The Interconnected Science Ecosystem (INTERSECT) Architecture

Christian Engelmann

Olga Kuchar, Swen Boehm, Michael Brim, Jack Lange, Thomas Naughton, Patrick Widener, and Ben Mintz

Alumni: Rohit Srivastava, Suhas Somnath, Scott Atchley, and Elke Arenholz

Contact: Christian Engelmann
engelmannc@ornl.gov

ORNL is managed by UT-Battelle LLC for the US Department of Energy
INTERSECT Initiative: Exascale System to Ecosystem

**Goal**
- Develop a scalable, integrated, and interoperable software framework to enable autonomous workflows, experiments, and smart connected ORNL laboratories

**Approach**
- Develop an open architecture
- Develop and integrate common software frameworks, tools, and services
- Demonstrate use cases to drive and exercise INTERSECT

**Successes**
- 20+ papers submitted and/or accepted
- 10+ software artifacts
- 6 INTERSECT demos including:
  - Autonomous Electron Microscope control loop
  - Digital twin for additive manufacturing
  - Automated flow chemistry
- Position ORNL for future DOE ecosystem development

---

**Software Development Frameworks and Services**

<table>
<thead>
<tr>
<th>Ecosystem Services</th>
<th>Control Plane</th>
<th>Data Plane</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exposing Resources</td>
<td>Message Abstraction Layer</td>
<td>Scientific Data Layer</td>
</tr>
<tr>
<td>Workflow Status / Adapter</td>
<td>MQTT</td>
<td>Other</td>
</tr>
<tr>
<td>Approval Status / Adapter</td>
<td>Data/Compute Services</td>
<td>Groom</td>
</tr>
<tr>
<td>Contingency Status / Adapter</td>
<td>Authent. / Author</td>
<td>Ponder</td>
</tr>
<tr>
<td>Autonomy Status / Adapter</td>
<td>Cyber Monitor</td>
<td>Other</td>
</tr>
</tbody>
</table>

**Integrate Frameworks and Capabilities into Autonomous Laboratories and Facilities**

- Autonomous chemistry lab for CO₂ conversion
- Autonomous electron microscopy for quantum materials

---

**Open Architecture Specifications**

- Message Abstraction Layer
- Scientific Data Layer
- Control Plane
- Data Plane

---

**Interact with INTERSECT into Future OLCF Ecosystems**

- Incorporate INTERSECT into future OLCF Ecosystems

---

**Ben Mintz, CS Director**
**Rob Moore, Exp Director**
Autonomous Experiments Today

~64 MB/s
“Streaming” Edge
FPGA*
Jetson Nano

~ MB-GB/run
GPU “Far” Edge
DGX-2
16 GPUs

~ 10+GB/job
Leadership Class
Summit
~27000 GPUs

Simulations and Model Refinements

Feedback for control

Instrument

Single Independent Smart Labs

*Image from National Instruments
Autonomous Labs of the Future

Interoperable Ecosystem is Required

Autonomous Materials Lab

Autonomous Electric Grid Lab

Interconnected Smart Labs

Autonomous Microscopy Lab
Scientist Really Want ...

Hey Google

“Hey Oakley,
Help me solve a science problem!”
Reality of Autonomous Laboratories

Human-Machine Interfaces provide data vis, instrument status and command/control

Software interprets input and manages experiments, processes, and labs

Data analysis, simulations, modeling using edge computing and high-performance computing

Results

Autonomous Feedback

Data Output

Synthesis

Multi-Modal Characterization

Manufacturing

Complex real-world systems

Broad Science Applications

Ecosystem solutions must be Scalable, Flexible, and Interoperable
INTERSECT Programmatic Structure

**Domain Science Projects**

- **Autonomous Chemistry Lab**
- **AutoflowS**
- **Autonomous Microscopy**
- **Additive Manufacturing**
- **Quantum Accelerator**
- **Electric Grid**

**Crosscutting Projects**

- **INTERSECT Architecture**
  - Develop an open architecture that is scalable across scientific domains
- **INTERSECT Software Development Framework**
  - Develop the software framework and tools required to interconnect systems
- **INTERSECT Integration**
  - Integrate networks, systems, and software across all projects

**Establish an autonomous robotic chemistry lab for catalytic synthesis that operates 24/7**

**Develop an autonomous flow chemistry system by combining in-situ analysis capabilities with AI enabled feedback**

**Establish data streaming, on-the-fly data analysis and simulation for AI enabled feedback for microscopes at CNMS**

**Autonomous additive manufacturing (AM) by combining AM build system, in-situ analysis, and on-the-fly simulations**

**Integrate a trapped ion quantum resource into the INTERSECT ecosystem for use as a quantum accelerator**

**Establish a scalable platform for hardware in the loop emulation of large-scale power grids to test new power controllers**
Guided by History

- Future Combat Systems (FCS) was the United States Army's principal modernization program from 2003 – 2009.
- The Boeing Company and Science Applications International Corporation (SAIC) worked together as the lead systems integrators, coordinating more than 550 contractors and subcontractors in 41 states.
- Estimated program losses range from $18-32B.
- RAND Analysis of FCS (https://www.rand.org/pubs/monographs/MG1206.html)
  - “an industry consortium led by Boeing and SAIC was effectively put in charge of overseeing its own performance”
  - Entrenched communities were evident in the FCS program.
  - Overreliance that the acquisition community could develop and integrate items using both evolutionary and unknown revolutionary technologies.
  - An emphasis on the integration of technologies and advanced concepts allows the enforcement of system-of-systems discipline and curbs conflicting influences.
  - FCS involved the largest integrated set of requirements the Army had ever developed, and it was extremely difficult to analyze and understand precisely how all of them would interoperate.
  - Significant technology development should occur early in a program.
  - Alternative technology assessment metrics can supplement technical readiness levels, which may be inadequate for some aspect of system-of-systems acquisitions.
  - Having too many connections to or being too highly dependent on outside programs can lead to significant risk.
  - Risk-mitigation strategies that incorporate system-of-systems engineering practices will facilitate risk mitigation across systems.

No single team has all the answers!
Early user engagement/adoption is critical!
Wild-wild west integration does not work!
Early technologies should focus integration!
Interoperability must be a primary goal!
Reuse software! Continuous Dev/Integration!
Technologies must be interchangeable!
Define compatibility or compliance!
Good engineering practices are key!
INTERSECT Software Ecosystem

Development and Operations (DevOps) Env.
• Replicates operational environment for sandbox software development
Adopter’s Web Portal
• Easy access to software capabilities

Smart Lab Marketplace

Abstract Service Bus and Common Messages
• System and Software Interoperability
• Software Reuse
Microservice Architecture
• Breaks Monolithic Software
• Incremental Software Development and/or Updates
• Reuse Individual Services

Ecosystem Software Services

Abstract Service Bus

Control Plane
Message Abstraction Layer
MQTT
RestAPIs

Data Plane
Scientific Data Layer
Globus
Stream Data

Exposing Resources
Orchestration
Autonomy / Automation
Cybersecurity

System Adapter
Subsystem Adapter
Workflow Manager
Approval Manager
Contingency Manager
Autonomy

Web Portal
Storage
Movement
AI Analysis
Simulation

Other

Cyber Monitor

Web Portal

DevOps Tools
Web Portal
Market Webpage

Containers
CI/CD

Cybersecurity
Authent. / Author.

Adopter's Web Portal
• Easy access to software capabilities

Messaging and Data Services

• Rapid Integration of New Technology with Limited Software Rewrite
Standard Requirements
• Interoperability Across Implementations
INTERSECT Architecture Overview

INTERSECT Architecture

Science Use Case Design Patterns
- Strategy patterns
- Experiment control
- Experiment steering
- Design of experiments
- Multi-experiment workflow
- Architectural Patterns
- Local vs. distributed

System of Systems Architecture
- User view
- Data view
- Logical view
- Physical view
- Operational view
- Standards view

Microservice Architecture
- Interaction patterns
- Capabilities catalog
- Orchestration and deployment patterns

Interconnected Smart Laboratories
- Automated/Autonomous
- Robot-controlled
- Chemistry
- Microscopy
- Additive Manufacturing
- Electric Grid Emulation

INTERSECT Software Development Kit

INTERSECT Integration

Science Use Case Design Patterns

System of Systems Architecture

Microservice Architecture

Interconnected Smart Laboratories

- Automated/Autonomous
- Robot-controlled
- Chemistry
- Microscopy
- Additive Manufacturing
- Electric Grid Emulation
The INTERSECT Open Architecture Specification
A written documentation of the INTERSECT Architecture, like a blueprint

• **Science Use Case Design Pattern Specification**
  – Abstract descriptions of the involved hardware and software components and their work, data and control flows.

• **System of Systems Architecture Specification**
  – Detailed design decisions about the involved hardware and software components from different points of view (e.g., logical, physical, operational, data, …)

• **Microservice Architecture Specification**
  – Detailed design decisions about software microservices, including their functionalities, capabilities, compositions, with control, work, and data flows.

• Current approach: 3 ORNL reports (PDF) released in intervals
Agile Development of the INTERSECT Architecture

- Iteratively develop and refine the INTERSECT Architecture
- Interact with the Software Development Kit, Integration and Domain Science Projects for
  - Requirements analysis
  - Feedback on drafts and releases
  - Assuring architecture compliance
  - Understanding implementation nuances

Fine-Grain Cycle for Specification Document Draft
Coarse-Grain Cycle for Specification Document Release
• Abstract descriptions of the involved hardware and software components and their work, data and control flows.
Why Design Patterns?

• Systematize recurring problems by describing generalized solutions based on best practices
• Offer solution templates to solve specific problems that may apply to different situations
• Provide different solution alternatives to specific problems
• Identify the key aspects of solutions and create abstract descriptions to develop reusable design elements
• Communicate problems and solutions with clear terms and abstract concepts
Science Use Case Design Patterns: Anatomy

• **Approach: Focus on the control problem**
  - Open vs. closed loop control
  - Single vs. multiple experiment control
  - Steering vs. designing experiments
  - Local vs. remote compute in the loop

• Universal patterns that describe solutions free of implementation details

• Patterns may exclude each other or may be combined with each other

• Described pattern properties:
  - Name, Problem, Context, Forces, Solution, Capabilities, Resulting Context, Related Patterns, Examples, and Known Uses

Figure: Single experiment control

Figure: Multi-experiment control
Science Use Case Design Patterns: Classification

- **Strategy patterns:** High-level solutions with different control features
- **Architectural patterns:** More specific solutions using different hardware/software architectural features

![Pattern classification scheme](image)
Science Use Case Design Patterns: Strategy Patterns

### Experiment Control
- **Executes an existing plan**
  - Open loop control
  - Automated operation
- **Executes an existing plan, depending on progress**
  - Closed loop control
  - Autonomous operation
  - Extends patterns:
    - Experiment Control

### Experiment Steering
- **Executes an existing plan**
  - Safety-Related Feedback Only
- **Executes an existing plan, depending on progress**
  - Safety-/Progress-Related Feedback

### Design of Experiments
- **Creates/executes a plan, based on prior result**
  - Closed loop control
  - Autonomous operation
  - Uses patterns:
    - Experiment Control
  - May use patterns:
    - Experiment Steering

### Multi-Experiment Workflow
- **Executes existing plans (workflow of experiments)**
  - Open loop control
  - Automated operation
  - Uses patterns:
    - Experiment Control
  - May use patterns:
    - Experiment Steering
    - Design of Experiments
Science Use Case Design Patterns: Architectural Patterns
Local vs. Distributed Experiment Control

Figure: Local Experiment Control

Figure: Distributed Experiment Control
Science Use Case Design Patterns: Architectural Patterns
Local vs. Distributed Experiment Steering

Figure: Local Experiment Steering

Figure: Distributed Experiment Steering
Science Use Case Design Patterns: Architectural Patterns
Local vs. Distributed Design of Experiments

Figure: Local Design of Experiments

Figure: Distributed Design of Experiments
Science Use Case Design Patterns: Architectural Patterns
Local vs. Distributed Multi-Experiment Workflow

**Figure: Local Multi-Experiment Workflow**

- Multi-Experiment Workflow Plan
- Multi-Experiment Workflow Controller
- Experiment Controller 1
- Local Test 1
  - Safety-Related Feedback Only
  - Experiment Plan 1
- Experiment Controller n
- Local Test n
  - Safety-Related Feedback Only
  - Experiment Plan n
- (Local or Distributed) Experiment Control

**Figure: Distributed Multi-Experiment Workflow**

- Multi-Experiment Workflow Plan
- Multi-Experiment Workflow Controller
- Experiment Controller 1
- Local Test 1
  - Safety-Related Feedback Only
  - Experiment Plan 1
- Experiment Controller n
- Local Test n
  - Safety-Related Feedback Only
  - Experiment Plan n
- Remote Test n
- (Local or Distributed) Experiment Control

Start → Local Experiment 1 → End
Local Experiment 2 → End
Local Experiment 3 → End
Local Experiment 4 → End
Local Experiment 5 → End

Start → Local Experiment 1 → End
Local Experiment 2 → End
Local Experiment 3 → End
Local Experiment 4 → End
Remote Experiment 5 → End
Science Use Case Design Patterns: Compositions

Figure: Strategy pattern composition

Figure: Architectural pattern composition
System of Systems Architecture Specification

- Detailed design decisions about the involved hardware and software components from different points of view.
Why System of Systems?

Common Architecture Elements

Independent systems enable systematic growth and eliminates monolithic systems.

Well defined/common interfaces enable rapid integration and digital twin simulations.

Common Messages

**System**
- SystemStatus
- SystemControlStatus
- SystemControlRequest
- SystemControlRequestStatus
- SystemTask
- SystemTaskStatus

**Subsystem**
- SubsystemStatus
- SubsystemControlRequest
- SubsystemControlRequestStatus
- X_Capability
- X_CapabilityStatus
- X_CapabilityCommand
- X_CapabilityCommandStatus
- X_CapabilityActivity

**Component**
- ComponentControlStatus
- ComponentCommand
- ComponentCommandStatus

Enable **Scalable, Flexible, and Interoperable** Development, Deployment and Operation
System of Systems Architecture Views

- User View
- Data View
- Operational View
- Logical View
- Physical View
- Standards View
## System of Systems Architecture: Stakeholder Roles

### Stakeholder Scenarios for Using this Document

For each role identified in Section 1.3.1, this section will contain a few short scenarios that explain how stakeholders in the role would use specific sections of the architecture document to help address their concerns.

### Table 1-1. Stakeholder roles and the views within this document.

<table>
<thead>
<tr>
<th>Role</th>
<th>User</th>
<th>Data</th>
<th>Operational</th>
<th>Logical</th>
<th>Physical</th>
<th>Standards</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application software developers</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Infrastructure software developers</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>End users</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Application and platform hardware engineers</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Security Engineers</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Communications engineers</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>System-of-system engineers</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Chief engineer/scientists</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Lead System Integrator</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>System Integration and test engineers</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>External test agencies</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Operational system managers</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Figure: A federated ecosystem for autonomous experiments and self-driving labs with a system of systems architecture*
System of Systems Architecture: Logical View

- Captures the logical composition of systems and their relationships and interactions
- Includes:
  - Definition of system concepts
  - Definition of system options
  - System resource flow requirements capture
  - Capability integration planning
  - System integration management
  - Operational planning

Figure: Relationships between infrastructure and logical systems and their services
System of Systems Architecture: Data View

- **Highlights the system's data needs and framework**
- Includes data flow between systems and data definitions, schemas and exchange sequence diagrams
- Does not include specifications for scientific, instrument, or experiment data

<table>
<thead>
<tr>
<th>Entity Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>User</td>
<td>A user of an INTERSECT-compliant system or application. May participate in authentication or authorization processes.</td>
</tr>
<tr>
<td>User Profile</td>
<td>Profile information (contact/address/miscellaneous) for an INTERSECT user.</td>
</tr>
<tr>
<td>Project</td>
<td>Accounting abstraction for resource allocation in an INTERSECT system.</td>
</tr>
<tr>
<td>Campaign</td>
<td>A collection of related experimental activity which uses INTERSECT resources. A Campaign is associated with a Project and may have multiple Users associated with it. Campaigns have explicit durations and discrete sets of resources assigned to them.</td>
</tr>
<tr>
<td>Campaign Result</td>
<td>Outcomes of INTERSECT Campaigns. There may be several different result states represented.</td>
</tr>
<tr>
<td>Campaign Error</td>
<td>‘Error’ outcomes for INTERSECT Campaigns. As with Campaign Result, there may be several different “flavors” of error/failure results.</td>
</tr>
<tr>
<td>Campaign Template</td>
<td>It may prove useful to memoize a Campaign structure as a template, so that it may be quickly replicated by users. Such replicated new Campaigns are assigned the templated INTERSECT resources.</td>
</tr>
<tr>
<td>Recipe</td>
<td>Users may also wish to reuse resource structures at a finer granularity than Campaign. Recipes allow this usage to be memoized.</td>
</tr>
<tr>
<td>Approved User Resources</td>
<td>Resource allocations are tracked with approval durations for each of Users, Administrators, and Operators.</td>
</tr>
<tr>
<td>Approved Administrator Resources</td>
<td></td>
</tr>
<tr>
<td>Approved Operator Resources</td>
<td></td>
</tr>
<tr>
<td>INTERSECT Resource Type</td>
<td>Additional information about an INTERSECT resource.</td>
</tr>
<tr>
<td>INTERSECT Resource Action</td>
<td>Detail on the operations/functions available from a given INTERSECT resource.</td>
</tr>
<tr>
<td>INTERSECT Resources</td>
<td>Experimental/physical, computational, or virtual facilities available within the INTERSECT system or application.</td>
</tr>
<tr>
<td>Computational Resource</td>
<td>Additional information about computational resources available to the INTERSECT system or application.</td>
</tr>
<tr>
<td>Resource Support</td>
<td>An INTERSECT resource may be large and complex, requiring specialized support procedures and/or personnel for operation. Computational resources, for example, may have multiple such support staff, organized into tiers or functional areas.</td>
</tr>
<tr>
<td>Resource Capability</td>
<td>Resources provide INTERSECT capabilities, which allow them to be composed into systems and applications within the INTERSECT Architecture.</td>
</tr>
</tbody>
</table>

Table 6.1. Names and descriptions of INTERSECT architecture data entities
System of Systems Architecture: Operational View

- Captures the tasks, activities, procedures, information exchanges/flows from the perspective of operations stakeholders

- Does not include formats for data exchanges or details of user applications

Figure: Components, interfaces, and message sequences involved in system status monitoring
System of Systems Architecture: Physical View

- Captures the underlying system components from the perspective of resource managers/owners, system administrators, network engineers, and facility space managers
- Includes descriptions and definitions of physical systems, networks, connectivity and organizational boundaries
- Does not include specifications for instruments, resources, experiments and data
- Proprietary information is not part of the open architecture documentation!

Figure: Schematic representation of resources at Oak Ridge National Laboratory’s Spallation Neutron Source
System of Systems Architecture: User View

• **Captures user-facing functionality**

• Does not include system-internal interactions

• Described activities:
  - Logging into dashboard
  - Experiment creation
  - Start experiment
  - Steer experiment
  - Experiment end

• Includes examples for graphical user interfaces

---

Figure: Examples of graphical user interfaces for different user interactions
System of Systems Architecture: Standards View

- **Captures the various standards including instruments specific standards, messaging standards, and other external standards**

- Provides a table of supported standards and other views or architecture elements that are impacted by each standard

- Provides a block diagram to illustrate exactly where each standard impacts a given system

### Table 3: Example of messaging standards maintained in the standards view

<table>
<thead>
<tr>
<th>Name</th>
<th>Version</th>
<th>Affected Views</th>
<th>Affected Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTERSECT Core Messages</td>
<td>1.0</td>
<td>Data, Logical, Operational</td>
<td>Microservice Capabilities: All</td>
</tr>
<tr>
<td>Compute Allocation Capability</td>
<td>1.0</td>
<td>Data, Logical</td>
<td>Microservice Capabilities: Application Execution, Container Execution, Host Command Execution</td>
</tr>
<tr>
<td>Compute Queue Capability</td>
<td>1.0</td>
<td>Data, Logical</td>
<td>Microservice Capabilities: Compute Queue Reservation</td>
</tr>
<tr>
<td>NION Swift API</td>
<td>0.16.3</td>
<td>Logical, Operational</td>
<td>Systems: Electron Microscopes</td>
</tr>
<tr>
<td>Robot Operating System (ROS)</td>
<td>2.rolling</td>
<td>Logical, Operational</td>
<td>Systems: Additive Manufacturing</td>
</tr>
</tbody>
</table>
Microservice Architecture Specification

- Detailed design decisions about software microservices, including their functionalities, capabilities, compositions, with control, work, and data flows.
Microservice Architecture: Microservice Capabilities

- System consists of
  - Subsystems, resources, and services
- Subsystem consists of
  - Services and resources
- Service consists of
  - Microservice capabilities

![Diagram of systems, subsystems, services, and microservices]

**Capability: Unique Capability Name**

**Description:** A short summary description of the domain of interest for this capability and the provided functionality.

**Related Capabilities:** Where applicable, provides references to related capabilities.
- **Extends:** A list of base capabilities that the functionality of this capability extends. A service implementing this capability must also implement the base capabilities.
- **Requires:** A list of required capabilities that are necessary to implement the functionality of this capability. The required capabilities are most often provided by other services, but may be implemented in the same service.

**Custom Data Type:** Where applicable, provides definitions of new data types or structures.

**Interactions:**
- **Command**
  - **MethodName()**
    - **Purpose:** A short description of the purpose of the current command method.
    - **Command Data:** A list of input data for the current method formatted as:
      - `dataName (DataType)` : A description of the data, including any format or value constraints.

- **Request-Reply**
  - **MethodName()**
    - **Purpose:** A short description of the purpose of the current request method.
    - **Request Data:** A list of input data for the current method formatted as:
      - `dataName (DataType)` : A description of the data, including any format or value constraints.
    - **Reply Data:** A list of output data for the current method formatted as:
      - `dataName (DataType)` : A description of the data, including any format or value constraints.

- **Asynchronous Event**
  - **EventName**
    - **Purpose:** A description of the activity or state change that generates this event.
    - **Event Data:** A list of data for the current event formatted as:
      - `dataName (DataType)` : A description of the data, including any format or value constraints.
Microservice Architecture: Interaction Patterns

- **Command / Acknowledgement**
  - Responds immediately

- **Request / Reply**
  - Responds after fulfilling the request

- **Asynchronous Event**
  - Status update or event information

- Can be mapped to asynchronous and RESTful client-server communication
  - Microservice architecture does not force a specific implementation

Figure: Command/acknowledgement, request/reply and asynchronous event interaction patterns for microservices
Microservice Architecture: Capabilities Catalog

- **Example: Data Management**
  - **Data Transfer**
    - File Transfer
    - Block Data Transfer
    - Streaming Data Transfer
    - Multi-party Data Transfer
  - **Data Storage**
    - File System Storage
    - Key-value Storage
    - Object Storage
    - Relational Database
    - Non-relational Database
    - ...
Microservice Architecture: Orchestration and Deployment

• Microservice orchestration
  – Asynchronous messaging or/and RESTful services
  – Conductor vs. choreography

• Microservice deployment
  – Sidecar pattern, Ambassador Proxy, and Service Mesh deployment patterns
Current Status

• **INTERSECT Open Architecture Specification**
  - Design pattern catalog that covers the science use cases in the INTERSECT Initiative
  - System-of-systems architecture specification with elements, communication and interfaces and some command and control and resource triad specifications
  - Initial microservice architecture that covers some INTERSECT science use cases

• **v0.5 released as 3 ORNL reports in Sept. 2022 (v0.8 latest internal version)**
  - INTERSECT Architecture: Use Case Design Patterns
  - INTERSECT Architecture: System of Systems Architecture
  - INTERSECT Architecture: Microservices Architecture
Future Roadmap: Capabilities to be Targeted

- **Campaign orchestration** (distributed and federated) and management templates (workflow repository)
- **Data plane architecture** (storage, movement, catalog, indexing, metadata, provenance, and asset management)
- **Standards view**: Requirements for INTERSECT and domain-specific standards (APIs, messages, and data formats)
- Architecture support for multi-tenancy (**multi-user**) and federation (**multi-site**)
- **Distributed and federated monitoring** architecture (for reliability, availability, serviceability and cybersecurity)
- **Error handling** concepts and interfaces (detection, notification, and isolation)
- **Resilience** concepts and interfaces (error/failure detection, notification, and mitigation)
- **Cybersecurity** architecture, including identity management adapters and access controls
- INTERSECT **documentation portal** targeting different audiences (e.g., developers and users)
- **Architecture for graphical user interfaces** that are independent from the business logic
- INTERSECT as part of ORNL’s Integrated Research Infrastructure
Autonomous Microscopy: Science Goal
Autonomous Microscopy: Science Use Case Design Patterns

• Strategy Pattern
  – Experiment Steering
  – Control of an ongoing STEM experiment via analysis of periodic experimental data

• Architectural Pattern
  – Distributed Experiment Steering
  – Local control of an ongoing STEM experiment via remote analysis of periodic experimental data

Figure: Strategy pattern: Experiment Steering

Figure: Architectural pattern: Remote Experiment Steering
Questions?