authorization will first be implemented against Lightweight Directory Access Control (LDAP) and local system methods. The current web-based user interface mitigates cross-site scripting (XSS) attacks by applying a strong Content Security Policy and utilizing input-sanitizing frameworks. For ensuring security of the INTESERCT-SDK software artifacts, automated static security scanning tools, such as Bandit [28] and Harbor [11], will be set up in CI/CD for software packages and container images, respectively.

4.3 Digital Twins

The last main focus of INTERSECT-SDK is developing instrument digital twins for the STEM use-case that can be deployed within our DevSecOps infrastructure. Ideally, instrument vendors would provide software, interfaces, and digital twins for their instruments, but this does not happen often. There were several major steps involved with creating and deploying the digital twin that we used for our STEM use-case.

[6] See https://github.com/GoogleCloudPlatform/terraform-folding-at-home

Containerization: The NionSwift control software for the microscopes uses plugins and provides the ability to simulate STEM instruments with generated data. Our first step was integrating NionSwift and its simulator plugins into a container that could be easily deployed for testing and demonstration. NionSwift uses a graphical user interface (GUI) which introduced some initial roadblocks because containers do not typically have display servers installed. We used an existing base image that includes a X-window server which exposes a GUI over an HTTP VNC connection. This proved to be a sufficient solution for the digital twin by allowing remote access to the GUI as long as the container ports were properly mapped.

Custom Plugin: The next major hurdle is remotely accessing the digital twin from the adapter. NionSwift does have some basic APIs, but these were not adequate for handling remote control. Luckily, NionSwift is open source and the INTERSECT Integration team had developed a prototype remote API using the Pyro library (Python Remote Objects) [27] We want to reuse as much code as possible for other scenarios, so we identified generic patterns of controlling the microscope that are useful for STEM workflows. We extended the proof-of-concept server to expose new capabilities and also created a plugin that can be directly integrated into NionSwift for both the instrument and the digital twin. This simplifies the workflow even further for scientists as it is directly integrated and automatically started with NionSwift. Our NionSwift adapter connects to the API and calls the proper functions when handling INTERSECT messages.

Security: Some applications may not have INTERSECT compatible security and we must take additional steps to properly integrate them. Pyro lacks an integrated security solution, which is a major concern for the digital twin and instrument. Any user could connect to the unsecured API and run commands on the microscope without authorization. We addressed this issue by incorporating a secure socket layer to both the digital twin and the adapter to ensure only the adapter could successfully connect. This uses standard public/private certificates to validate a connecting user against a list of allowed keys. These certificates could be passed to users that need to control the microscope remotely, but ideally they should only be installed on trusted systems like the adapter.

5 Related Work

The scientific community has recently emphasized the need for intelligent systems, instruments, and facilities for enabling the science breakthroughs with autonomous experiments, self-driving laboratories, and supporting AI-driven design, discovery and evaluation [7]. Today, many scientific experiments are time and labor intensive [6]. A majority of effort is spent designing ad-hoc software stacks and data processing tools specifically for the experiments [18]. This specialization also creates interoperability challenges among scientific software tools

leading to reduced automation and integration across multiple science disciplines [9]. In addition, scaling the experiments would be difficult, as the teams responsible for developing the software are domain scientists and might have inadequate experience software engineering. In brief, it would be helpful to decouple the domain science from the science of software responsible for executing the experiments. Earlier, researchers aiming to enhance interactions between traditional high performance computer resources and experimental equipment have developed a Software Framework for Federated Science Instruments [22]. The main objective of this work is the development of an interconnected SDK that would relinquish the software development by scientific domain experts, and instead benefit from a wide array of inter-connected infrastructure.

Both the System of Systems (SoS) Computing Architecture [9, 1] and Artificial Intelligence brings transformative change to the scientific community and foster novel discoveries. This change largely be fueled by the scientific applications that would operate at the intersection of problem and solution space across varied domains, including materials, neutron science, systems biology, energy, and national security. While successes have been achieved by applying AI, albeit owing to its complexity and technological difficulty, AI is still limited to its domain experts and core practitioners. With the arrival of automated machine learning (AutoML) [8, 3, 17], there is now an unparallel opportunity to democratize the use of AI across all scientific domains and give its power in the hands of non-practitioners, too. As part of further work, we plan to extend the functionality by deploying machine learning packages and scale it through what is increasingly referred to as Machine Learning Operations (MLOps) [35] This includes model training and optimization, endpoint deployment, and endpoint monitoring for the application using ML-related functionalities.

Over the last 15 years or so, the community has recognized the key roles that Development and Operations (DevOps) play in bridging the gap between software development and deployment into production [3, 17, 37]. However, a skills and knowledge gap exists to realize the necessity of DevOps for scientific software development. Software for scientific experiments should require prototyping and flexibility to accommodate advances in the instrument setup and innovative, original experiments [12, 20, 21]. A great deal of progress has been made by trying to adopt standardized SDKs [1, 8] for science. However, a feedback loop is required to rapidly adapt and modify for cutting-edge developments due to the novelty [23, 2, 31]. DevOps plays a crucial role in moving developers' changes from version control to a released software artifact, allowing scientists to run the latest version of the prototype. Bayser et al. [3] coined the term ResearchOps in their work where they outline the importance of a continuous cycle for research, development, and operation. This allowed for the latest prototype versions to be available in practice at the IBM Research Brazil lab (i.e., short release cycles) [3]. Our work will be the first to connect such a wide variety of devices, HPC environments, and edge federated instruments [4, 16] together under a single workflow. This brings a seamless and secure infrastructure for multi-domain science that drives the future of federated research [14, 15].

6 Conclusion

The main outcome of this work is a process toward a robust INTERSECT-SDK, a development environment for building scientific applications on an open architecture specification allowing for autonomous experiments and "self-driving" controls. The design and development of INTERSECT as a platform of transformative sciences underpins the need for a broader collaboration and coordination from computing and domain science. The proposed INTERSECT-SDK will enable the optimal use of federated architectures connecting disparate instruments and AI-driven compute for novel scientific discoveries. In addition, this will allow the research community to run multi-domain experiments virtually from distant locations, control such instruments, and process data by combining the optimal workflow. The DevSecOps capability demonstrated through this work will allow for application development and execution in an end-to-end secure environment and across different levels of access hierarchies and instruments.

Additionally, we showcased the efficacy of the proposed SDK through a case-study on Scanning Transmission Electron Microscopy. This example is an archetype of future interconnected science applications that connect experimental hardware (e.g., microscopes) and AI-guided experimental control systems to speed up and improve the repeatability of scientific discovery.

The other core outcome of this work is an articulation of the software engineering best practices for the development of scientific workflow for autonomous experiments. We propose and demonstrate early results for a complete software life-cycle framework that addresses the need to automate the development, integration and deployment of software applications in a repeatable and reproducible way. In subsequent work, we plan to complete the development of the proposed SDK with interfaces to connect with standard scientific instruments from multiple domains. In addition, we plan to develop APIs which allow for extending the SDK to new instruments. The SDK will be released as open-source, allowing the research community to benefit in their R&D work.

Acknowledgement. This manuscript has been authored by UT-Battelle, LLC under Contract No. DEAC05-00OR22725 with the U.S. Department of Energy. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a nonexclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access.

References

1. Roscoe Bartlett, Irina Demeshko, Todd Gamblin, and et al. xsdk foundations: Toward an extreme-scale scientific software development kit. *Supercomputing Frontiers and Innovations*, 4:69–82, 2 2017.

2. Susan M Baxter, Steven W Day, Jacquelyn S Fetrow, and Stephanie J Reisinger. Scientific software development is not an oxymoron. *PLOS Computational Biology*, 2:1–4, 2006.
3. Maximilien De Bayser, Leonardo G. Azevedo, and Renato Cerqueira. Researchops: The case for devops in scientific applications. *Proceedings of the 2015 IFIP/IEEE International Symposium on Integrated Network Management, IM 2015*, pages 1398–1404, 6 2015.
4. Jack Brassil and Irene Kopaliani. Cloudjoin: Experimenting at scale with hybrid cloud computing. In *2020 IEEE 3rd 5G World Forum (5GWF)*, pages 467–472, 2020.
5. CADES. CADES OpenStack cloud computing, 2022.
6. Marcelo Cataldo, Audris Mockus, Jeffrey A. Roberts, and James D. Herbsleb. Software dependencies, work dependencies, and their impact on failures. *IEEE Transactions on Software Engineering*, 35(6):864–878, 2009.
7. Kate Crawford, Meredith Whittaker, Madeleine Clare Elish, Solon Barocas, Aaron Plasek, and Kadija Ferryman. The ai now report. *The Social and Economic Implications of Artificial Intelligence Technologies in the Near-Term*, 2016.
8. Rafael Ferreira da Silva, Henri Casanova, Kyle Chard, and et al. Workflows community summit: Advancing the state-of-the-art of scientific workflows management systems research and development. *arXiv preprint arXiv:2106.05177*, 6 2021.
9. Stephen G. Eick, Todd L. Graves, Alan F. Karr, Audris Mockus, and Paul Schuster. Visualizing software changes. *IEEE Transactions on Software Engineering*, 28, 2002.
10. Google Developers. Protocol Buffers, 2022.
11. Harbor. Harbor Website, 2022.
12. Orit Hazzan and Yael Dubinsky. The agile manifesto. *SpringerBriefs in Computer Science*, 9:9–14, 2014.
13. Jonathan Hines. Ornl adds powerful ai appliances to computing portfolio – oak ridge leadership computing facility. https://www.olcf.ornl.gov/2019/02/06/ornl-adds-powerful-ai-appliances-to-computing-portfolio/, Aug 2019. (Accessed on 06/26/2022).
14. Li Li, Yuxi Fan, Mike Tse, and Kuo-Yi Lin. A review of applications in federated learning. *Computers and Industrial Engineering*, 149:106854, 2020.
15. Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. Federated learning: Challenges, methods, and future directions. *IEEE Signal Processing Magazine*, 37:50–60, 2020.
16. Wenbin Li and Matthieu Liewig. A survey of ai accelerators for edge environment. In Álvaro Rocha, Hojjat Adeli, Luís Paulo Reis, Sandra Costanzo, Irena Orovic, and Fernando Moreira, editors, *Trends and Innovations in Information Systems and Technologies*, pages 35–44. Springer International Publishing, 2020.
17. Lucy Ellen Lwakatare, Pasi Kuvaja, and Markku Oivo. Dimensions of devops bt - agile processes in software engineering and extreme programming. pages 212–217. Springer International Publishing, 2015.
18. Addi Malviya-Thakur and Gregory Watson. Dynamics of scientific software teams. *Collegeville*, 2021.
19. Chris Meyer, Niklas Dellby, Jordan A Hachtel, Tracy Lovejoy, Andreas Mittelberger, and Ondrej Krivanek. Nion swift: Open source image processing software for instrument control, data acquisition, organization, visualization, and analysis using python. *Microscopy and Microanalysis*, 25(S2):122–123, 2019.

20. Audris Mockus, Roy T. Fielding, and James D. Herbsleb. Two case studies of open source software development: Apache and mozilla. *ACM Transactions on Software Engineering and Methodology*, 11, 2002.
21. Audris Mockus and David M. Weiss. Predicting risk of software changes. *Bell Labs Technical Journal*, 5, 2000.
22. Thomas Naughton, Seth Hitefield, Lawrence Sorrillo, Nageswara Rao, James Kohl, Wael Elwasif, Jean-Christophe Bilheux, Hassina Bilheux, Swen Boehm, Jason Kincl, et al. Software framework for federated science instruments. In *Smoky Mountains Computational Sciences and Engineering Conference*, pages 189–203. Springer, 2020.
23. Luke Nguyen-Hoan, Shayne Flint, and Ramesh Sankaranarayana. A survey of scientific software development. In *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*. Association for Computing Machinery, 2010.
24. Nion Co. Nion Swift User's Guide, 2022.
25. Colin Ophus, Peter Ercius, Michael Sarahan, Cory Czarnik, and Jim Ciston. Recording and using 4d-stem datasets in materials science. *Microscopy and Microanalysis*, 20(S3):62–63, 2014.
26. Stephen J Pennycook and Peter D Nellist. *Scanning transmission electron microscopy: imaging and analysis*. Springer Science & Business Media, 2011.
27. Library Pyro. Github - irmen/pyro5: Pyro 5 - python remote objects for modern python versions. https://github.com/irmen/Pyro5, 2022. (Accessed on 06/27/2022).
28. Python Code Quality Authority (PyCQA). PyCQA's Bandit GitHub repository, 2022.
29. RabbitMQ. RabbitMQ Website, 2022.
30. Kevin M Roccapriore, Ondrej Dyck, Mark P Oxley, Maxim Ziatdinov, and Sergei V Kalinin. Automated experiment in 4d-stem: exploring emergent physics and structural behaviors. *ACS nano*, 2022.
31. Judith Segal and Chris Morris. Developing scientific software. *IEEE Software*, 25:18–20, 2008.
32. Slate. Slate: Kubernetes cluster with access to Summit, 2021.
33. Suhas Somnath, Rama K Vasudevan, et al. Building an integrated ecosystem of computational and observational facilities to accelerate scientific discovery. In *Smoky Mountains Computational Sciences and Engineering Conference*, pages 58–75. Springer, 2021.
34. Nion Swift. Nion swift. https://github.com/nion-software/nionswift, Mar 2022. (Accessed on 06/26/2022).
35. Mark Treveil, Nicolas Omont, Clément Stenac, Kenji Lefevre, Du Phan, Joachim Zentici, Adrien Lavoillotte, Makoto Miyazaki, and Lynn Heidmann. *Introducing MLOps*. O'Reilly Media, 2020.
36. Robert Zager and John Zager. Ooda loops in cyberspace: A new cyber-defense model. *Small Wars Journal*, 10 2017.
37. F Zhao, X Niu, S L. Huang, and L Zhang. Reproducing scientific experiment with cloud devops. In *2020 IEEE World Congress on Services (SERVICES)*, pages 259–264, 2020.