turbineTelemetryService

Service Description

Abstract

This document describes the **turbineTelemetryService** service: an Arrowhead service that provides telemetry ingestion, retrieval and basic query capabilities for wind-turbine sensors (vibration, rotor speed, generated power, temperatures, and GPS position). The service is intended for both research/prototyping (e.g. supervised anomaly detection and predictive maintenance experiments) and light-weight operational monitoring within an Arrowhead-compliant system.

Contents

1 Overview

This document describes the **turbineTelemetryService** service. The service enables collection, storage (short-term), and query of telemetry samples produced by sensors attached to a wind turbine. It is intentionally implementation-neutral: it specifies abstract operations and data types that an Arrowhead-compatible provider should expose while leaving storage and transport details to concrete implementations.

Primary use-cases:

- Real-time ingestion of sensor samples from turbine edge gateways.
- Retrieval of the latest telemetry for monitoring dashboards.
- Time-window queries for analytics and model training.
- Lightweight health check (Echo).

The rest of this document is organized as follows. In Section 2, we describe the abstract message operations provided by the service. In Section **??**, we present the data types used by the mentioned operations.

1.1 Significant Prior Art

This service is inspired by typical industrial telemetry APIs and cloud telemetry ingestion patterns (message-based ingestion, time-series stores). It aligns with Arrowhead's service registration and discovery patterns and is suitable to integrate with data science experiments such as predictive maintenance models (vibration analysis, anomaly detection) commonly used in academic/industrial projects.

1.2 How This Service Is Meant to Be Used

A typical workflow:

1. A turbine edge gateway registers with the Arrowhead Service Registry and calls the `registerTelemetry` operation to push telemetry samples to the data ingestor.
2. Monitoring dashboards or analytics components call `getSensorData` for live displays.
3. Operators use the `echo` operation to verify liveness of the service.

1.3 Important Delimitations and Dependencies

- This SD defines abstract operations and message formats only. Persistence (time-series database, retention policies) and secure transport (TLS, OAuth2) are implementation details.
- Integration with Arrowhead Service Registry and Authorization Framework (secure token exchange) is assumed but not mandated at the protocol level.

2 Service Interface

This section describes the interfaces to the **turbineTelemetryService** service. Each subsection names an abstract operation, an input type and an output type, in that order. Input and output types are only denoted when accepted or returned, respectively, by the interface in question.

All abstract data types named in this section are defined in Section **??**.

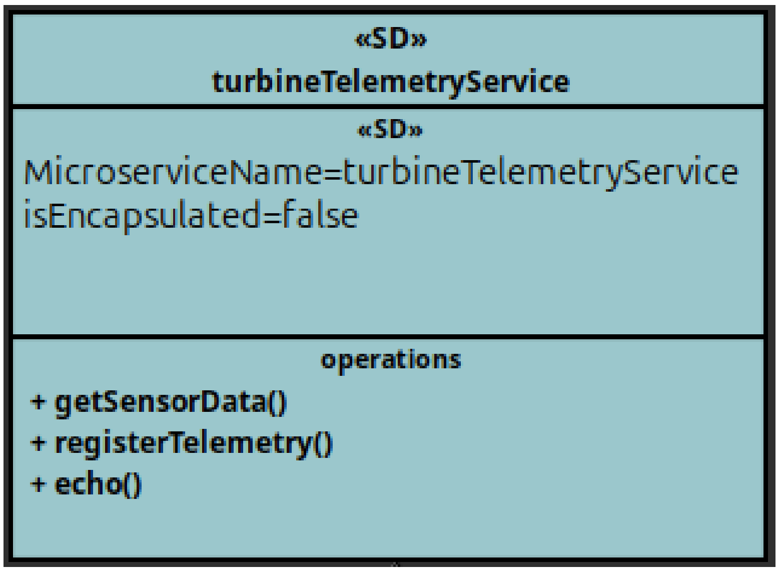

Figure 1: SysML block diagram example showing a monitoring consumer discovering the telemetry service via Arrowhead Service Discovery.

The following interface operations are available.

2.1 operation registerTelemetry (turbineTelemetryService) : turbineTelemetryService

The `registerTelemetry` operation is used by edge gateways or sensor agents to submit one or multiple telemetry samples. A `turbineTelemetryService` contains sensor identifiers, a UTC timestamp and a set of measured values (vibration, rotorSpeed, power, temperature, GPS coordinates, etc.). The service returns a `turbineTelemetryService` indicating acceptance and optionally an ingestion identifier.

2.2 operation getSensorData (turbineTelemetryService) : turbineTelemetryService

The `getSensorData` operation returns the most recent telemetry sample available. This operation is optimized for dashboard updates and health-checks.

2.3 operation echo () : StatusCodeKind

The `echo` operation provides a simple liveness check. The service returns a standard `StatusCodeKind` indicating OK or an error state.

3 Information Model

This section provides UML activity diagrams describing the behavior of each major operation of the **turbineTelemetryService** service. Each diagram illustrates how the service interacts with Arrowhead core systems and external clients during its execution.

3.1 Service Discovery and Authorization Workflow

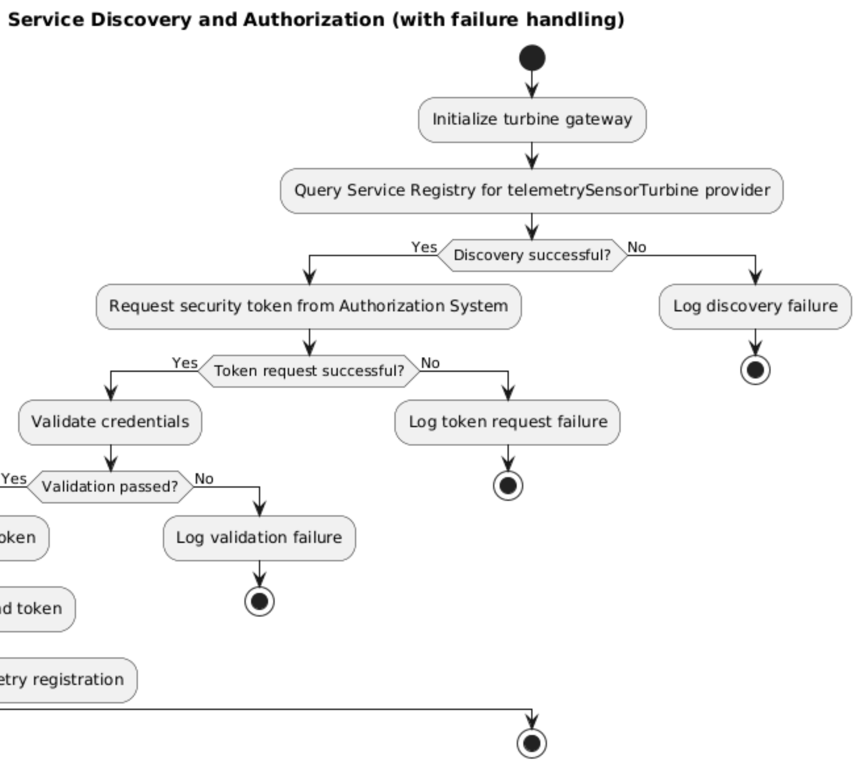
Service Discovery and Authorization (with failure handling)
Initialize turbine gateway
Query Service Registry for telemetrySensorTurbine provider
Yes
Discovery successful?
No
Request security token from Authorization System
Log discovery failure
Yes
Token request successful?
No
Validate credentials
Log token request failure
Yes
Validation passed?
No
Issue access token
Log validation failure
Store endpoint and token
Gateway ready for telemetry registration

Figure 2: Activity diagram showing service discovery and authorization prior to telemetry transmission.

Swimlanes: *Edge Gateway, Service Registry, Authorization System*

3.2 registerTelemetry Operation

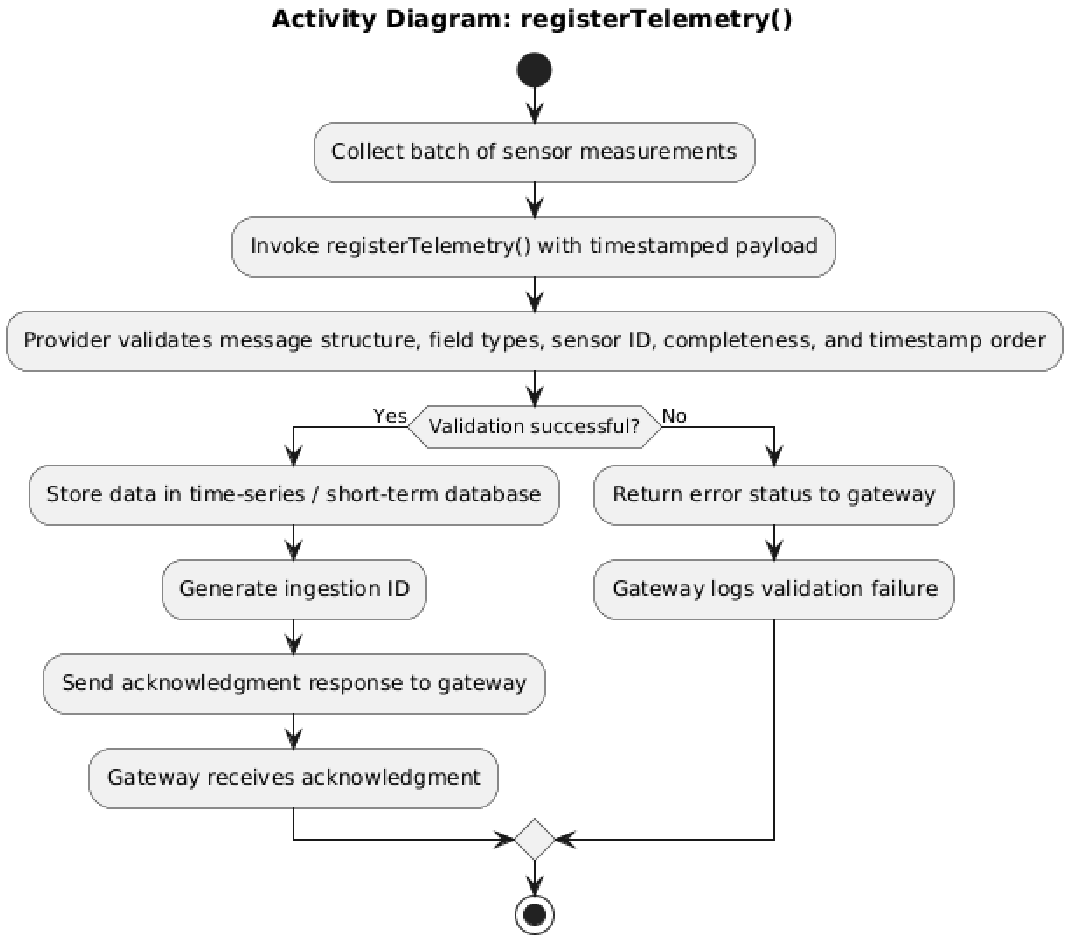
Activity Diagram: registerTelemetry()
Collect batch of sensor measurements
Invoke registerTelemetry() with timestamped payload
Provider validates message structure, field types, sensor ID, completeness, and timestamp order
Validation successful?
Yes
No
Store data in time-series / short-term database
Generate ingestion ID
Send acknowledgment response to gateway
Gateway receives acknowledgment
Return error status to gateway
Gateway logs validation failure

Figure 3: Activity diagram for the `registerTelemetry()` operation.

Swimlanes: *Edge Gateway, Telemetry Provider*

3.3 getSensorData Operation

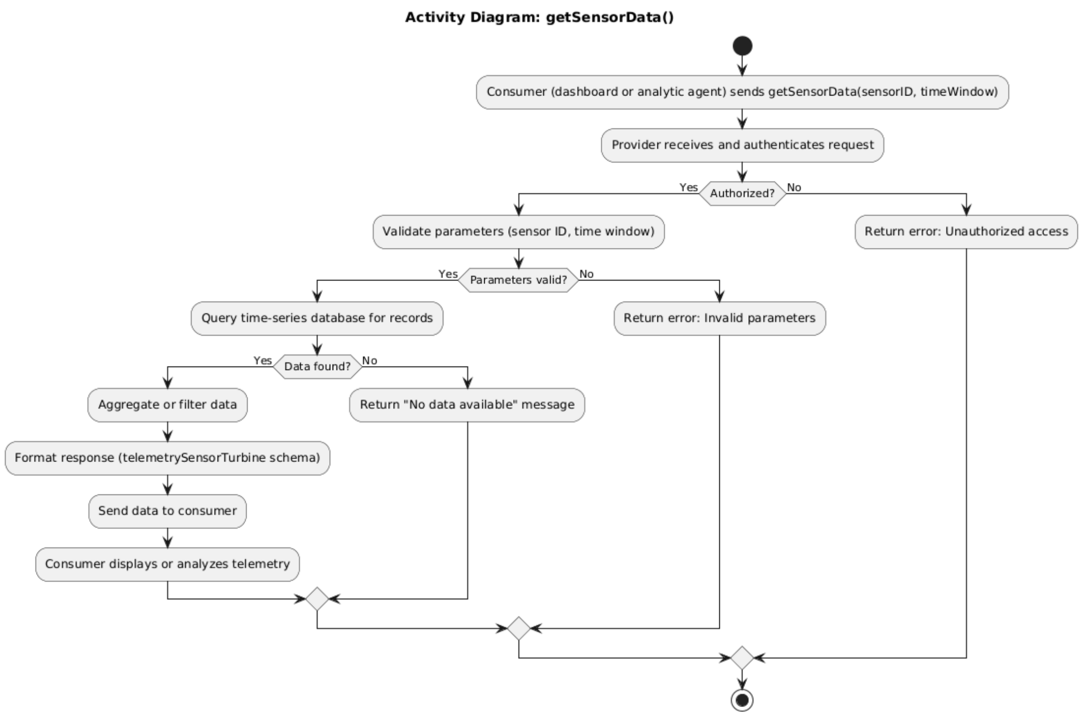
Activity Diagram: getSensorData()
Consumer (dashboard or analytic agent) sends getSensorData(sensorID, timeWindow)
Provider receives and authenticates request
Authorized?
Yes
No
Validate parameters (sensor ID, time window)
Return error: Unauthorized access
Parameters valid?
Yes
No
Query time-series database for records
Return error: Invalid parameters
Data found?
Yes
No
Aggregate or filter data
Return "No data available" message
Format response (telemetrySensorTurbine schema)
Send data to consumer
Consumer displays or analyzes telemetry

Figure 4: Simplified activity diagram for the `getSensorData()` operation.

Swimlanes: *Consumer (Dashboard/Analytics), Telemetry Provider*

3.4 echo Operation

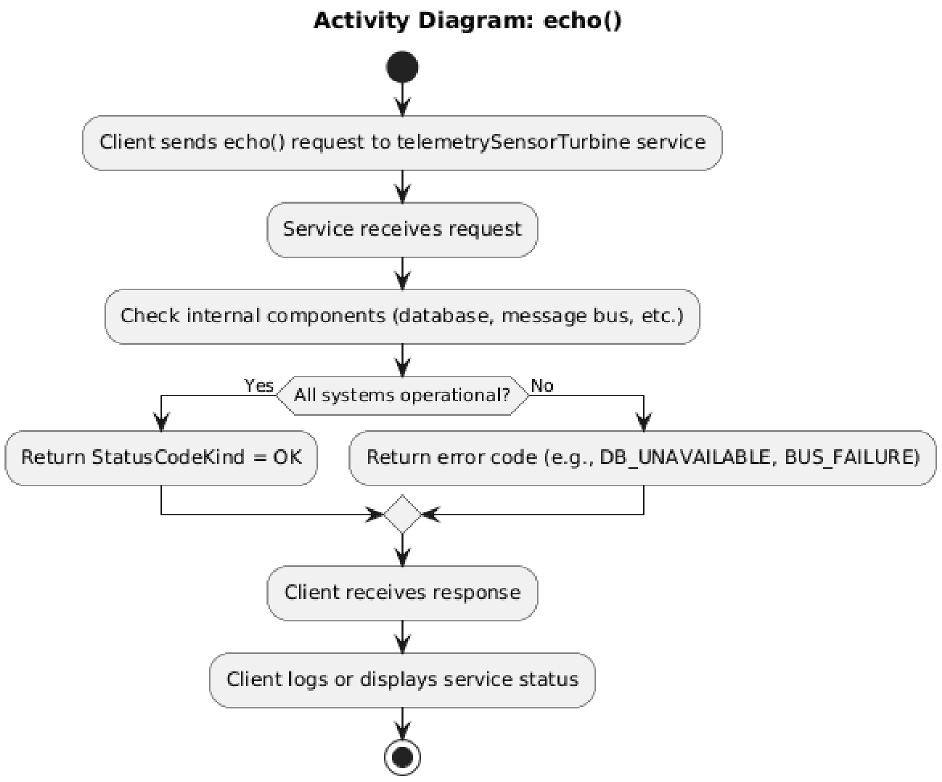
Activity Diagram: echo()
Client sends echo() request to telemetrySensorTurbine service
Service receives request
Check internal components (database, message bus, etc.)
All systems operational?
Yes
No
Return StatusCodeKind = OK
Return error code (e.g., DB_UNAVAILABLE, BUS_FAILURE)
Client receives response
Client logs or displays service status

Figure 5: Activity diagram for the `echo()` operation, showing service health verification and status reporting.

Swimlanes: *Any Client, Telemetry Provider*

3.5 Overall Lifecycle Overview

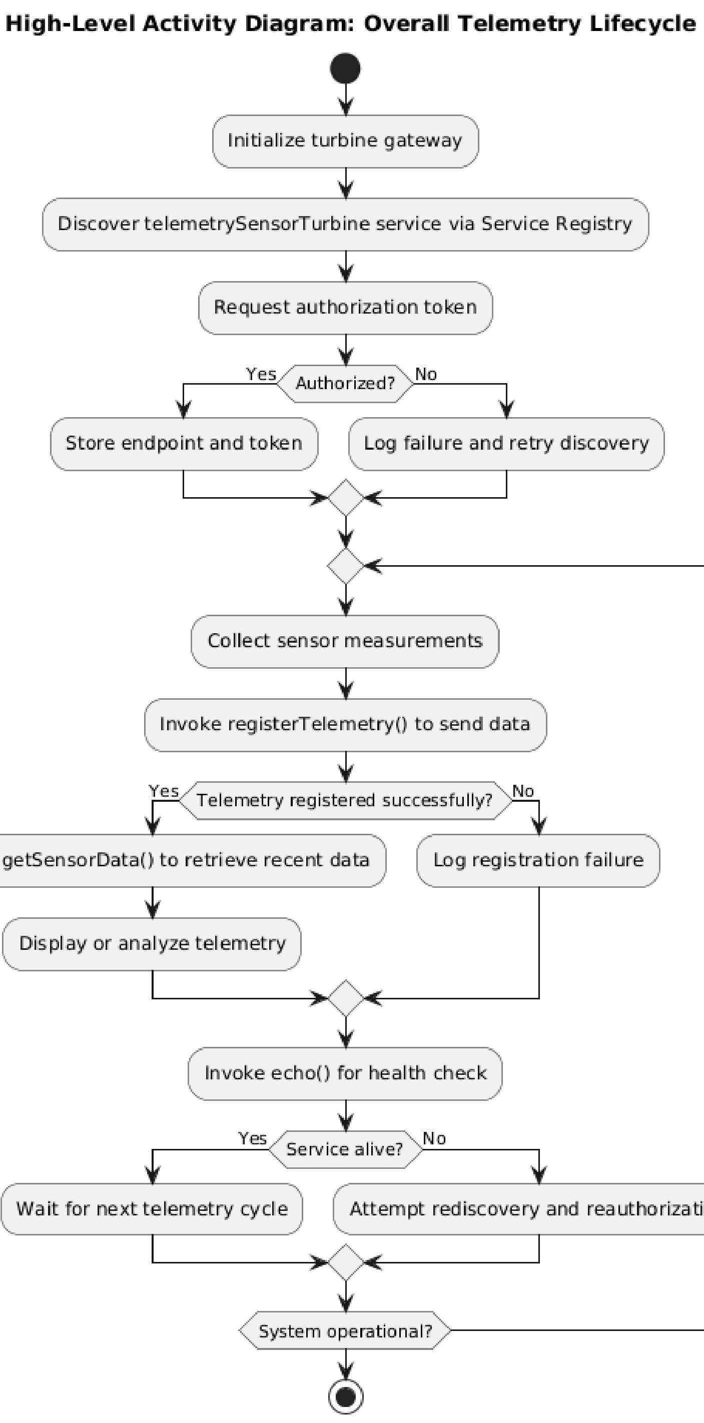
High-Level Activity Diagram: Overall Telemetry Lifecycle
Initialize turbine gateway
Discover telemetrySensorTurbine service via Service Registry
Request authorization token
Authorized?
Yes
No
Store endpoint and token
Log failure and retry discovery
Collect sensor measurements
Invoke registerTelemetry() to send data
Telemetry registered successfully?
Yes
No
Invoke getSensorData() to retrieve recent data
Log registration failure
Display or analyze telemetry
Invoke echo() for health check
Service alive?
Yes
No
Wait for next telemetry cycle
Attempt rediscovery and reauthorization
System operational?

Figure 6: High-level activity diagram summarizing the complete telemetry lifecycle: discovery, authorization, telemetry registration, retrieval, and health-check.