A Microservices Architecture Toolkit for Interconnected Science Ecosystems

Michael J. Brim*, Lance Drane†, Marshall McDonnell†, Christian Engelmann†, and Addi Malviya Thakur†
*National Center for Computational Sciences, †Computer Science and Mathematics Division
Oak Ridge National Laboratory
Oak Ridge, Tennessee, U.S.A.
{brimmj,dranelt,mcdonnellmt,engelmannc,malviyaa}@ornl.gov

***Abstract*—Microservices architecture is a promising approach for developing reusable scientific workflow capabilities for integrating diverse resources, such as experimental and observational instruments and advanced computational and data management systems, across many distributed organizations and facilities. In this paper, we describe how the INTERSECT Open Architecture leverages federated systems of microservices to construct interconnected science ecosystems, review how the INTERSECT software development kit eases microservice capability development, and demonstrate the use of such capabilities for deploying an example multi-facility INTERSECT ecosystem.**

***Index Terms*—Software architecture, Scientific computing, Application programming interfaces**

I. Introduction

The Interconnected Science Ecosystem (INTERSECT) initiative at Oak Ridge National Laboratory (ORNL) seeks to research and develop technologies that allow scientific workflows to seamlessly incorporate scientific instruments and advanced computational and data management resources located across diverse facilities and organizations. The initiative's overall goal is to pioneer science ecosystems that enable autonomous, multi-modal experimentation and advanced data analysis through interconnected "Smart Labs of the Future" [1]. INTERSECT is structured to advance the state-of-the-art for both domain science-focused and cross-cutting efforts. The domain science projects seek to enable autonomous experimentation and incorporation of machine learning for data-driven experiment steering or design of experiments, while the cross-cutting projects address challenges common to a wide variety of science use cases. The initial cross-cut projects focus on the *Architecture*, the *Software Development Environment*, and *Integration*. The Architecture project is defining an open architecture specification based on a federated system-of-systems approach to interconnected science ecosystems [2]. The Software Development Environment project is creating the reusable software infrastructure and application programming interfaces (APIs) as a software development kit (SDK) to support the secure deployment and operation of interconnected science ecosystems [3]. The Integration project works closely with the domain science projects to deploy or update hardware and software infrastructure at laboratories and facilities to enable connections to INTERSECT ecosystems, and to develop virtual infrastructure twin technologies that enable software development without impact to production science [4].

System-of-systems (SoS) is a core design methodology for the INTERSECT Open Architecture [5] as it simplifies the creation of complex systems with many interacting components by decomposition into smaller, well-defined systems [6]. This methodology has several beneficial characteristics including managerial and operational independence of systems, evolutionary development of the independent systems, and support for emergent behaviors through new forms of system composition [7]. Similarly, microservices architecture is a design methodology for structuring a distributed application as a networked collection of loosely-coupled services that are independently developed, maintained, and operated. Microservices are thus a natural choice for developing reusable capabilities that can be composed in a SoS manner to support scientific workflows in interconnected science ecosystems.

In this paper, we describe our ongoing work to demonstrate the use of the INTERSECT SDK [8] to develop and deploy microservice capabilities defined in the INTERSECT Architecture Specification. First, we summarize the INTERSECT architectural design for constructing interconnected science ecosystems via microservices (§II). Next, we discuss the INTERSECT SDK features that promote ease of development and reuse of microservices (§III). Finally, we demonstrate the development and deployment of INTERSECT infrastructure microservice capabilities for distributed system management and registration using an enhanced version of the SDK (§IV).

II. The INTERSECT Microservice Architecture

As previously introduced, microservices architecture is a design methodology for structuring a distributed application (e.g., a scientific workflow incorporating distributed resources) as a networked collection of loosely-coupled services. Each microservice provides a specific application programming interface (API) that is tailored to its domain, which ensures a clear separation of concerns between differing microservices,

979-8-3503-5554-3/24/$31.00 ©2024 IEEE
DOI 10.1109/SCW63240.2024.00259

prevents duplicate functionality, and encourages reuse. The API methods and associated data (i.e., input and output parameters) are defined by the *microservice contract*, which documents the purpose for each service method and its data types and permitted values. A microservice may have several different implementations, where each implementation provides the same contract but uses different underlying technologies or supports a particular deployment environment. When multiple implementations are available, a scientific workflow can choose the implementation most suitable for its environment or application needs.

Within the INTERSECT Architecture Specification, the Microservice Architecture [9] provides a catalog of infrastructure and experiment-specific microservices that may be useful within an interconnected science ecosystem. *Infrastructure microservices* represent common service functionality and capabilities, such as data management, computing, system management, and workflow orchestration that are likely to be generally useful across many science ecosystems without the need for customization. *Experiment-specific microservices*, on the other hand, represent services whose implementation may require detailed application domain knowledge, such as experiment planning or steering services that require knowledge of experiment-specific control parameters and their associated constraints. The INTERSECT Architecture Specification includes science use case design patterns [10] that help identify the relevant infrastructure and experiment-specific microservices for a given science workflow. Figure 1 depicts a generalized INTERSECT ecosystem that provides machine-in-the-loop intelligence based on experiment-specific services for experiment steering or design of experiments. The infrastructure services shown on the right may be used throughout the experimental cycle for any associated computing, data management, or workflow management aspects.

A. Federated Systems of Microservices

All INTERSECT microservices are defined to facilitate composition within federated SoS architectures [11] through adoption of a common hierarchy of organizations, facilities, systems, services, and resources. As shown in Figure 2, each INTERSECT system is owned by a single organization and optional facility, and represents a logical collection of INTERSECT services and associated resources. Each service provides utility in the form of a set of microservice capabilities, where each capability corresponds to a specific microservice contract. The INTERSECT architecture supports both long-lived ecosystems based on shared multi-user services, as well as campaign-specific ecosystems constructed from dynamically deployed private services.

Resources include physical infrastructure such as computing systems and scientific instruments as well as external services (e.g., cloud computing services) and data or information repositories. All INTERSECT activities involving system resources are facilitated through service interactions. A given resource may be exclusive to a system or shared amongst systems within a facility or organization.

A system may also include subsystems, which are self-contained systems that are used by the parent system. Subsystems typically exist to maintain operational independence over a group of related services that provide access to specific system resources.

B. Microservice Capability Definitions

To enable SoS composition of INTERSECT microservices, it is crucial to understand the types of interactions a given microservice may reasonably expect from one of its clients. In Figure 3, we show three common patterns that substantively cover the expected interactions: *Command*, *Request-Reply*, and *Asynchronous Status or Event*. The Command interaction pattern involves the client asking the microservice to do something. The microservice typically responds immediately with a simple acknowledgement that the command has been received successfully or some error status indicating why the command was not acceptable. A command may initiate an activity within the microservice, but that activity is not ordered with respect to the acknowledgment. Commands are thus asynchronous interactions from the client perspective. The Request-Reply interaction pattern has the client making a request of the microservice that includes an expected reply containing pertinent information or data related to the request. Because the reply is not sent until the request has been fully processed, this is a synchronous interaction pattern from the client perspective. Finally, the Asynchronous Status or Event interaction pattern represents cases where the microservice generates status or event information that is broadcast to any interested parties at irregular intervals as a result of internal operational state changes or ongoing activities. Events are informational in nature and there is no expectation that the message must be delivered. However, status messages are typically associated with activities initiated by clients, and therefore must provide some limited form of message durability to ensure delivery to at least one interested party.

Based on these common interaction patterns, all microservice capabilities within the INTERSECT Architecture Specification are defined in a uniform fashion. Each capability definition is intended to serve as the basis for the microservice contract for all implementations. In each capability definition, the proposed functionality (i.e., API methods or associated data) is grouped by the corresponding microservice interaction pattern. The data types used in definitions are generic names for common types and structures supported by the data models of most data schema standards (e.g., JSON Schema and XML Schema Definition). Where applicable, capability definitions also specify relationships to other microservice capabilities, such as whether the capability extends the functionality of another capability or has dependencies on other capabilities.

III. A Software Development Kit for INTERSECT Microservices

At the launch of the INTERSECT initiative in October of 2021, the three cross-cut and four domain science projects

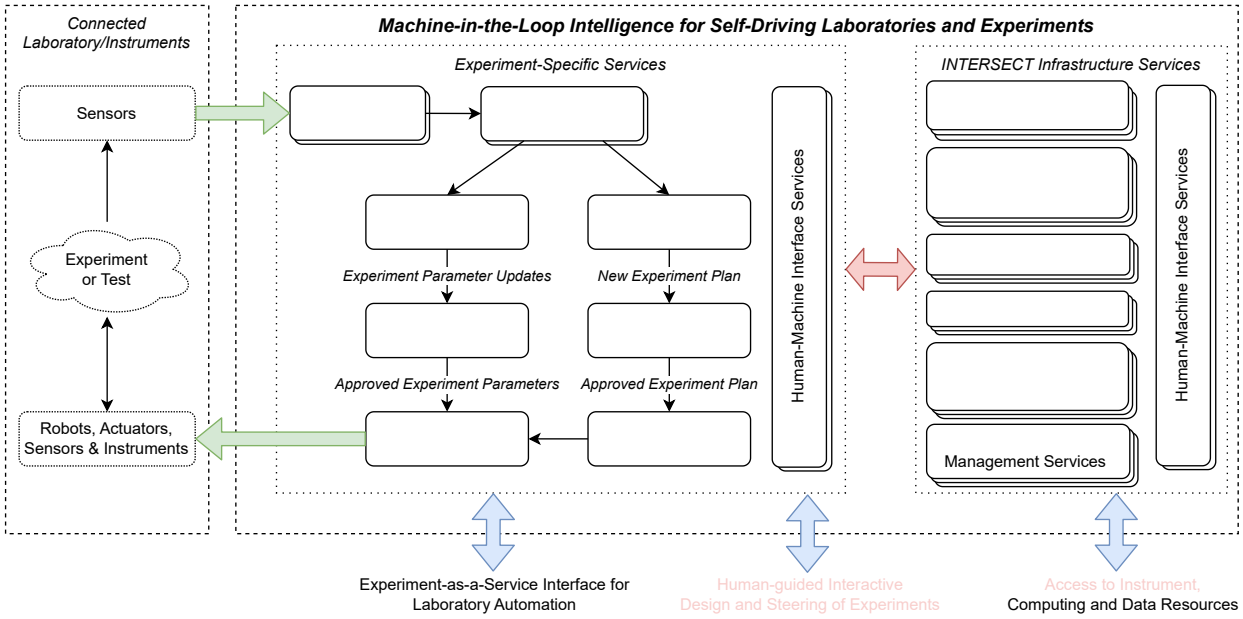

Fig. 1. A Microservice-oriented Approach to Machine-in-the-Loop Interconnected Science Ecosystems

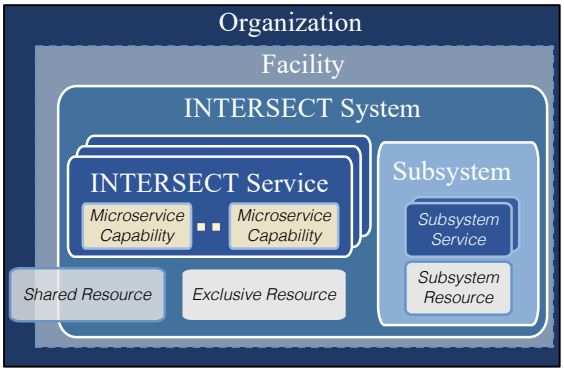

Fig. 2. INTERSECT System, Service, and Resource Hierarchy

started simultaneously. To ensure each project could demonstrate progress, an agile approach was adopted that allowed each project to rapidly work toward their individual milestones while also requiring regular meetings between projects to cross-pollinate ideas, methods, and common requirements for interconnected science ecosystems. Because the INTERSECT architecture was just starting to be defined, the Software Development Environment project initially focused on creating foundational software infrastructure to enable interconnection between scientific instruments and computing systems for machine-in-the-loop experiment steering, using the autonomous scanning transmission electron microscopy (STEM) domain science project as the guiding use case.

The initial SDK [12] that resulted from this focus provided the INTERSECT control plane communication infrastructure

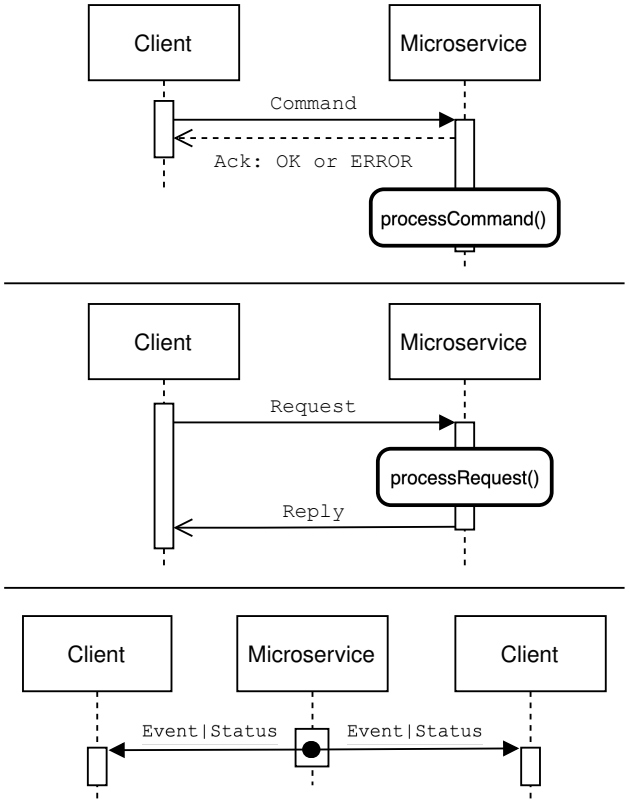

Fig. 3. INTERSECT Microservice Interaction Patterns: (top) Command, (middle) Request-Reply, (bottom) Asynchronous Event or Status

based on a RabbitMQ message broker and a message encapsulation library using Google protocol buffers for cross-language portability. It also included a minimal Python SDK client library that simplified secure interactions with the broker and seamlessly handled message encapsulation and validation. The client library was intended for use in adapters [13], which are services that bridge between INTERSECT and external software used to interact with instrument or computing resources. In this early version of the SDK, each use case utilized its own set of message definitions, and adapters provided callback functions that process incoming messages based on their type. Feedback from domain scientists wishing to develop new INTERSECT capabilities using this version of the SDK noted a desire for the client library to support function-based, rather than message-based, definition of the API and for transparent definition of the associated messages based on function parameters and return types. An API-oriented SDK client library would also ease prototype development for the catalog of microservice capabilities being defined in the INTERSECT Architecture Specification. To address these concerns a significant restructuring of the SDK was undertaken.

The SDK was modified so that users interact with instances of classes representing core architecture concepts such as services, microservice capabilities, and clients. The `IntersectService` class provides the INTERSECT service abstraction of a persistent entity that is registered within an ecosystem to process interactions with a microservice capability implemented by the service. The `IntersectClient` class provides a minimal interface for sending messages to INTERSECT services and declaring callback functions that should be used to process responses or events from services. The services for which a client wishes to receive events are provided as a list at initialization, and may be modified throughout the client's lifetime. Both `IntersectService` and `IntersectClient` continue to hide all interactions with the underlying INTERSECT control and data planes. However, they both require a configuration to be provided at instantiation that contains all the necessary information for connecting to the control and data planes. In the case of services, the configuration information also defines the SoS coordinates (i.e., the names of the organization, facility, system, subsystem, and service) that may be used by clients to make requests of specific services.

Function-oriented definition of microservice capability APIs are supported through use of SDK-defined Python function decorators within a class that derives from `IntersectBaseCapabilityImplementation`. The `@intersect_message` decorator is added to each class method associated with a microservice API function. All decorated class methods are required to use the same function signature pattern shown in Figure 4, where the method has at most one argument and returns a single value. Arguments and return values are required to be annotated with their associated type, which may be a built-in Python type or structure or a user-defined class. The revised SDK uses Pydantic for message data serialization and validation. Thus, user-defined classes must be a Python `dataclass`, `TypedDict`, or derived from the Pydantic `BaseModel` class. The SDK also uses Pydantic and the Python `inspect` module to automatically generate AsyncAPI-compatible schemas [14] for capability APIs.

A `@intersect_message` decorated method supports microservice capability API functions providing *Command* or *Request-Reply* interactions. The SDK also provides a `@intersect_event` decorator that can be used to signify a method may generate a microservice capability *Asynchronous Event* but is not otherwise part of the capability API. The name and value type for each event is specified within an `events` dictionary parameter on the decorator. The `@intersect_message` decorator also supports this parameter for capability API functions that also emit events.

Client callback functions for both response and event handling must use a defined function signature that has parameters for the message or event data as well as its source (i.e., the service and generating method) and returns an `IntersectClientCallback` structure. This structure includes a list of any new messages that should be sent and lists that alter the set of services for which the client has event subscriptions.

The restructured SDK with these modifications was made available to users with the release of version 0.6.0. The SDK code repository has been made public by a move from code.ornl.gov, a restricted-access ORNL GitLab service, to its new home on GitHub [8].

IV. Demonstration of INTERSECT Microservices Development and Deployment

In this section, we demonstrate the use of an extended version of the SDK to deploy systems within an INTERSECT ecosystem. First, we review the architectural design for managing INTERSECT systems and registering them within an ecosystem. We then discuss two recently developed extensions to the SDK that are critical to supporting this design. Finally, we discuss the implementation of the related microservice capabilities from the INTERSECT Architecture Specification and describe an example INTERSECT ecosystem deployment.

A. INTERSECT System Management and Registration

Within the INTERSECT Architecture Specification, *System Management and Monitoring Services* are responsible for control and inspection of INTERSECT systems, subsystems, and services. The architecture does not prescribe the granularity of an INTERSECT system, leaving that decision up to the operators of the system that integrate it for use within INTERSECT ecosystems. Thus, a single INTERSECT system may logically represent a wide range of services and associated resources. For example, a system operated by an observational facility may only expose capabilities for data acquisition and control of an individual scientific instrument, while a computational facility may provide a single system that incorporates multiple compute and data storage resources. In the latter case, it is appropriate to utilize subsystems to

```
class MyCapability(IntersectBaseCapabilityImplementation):
    # Decorated method for a microservice capability Command or Request-Reply
    @intersect_message()
    def api_method_name(self, parameters : <type>) -> <type>:
        ...

    # Decorated method and generation of a microservice capability Asynchronous Event
    @intersect_event(events={'event_name': IntersectEventDefinition(event_type=int)})
    def other_method_name(self, ...):
        ...
        event_value : int = 1
        self.intersect_sdk_emit_event('event_name', event_value)
        ...
```

Fig. 4. INTERSECT SDK Decorators for Microservice Capability Interactions

enable finer-grained control and inspection of resources that are operationally independent.

Each INTERSECT system is expected to deploy a designated *System Management* service that coordinates all aspects related to system information management, control of services and subsystems, and status monitoring of associated resources, services, and subsystems for the duration of the system's connection to the INTERSECT ecosystem. These responsibilities are handled by two microservice capabilities the service must provide. The `SystemManager` capability provides interfaces for aggregate control and status of all subsystems, resources, and services for the system, while the `SystemInformationCatalog` capability maintains useful information about each subsystem, service, and resource.

When a system is newly introduced to an ecosystem, its management service must register the system. Figure 5 shows a simplified sequence diagram for this initial system registration procedure; the full details of microservice interactions involved are available in the "Registration of INTERSECT Systems, Services, and Resources" appendix of [9]. The INTERSECT architecture permits a hierarchy of coordinating services providing the `SystemsRegistrar` capability. In Figure 5, we assume each distinct INTERSECT operational domain (e.g., an organization or facility) provides a registry service. The management service first requests a universally unique identifier (UUID) for the system from the registrar. It then registers each of its local resources by requesting a resource UUID from the registrar, then recording details about the resource in its information catalog.

The service registration sequence is shown in Figure 6, and follows a similar pattern to system resource registration. Although not shown in Figure 5, the management service also follows this sequence to register itself with the ecosystem to provide its contact details so that other members of the ecosystem may query it to obtain information on the system's underlying services, resources, and subsystems. Once its registration is complete, the management service begins accepting requests from other services wishing to join the system. These requests include information about the microservice capabilities provided by the new service and any system resources used. The management service may optionally use a secret key to allow only those services providing the key to join the system. Key distribution is expected to occur using external mechanisms such as those found in container orchestration infrastructures such as Kubernetes [15].

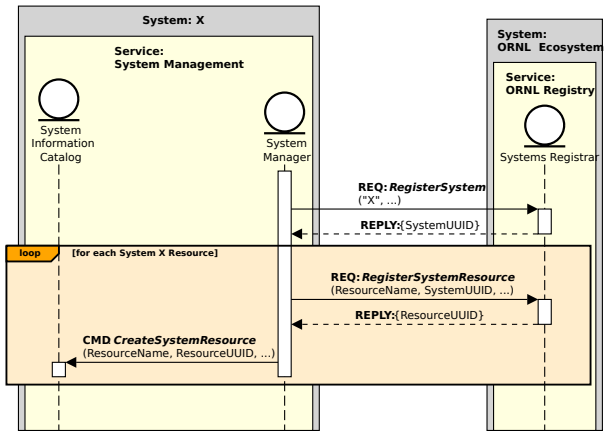

Fig. 5. INTERSECT System Registration Sequence

B. SDK Enhancements

Many of the microservice capabilities in the INTERSECT Architecture Specification have dependencies on other capabilities. The most common form of dependency is a *required* capability, which means that the current microservice must make requests of the dependent capability in order to fully support its own functionality. The other form of dependency is a capability *extension*, which means that the current microservice provides functionality that extends that of the dependent (e.g., the `ComputeQueueReservation` capability adds reservation functionality to the the `ComputeQueue` capability [9]). For extensions, the base capability is assumed to be provided by the same parent service.

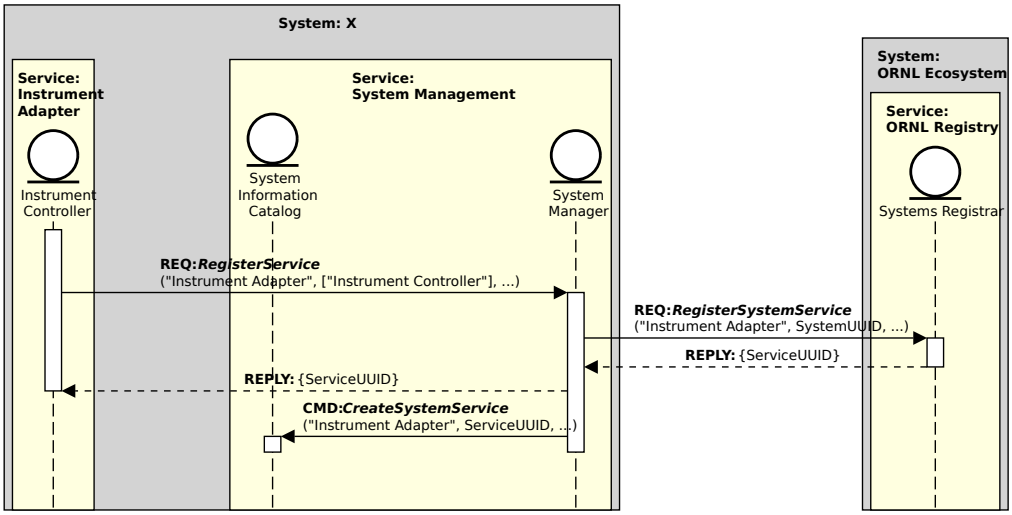

Fig. 6. INTERSECT Service Registration Sequence

Although the SDK supports all three microservice interaction patterns beginning with the release of version 0.6.0, there are still two limitations for users that wish to develop services similar to the *System Management* service. First, due to its focus on client-server interactions, the current SDK does not provide the mechanisms for a service to make a request of another service and process any response. This prevents implementation of services containing capabilities with required dependencies. Second, an `IntersectService` assumes it provides a single microservice capability, which precludes the implementation of microservice capability extensions or services providing multiple capabilities for ease of deployment.

To address these limitations and enable our demonstration of INTERSECT microservice capabilities for system registration and management, the SDK has been enhanced to provide the needed functionality, which we refer to as *service-to-service requests* and *multi-capability services*. The current implementation of these enhancements is available on the `mjb-arch-sdk` branch of the public code repository.

Service-to-service requests are enabled through the addition of a structure mapping unique request identifiers to a newly defined external request object. Any capability implementation can call the `create_external_request()` method on its parent service to create a unique external request from a client message and an optional response handler function. An additional thread of execution in the service periodically scans the external requests and identifies the newly added requests to send. The unique request identifier is included in the outgoing request message and the target service includes the identifier in the associated response message. Responses are received via a separate message broker channel than that used for incoming requests. If a response handler function is provided, the external request thread calls the function with the response message when it is received.

Multi-capability services are supported by initializing each `IntersectService` with a set of capability implementations rather than a single one. To map service requests to a specific capability, each capability implementation must now specify its advertised name and client messages prefix the exposed capability method name with the advertised name (e.g., `AdvertisedCapability.method_name`) in their target operation field. The SDK's automatic capability API inspection and schema generation was also updated to incorporate the advertised capability prefix. This use of advertised names aligns with the INTERSECT Architecture Specification's design for permitting multiple concurrent implementations of the same microservice contract.

C. Example Deployment of an Interconnected Science Ecosystem

We demonstrate the deployment of an ORNL-based INTERSECT ecosystem including three U.S. Department of Energy Office of Science user facilities, the Oak Ridge Leadership Computing Facility (OLCF), the Center for Nanophase Materials Science (CNMS), and the Spallation Neutron Source (SNS). Within each facility, we create an example INTERSECT system that represents a group of related resources. The OLCF 'frontier' system includes resources for the Frontier HPC system, the Orion parallel file system, and two file systems for user and project storage. The CNMS 'stem' system includes scanning transmission electron microscopy instruments. The SNS 'first-target-station' system has a sample of the various neutron scattering instruments.

All services and supporting infrastructure are deployed in a local workstation development environment. The demo deployment script and microservice capability implementations used in this demonstration are available from the associated development branch. First, we deploy the supporting infrastructure. The INTERSECT SDK control and data planes are started using the provided Docker Com-

pose configuration. An instance of the neo4j graph database is deployed via Docker and configured via its web interface for use as the backend technology to store information about the ecosystem. Next, we launch two ORNL domain services that provide the `SystemsRegistrar` and `EntityRelationCatalog` capabilities respectively. Finally, we launch the 'system-manager' service within the 'infrastructure-management' subsystem for each of the three facility systems. After each of the system management services has completed its initial registration sequence, we query the neo4j instance to obtain the ecosystem information hierarchy graph shown in Figure 7. Further details about each node or relation in the graph can be obtained by clicking on the target. Figure 8 shows the information available for a system management service node, which includes its supported capabilities.

V. Related Work

Frameworks for microservices development and deployment are increasingly popular for cloud-native applications. Service-mesh frameworks like Istio [16] automatically provide key functionality such as secure communication, service discovery, traffic observability, and horizontal scaling and load balancing that help developers focus on core microservice functionality. They are however unsuitable technology for federated ecosystems involving multiple organizations, as no single organization is trusted to serve as the authority for security, policy, and configuration decisions. Further, they are designed exclusively for cloud environments that are not widely deployed within HPC and scientific user facilities. The INTERSECT SDK may be considered an early-stage federated service mesh technology.

The Globus Compute platform [17] provides Python function-as-a-service and remote command execution capabilities that allow distributed computation on a wide variety of resources including user workstations, cloud environments, and HPC systems. To secure network communication and remote execution in federated ecosystems, it relies on the same Globus user authentication widely used for large-scale data transfers in scientific workflows. Globus Compute currently expects functions or commands to have relatively short lifetimes, and thus is not a good fit for deploying long-lived services for repeated use by many scientific workflows. However, Globus Compute may be useful for implementing certain INTERSECT computing capabilities (e.g., `HostCommandExecution`).

VI. Conclusion and Future Work

This paper has described the key design aspects for constructing INTERSECT ecosystems using microservices architecture and how those aspects are reflected in the current INTERSECT SDK. We discussed two current SDK deficiencies for developing microservice capabilities and our enhancements that address them. Finally, we demonstrated the use of prototype microservice capabilities for system registration and management to deploy an example multi-facility INTERSECT ecosystem.

The enhancements for service-to-service requests and multi-capability services are currently under review for incorporation in the next release of the INTERSECT SDK. We also plan to adapt the existing SDK-based services developed for the INTERSECT domain science use cases, including autonomous additive manufacturing and electron microscopy and automated chemical synthesis and characterization, to take advantage of these enhancements.

Work continues to develop prototype implementations for the full set of microservice capabilities defined in the INTERSECT Architecture Specification. In particular, near term efforts are targeting compute capabilities that leverage DOE computing facility APIs such as the NERSC SuperFacility API [18]. Longer term, we wish to create a community repository for INTERSECT microservice capability implementations and associated tooling the makes it easy to discover and integrate capabilities within user-developed services.

Acknowledgments

Research sponsored by the Laboratory Directed Research and Development Program of Oak Ridge National Laboratory, managed by UT-Battelle, LLC, for the U.S. Department of Energy. This research used resources from the ORNL Research Cloud Infrastructure at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725

References

[1] Oak Ridge National Laboratory, "INTERSECT — ORNL," July 2024. [Online]. Available: https://www.ornl.gov/intersect

[2] ——, "An Open Federated Architecture for the Laboratory of the Future — ORNL," July 2024. [Online]. Available: https://www.ornl.gov/project/open-federated-architecture-laboratory-future

[3] ——, "Software Development Environment Project — ORNL," July 2024. [Online]. Available: https://www.ornl.gov/project/software-development-environment-project

[4] ——, "Integration Project — ORNL," July 2024. [Online]. Available: https://www.ornl.gov/project/integration-project

[5] C. Engelmann, O. Kuchar, S. Boehm, M. J. Brim, T. Naughton, S. Somnath, S. Atchley, J. Lange, B. Mintz, and E. Arenholz, "The INTERSECT open federated architecture for the laboratory of the future," in *Accelerating Science and Engineering Discoveries Through Integrated Research Infrastructure for Experiment, Big Data, Modeling and Simulation*, vol. 1690. Springer, Cham, Aug. 24-25, 2022, pp. 173–190.

[6] W. H. J. Manthorpe Jr., "The emerging joint system of systems: A systems engineering challenge and opportunity for apl," *John Hopkins APL Technical Digest*, vol. 17, no. 3, pp. 305–313, Jul. 1996.

[7] M. W. Maier, "Architecting principles for system-of-systems," *Systems Engineering*, vol. 1, no. 4, pp. 267–284, Nov. 1998.

[8] INTERSECT SDK Developers, "INTERSECT-SDK," August 2024. [Online]. Available: https://github.com/INTERSECT-SDK/python-sdk

[9] M. J. Brim and C. Engelmann, "INTERSECT architecture specification: Microservice architecture (v.0.9)," Oak Ridge National Laboratory, Tech. Rep. ORNL/TM-2023/3171, 9 2023. [Online]. Available: https://www.osti.gov/biblio/2333815

[10] C. Engelmann and S. Somnath, "INTERSECT architecture specification: Use case design patterns (v.0.9)," Oak Ridge National Laboratory, Tech. Rep. ORNL/TM-2023/3133, 9 2023. [Online]. Available: https://www.osti.gov/biblio/2229218

[11] O. A. Kuchar, S. Boehm, T. Naughton III, S. Somnath, B. Mintz, J. Lange, S. Atchley, R. Srivastava, and P. Widener, "INTERSECT architecture specification: System-of-systems architecture (v.0.9)," Oak Ridge National Laboratory, Tech. Rep. ORNL/TM-2023/3168, 9 2023. [Online]. Available: https://www.osti.gov/biblio/2333813

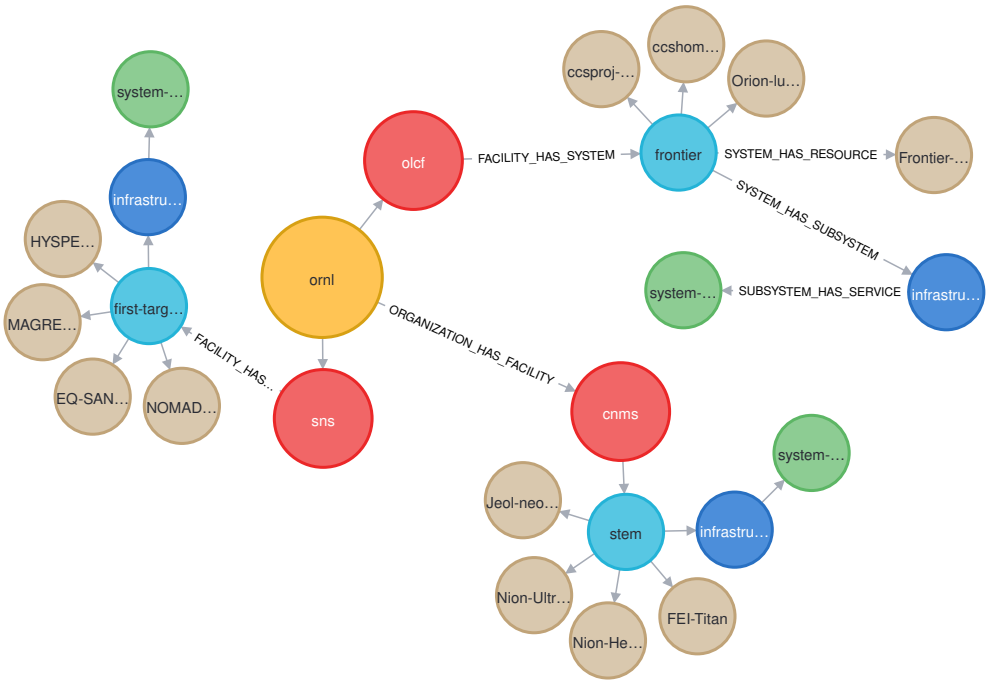

Fig. 7. Graph Database (neo4j) Rendering of Example Ecosystem Hierarchy

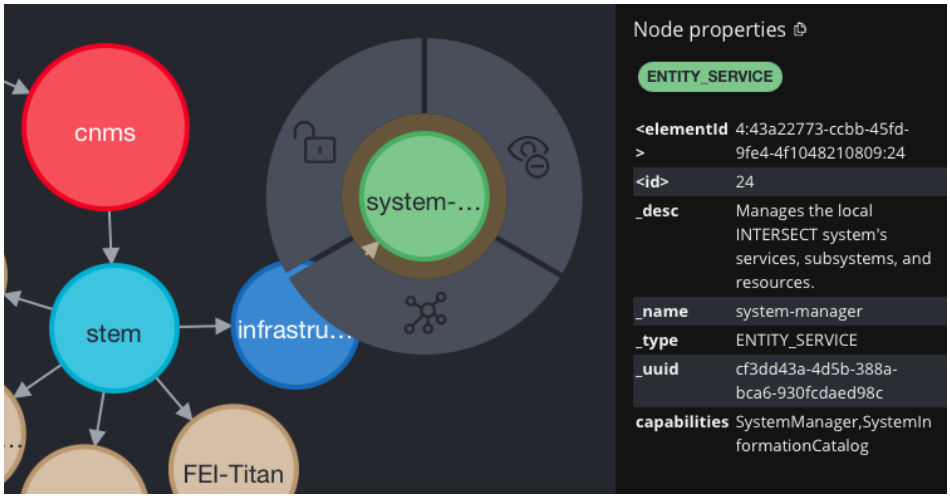

Fig. 8. Graph Database (neo4j) Information for Node Selection

[12] A. M. Thakur, S. Hitefield, M. McDonnell, M. Wolf, R. Archibald, L. Drane, K. Roccapriore, M. Ziatdinov, J. McGaha, R. Smith, J. Hetrick, M. Abraham, S. Yakubov, G. Watson, B. Chance, C. Nguyen, M. Baker, R. Michael, E. Arenholz, and B. Mintz, "Towards a software development framework for interconnected science ecosystems," in *Accelerating Science and Engineering Discoveries Through Integrated Research Infrastructure for Experiment, Big Data, Modeling and Simulation*. Cham: Springer Nature Switzerland, 2022, pp. 206–224.

[13] J. McGaha, M. McDonnell, L. Drane, S. Hitefield, G. Wiggins, G. Cage, R. Smith, M. Brim, M. Abraham, R. Archibald, and A. M. Thakur, "Enabling interconnected science workflows through an adapter approach," in *US Research Software Engineers Association Conference (US-RSE 23)*, Oct. 2023, p. 49.

[14] AsyncAPI Project a Series of LF Projects, LLC, "3.0.0 — AsyncAPI Initiative for event-driven APIs," July 2024. [Online]. Available: https://www.asyncapi.com/docs/reference/specification/v3.0.0

[15] The Kubernetes Authors, "Distribute Credentials Securely Using Secrets," August 2023. [Online]. Available: https://kubernetes.io/docs/tasks/inject-data-application/distribute-credentials-secure/

[16] Istio Authors, "Istio / What is Istio?" August 2024. [Online]. Available: https://istio.io/latest/docs/overview/what-is-istio/

[17] The University of Chicago, "Globus Compute 2.25.0 documentation," August 2024. [Online]. Available: https://globus-compute.readthedocs.io/en/latest/index.html

[18] National Energy Research Scientific Computing Center, "NERSC SuperFacility API - Swagger UI," August 2024. [Online]. Available: https://api.nersc.gov/api/v1.2/