Unmanned Maritime Autonomy Architecture (UMAA)
Engineering Operations (EO)
Interface Control Document (ICD)
(UMAA-SPEC-EOICD)

Version 6.0

6 June 2024

Contents

List of Figures

List of Tables

1 Scope

1.1 Identification

This document defines a set of services as part of the Unmanned Maritime Autonomy Architecture (UMAA). The services and their corresponding interfaces covered in this ICD encompass the functionality to interact with the Hull, Mechanical & Electrical (HM&E) systems of an Unmanned Maritime Vehicle (UMV) (surface or undersea). As such, it includes low-level control and/or status of individual components on the unmanned vehicle such as the propulsor, battery, engine, gong, and fin. It includes physical constraints and specifications of the system, and its components. Also, it includes fault handling and health of the unmanned vehicle. Services provided here may be invoked by the Maneuver Operations ICD services depending on how those higher level services are implemented (i.e. either using intrinsic vehicle capabilities or interacting at the component level with the unmanned vehicle component services included herein). This document is generated automatically from data models that define its services and their interfaces as part of the Unmanned Systems (UxS) Control Segment (UCS) Architecture as extended by UMAA to provide autonomy services for unmanned vehicles.

To put each ICD in context of the UMAA Architecture Design Description (ADD), the UMAA functional decomposition mapping to UMAA ICDs is shown in Figure 1.

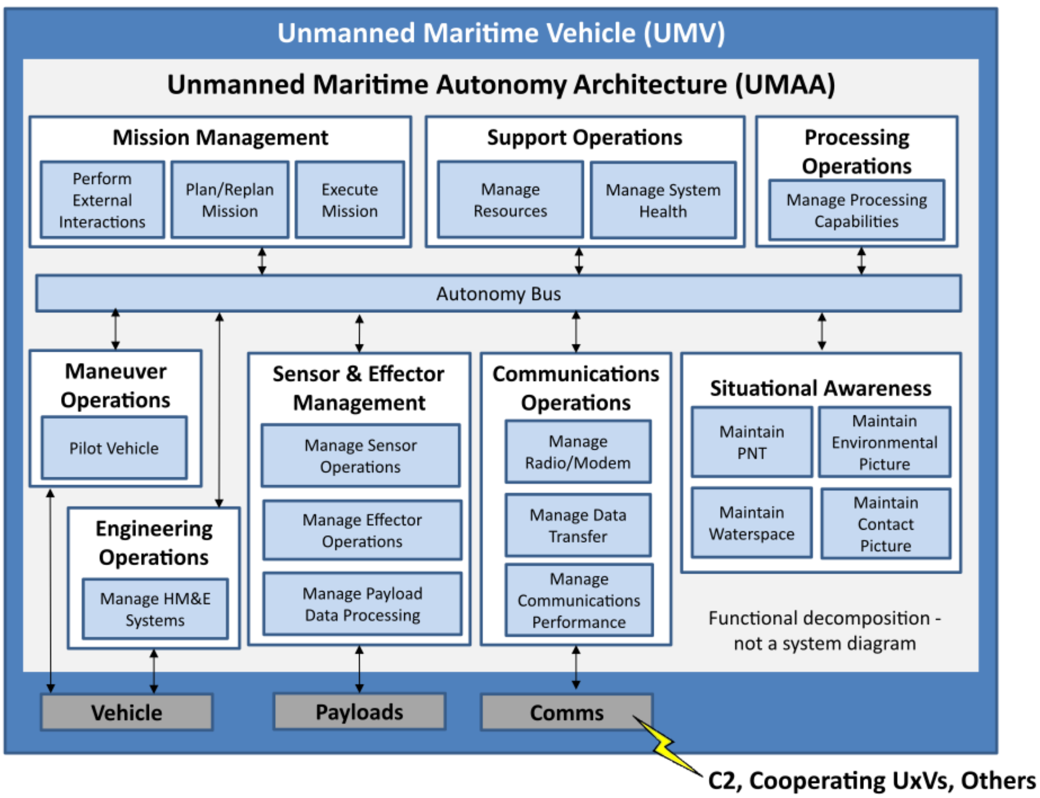

Figure 1: UMAA Functional Organization.

1.2 Overview

The fundamental purpose of UMAA is to promote the development of common, modular, and scalable software for unmanned vehicles that is independent of a particular autonomy implementation. Unmanned Maritime Systems (UMSs) consist of Command and Control (C2), one or more unmanned vehicles, and support equipment and software (e.g. recovery system, Post Mission Analysis applications). The scope of UMAA is focused on the autonomy that resides on-board the unmanned vehicle. This includes the autonomy for all classes of unmanned vehicles and must support varying levels of communication in mission (i.e., constant, intermittent, or none) with external systems. To enable modular development and upgrade of the functional capabilities of the on-board autonomy, UMAA defines eight high-level functions. These core functions include: Communications Operations, Engineering Operations, Maneuver Operations, Mission Management, Processing Operations, Sensor and Effector Operations, Situational Awareness, and Support Operations. In each of these areas, it is anticipated that new capabilities will be required to satisfy evolving Navy missions over time. UMAA seeks to define standard interfaces for these functions so that individual programs can leverage capabilities developed to these standard interfaces across programs that meet the standard interface specifications. Individual programs may group services and interfaces into components in different ways to serve their particular vehicle's needs. However, the entire interface defined by UMAA will be required as defined in the ICDs for all services that are included in a component. This requirement is what enables autonomy software to be ported between heterogeneous UMAA-compliant vehicles with their disparate vendor-defined vehicle control interfaces without recoding to a vehicle-specific interface.

Engineering Operations (EO) includes the services required to interact with the physical systems of an unmanned vehicle. Figure 2 depicts an example of various services in this EO ICD in relation to the maneuvering behavior services (in the Maneuver Operations ICD) and navigation sensing services (in the Situational Awareness ICD).

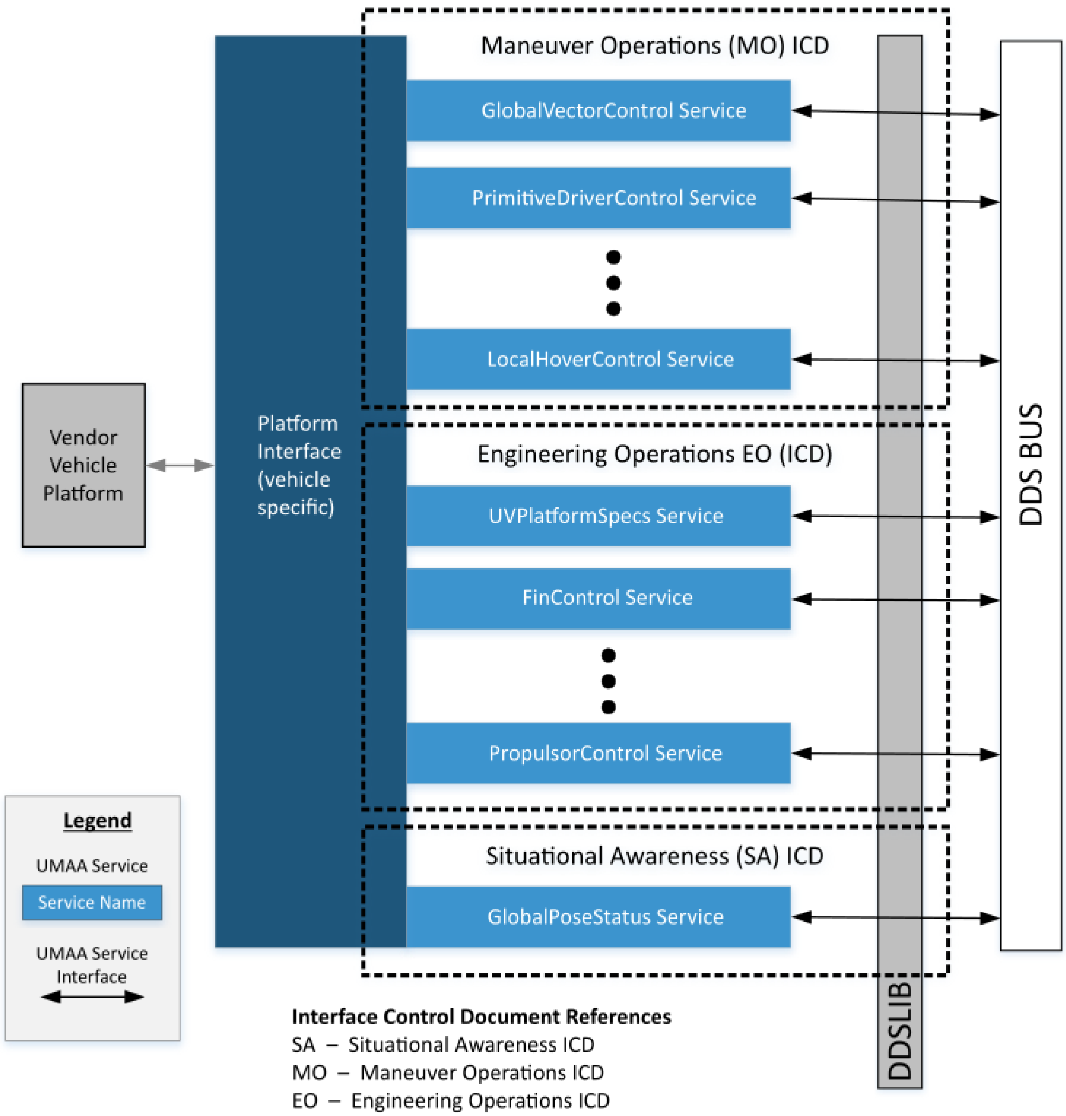

Figure 2: UMAA Services and Interfaces Example.

1.3 Document Organization

This interface control document is organized as follows:

Section 1 – Scope: A brief purview of this document

Section 2 – Referenced Documents: A listing of associated of government and non-government documents and standards

Section 3 – Introduction to Data Model, Services, and Interfaces: A description of the common data model across all services and interfaces

Section 4 – Introduction to Coordinate Reference Frames and Position Model: An overview of the reference frame model used by UMAA

Section 5 – Flow Control: A description of different flow control patterns used throughout UMAA

Section 6 – Engineering Operations (EO) Services and Interfaces: A description of specific services and interfaces for this ICD

2 Referenced Documents

The documents in the following table were used in the creation of the UMAA interface design documents. Not all references may be applicable to this particular document.

Table 1: Standards Documents

Title	Release Date
A Universally Unique IDentifier (UUID) URN Namespace	July 2005
Data Distribution Service for Real-Time Systems Specification, Version 1.4	March 2015
Data Distribution Service Interoperability Wire Protocol (DDSI-RTPS), Version 2.3	April 2019
Object Management Group Interface Definition Language Specification (IDL)	March 2018
Extensible and Dynamic Topic Types for DDS, Version 1.3	February 2020
UAS Control Segment (UCS) Architecture, Architecture Description, Version 2.4	27 March 2015
UCS Architecture, Conformance Specification, Version 2.2	27 September 2014
UCS-SPEC-MODEL v3.4 Enterprise Architect Model	27 March 2015
UCS Architecture, Architecture Technical Governance, Version 2.5	27 March 2015
System Modeling Language Specification, Version 1.5	May 2017
Unified Modeling Language Specification, Version 2.5.1	December 2017
Interface Definition Language (IDL), Version 4.2	March 2018
U.S. Department Of Homeland Security, United States Coast Guard "Navigation Rules International-Inland" COMDTINST M16672.2D	March 1999
IEEE 1003.1-2017 - IEEE Standard for Information Technology–Portable Operating System Interface (POSIX(R)) Base Specifications, Issue 7	December 2017
Guard, U. C. (2018). Navigation Rules and Regulations Handbook: International—Inland. Simon and Schuster.	June 2018
Department of Defense Interface Standard: Joint Military Symbology (MIL-STD-2525D Appendix A)	10 June 2014
DOD Dictionary of Military and Associated Terms	August 2018

Table 2: Government Documents

Title	Release Date
Unmanned Maritime Autonomy Architecture (UMAA) Architecture Design Description (ADD), Version 1.0	January 2019
Manual for the Submission of Oceanographic Data Collected by Unmanned Undersea Vehicles (UUVs)	October 2018

3 Introduction to Data Model, Services, and Interfaces

3.1 Data Model

A common data model is at the heart of UMAA. The common data model describes the entities that represent system state data, the attributes of those entities and relationships between those entities. This is a "data at rest" view of system-level information. It also contains data classes that define types of messages that will be produced by components, or a "data in motion" view of system-level information.

The common data model and coordinated service interfaces are described in a Unified Modeling Language (UML^{TM}) modeling tool and are represented as UML^{TM} class diagrams. Interface definition source code for messages/topics and other interface definition products and documentation will be automatically generated from the common data model so that they are consistent with the data model and to ensure that delivered software matches its interface specification.

The data model is maintained as a Multi-Domain Extension (MDE) to the UCS Architecture and will be maintained under configuration control by the UMAA Board as UCSMDE and will be incrementally integrated into the core UCS standard. Section 6 content is automatically generated from this data model, as are other automated products such as IDL that are used for automated code generation.

3.2 Definitions

UMAA ICDs follow the UCS terminology definitions found in the UCS Architecture Description v2.4. The normative (required) implementation to satisfy the requirements of a UMAA ICD is to provide service and interface specification compliance. Components may group services and required interfaces in any manner so long as every service meets its interface specifications. Figure 3 shows a particular grouping of services into components. The interfaces are represented by the blue and green lines and may equate to one or more independent input and output interfaces for each service. The implementation of the service into software components is left up to the individual system development. Given this context, section 6 correspondingly defines services with their interfaces and not components.

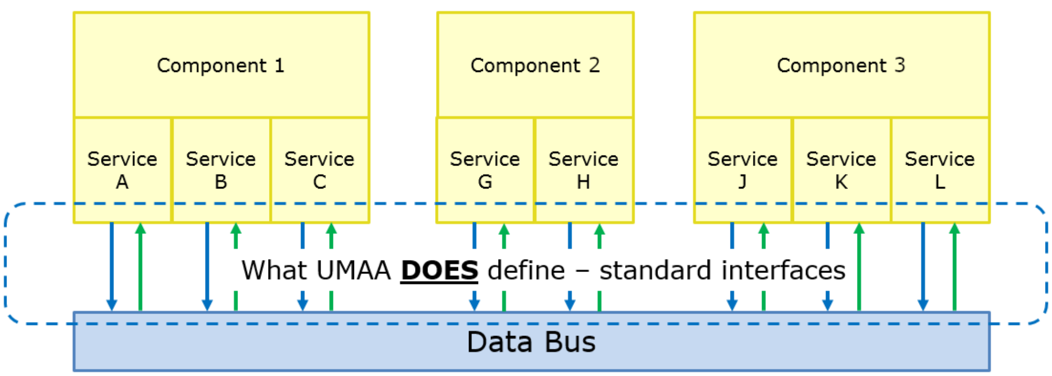

Figure 3: Services and Interfaces Exposed on the UMAA Data Bus.

Services may use other services within this ICD, or in other UMAA defined ICDs, to provide their capability. Additionally, components for acquisition and development may span multiple ICDs. An example of this would be a commercial radar that provides both status and control of the unit via the radar's software Application Programming Interface (API).

3.3 Data Distribution Service (DDS^{TM})

The data bus supporting autonomy messaging (as seen in Figure 3) is implemented via DDS^{TM}. DDS is a middleware protocol and API standard for data-centric connectivity from the Object Management Group (OMG). It integrates the components of a system together, providing low-latency data connectivity, extreme reliability, and a scalable architecture. In a distributed system, middleware is the software layer that lies between the operating system and applications. It enables the various system components to more easily communicate and share data. It simplifies the development of distributed systems by letting software developers focus on the specific purpose of their applications rather than the mechanics of passing information between applications and systems. The DDS specification is fully described in free reference material on the OMG website and there are both open source and commercially available implementations.

3.4 Naming Conventions

UMAA services are modeled within the UCS Architecture under the Multi-Domain Extension (MDE). The UCS Architecture uses SoaML concepts of participant, serviceInterface, service port, and request port to describe the interfaces that make up a service and show how the service is used. Each service defines the capability it provides as well as required interfaces. Each interface consists of an operation that accepts a single message (A SoaML MessageType). In SoaML, a MessageType is defined as a unit of information exchanged between participant Request and Service ports via ServiceInterfaces. Instances of a MessageType are passed as parameters in ServiceInterface operations. (Reference: UCS Architecture, Architecture Technical Governance)

To promote commonality across service definitions, a common way of naming services and their sets of operations and messages has been adopted for defining services within UCS-MDE. The convention uses the Service Base Name <SBN> and an optional Function Name [FN] to derive all service names and their associated operations and messages. As this is meant to be a guide, services might not include all of the defined operations and messages and their names might not follow the convention where a more appropriate name adds clarity.

Furthermore, services in UMAA are not required to be defined as indicated in Table 3 when all parts of the service capabilities are required for the service to be meaningful (such as ResourceAllocation).

Additionally, note that for UMAA not all operations defined in UCS-MDE result in a message being published to the DDS bus, e.g., since DDS uses publish/subscribe, most query operations result in a subscription to a topic and do not actually publish the associated request message. In the case of cancel commands, there is no associated implementation of the cancel<SBN>[FN]CommandStatus as it is just the intrinsic response of the DDS dispose function; so, it is essentially a NOOP (no operation) in implementation. The conventions used to define UCS-MDE services are as follows:

Service Name
- <SBN>[FN]Config
- <SBN>[FN]Control
- <SBN>[FN]Specs
- <SBN>[FN]Status OR Report

where the SBN should be descriptive of the task or information provided by the service. Note that the FN is optional and only included if needed to clarify the function of the service. The suffixes Status and Report are interchangeable. If a "Report" is a more appropriate description of the service, it can be used in lieu of "Status".

Table 3: Service Requests and Associated Responses

	Service Requests (Inputs)	**Service Responses (Outputs)**
Config	set<SBN>[FN]Config	report<SBN>[FN]ConfigCommandStatus
	query<SBN>[FN]ConfigAck	report<SBN>[FN]ConfigAck
	query<SBN>[FN]Config	report<SBN>[FN]Config
	cancel<SBN>[FN]Config	report<SBN>[FN]CancelConfigCommandStatus
	query<SBN>[FN]ConfigExecutionStatus	report<SBN>[FN]ConfigExecutionStatus
Control	set<SBN>[FN]	report<SBN>[FN]CommandStatus
	query<SBN>[FN]CommandAck	report<SBN>[FN]CommandAck
	cancel<SBN>[FN]Command	report<SBN>[FN]CancelCommandStatus
	query<SBN>[FN]ExecutionStatus	report<SBN>[FN]ExecutionStatus
Specs	query<SBN>[FN]Specs	report<SBN>[FN]Specs
Status OR Report	query<SBN>[FN]	report<SBN>[FN]

Service Requests (operation:message)

set<SBN>[FN]Config:<SBN>[FN]ConfigCommandType
query<SBN>[FN]Config:<SBN>[FN]ConfigRequestType[1]
set<SBN>[FN]:<SBN>[FN]CommandType
query<SBN>[FN]CommandAck:<SBN>[FN]CommandAckRequestType[1]
cancel<SBN>[FN]Command:<SBN>[FN]CancelCommandType[1]
cancel<SBN>[FN]Config:<SBN>[FN]CancelConfigType[1]
query<SBN>[FN]ExecutionStatus:<SBN>[FN]ExecutionStatusRequestType[1]
query<SBN>[FN]ConfigExecutionStatus:<SBN>[FN]ConfigExecutionStatusRequestType[1]
query<SBN>[FN]ConfigAck:<SBN>[FN]ConfigAckRequestType[1]
query<SBN>[FN]Specs:<SBN>[FN]SpecsRequestType[1]
query<SBN>[FN]:<SBN>[FN]RequestType [1] [2]

Service Responses (operation:message)
report<SBN>[FN]ConfigCommandStatus:<SBN>[FN]ConfigCommandStatusType
report<SBN>[FN]Config:<SBN>[FN]ConfigReportType
report<SBN>[FN]ConfigAck:<SBN>[FN]ConfigAckReportType
report<SBN>[FN]CommandStatus:<SBN>[FN]CommandStatusType
report<SBN>[FN]CommandAck:<SBN>[FN]CommandAckReportType
report<SBN>[FN]CancelCommandStatus:<SBN>[FN]CancelCommandStatusType[1]
report<SBN>[FN]CancelConfigCommandStatus:<SBN>[FN]CancelConfigCommandStatusType[1]
report<SBN>[FN]ExecutionStatus:<SBN>[FN]ExecutionStatusReportType
report<SBN>[FN]ConfigExecutionStatus:<SBN>[FN]ConfigExecutionStatusReportType
report<SBN>[FN]Specs:<SBN>[FN]SpecsReportType
report<SBN>[FN]:<SBN>[FN]ReportType

where,

- Config (Configuration) Command/Report – This is the setup of a resource for operation of a particular task. Attributes may be static or variable. Examples include: maximum RPM allowed, operational sonar frequency range allowed, and maximum allowable radio transmit power.
- Command Status – This is the current state of a particular command (either control or configuration).
- Command – This is the ability to influence or direct the behavior of a resource during operation of a particular task. Attributes are variable. Examples include a vehicle's speed, engine RPM, antenna raising/lowering, and controlling a light or gong.
- Command Ack (Acknowledgement) Report – This is the command currently being executed.
- Cancel – This is the ability to cancel a particular command that has been issued.
- Execution Status Report – This is the status related to executing a particular command. Examples associated with a waypoint command include cross track error, time to achieve, and distance remaining.
- Specs (Specifications) Report – Provides a detailed description of a resource and/or its capabilities and constraints. Attributes are static. Examples include: maximum RPM of a motor, minimum frequency of a passive sonar sensor, length of the unmanned vehicle, and cycle time of a radar.
- Report – This is the current information being provided by a resource. Examples include vehicle speed, rudder angle, current waypoint, and contact bearing.

3.5 Namespace Conventions

Each UMAA service and the messages under the service can be accessed through their appropriate UMAA namespace. The namespace reflects the mapping of a specific service to its parent ICD, and the parent ICD's mapping to the overall UMAA Design Description. For example:

Access the Primitive Driver Control service under Maneuver Operations:

[1] These message types are required for UCS model rules of construction, but are not implemented as messages in the UMAA specification.
[2] At this time, there are no Requests in the specification. This will be the message format when Requests have been added.

UMAA::MO::PrimitiveDriverControl
Access the ContactReport Service under Situational Awareness:
UMAA::SA::ContactReport

The UMAA model uses common data types that are re-used through the model to define service interface topics, interface topics, and other common data topics. These data types are not intended to be directly utilized but, for reference, they can be accessed in the same manner:

Access the common UMAA Status Message Fields:
UMAA::UMAAStatus
Access the common UMAA GeoPosition2D (i.e., latitude and longitude) structure:
UMAA::Common::Measurement::GeoPosition2D

3.6 Cybersecurity

The UMAA standard addressed in this ICD is independent from defining specific measures to achieve Cybersecurity compliance. This UMAA ICD does not preclude the incorporation of security measures, nor does it imply or guarantee any level of Cybersecurity within a system. Cybersecurity compliance will be performed on a program-specific basis and compliance testing is outside the scope of UMAA.

3.7 GUID algorithm

The UMAA standard utilizes the Globally Unique IDentifier (GUID), conforming to the variant defined in RFC 4122 (variant value of 2). Generators of GUIDs may generate GUIDs of any valid, RFC 4122-defined version that is appropriate for their specific use case and requirements. (Reference: A Universally Unique IDentifier (UUID) URN Namespace)

3.8 Large Collections

The UMAA standard defines Large Collections, which are collections of decoupled but related data. Large Collections provide the ability to update one or more elements of the collection without republishing the entire collection to the DDS bus. This avoids two problems related to using an unbounded sequence type in a DDS message: 1) resource consumption growing as the collection is appended to or updated, and 2) DDS implementation-specific limitations on unbounded sequences. There are two implementations of a Large Collection: the Large Set (unordered) and the Large List (ordered).

In both Large Collection implementations, there are two important abstractions: the collection metadata and collection element type. Because Large Collections are specific to the UMAA PSM, the type definitions for the collection metadata and collection element are not part of MDE, and the IDL definitions of these types are generated separately. A particular UMAA message that has a Large Collection attribute will reference the metadata type (LargeSetMetadata or LargeListMetadata). The collection element type is defined under the same namespace as the message that uses it, and follows the naming pattern <parent message name><attribute name><collection type>Element. Each element of the collection is published as a separate message on the DDS bus, and can be tracked back to their related collection using the setID or listID. Users can also trace an element in a set to the source attribute (a NumericGUID) of the Service Provider that generated the report with this set using the collection metadata.

3.8.1 Necessary QoS

To achieve the Large Collection consistency in the update process described below, ordering of samples on the collection element type topic is necessary. Therefore, publishers and subscribers to the collection element type topic must use the `PRESENTATION` QoS policy with an `access_scope` of `DDS_TOPIC_PRESENTATION_QOS` and `ordered_access`.

Note that Large Collection Metadata and Elements are sent on separate DDS topics. DDS QoS does not guarantee ordering across topics. For this reason, implementations must be able to handle cases where elements arrive before or after the associated metadata. Memory must be allocated to await the proper metadata and associated elements.

3.8.2 Creating Large Collections

To create a large collection, a series of element messages and a metadata message must be sent from one DDS participant (the sender) to another (the receiver). The messages should be buffered on the receiving side until a synchronization point is reached which indicates an atomic update. That is, when both a metadata message and an element message corresponding by list ID, timestamp, and last element ID have been received, yield a complete collection. Figure 4 shows the sequence of exchanges to establish a collection with 3 elements.

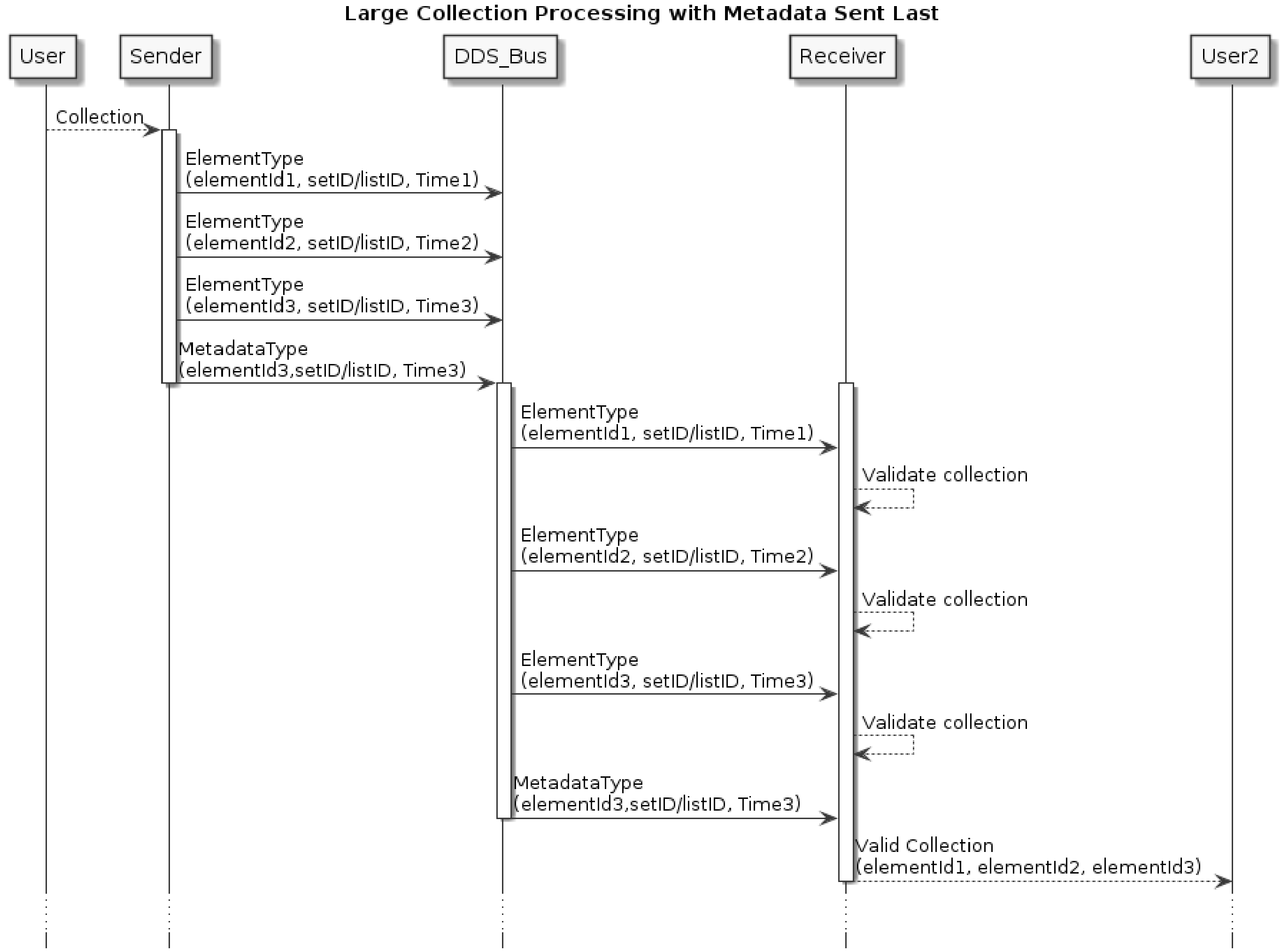

Figure 4: Sequence Diagram for initialization of a Large Collection with 3 elements.

The same collection could be established where the element data arrives after the metadata, creating the same list as depicted in figure 5.

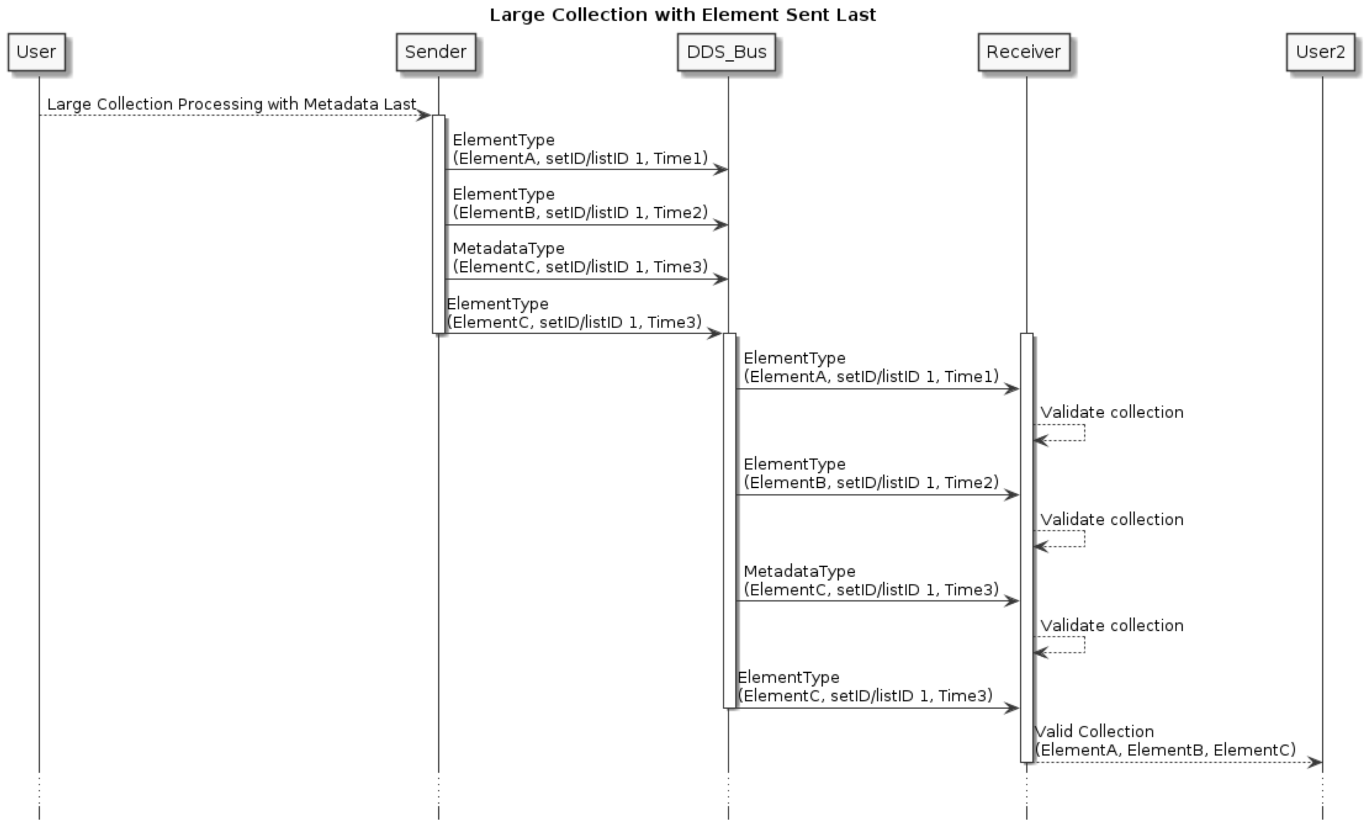

Figure 5: Sequence Diagram for initialization of a Large Collection with 3 elements.

3.8.3 Updating Large Collections

When elements of the collection are updated, the metadata must be updated as well to signal a change in the set. The `updateElementID` is updated to match the elementID of the element whose reception signals the end of the atomic update of the collection. Because of the requirement of an ordered topic described above, this will be the element that is updated last chronologically. The metadata `updateElementTimestamp` must be updated to the timestamp of the same element that signals the end of the update.

The set can be updated as a batch (multiple elements in a single "update cycle," as determined by the provider). This allows for a coarse synchronization: data elements that do not match the metadata `updateElementID` and `updateElementTimestamp` can be assumed to be part of an in-progress update cycle. Consumers can choose to immediately act on those data individually or wait until the matching element is received to signal that the complete update cycle has finished and consider the set as a whole. Note that the coarseness of synchronization is service-dependent: in some cases an intermediate view of a collection update may be logically incorrect to act upon.

Figure 6 shows the sequence of exchanges to update a collection of 3 elements and add a 4th element.

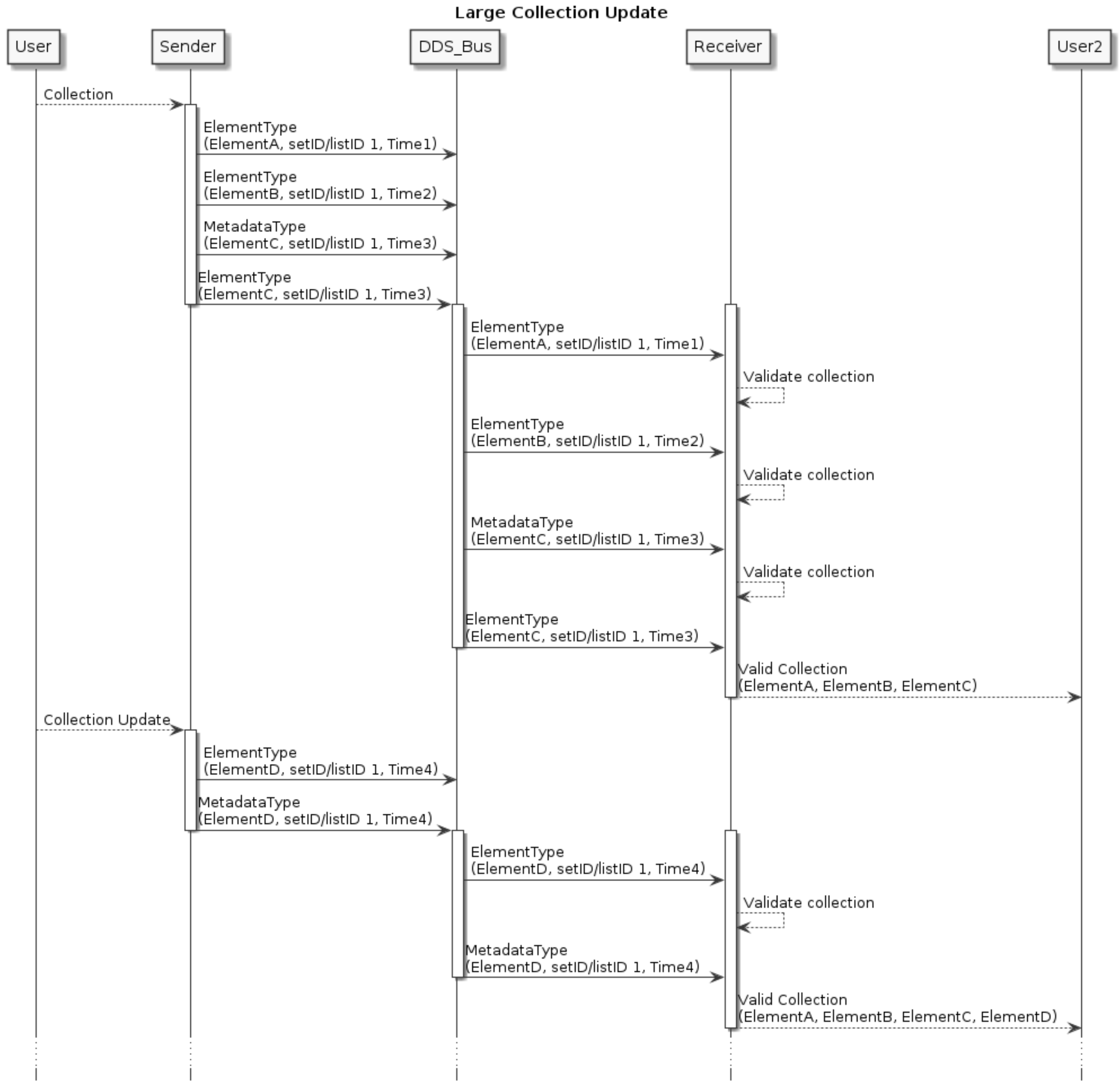

Figure 6: Sequence Diagram for update of Large Collection.

Figure 7 shows the sequence of exchanges to update an element of a collection multiple times.

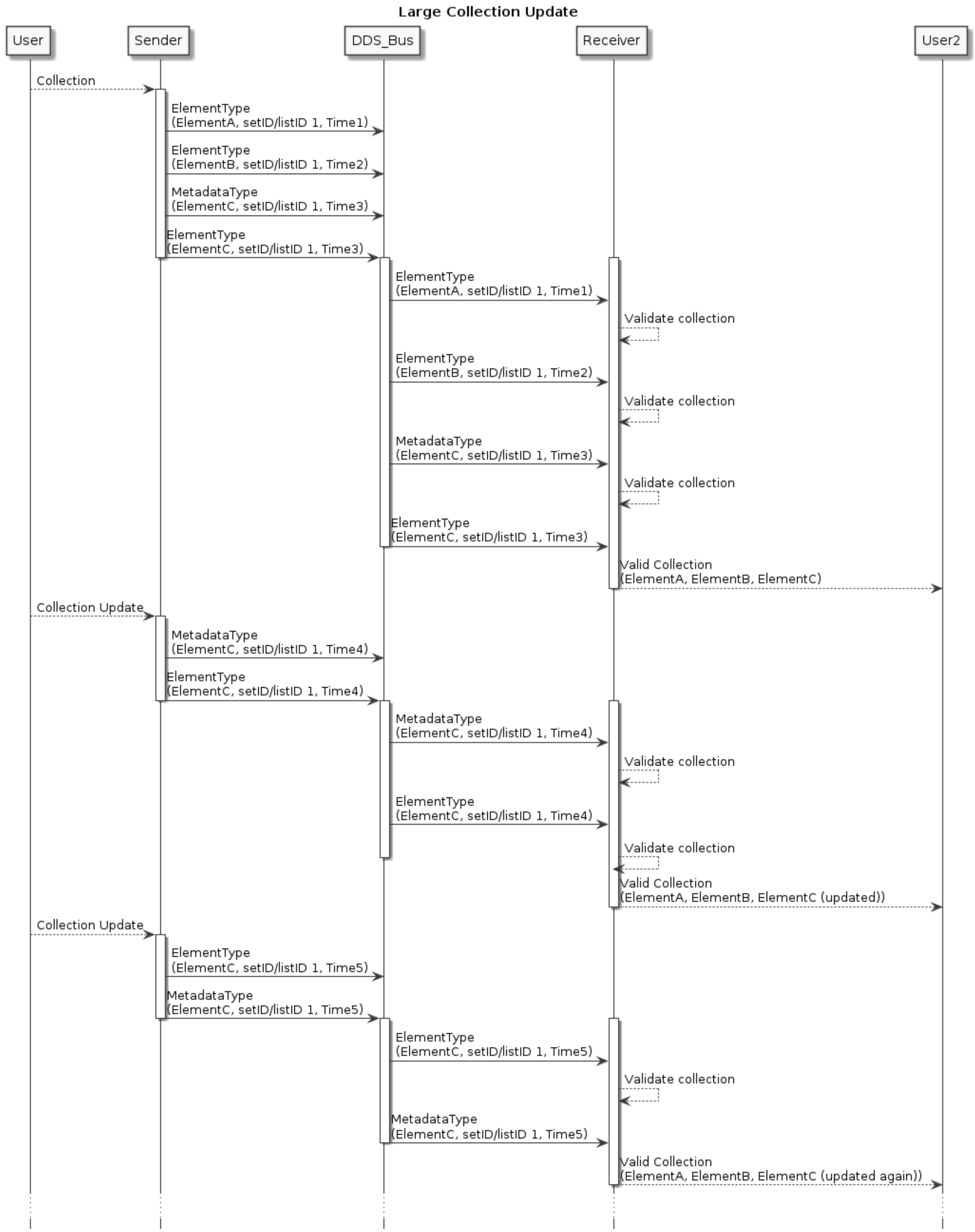

Figure 7: Sequence Diagram for update of an element of a Large Collection multiple times.

3.8.4 Removing an element from Large Collections

To remove an element from a collection, dispose of the element on the element topic and re-publish the metadata. Multiple deletes and inserts can happen for a single metadata update. In the case where the final element of the collection is deleted, the updateElementTimestamp should be unset in the metadata.

Figure 8 shows the sequence of exchanges to delete an element from a Large Collection.

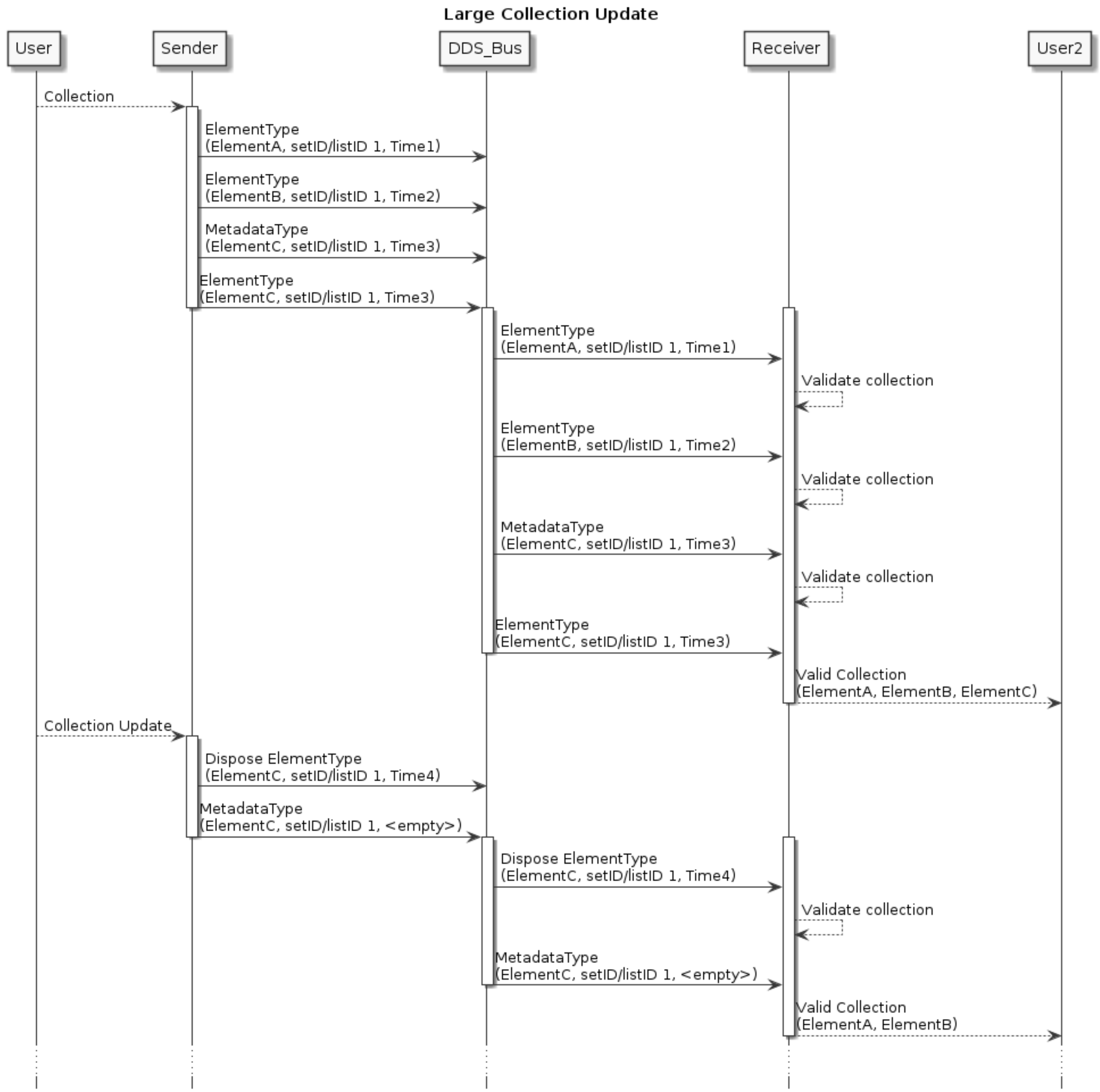

Figure 8: Sequence Diagram for delete of element from Large Collection.

For Large Lists, it may be necessary to update the nextElementID references during delete operations to ensure that the list is still valid. This would cause multiple element messages to be sent along with updated metadata.

3.8.5 Specifying an Empty Large Collection

A particular Large Collection can be empty during initial creation. This is indicated by publishing metadata with a `size` of zero and an `updateElementID` set to the Nil UUID. As specified in section 4.1.7 of the referenced document "A Universally Unique IDentifier (UUID) URN Namespace", this is a "special form of UUID that is specified to have all 128 bits set to zero".

Figure 9 shows the sequence of exchanges to establish an initially empty Large Collection.

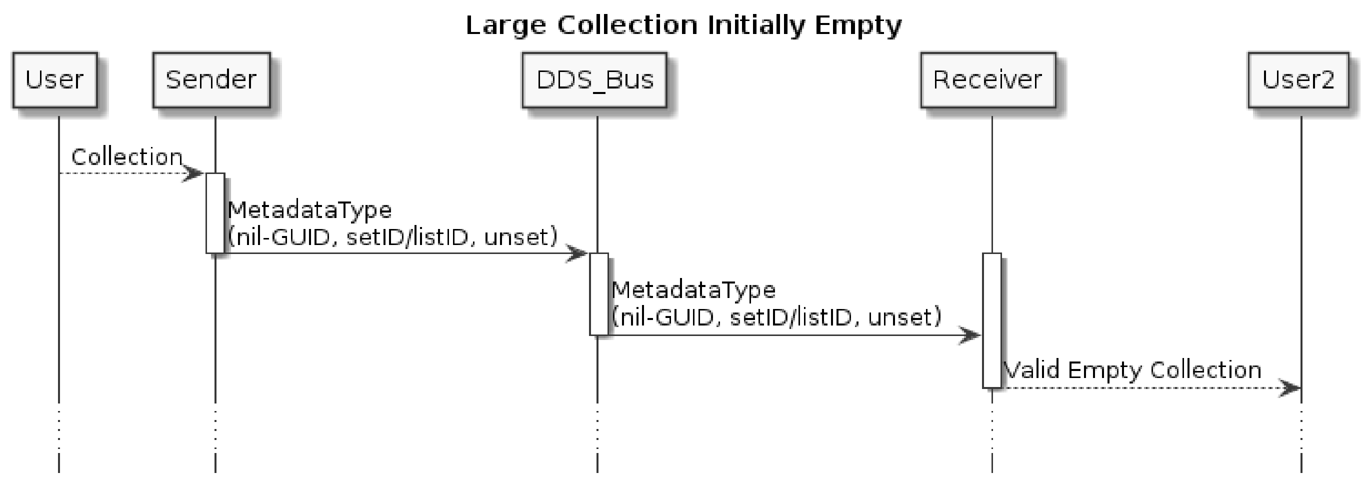

Figure 9: Sequence Diagram for initialization of an empty Large Collection.

3.8.6 Large Set Types

The following details the LargeSetMetadata structure:

Table 4: LargeSetMetadata Structure Definition

Attribute Name	Attribute Type	Attribute Description
setID	NumericGUID	Identifies the Large Set instance this metadata relates to.
updateElementID	NumericGUID	This field references the element ID of the set element whose reception signals the end of an atomic update to this set. This elementID must be used in conjunction with the updateElementTimestamp below to fully identify when the atomic update has completed and the set is stable.
updateElementTimestamp†	DateTime	This field identifies the elementTimestamp of the element, referenced above by updateElementID, that signals the end of an atomic update to this set. This field will be empty in the event that the element update results from a DDS dispose.
size	LargeCollectionSize	Indicates the number of elements associated with this set after the atomic update is complete.

An example element type is shown below, where a `FooReportType` message has a Large Set attribute called "items" whose type is `BarType`

Table 5: Example FooReportTypeItemsSetElement Structure Definition

Attribute Name	Attribute Type	Attribute Description
element	BarType	The value of the set element.
setID*	NumericGUID	Identifies the Large Set instance this element relates to.
elementID*	NumericGUID	Uniquely identifies this element within the set and across all large collection elements that currently exist on the DDS bus.
elementTimestamp	DateTime	The timestamp of this element.

3.8.7 Large List Types

The following details the LargeListMetadata structure:

Table 6: LargeListMetadata Structure Definition

Attribute Name	Attribute Type	Attribute Description
listID	NumericGUID	Identifies the Large List instance this metadata relates to.
updateElementID	NumericGUID	This field references the element ID of the list element whose reception signals the end of an atomic update to this list. This elementID must be used in conjunction with the updateElementTimestamp below to fully identify when the atomic update has completed and the list is stable.
updateElementTimestamp†	DateTime	This field identifies the elementTimestamp of the element, referenced above by updateElementID, that signals the end of an atomic update to this list. This field will be empty in the event that the element update results from a DDS dispose.
startingElementID	NumericGUID	This field identifies the list element, tying to its elementID, that is sequentially first in the list. This is provided for convenience when iterating through the linked list using the nextElementID field.
size	LargeCollectionSize	Indicates the number of elements associated with this set after the atomic update is complete.

An example element type is shown below, where a `FooReportType` message has a Large List attribute called "items" whose type is `BarType`

Table 7: Example FooReportTypeItemsListElement Structure Definition

Attribute Name	Attribute Type	Attribute Description
element	BarType	The value of the list element.
listID*	NumericGUID	Identifies the Large List instance this element relates to.
elementID*	NumericGUID	Uniquely identifies this element within the list and across all large collection elements that currently exist on the DDS bus.
elementTimestamp	DateTime	The timestamp of this element.

Attribute Name	Attribute Type	Attribute Description
nextElementID†	NumericGUID	This field references to the elementID of the element that logically follows this element in the linked list. This is empty if this element is sequentially last.

3.9 Generalizations and Specializations

The UMAA standard makes use of generalization/specialization relationships when defining data types. The generalization/specialization relationship is one where a generalization data structure is defined to contain attributes that are common across some entity and specialization data structures are defined to contain attributes that are specific to a particular type of that entity. This relationship can be modeled as inheritance in UML as shown below.

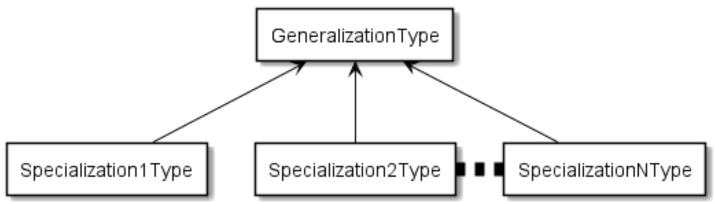

Figure 10: Generalization/Specialization UML diagram.

When the data type of an attribute within a message is a generalization, it is defined to be that generalization plus the data type of one of its specializations. In order to support this relationship, the generalization data structure and its specialization data structure are published to separate topics along with additional metadata linking the two topics. Specifically, the generalization data structure includes: specializationTopic, specializationID, and specializationTimestamp; and the specialization data structure includes: specializationID and specializationTimestamp. The specializationTopic specifies the topic name of the particular specialization, and the specializationID and specializationTimestamp must be equivalent in each topic, respectively, in order to establish the generalization/specialization relationship.

3.9.1 Creating a generalization/specialization

To create a generalization/specialization, both the GeneralizationType and SpecializationType topics must be sent from one DDS participant (the sender) to another (the receiver). The topics should be buffered on the receiving side until a synchronization point is reached that indicates an atomic update.

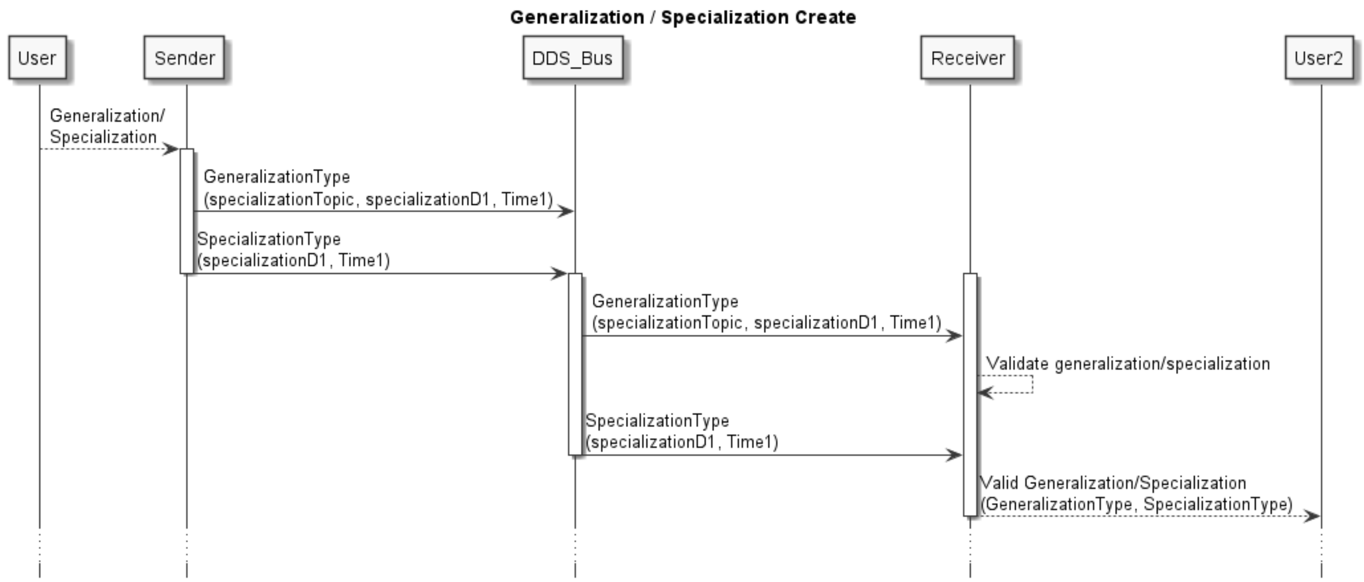

Figure 11: Sequence diagram for creating a generalization/specialization.

3.9.2 Updating a generalization/specialization

An update to a generalization/specialization can occur when there is a change in either data structure. In order for the update to be complete, the specializationTimestamp must be updated in both the GeneralizationType and the SpecializationType, and again they must be equal. Note that if a generalization/specialization exists within a large set or large list that their respective metadata must also be updated as defined in Section 3.8.

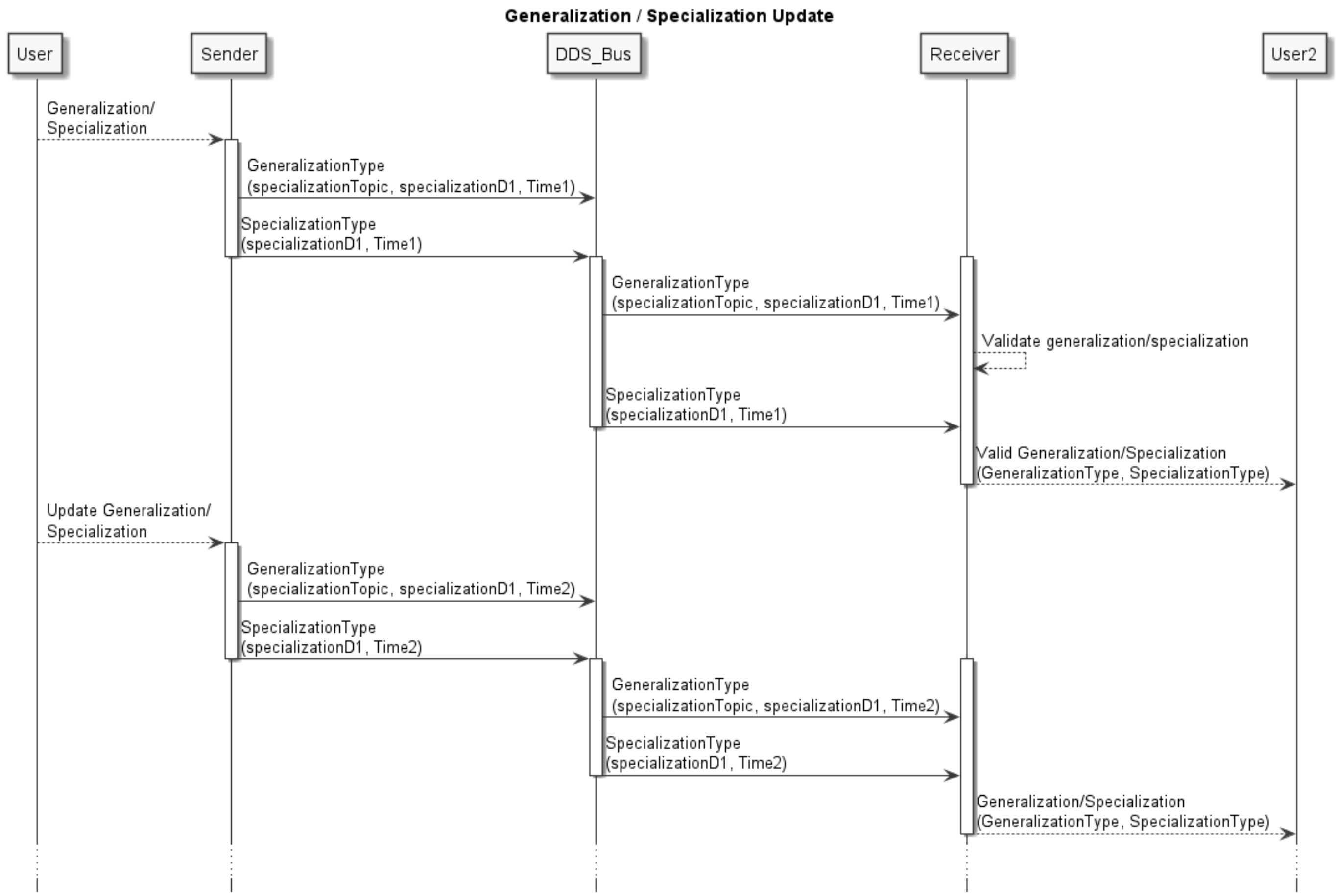

Figure 12: Sequence diagram for updating a generalization/specialization.

3.9.3 Removing a generalization/specialization

To remove a generalization/specialization, both topics must be disposed. Again, note that if a generalization/specialization exists within a large set or large list that their respective metadata must also be updated as defined in Section 3.8.

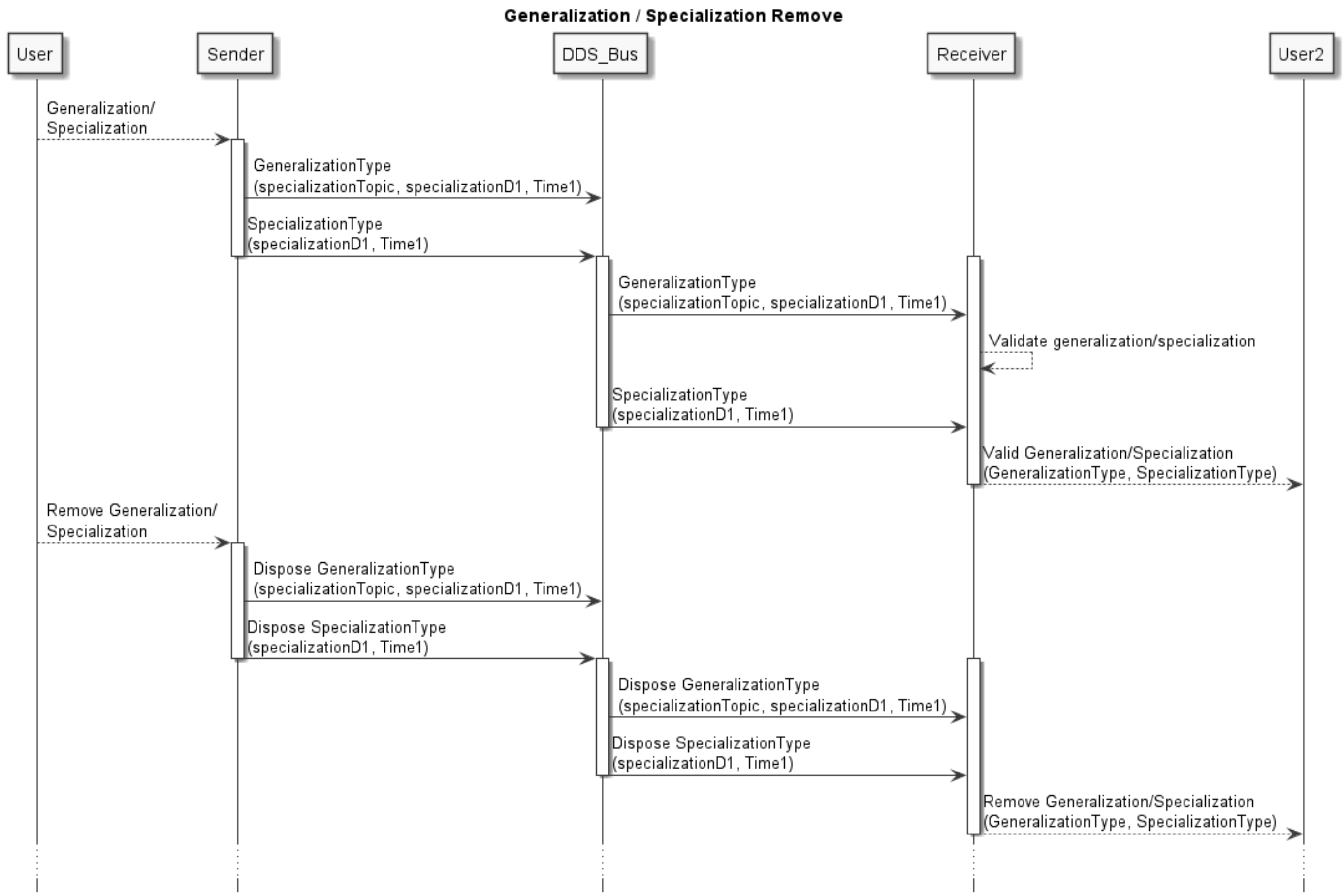

Figure 13: Sequence diagram for removing a generalization/specialization.

4 Introduction to Coordinate Reference Frames and Position Model

4.1 Platform Reference Frame

In the following Service Definitions, we use the parameters yaw, pitch, and roll to define the platform orientation with respect to the specified reference frame. Each parameter is described as a rotation around a given axis: Yaw about the Z axis. Pitch about the Y axis. Roll about the X axis. A UUV is shown in the diagrams because it has more degrees for freedom for its pose and motion, however, the terminology equally applies to both USVs and UUVs.

The axes are defined as:

- X - Positive in the forward direction, negative in the aft.
- Y - Positive in the starboard direction, negative in the port.
- Z - Positive in the down direction, negative in the up.

Additionally, rotations about all axes follow the right-hand rule.

4.2 Earth-Centered Earth-Fixed Frame

The Earth-Centered Earth-Fixed (ECEF) frame is a global reference frame with its origin at the center of the ellipsoid modeling the Earth's surface (Figure 14). The Z-axis points along the Earth's axis of rotation through the North Pole. The X-axis points from the origin to the intersection of the equator with the prime meridian, which defines 0° longitude. The Y-axis completes the right-handed orthogonal system, intersecting the equator at the 90° east meridian.

4.3 North-East-Down Frame

The North-East-Down (NED) frame is defined with an origin at the object described by the navigation solution. The Down axis is defined as normal to the surface of the reference ellipsoid in the direction pointing towards the interior of the Earth. The North axis is the projection of the line from the object to the north pole onto the plane orthogonal to the Down axis. The East axis completes the right-handed orthogonal system and points in the East direction.

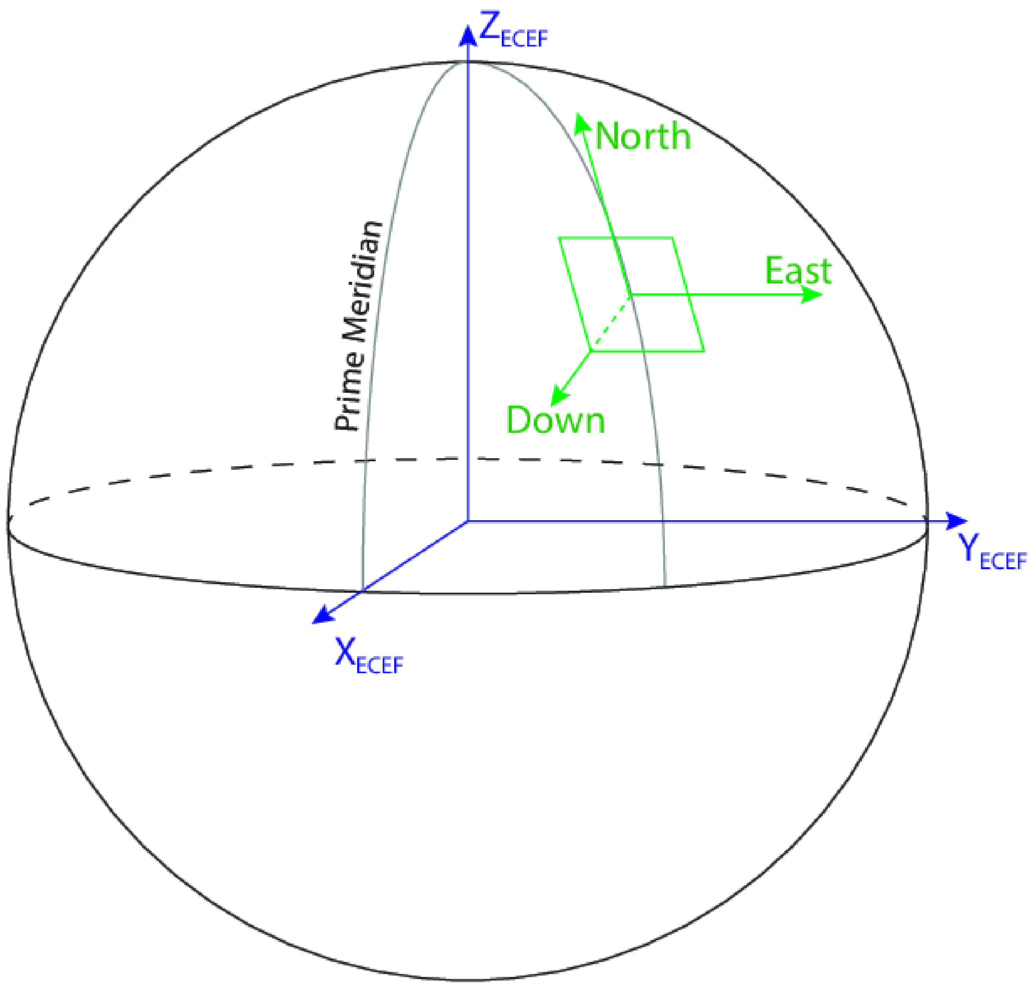

Figure 14: Origins and axes of the Earth-Centered Earth-Fixed (ECEF) and North-East-Down (NED) frames.

4.4 WGS 84

The World Geodetic System (WGS) 1984 defines a standard coordinate system for the Earth. It represents the Earth as an oblate spheroid, and defines the mapping between latitude-longitude-altitude (LLA) coordinates and Cartesian ECEF coordinates. GPS reports positions in WGS 84 LLA coordinates. It has become the standard datum for navigation.

While the UMAA services typically make use of the coordinate systems defined by WGS 84, it also defines an Earth Gravity Model (EGM) and a World Magnetic Model (WMM) which are updated regularly.

4.5 Vehicle Orientation

Determining the orientation of the vehicle (Figure 15) with respect to any reference frame is carried out via the following procedure (Figure 16).

1. Align the vehicle's longitudinal or X axis with the reference frame X axis. In the global reference frame, this is the north direction.
2. Align the vehicle's down or Z axis with the reference frame's Z axis. In the global reference frame, this is the gravity direction.
3. Ensure that the vehicle's transverse or Y axis is aligned with the reference frame's Y axis. In the global reference frame, this is the east direction.
4. Rotate the vehicle about the vehicle's Z axis by the yaw angle (Figure 17).

5. Rotate the vehicle about the vehicle's newly oriented Y axis by the pitch angle (Figure 18).
6. Rotate the vehicle about the vehicle's newly oriented X axis by the roll angle (Figure 19).

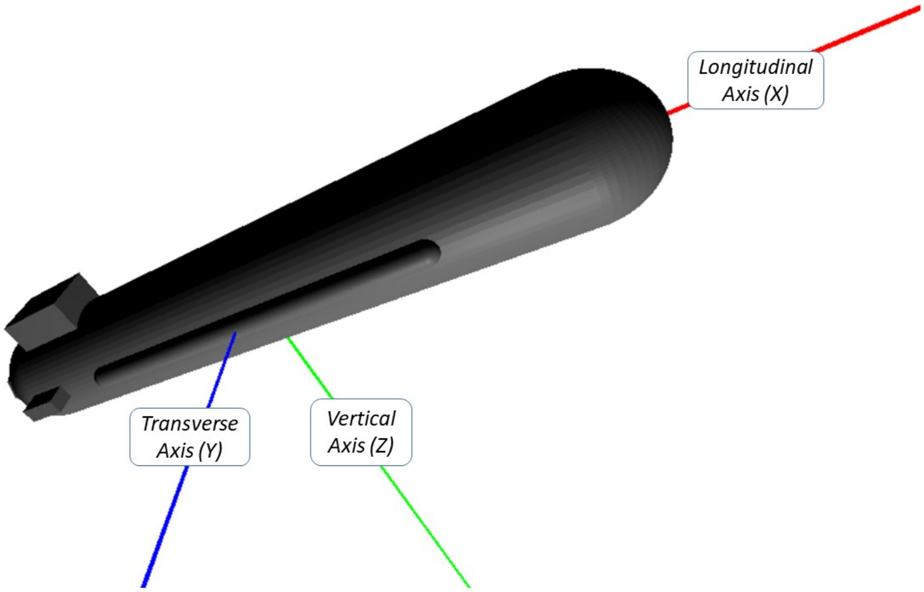

Figure 15: Define the Vehicle Coordinate System

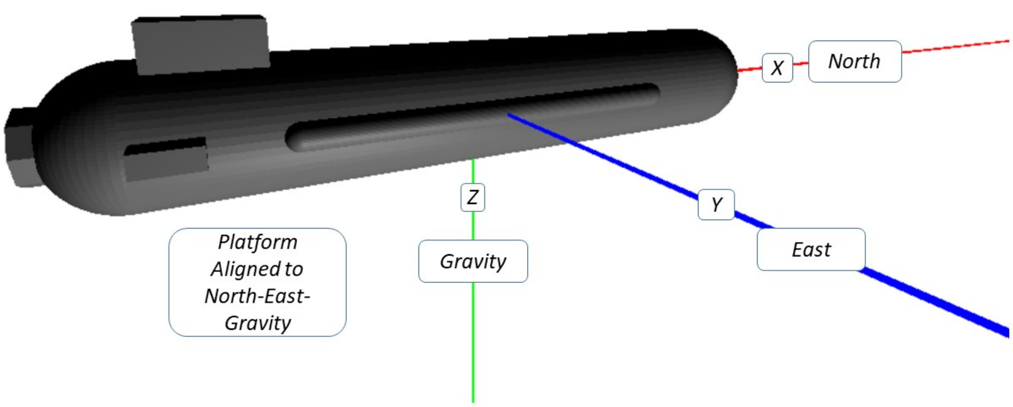

Figure 16: Align the Vehicle with the Reference Frame Axes.

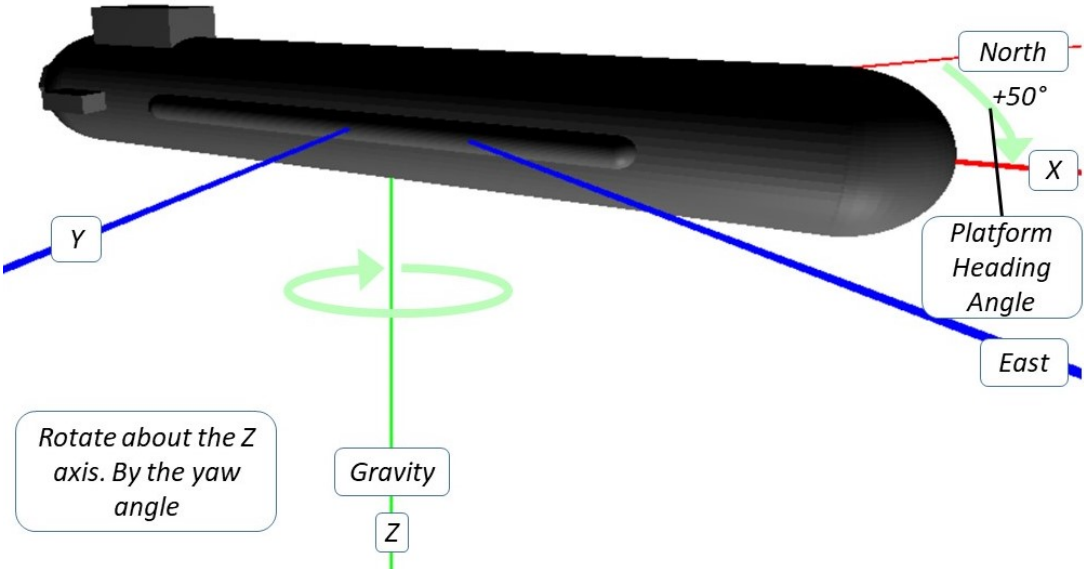

Figure 17: Rotate the Vehicle by the Yaw Angle.

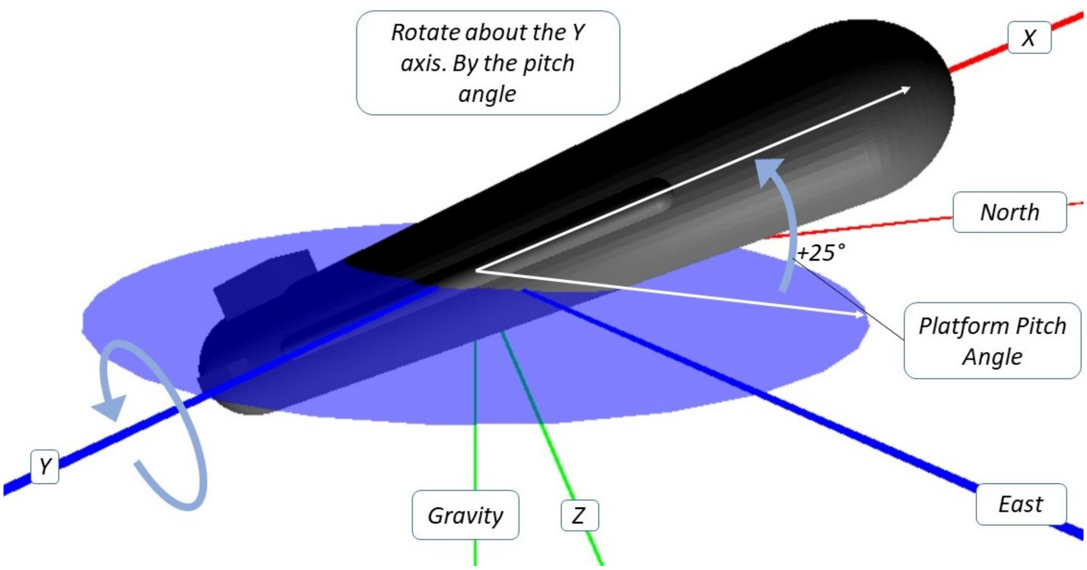

Figure 18: Rotate the Vehicle by the Pitch Angle.

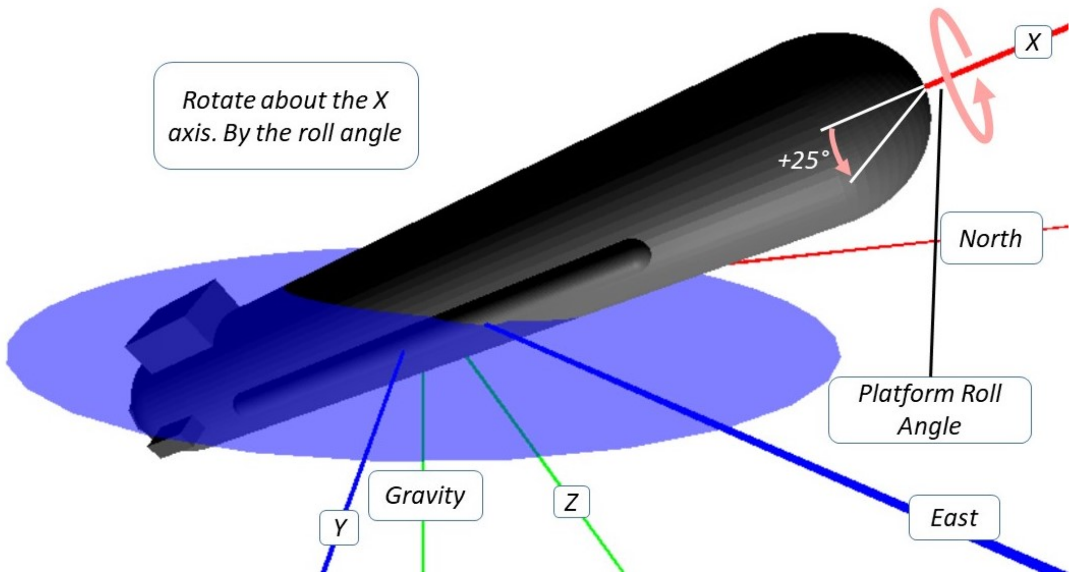

Figure 19: Rotate the Vehicle by the Roll Angle.

4.6 Vehicle Coordinate Reference Frame Origin

UMAA does not specify a required origin for the vehicle coordinate reference frame. However, certain applications may benefit from defining a specific origin such as the registration of multiple sensors with associated offsets for data fusion. Possible origins include the keel/transom intersection, bow/waterline intersection, center of gravity, center of buoyancy and location of INS. A few examples follow.

Definitions

- Keel Transom Intersection
  - Beam at Waterline (BWL) - The maximum distance of the vehicle at the waterline, the distance along the Y axis of the widest point of the hull where it meets the waterline.
  - Design Waterline (DWL) - The line representing the waterline on the vehicle at designed load in summer temperature.
  - Keel - The principal fore-and-aft component of a ship's framing, located along the centerline of the bottom and connected to the stem and stern frames.
  - Length at Waterline (LWL) - The measured distance of the vehicle at the level where it sits in the water, measured along the X axis.
  - Transom - The aftermost transverse flat or shaped plating enclosing the hull.

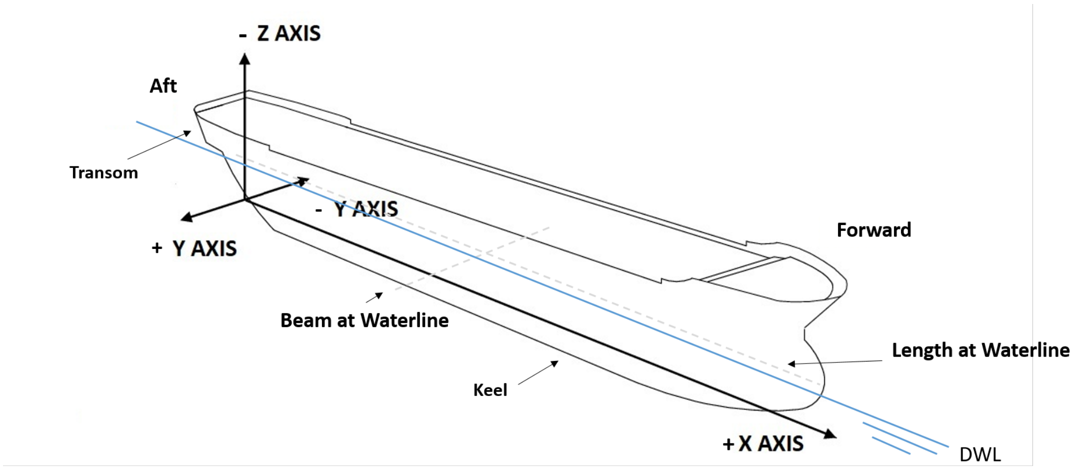

Figure 20: Keel Transom Intersection Origin Location on a USV as Example

- Center of Buoyancy
  - X - The Longitudinal Center of Buoyancy (LCB) when fully submerged.
  - Y - The symmetrical centerline.
  - Z - The Vertical Center of Buoyancy (VCB) when fully submerged.

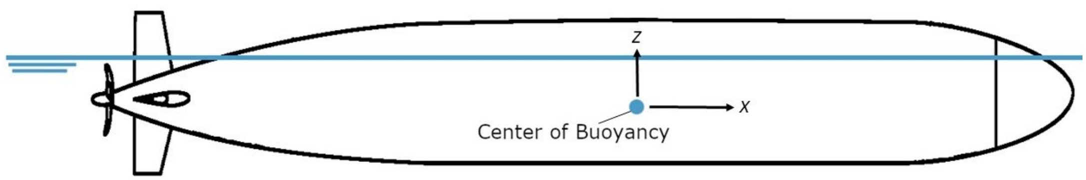

Figure 21: Center of Buoyancy Origin Location on a UUV as Example.

5 Flow Control

5.1 Command / Response

This section defines the flow of control for command/response over the DDS bus. A command/response controls a specific service. While the exact names and processes will depend on the specific service and command being executed, all command/responses in UMAA follow a similar pattern. A notional "Function" command `FunctionCommand` is used in the following examples. As will be described in subsequent paragraphs, DDS publish/subscribe methods are used in implementations to issue commands and responses.

To direct a `FunctionCommand` at a specific Service Provider, UMAA includes a `destination` GUID in all commands. A Service Provider is required to respond to all `FunctionCommand`s where the `destination` is the same as the Service Provider's ID. The Service Consumer will also create a `sessionID` for the command when commanded. The `sessionID` is used to track the command execution as a key into other command-related messages. The `sessionID` must be unique across all `FunctionCommand` instances that are active (i.e. currently on the DDS bus), otherwise the Service Provider will consider the `FunctionCommand` to be a command update (see Section 5.1.4.2). Once a `FunctionCommand` is removed from the DDS bus as part of the Command Cleanup process (see Section 5.1.5), its `sessionID` may be reused for future commands without triggering a command update; therefore it is not necessary for a Service Provider to maintain a complete history of `sessionID`s.

Service Provider and Service Consumer terminology in the following sections is adopted from the OMG Service-oriented architecture Modeling Language (SoaML).

To initialize, a Service Provider (controllable resource) subscribes to the `FunctionCommand` DDS topic. At startup or right before issuing a command, the Service Consumer (controlling resource) subscribes to the `FunctionCommandStatus` DDS topic. Optionally, the Service Consumer may also subscribe to the `FunctionCommandAckReport` to monitor which command is currently being executed, and the `FunctionExecutionStatusReport` (if defined for the Function service) that provides reporting on function-specific data status.

Both Service Providers and Service Consumers are required to recover or clean up any previous persisted commands on the bus during initialization.

To execute a command, the Service Consumer publishes a `FunctionCommandType` to the DDS bus. The Service Provider will be notified and will begin processing the request. During each phase of processing, the Service Provider will provide updates to the Service Consumer via published updates to a related `FunctionCommandStatus` topic. Command responses are correlated to their originating command via the `sessionID`. If a command with a duplicate `sessionID` is received, the Service Provider will regard this as a command update, and follow the flow control detailed in Section 5.1.4.2. Command status updates are provided in the command responses via the `commandStatus` field with additional details included in the `commandStatusReason` field. The Service Provider will also publish the current executing command to the `FunctionCommandAckReport` topic. When defined for the Function service, the Service Provider must also publish the `FunctionExecutionStatusReport` topic and update it as appropriate throughout the execution of the command.

The required state transitions for the `commandStatus` field are shown in Figure 22. Commands may complete normally, or they may terminate early due to failure (Section 5.1.4.4) or cancellation (Section 5.1.4.5). The state machine for a command can also be reset to `ISSUED` via a command update (Section 5.1.4.2). If there is not a self-transition indicated in the diagram, you cannot republish that state in a message. Every command must transition through the states as defined. For example, it is a violation to transition from `ISSUED` to `EXECUTING` without transitioning through `COMMANDED`. Even in the case where there is no logic executing between the `ISSUED` and `EXECUTING` states, the Service Provider is required to transition through `COMMANDED`. This ensures consistent behavior across different Service Providers, including those that do require the `COMMANDED` state.

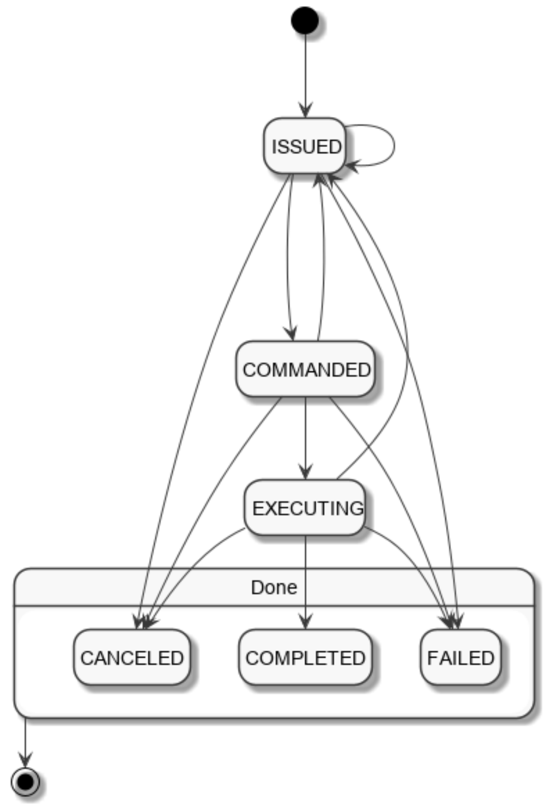

Figure 22: State transitions of the `commandStatus` as commands are processed.

As described above, each time a command transitions to a new state, a `FunctionCommandStatus` message is published containing the updated `commandStatus` and a `commandStatusReason` that indicates why the state transition happened. The table below shows all valid `commandStatusReason` values for each `commandStatus` transition.

	Ending State					
Starting State	ISSUED	COMMANDED	EXECUTING	COMPLETED	FAILED	CANCELED
Initial State	SUCCEEDED	—	—	—	—	—
ISSUED	UPDATED	SUCCEEDED	—	—	VALIDATION_FAILED RESOURCE_FAILED INTERRUPTED TIMEOUT SERVICE_FAILED	CANCELED
COMMANDED	UPDATED	—	SUCCEEDED	—	RESOURCE_REJECTED INTERRUPTED TIMEOUT SERVICE_FAILED	CANCELED
EXECUTING	UPDATED	—	—	SUCCEEDED	OBJECTIVE_FAILED RESOURCE_FAILED INTERRUPTED TIMEOUT SERVICE_FAILED	CANCELED
COMPLETED	—	—	—	—	—	—
FAILED	—	—	—	—	—	—
CANCELED	—	—	—	—	—	—

Figure 23: Valid `commandStatusReason` values for each `commandStatus` state transition. Entries marked with a (—) indicate that the state transition is invalid.

In the following sections, the sequence diagrams demonstrate different exchanges between a Service Consumer and Service Provider. Within the diagrams, the dashed arrows represent implementation-specific communications that are outside of UMAA's scope. These sequence diagrams are just an example of one possible implementation. Other implementations may have different communication patterns between the Service Provider and the Resource or be implemented completely within the Service Provider process itself (no dependency on an external Resource). Likewise, the interactions between the User and Service Consumer may follow similar or different patterns. However, the UMAA-defined exchanges with the DDS bus between the Service Consumer and Service Provider must happen in the order shown within the sequence diagrams.

5.1.1 High-Level Flow

The high-level flow of a command sequence is shown in Figure 24 and can be described as follows:

1. The Command Startup Sequence is performed.
2. For each command to be executed:
   (a) The Command Start Sequence is performed.
   (b) The command is executed (sequence depends on the execution path, i.e., success, failure, or cancel).
   (c) The Command Cleanup Sequence is performed.
3. The Command Shutdown Sequence is performed.

The `ref` blocks will be defined in later sequence diagrams. Note that the duration of the system execution for any particular `FunctionCommandType` is defined by the combination of the Service Provider(s) and Service Consumer(s) in the system and may not be identical to the overall system execution duration. For example, providers may only be available to execute certain commands during specific mission phases or when certain hardware is in specific configurations. This Command Startup Sequence is not required to happen during a system startup phase. The only requirement is that it must be completed by at least one Service Provider and one Service Consumer before any `FunctionCommandType` commands can be fully executed. Likewise, the Command Shutdown sequence may occur at any time the `FunctionCommandType` will no longer be supported. There is no requirement stating that the Command Shutdown Sequence only be performed during a system shutdown phase.

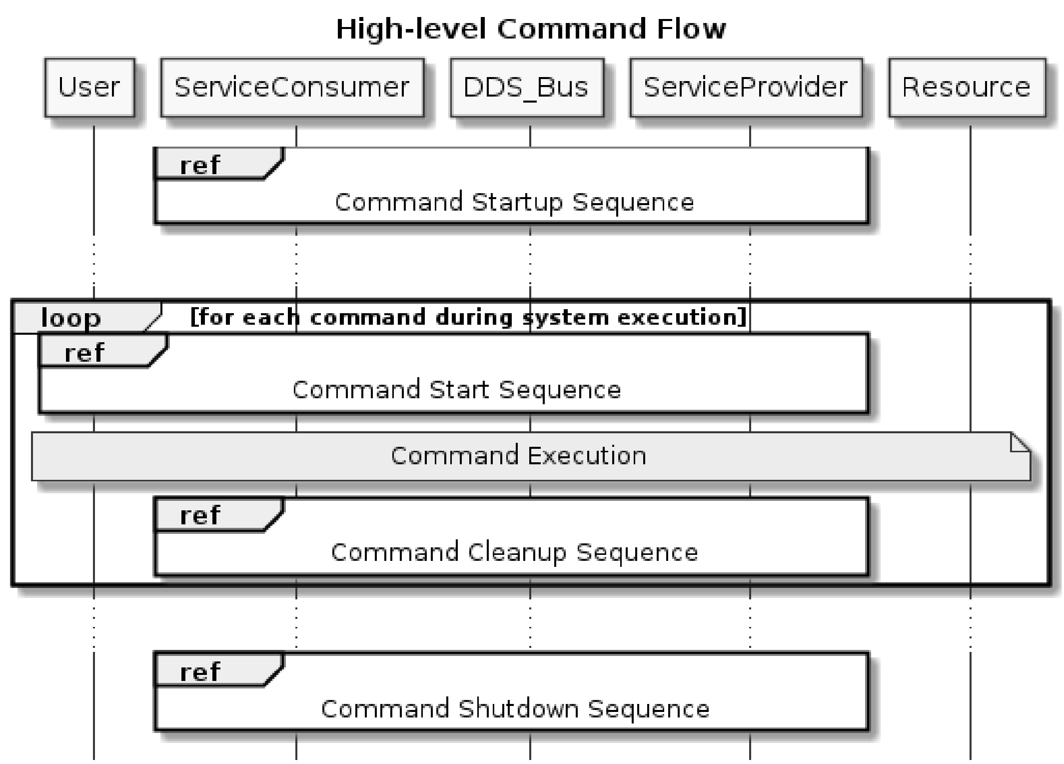

Figure 24: Sequence Diagram for the High-Level Description of a Command Execution.

5.1.2 Command Startup Sequence

As part of initialization both the Service Provider and Service Consumer are required to perform a startup sequence. This startup prepares the Service Provider to execute commands and the Service Consumer to request commands and monitor the progress of those requested commands.

The Service Provider and Service Consumer can initialize in any order. Commands will not be completely executed until both have completed their initialization. The sequence diagram is shown in Figure 25.

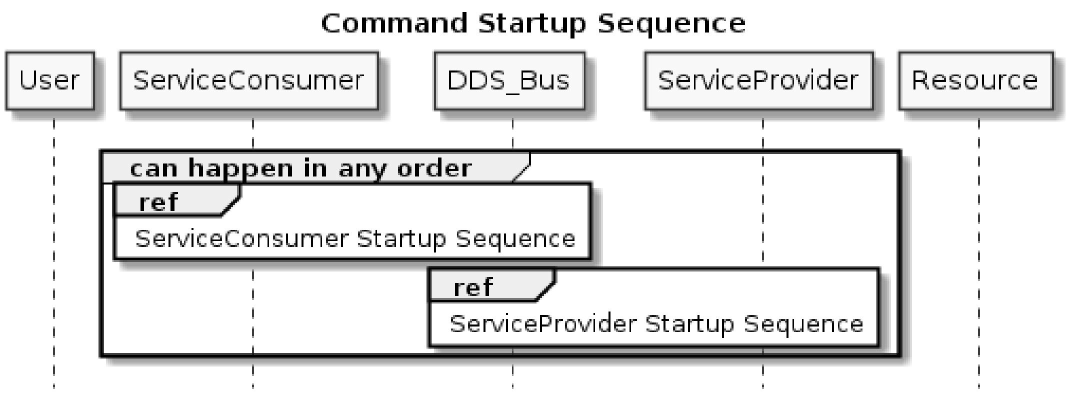

Figure 25: Sequence Diagram for Command Startup.

5.1.2.1 Service Provider Startup Sequence

During startup, the Service Provider is required to register as a publisher to the `FunctionCommandStatus`, `FunctionCommandAckReport`, and (if defined for the Function service) the `FunctionExecutionStatusReport` topics.

The Service Provider is also required to subscribe to the `FunctionCommand` topic to be notified when new commands are published.

Finally, the Service Provider is required to handle any existing `FunctionCommandType` commands persisted on the DDS bus with the Service Provider's ID. For each command, if the Service Provider can and wishes to recover, it can continue to execute the command. To obtain the last published state of the command, the Service Provider must subscribe to the `FunctionCommandStatusType`. The Service Provider will continue following the normal status update sequence, picking up from the last status on the bus. If the Service Provider cannot or chooses not to continue processing the command, it must fail the command by publishing a `FunctionCommandStatus` with a `commandStatus` of `FAILED` and a `reason` of `SERVICE_FAILED`.

The Service Provider Startup sequence is shown in Figure 26.

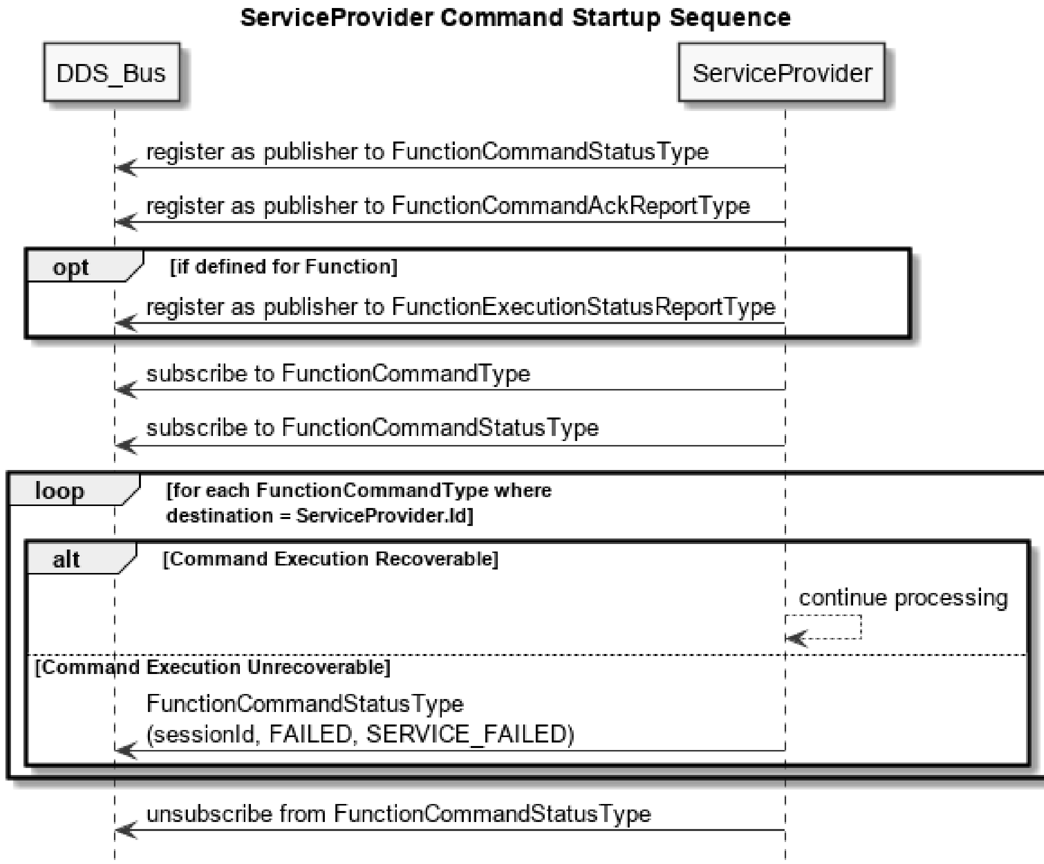

Figure 26: Sequence Diagram for Command Startup for Service Providers.

5.1.2.2 Service Consumer Startup Sequence During startup, the Service Consumer is required to register as a publisher of the `FunctionCommandType`.

The Service Consumer is also required to subscribe to the `FunctionCommandStatusType` to monitor the execution of any published commands. The Service Consumer can optionally register for the `FunctionCommandAckReportType` and, if defined for the Function service, the `FunctionExecutionStatusReportType` if it desires to track additional status of the execution of commands.

Finally, the Service Consumer is required to handle any existing `FunctionCommandType` commands persisted on the DDS bus with this Service Consumer's ID. To find existing `FunctionCommandType`s on the bus, it must first subscribe to the topic. If the Service Consumer can and wishes to recover, it can continue to monitor the execution of the command. If the Service Consumer cannot or chooses not to continue the execution of the command, it must cancel the command via the normal command cancel method.

The Service Consumer Startup sequence is shown in Figure 27.

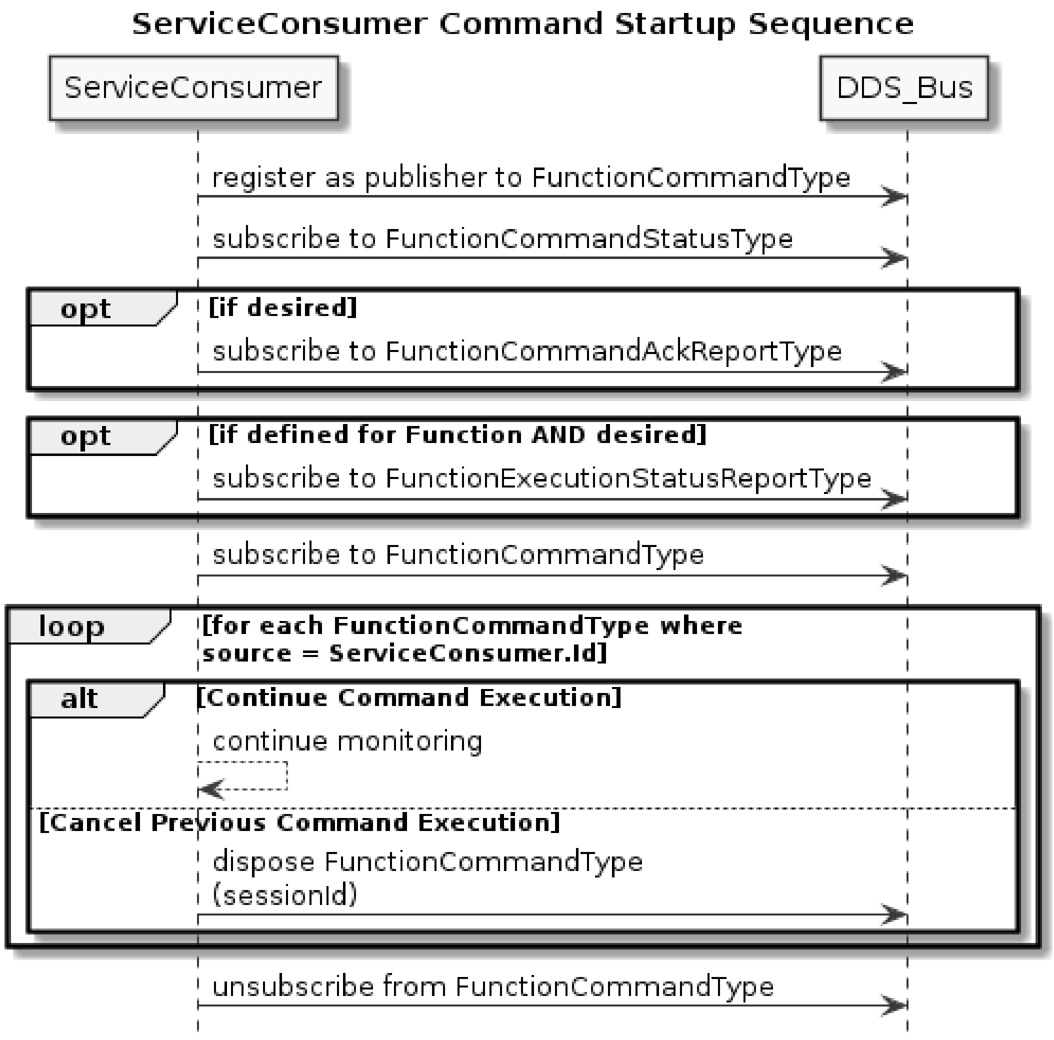

Figure 27: Sequence Diagram for Command Startup for Service Consumers.

5.1.3 Command Execution Sequences

Once both the Service Provider and Service Consumer have performed the startup sequence, the system is ready to begin issuing and executing commands.

5.1.4 Command Start Sequence

The initial start sequence to execute a single new command follows this pattern:

1. The User of the Service Consumer issues a request for a command to be executed.
2. The Service Consumer publishes the `FunctionCommandType` with a unique session ID, the source ID of the Service Consumer, and the destination ID of the desired Service Provider.
3. The Service Provider, upon notification of the new `FunctionCommandType`, publishes a new `FunctionCommandStatusType` with (1) the same session ID as the new `FunctionCommandType`, (2) the status of `ISSUED` and (3) the reason of `SUCCEEDED` to notify the Service Consumer it has received the new command.

The Command Start Sequence for a new command is shown in Figure 28. This pattern will be repeated each time a new command is requested. Note that the Command Start Sequence differs if the `FunctionCommandType` has a `sessionID` that matches another `FunctionCommandType` that currently exists on the DDS bus. This is considered a command update and detailed in Section 5.1.4.2.

After the Command Start Sequence, the sequence can take different paths depending on the actual execution of the command, detailed from Section 5.1.4.1 to Section 5.1.4.5, but they do not enumerate all of the possible execution paths. Other paths (e.g., an objective failing) will follow a similar pattern to other failures; all are required to follow the state diagram shown in Figure 22 and eventually end with the Command Cleanup Sequence (shown in Figure 35).

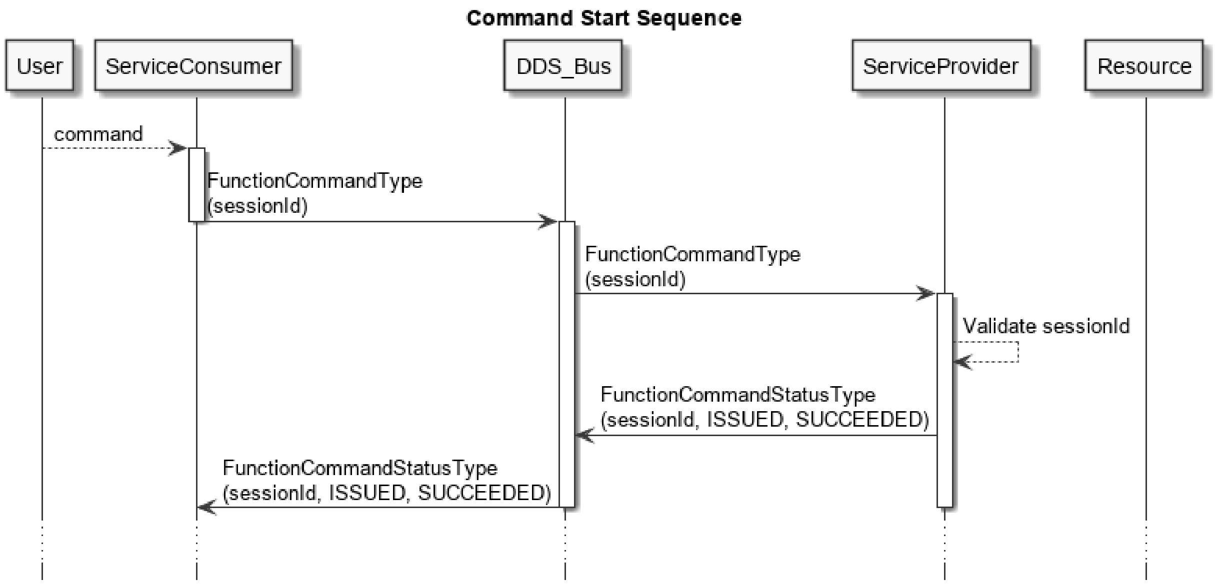

Figure 28: Sequence Diagram for the Start of a Command Execution.

5.1.4.1 Command Execution Once a Service Provider starts to process a command, the Command Execution sequence is:

1. The Service Provider publishes a `FunctionCommandAckReportType` with matching session ID and parameters as the `FunctionCommandType` it is starting to process.
2. The Service Provider performs any validation and negotiation with backing resources as necessary. Once the command is ready to be executed, the Service Provider publishes a `FunctionCommandStatusType` with a status `COMMANDED` and reason `SUCCEEDED` to notify the Service Consumer that the command has been validated and commanded to start execution.
3. Once the command has begun executing, the Service Provider publishes a `FunctionCommandStatusType` with a status `EXECUTING` and reason `SUCCEEDED` to notify the Service Consumer that the command has been validated and commanded to start.
4. If the Function has a defined `FunctionExecutionStatusReportType`, the Service Provider must publish a new instance with matching session ID as the associated `FunctionCommandType`. The `FunctionExecutionStatusReportType` must be updated by the Service Provider throughout the execution as dictated by the definitions of the command-specific attributes in the execution status report.

The command execution sequence is shown in Figure 29. This sequence holds until the command completes execution.

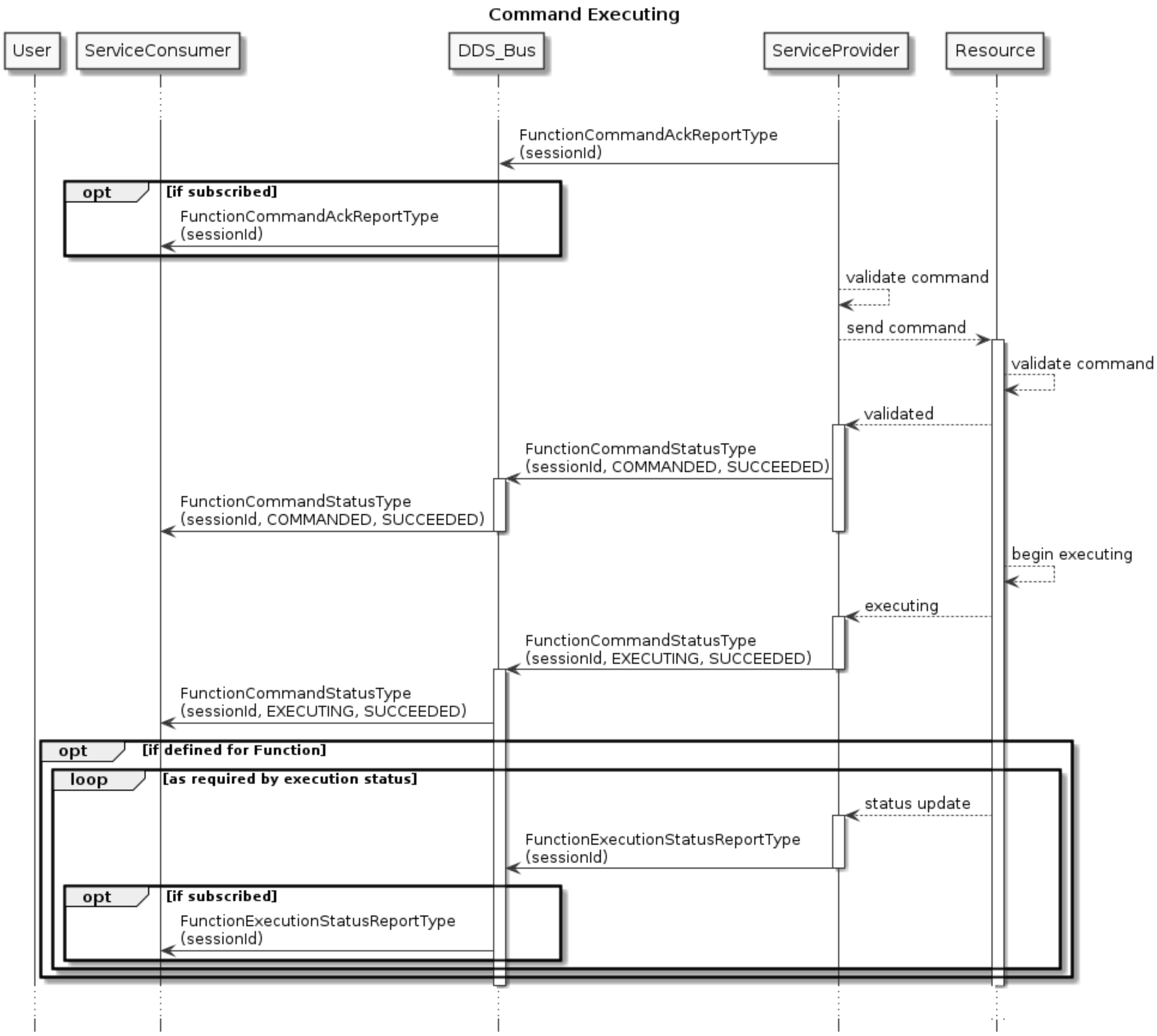

Figure 29: Beginning Sequence Diagram for a Command Execution.

The normal successful conclusion of a command being executed in some cases is initiated by the Service Consumer (an endless GlobalVector command concluded by canceling it) and in other cases is initiated by the Service Provider (a GlobalWaypoint commanded concluded by reaching the last waypoint). Unless otherwise explicitly stated, it is assumed the Service Provider will be able to identify the successful conclusion of a command. In the cases where commands are defined to be indeterminate the Service Consumer must cancel the command when the Service Consumer no longer desires the command to be executed.

5.1.4.2 Updating a Command An updated command is defined as a command with a source ID and session ID identical to the current command being processed by the Service Provider, but whose timestamp is newer than the current command. Only a command that is in a non-terminal state may be updated - otherwise, the Service Consumer must follow the normal command cleanup process and issue a new command with an updated unique session ID. If a command is in a terminal state, the Service Provider must ignore an update to that command.

When the Service Provider receives an updated command, it is required to take one of two possible actions:

1. If the current command is in a non-terminal state (`ISSUED`, `COMMANDED`, or `EXECUTING`), then the Service Provider publishes a `FunctionCommandStatusType` with a status `ISSUED` and reason `UPDATED`. The state machine then restarts and proceeds through the normal command flow detailed in 5.1.4. The Service Provider must consider the updated command as an entirely new command, resetting any internal state related to the command (e.g. a timer that keeps track of command timeout).

2. If the current command is in a terminal state (`COMPLETED`, `CANCELED`, or `FAILED`), then the updated command cannot be processed, and the Service Provider must publish a `FunctionCommandStatusType` with a status `FAILED` and follow the normal command cleanup process.

The flow control for command update is detailed below:

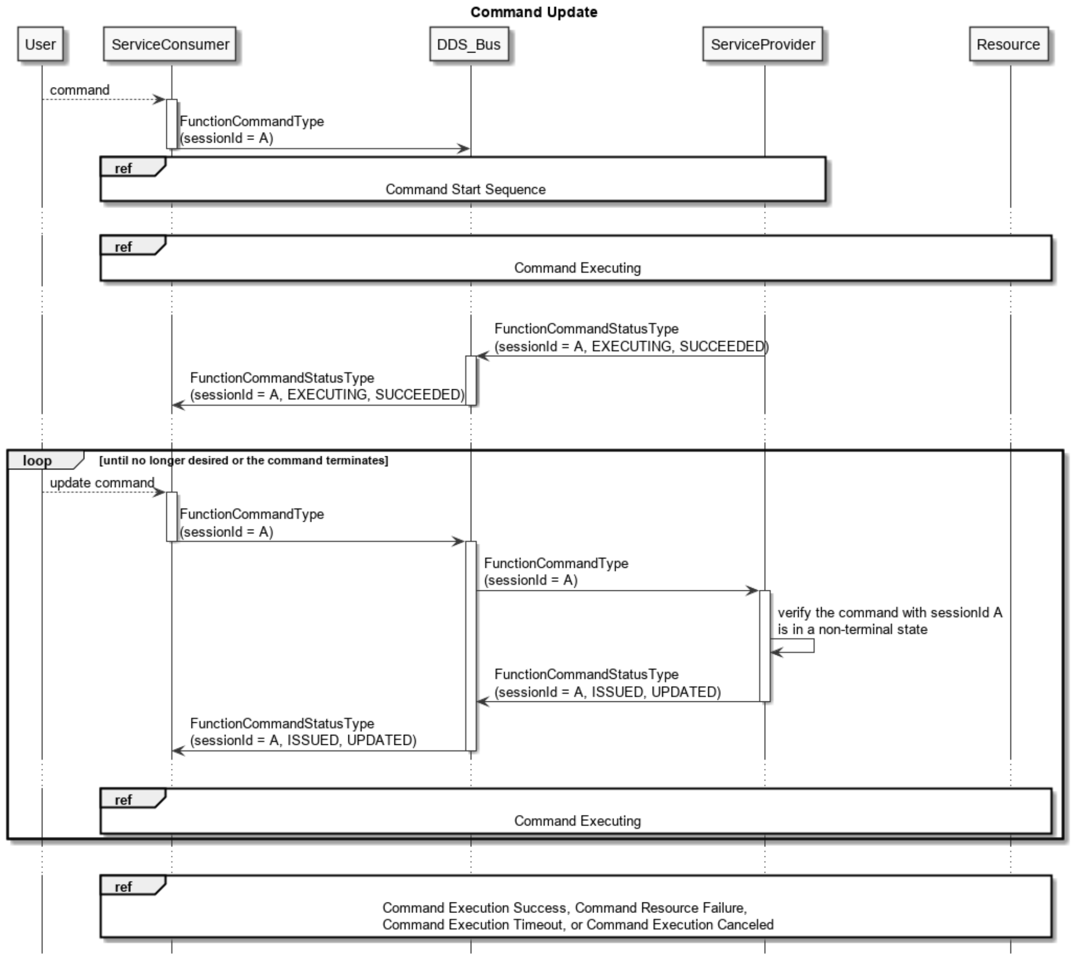

Figure 30: Sequence Diagram for Command Update.

5.1.4.3 Command Execution Success

When the Service Provider determines a command has successfully completed, it must update the associated `FunctionCommandStatusType` with as status of `COMPLETED` and reason of `SUCCEEDED`. This signals to the Service Consumer that the command has completed successfully.

The Command Execution Success sequence is shown in Figure 31.

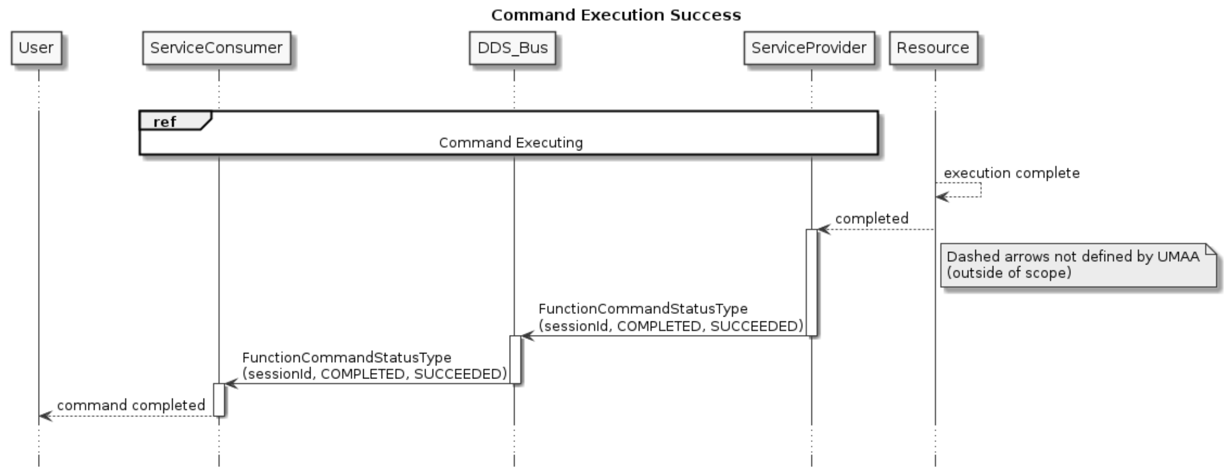

Figure 31: Sequence Diagram for a Command That Completes Successfully.

5.1.4.4 Command Execution Failure The command may fail to complete for any number of reasons including software errors, hardware failures, or unfavorable environmental conditions. The Service Provider may also reject a command for a number of reasons including inability to perform the task, malformed or out of range requests, or a command being interrupted by a higher priority process. In all cases, the Service Provider must publish a `FunctionCommandStatusType` with an identical `sessionID` as the originating `FunctionCommandType` with a status of `FAILED` and the reason that reflects the cause of the failure (`VALIDATION_FAILED, SERVICE_FAILED, OBJECTIVE_FAILED,` etc).

Figure 32 and Figure 33 provide examples where a command has failed.

In the first example, the backing Resource failed and the Service Provider is unable to communicate with it. In this case, the Service Provider will report a `FunctionCommandStatusType` with a status of `FAILED` and a reason of `RESOURCE_FAILED`. This is shown in Figure 32.

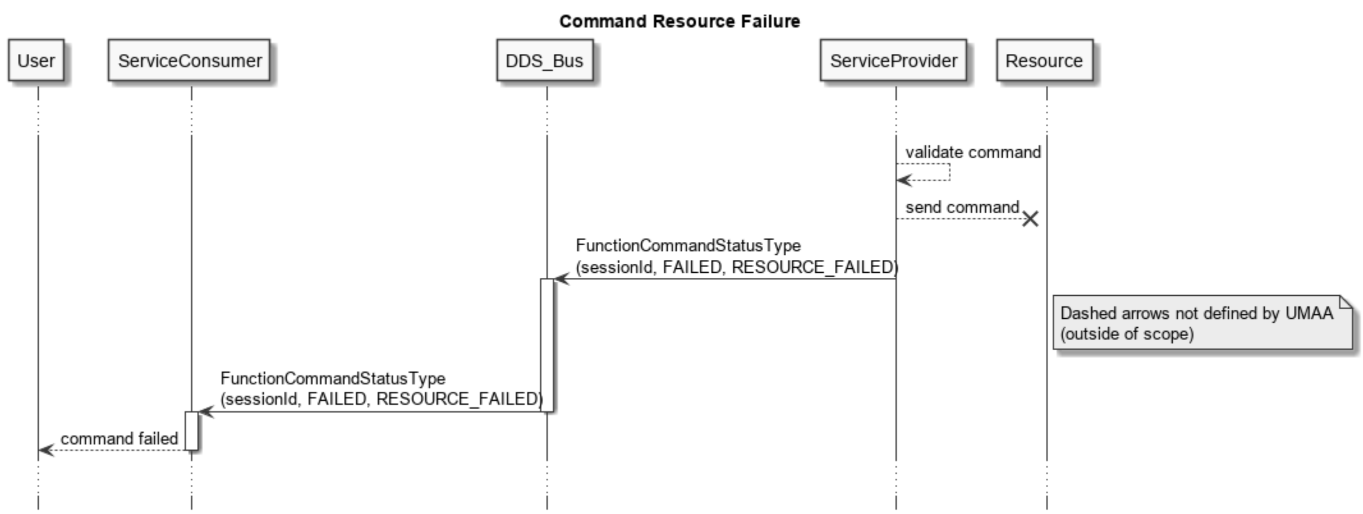

Figure 32: Sequence Diagram for a Command That Fails due to Resource Failure.

In the second example, the Resource takes too long to respond, so the Service Provider cancels the request and reports a `FunctionCommandStatusType` with a status of `FAILED` and a reason of `TIMEOUT`. This is shown in Figure 33.

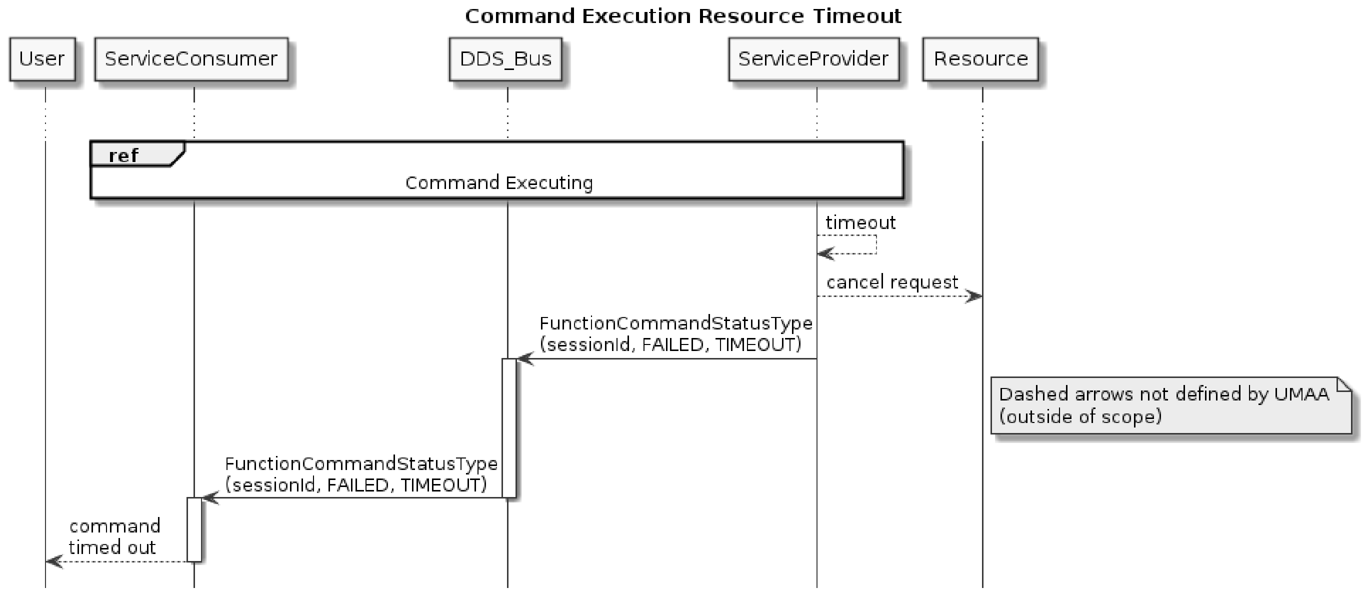

Figure 33: Sequence Diagram for a Command That Times Out Before Completing.

Other failure conditions will follow a similar pattern: when the failure is recognized, the Service Provider will publish a `FunctionCommandStatusType` with a status of `FAILED` and a reason that reflect the cause of the failure.

5.1.4.5 Command Canceled The Service Consumer may decide to cancel the command before processing is finished. To signal a desire to cancel a command, the Service Consumer disposes of the existing `FunctionCommandType` from the DDS bus before the execution is complete. When notified of the command disposal, and if the Service Provider is able to cancel the command, it should respond to the Service Consumer with a `FunctionCommandStatusType` with both the status and reason as `CANCELED`. At this point, the DDS bus should dispose of the `FunctionCommandStatusType`, the `FunctionCommandAckReportType` and, (if defined for the Function service) the `FunctionExecutionStatusReportType`. This is shown in Figure 34. If the command cannot be canceled, then the Service Provider can continue to update the command status until the execution is completed. Reporting will include `FunctionCommandStatusType` with a status of `COMPLETED` and a reason of `SUCCEEDED`. Then, the DDS bus should dispose of the `FunctionCommandStatusType`, the `FunctionCommandAckReportType`, and (if defined for the Function service) the `FunctionExecutionStatusReportType`.

There is no new, unique, or specific status message response to a cancel command from the Service Provider. The cancel command status can be inferred through the corresponding `FunctionCommandStatusType` status and reason updates.

On loss of liveliness of a Service Provider while executing a command, all Service Consumers must cancel (dispose) all in-process commands with that Service Provider.

On loss of liveliness of a Service Consumer while executing a command, all Service Providers must treat the command as canceled. This means the service should report the `CANCELED` status for the command, and then dispose the command status, ack, and execution status (if one exists).

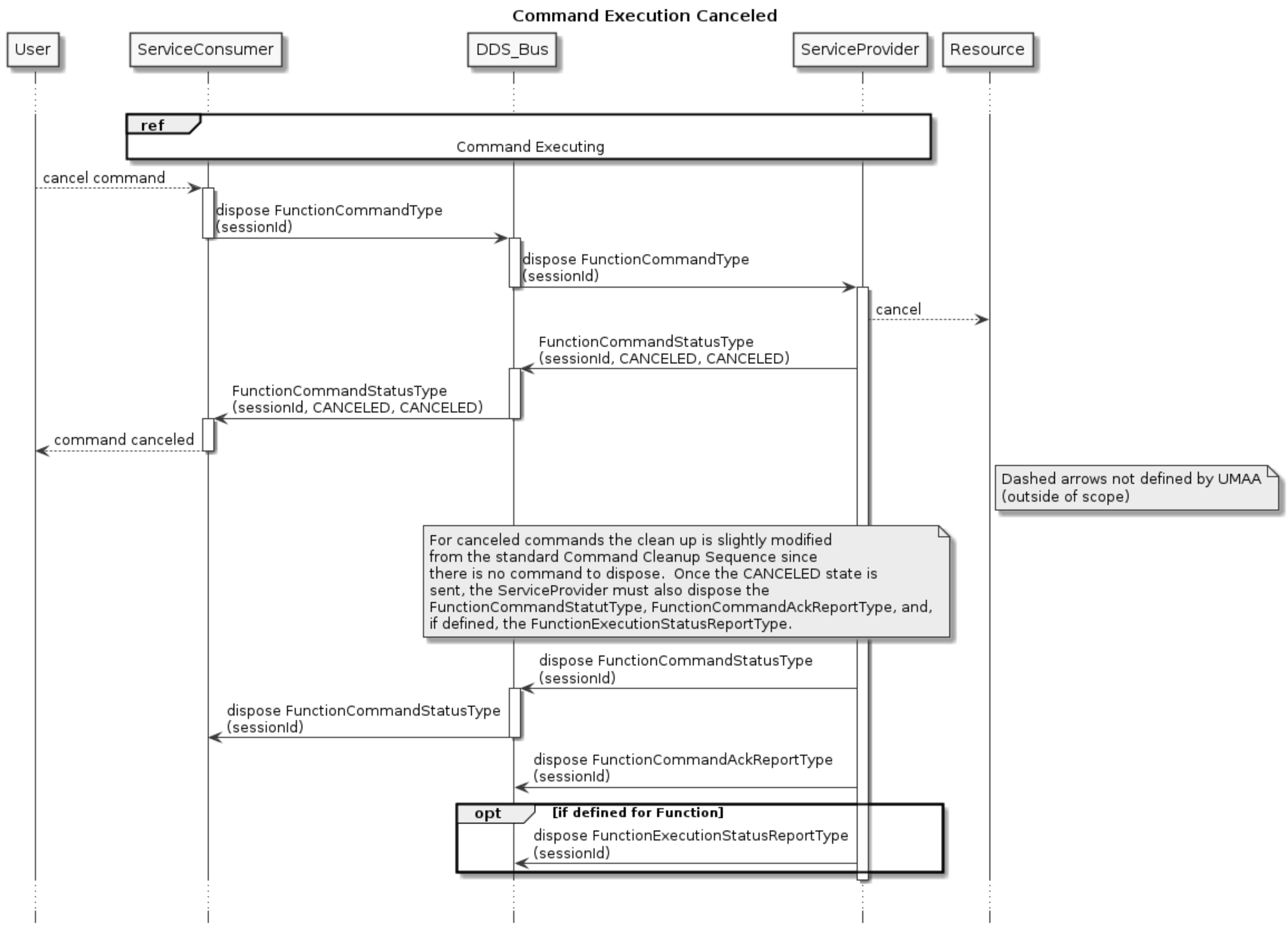

Figure 34: Sequence Diagram for a Command That is Canceled by the Service Consumer Before the Service Provider can Complete It.

5.1.5 Command Cleanup

The Service Consumer and Service Provider are responsible for disposing of corresponding data that is published to the DDS bus when the command is no longer active. With the exception of a canceled command, the signal that a `FunctionCommandType` can be disposed is when the `FunctionCommandStatusType` reports a terminal state (`COMPLETED` or `FAILED`)[3]. In turn, the signal that a `FunctionCommandStatusType`, `FunctionCommandAckReportType`, and (if defined for the Function service) the `FunctionExecutionStatusReportType` can be disposed is when the corresponding `FunctionCommandType` has been disposed. This is shown in Figure 35.

[3] While `CANCELED` is also a terminal state, the `CANCELED` command cleanup is handled specially as part of the cancelling sequence and, as such, does not need to be handled here.

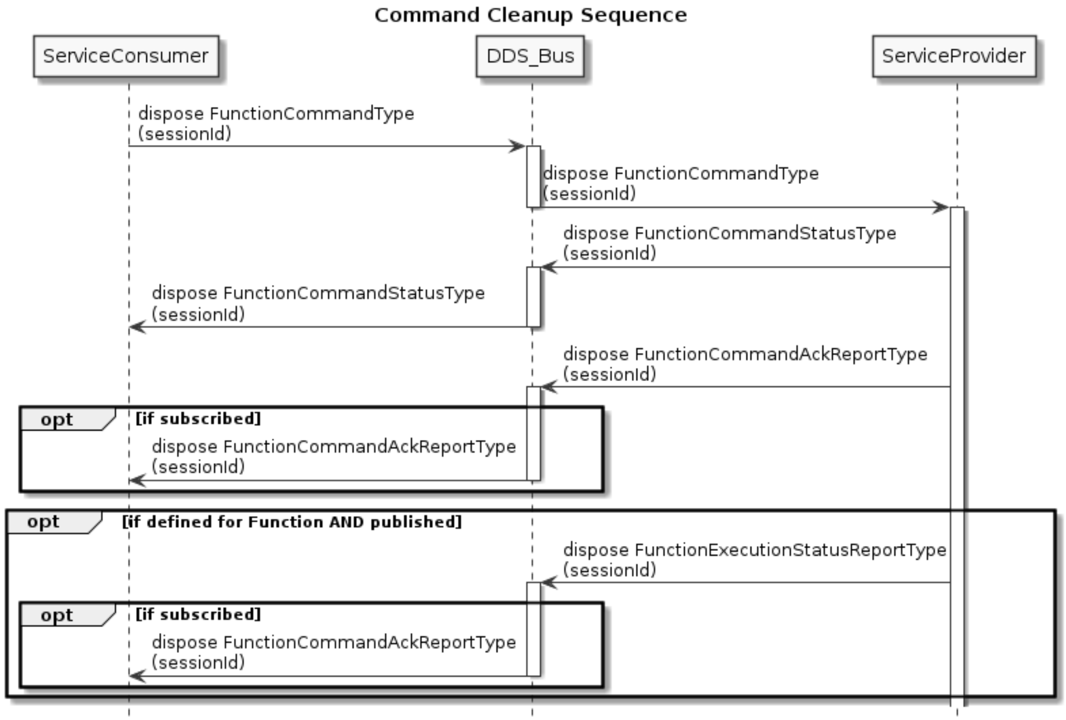

Figure 35: Sequence Diagram Showing Cleanup of the Bus When a Command Has Been Completed and the Service Consumer No Longer Wishes to Maintain the Commanded State.

5.1.6 Command Shutdown Sequence

As part of shutdown, both the Service Provider and Service Consumer are required to perform a shutdown sequence. This shutdown cleans up resources on the DDS bus and informs the system that the Service Provider and Service Consumer are no longer available.

The Service Provider and Service Consumer can shut down in any order. The sequence diagram is shown in Figure 36.

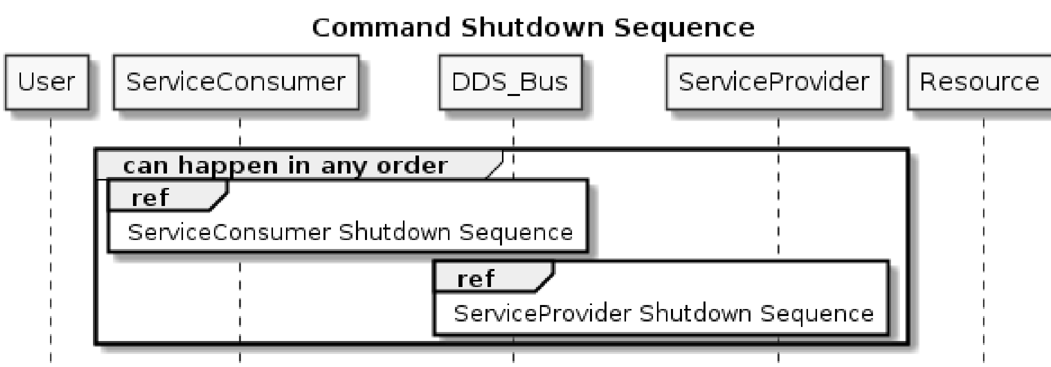

Figure 36: Sequence Diagram for Command Shutdown.

5.1.6.1 Service Provider Shutdown Sequence

During shutdown, the Service Provider is required to fail any incomplete requests and then unregisters as a publisher of the `FunctionCommandStatusType`, `FunctionCommandAckReportType`, and (if defined for the Function service) the `FunctionExecutionStatusReportType`.

The Service Provider is also required to unsubscribe from the `FunctionCommandType`.

The Service Provider Shutdown sequence is shown in Figure 37.

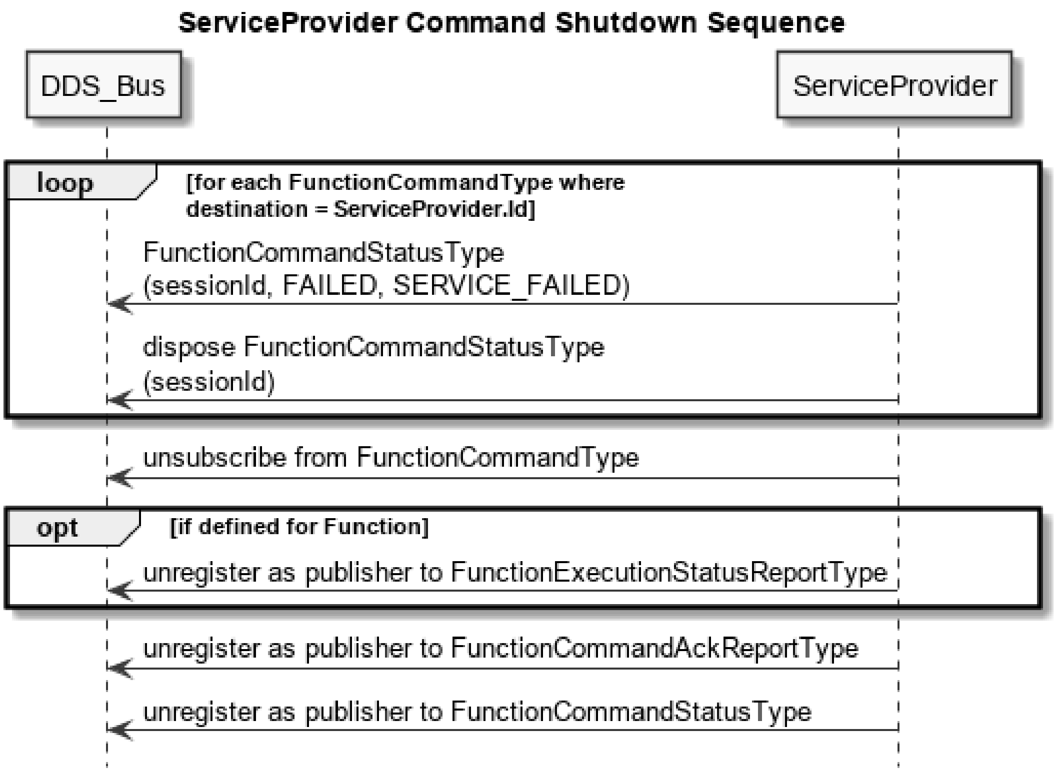

Figure 37: Sequence Diagram for Command Shutdown for Service Providers.

5.1.6.2 Service Consumer Shutdown Sequence During shutdown, the Service Consumer is required to cancel any incomplete requests and then unregister as a publisher of the `FunctionCommandType`.

The Service Consumer is also required to unsubscribe from the `FunctionCommandStatusType`, the `FunctionCommandAckReportType` if subscribed, and the `FunctionExecutionStatusReportType` if defined for the Function service and subscribed.

The Service Consumer Shutdown sequence is shown in Figure 38.

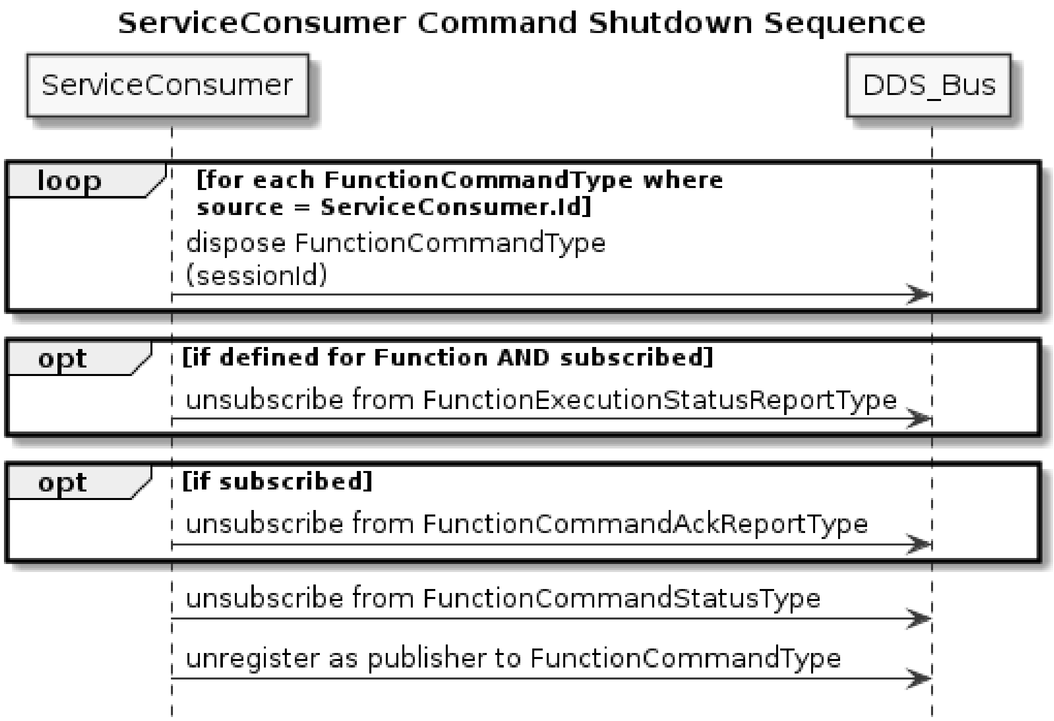

Figure 38: Sequence Diagram for Command Shutdown for Service Consumers.

5.2 Request / Reply

This section defines the flow of control for request/reply over the DDS bus. A request/reply is used to obtain data or status from a specific Service Provider.

A Service Provider is required to reply to all requests it receives. In the case of requests with no query data, this is accomplished via a DDS subscribe. In the case of a request with associated query data, a message with the query data must be published by the requester. To direct a request at a specific Service Provider or set of services, UMAA defines a `destination` GUID as part of requests.

The sequence diagrams in Sections 39 through 43 demonstrate different exchanges between a Service Consumer and Service Provider. Within the diagrams, the dashed arrows represent implementation-specific communications that are outside of UMAA's scope. Additionally, these sequence diagrams are examples of one possible implementation. Other implementations may have different communication patterns between the Service Provider and the Resource, or be implemented completely within the Service Provider process itself (no external Resource). However, in all implementations, UMAA-defined exchanges with the DDS bus between the Service Consumer and Service Provider must happen in the order shown within the sequence diagrams.

5.2.1 Request/Reply without Query Data

Figure 39 shows the sequence of exchanges in the case where there is no specific query data (i.e., the service is always just providing the current data to the bus).

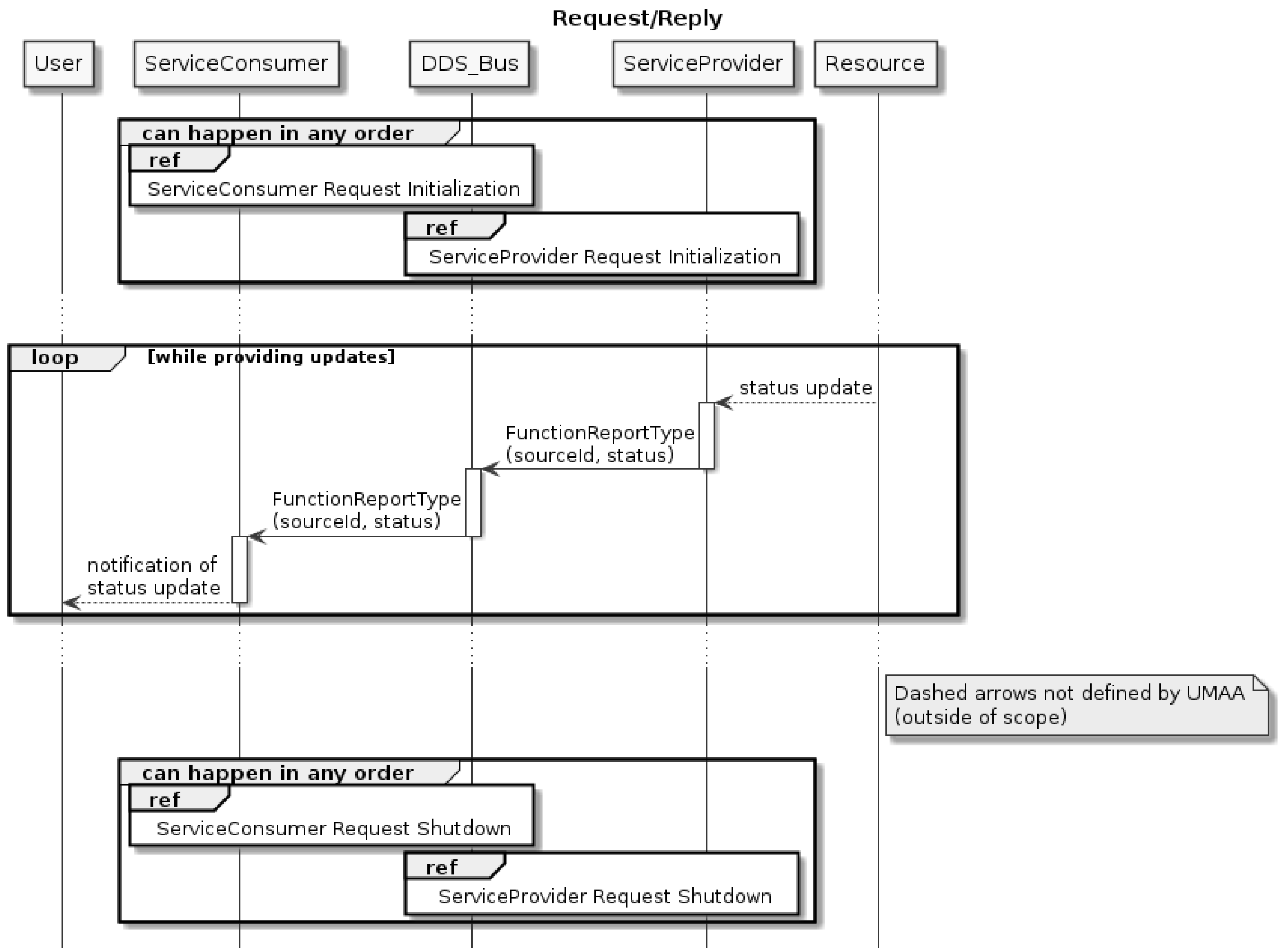

Figure 39: Sequence Diagram for a Request/Reply for Report Data That Does Not Require any Specific Query Data.

5.2.1.1 Service Provider Startup Sequence The Service Provider registers as a publisher of `FunctionReportType`s to be able to respond to requests. The Service Provider must also handle reports that exist on the bus from a previous instantiation, either by providing an immediate update or, if the status is unrecoverable, disposing of the old `FunctionReportType`. This is shown in Figure 40.

As `FunctionReportType` updates are required (either through event-driven changes or periodic updates), the Service Provider publishes the updated data. The DDS bus will deliver the updates to the Service Consumer.

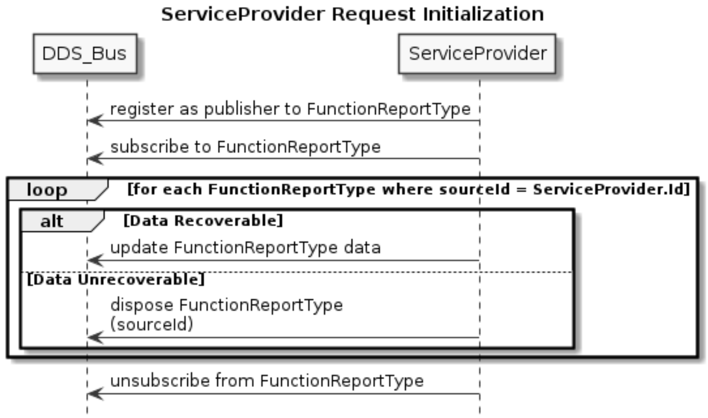

Figure 40: Sequence Diagram for Initialization of a Service Provider to Provide `FunctionReportTypes`.

5.2.1.2 Service Consumer Startup Sequence The Service Consumer subscribes to the `FunctionReportType` to signal an outstanding request for updates. This is shown in Figure 41.

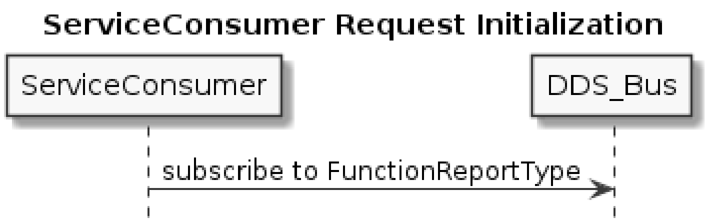

Figure 41: Sequence Diagram for Initialization of a Service Consumer to Request `FunctionReportTypes`.

5.2.1.3 Service Provider Shutdown To no longer provide `FunctionReportTypes`, the Service Provider disposes of the `FunctionReportType` and unregisters as a publisher of the data (shown in Figure 42).

Figure 42: Sequence Diagram for Shutdown of a Service Provider.

5.2.1.4 Service Consumer Shutdown To no longer request `FunctionReportTypes`, the Service Consumer unsubscribes from `FunctionReportType` (shown in Figure 43).

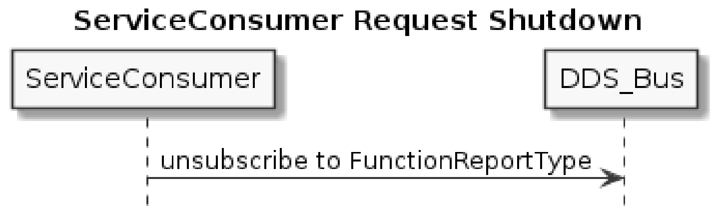

Figure 43: Sequence Diagram for Shutdown of a Service Consumer.

5.2.2 Request/Reply with Query Data

Currently, UMAA does not define any request/reply interactions with query data, but it is expected that some will be defined. When defined, this section will be expanded to describe how they must be used.

6 Engineering Operations (EO) Services and Interfaces

6.1 Services and Interfaces

The interfaces in the following subsections describe how each UCS-UMAA topic is defined by listing the name, namespace, and member attributes. The "name" corresponds with the message name of a given service interface. The "namespace" defines the scope of the "name" where similar commands are grouped together. The "member attributes" are fields that can be populated with differing data types, e.g. a generic "depth" attribute could be populated with a double data value. Note that using a UCS-UMAA "Topic Name" requires using the fully-qualified namespace plus the topic name.

Each interface topic is referenced by a UMAA service and is defined as either an input or output interface.

Attributes ending in one or more asterisk(s) denote the following:
* = Key (annotated with @key in IDL file; vendors may use different notation to indicate a key field)
† = Optional (annotated with @optional in IDL file; vendors may use different notation to indicate an optional field)

Optional fields should be handled as described in the UMAA Compliance Specification.

Commands issued on the DDS bus must be treated as if they are immutable in UMAA and, therefore, if updated (treated incorrectly as mutable), the resulting service actions are indeterminate and flow control protocols are no longer guaranteed.

Operations without DDS Topics

⊕ = Operations that are handled directly in DDS

query<...> - All query operations are used to retrieve the correlated report message. For UMAA, this operation is accomplished through subscribing to the appropriate DDS topic.

cancel<...> - All cancel operations are used to nullify the current command. For UMAA, this operation is accomplished through the DDS dispose action on the publisher.

report<...>CancelCommandStatus - All cancel reports are included here to show completeness of the MDE model mapping to UMAA. For UMAA, this operation is not used. Instead, the cancel status is inferred from the associated command status. If the cancel command is successful, the corresponding command will fail with a command status and reason of CANCELED. If the corresponding command status reports COMPLETED, then this cancel command has failed.

6.1.1 AnchorControl

The purpose of this service is to provide the operations and interfaces to control the anchor on the vehicle. Three modes of operation: stop, lower, and raise are supported per anchor. When canceled, stops at current position.

Table 8: AnchorControl Operations

Service Requests (Inputs)	Service Responses (Outputs)
setAnchor	reportAnchorCommandStatus
queryAnchorCommandAck⊕	reportAnchorCommandAck
cancelAnchorCommand⊕	reportAnchorCancelCommandStatus⊕

See Section 6.1 for an explanation of the inputs and outputs marked with a ⊕.

6.1.1.1 reportAnchorCommandAck

Description: This operation is used to provide the Anchor commanded values.

Namespace: UMAA::EO::AnchorControl

Topic: AnchorCommandAckReportType

Data Type: AnchorCommandAckReportType

Table 9: AnchorCommandAckReportType Message Definition

Attribute Name	Attribute Type	Attribute Description
Additional fields included from UMAA::UMAACommandStatusBase		
command	AnchorCommandType	The source command.

6.1.1.2 reportAnchorCommandStatus

Description: This operation is used to report the status of the associated command message.

Namespace: UMAA::EO::AnchorControl

Topic: AnchorCommandStatusType

Data Type: AnchorCommandStatusType

Table 10: AnchorCommandStatusType Message Definition

Attribute Name	Attribute Type	Attribute Description
Additional fields included from UMAA::UMAACommandStatus		

6.1.1.3 setAnchor

Description: This operation is used to set the control parameters for the Anchor service.

Namespace: UMAA::EO::AnchorControl

Topic: AnchorCommandType

Data Type: AnchorCommandType

Table 11: AnchorCommandType Message Definition

Attribute Name	Attribute Type	Attribute Description
Additional fields included from UMAA::UMAACommand		
action	AnchorActionEnumType	Defines the attributes used to control the anchor.

6.1.2 AnchorSpecs

The purpose of this service is to report the anchor specifications on the vehicle.

Table 12: AnchorSpecs Operations

Service Requests (Inputs)	Service Responses (Outputs)
queryAnchorSpecs⊕	reportAnchorSpecs

See Section 6.1 for an explanation of the inputs and outputs marked with a ⊕.

6.1.2.1 reportAnchorSpecs

Description: This operation is used to report the specification parameters for the Anchor service.

Namespace: UMAA::EO::AnchorSpecs

Topic: AnchorSpecsReportType

Data Type: AnchorSpecsReportType

Table 13: AnchorSpecsReportType Message Definition

Attribute Name	Attribute Type	Attribute Description
Additional fields included from UMAA::UMAAStatus		
anchorDescription	StringShortDescription	A description of the anchor.
anchorHoldingPower	Mass	Defines the anchor holding power as determined by full scale anchor drag tests in a firm sand bottom.
anchorHoldingPowerRatio	Ratio	The anchor holding power ratio is the ratio of the anchor holding power to the anchor size.
anchorKind	AnchorKindEnumType	Defines the type of anchor (e.g., commercial stockless, standard navy stockless, etc.).
anchorLocation	AnchorLocationEnumType	Defines the anchor location (i.e., bower anchor, stern anchor, keel anchor).
anchorSize	Mass	Defines the anchor size and is expressed in terms of the mass of the anchor.
rodeLength	Distance	The length of chain/rope that can be paid out by the anchoring system.
rodeSize	Distance	For a chain rode it defines the link or chain size (the nominal diameter of the link material in the grip area). For a rope rode it defines the diameter of the rope.
rodeType	AnchorRodeEnumType	The rode type of the anchoring system.
rodeWorkingLoadLimit	Force	The rated working load limit for the chain/rope of the anchoring system.

6.1.3 AnchorStatus

The purpose of this service is to provide the operations and interfaces to monitor the anchor on the vehicle.

Table 14: AnchorStatus Operations

Service Requests (Inputs)	Service Responses (Outputs)
queryAnchor⊕	reportAnchor

See Section 6.1 for an explanation of the inputs and outputs marked with a ⊕.

6.1.3.1 reportAnchor

Description: This operation is used to report the data parameters for the Anchor service.

Namespace: UMAA::EO::AnchorStatus

Topic: AnchorReportType

Data Type: AnchorReportType

Table 15: AnchorReportType Message Definition

Attribute Name	Attribute Type	Attribute Description
Additional fields included from UMAA::UMAAStatus		
rodeLengthPaidOut	Distance	The current length of chain/rope that is paid out.
state	AnchorStateEnumType	The current operational state of the anchor.

6.1.4 BatterySpecs

The purpose of this service is to provide the battery specifications.

Table 16: BatterySpecs Operations

Service Requests (Inputs)	Service Responses (Outputs)
queryBatterySpecs⊕	reportBatterySpecs

See Section 6.1 for an explanation of the inputs and outputs marked with a ⊕.

6.1.4.1 reportBatterySpecs

Description: This operation is used to report the system specifications of the batteries of the vehicle.

Namespace: UMAA::EO::BatterySpecs

Topic: BatterySpecsReportType

Data Type: BatterySpecsReportType

Table 17: BatterySpecsReportType Message Definition

Attribute Name	Attribute Type	Attribute Description
Additional fields included from UMAA::UMAAStatus		
cellMinimumVoltage	PowerBusVoltage	The minimum safe voltage of a cell in the battery system.
maxCapacity	Charge	The Coulombs available by the battery system when fully charged.
maxChargingCurrent	PowerBusCurrent	The maximum charging current of the battery system.
maxChargingTemp	Temperature	The maximum charging temperature.
maxOutputCurrent	PowerBusCurrent	The maximum output current of the battery system.
maxPulsedChargeCurrent†	BatteryCurrent	The maximum pulse charge that can be handled based on specified duration.
maxPulsedChargeCurrentDuration†	BatteryCurrentDuration	The maximum time that the pulse charge current can be handled.
maxStorageTemp	Temperature	The maximum storage temperature.
maxTemperature	Temperature	The maximum operating temperature.
maxVoltage	PowerBusVoltage	The maximum voltage of the battery system.
minChargeCycles†	BatteryCycles	The number of charge cycles before the battery is under the minimum charge specification.
minChargingTemp	Temperature	The minimum charging temperature.
minStorageTemp	Temperature	The minimum storage temperature.
minTemperature	Temperature	The minimum operating temperature.
minVoltage	PowerBusVoltage	The minimum voltage of the battery system.
name	StringShortDescription	The description of the battery system.
nominalCapacity	AmpHours	The nominal capacity at 1C rate.
nominalEnergy	WattHours	The nominal energy at 1C rate.
nominalVoltage	PowerBusVoltage	The nominal voltage of the battery system.
peakDischargeCurrent	PowerBusCurrent	The maximum current discharge for up to 10 seconds.

6.1.5 BatteryStatus

The purpose of this service is to provide the current battery status.

Table 18: BatteryStatus Operations

Service Requests (Inputs)	Service Responses (Outputs)
queryBattery⊕	reportBattery

See Section 6.1 for an explanation of the inputs and outputs marked with a ⊕.

6.1.5.1 reportBattery

Description: This operation is used to report the current status of battery of the vehicle.

Namespace: UMAA::EO::BatteryStatus

Topic: BatteryReportType

Data Type: BatteryReportType

Table 19: BatteryReportType Message Definition

Attribute Name	Attribute Type	Attribute Description
Additional fields included from UMAA::UMAAStatus		
cells→listID	LargeList<BatteryCellDataType>	The charge data for each cell in the battery. This attribute is implemented as a large list, see subsection 3.8 for an explanation. The associated topic is UMAA::EO::BatteryStatus::BatteryReportTypeCellsListElement.
chargeRemaining†	EnergyPercent	The amount of charge remaining.
current†	PowerBusCurrent	The runtime current of the battery.
energyUsageRate†	ElectricalPower	The rates of power at a moment in time.
hours†	DurationHours	The total runtime of the battery in its lifetime.
state	PowerPlantStateEnumType	Describes the current power plant state.
temp†	Temperature	The current temperature of the battery.
voltage†	PowerBusVoltage	The runtime voltage of the battery.

6.1.6 EngineControl

The purpose of this service is to provide the control of the engine on the vehicle.

Table 20: EngineControl Operations

Service Requests (Inputs)	Service Responses (Outputs)
setEngine	reportEngineCommandStatus
queryEngineCommandAck⊕	reportEngineCommandAck
cancelEngineCommand⊕	reportEngineCancelCommandStatus⊕

See Section 6.1 for an explanation of the inputs and outputs marked with a ⊕.

6.1.6.1 reportEngineCommandAck

Description: This operation is used to report the commanded values to the engine of the vehicle.

Namespace: UMAA::EO::EngineControl

Topic: EngineCommandAckReportType

Data Type: EngineCommandAckReportType

Table 21: EngineCommandAckReportType Message Definition

Attribute Name	Attribute Type	Attribute Description
Additional fields included from UMAA::UMAACommandStatusBase		
command	EngineCommandType	The source command.

6.1.6.2 reportEngineCommandStatus

Description: This operation is used to report the status of engine command.

Namespace: UMAA::EO::EngineControl

Topic: EngineCommandStatusType

Data Type: EngineCommandStatusType

Table 22: EngineCommandStatusType Message Definition

Attribute Name	Attribute Type	Attribute Description
Additional fields included from UMAA::UMAACommandStatus		

6.1.6.3 setEngine

Description: This operation is used to control the engines of the vehicle. The consumer must perform a "cancel" of the command to initiate the end of command execution as this command has no determinate end of execution.

Namespace: UMAA::EO::EngineControl

Topic: EngineCommandType

Data Type: EngineCommandType

Table 23: EngineCommandType Message Definition

Attribute Name	Attribute Type	Attribute Description
Additional fields included from UMAA::UMAACommand		
plugState†	OnOffStatusEnumType	The desired glow plug state.

Attribute Name	Attribute Type	Attribute Description
propulsion†	PropulsionType	The desired propulsive value of the engine, when the commanded state is ON.
state	IgnitionControlEnumType	The desired power state of the subsystem.

6.1.7 EngineStatus

The purpose of this service is to report the current status of the engine on the vehicle.

Table 24: EngineStatus Operations

Service Requests (Inputs)	Service Responses (Outputs)
queryEngine⊕	reportEngine

See Section 6.1 for an explanation of the inputs and outputs marked with a ⊕.

6.1.7.1 reportEngine

Description: This operation is used to report the current status of engines of the vehicle.

Namespace: UMAA::EO::EngineStatus

Topic: EngineReportType

Data Type: EngineReportType

Table 25: EngineReportType Message Definition

Attribute Name	Attribute Type	Attribute Description
Additional fields included from UMAA::UMAAStatus		
coolantLevel†	VolumePercent	The engine coolant level.
coolantPressure†	PressureKiloPascals	The engine coolant pressure.
coolantTemp†	Temperature	The temperature of the engine coolant.
engineTemp†	Temperature	The temperature of the engine.
exhaustTemp†	Temperature	The engine exhaust temperature.
glowPlugIndicator†	boolean	The glow plug indicator state.
glowPlugState†	OnOffStatusEnumType	The glow plug state.
glowPlugTemp†	Temperature	The glow plug temperature.
glowPlugTimeRemaining†	DurationSeconds	The time remaining for the glow plug to be on.
hours†	DurationHours	The total runtime of the engine in its lifetime.
manifoldAirTemp†	Temperature	The engine manifold air temperature.
manifoldPressure†	PressureKiloPascals	The engine manifold pressure.
oilLevel†	VolumePercent	The engine oil level.

Attribute Name	Attribute Type	Attribute Description
oilPressure†	PressureKiloPascals	The engine oil pressure.
oilTemp†	Temperature	The engine oil temperature.
percentOilPressure†	PressurePercent	The engine oil pressure.
RPM†	EngineSpeed	The engine RPM.
state	IgnitionStateEnumType	The current ignition state.
throttle†	Effort	The engine throttle. Negative values will be regarded as zero for non-reversible engines.

6.1.8 FuelTankSpecs

The purpose of this service is to provide the current fuel tank specifications.

Table 26: FuelTankSpecs Operations

Service Requests (Inputs)	Service Responses (Outputs)
queryFuelTankSpecs⊕	reportFuelTankSpecs

See Section 6.1 for an explanation of the inputs and outputs marked with a ⊕.

6.1.8.1 reportFuelTankSpecs

Description: This operation is used to report the system specifications of the fuel tanks of the vehicle.

Namespace: UMAA::EO::FuelTankSpecs

Topic: FuelTankSpecsReportType

Data Type: FuelTankSpecsReportType

Table 27: FuelTankSpecsReportType Message Definition

Attribute Name	Attribute Type	Attribute Description
Additional fields included from UMAA::UMAAStatus		
capacity	VolumeCubicMeter	The maximum capacity of the fuel tank.
name	StringShortDescription	The name of the fuel tank.

6.1.9 FuelTankStatus

The purpose of this service is to provide the current fuel tank status.

Table 28: FuelTankStatus Operations

Service Requests (Inputs)	Service Responses (Outputs)
queryFuelTank⊕	reportFuelTank

See Section 6.1 for an explanation of the inputs and outputs marked with a ⊕.

6.1.9.1 reportFuelTank

Description: This operation is used to report the current status of fuel tanks of the vehicle.

Namespace: UMAA::EO::FuelTankStatus

Topic: FuelTankReportType

Data Type: FuelTankReportType

Table 29: FuelTankReportType Message Definition

Attribute Name	Attribute Type	Attribute Description
Additional fields included from UMAA::UMAAStatus		
fuelLevel	VolumePercent	The amount of fuel remaining in the tank.
waterInFuel†	boolean	The detection of water in the fuel. This will be true when the detected water is above specified threshold.

6.1.10 GeneratorSpecs

The purpose of this service is to provide the current generator specifications.

Table 30: GeneratorSpecs Operations

Service Requests (Inputs)	Service Responses (Outputs)
queryGeneratorSpecs⊕	reportGeneratorSpecs

See Section 6.1 for an explanation of the inputs and outputs marked with a ⊕.

6.1.10.1 reportGeneratorSpecs

Description: This operation is used to report the system specifications of the generators of the vehicle.

Namespace: UMAA::EO::GeneratorSpecs

Topic: GeneratorSpecsReportType

Data Type: GeneratorSpecsReportType

Table 31: GeneratorSpecsReportType Message Definition

Attribute Name	Attribute Type	Attribute Description
Additional fields included from UMAA::UMAAStatus		
maxCurrent	PowerBusCurrent	The maximum current supported by the generator.
maxPower	ElectricalPower	The maximum power this generator can produce.
name	StringShortDescription	The name of the generator unit.
ratedPower	ElectricalPower	The amount of generated power over a long period of operation.
ratedVoltage	PowerBusVoltage	The highest voltage provided by the generator during normal operation.

6.1.11 GeneratorStatus

The purpose of this service is to provide the current generator status.

Table 32: GeneratorStatus Operations

Service Requests (Inputs)	Service Responses (Outputs)
queryGenerator⊕	reportGenerator

See Section 6.1 for an explanation of the inputs and outputs marked with a ⊕.

6.1.11.1 reportGenerator

Description: This operation is used to report the current status of generators of the vehicle.

Namespace: UMAA::EO::GeneratorStatus

Topic: GeneratorReportType

Data Type: GeneratorReportType

Table 33: GeneratorReportType Message Definition

Attribute Name	Attribute Type	Attribute Description
Additional fields included from UMAA::UMAAStatus		
current	PowerBusCurrent	The current of the generator. A negative current indicates that the generator is sourcing, a positive current indicates that it is sinking.
state	PowerPlantStateEnumType	The state of the generator unit.
voltage	PowerBusVoltage	The actual voltage of the generator.

6.1.12 MastControl

The purpose of this service is to provide the operations and interfaces to control the mast position of the vehicle.

Table 34: MastControl Operations

Service Requests (Inputs)	Service Responses (Outputs)
setMast	reportMastCommandStatus
queryMastCommandAck⊕	reportMastCommandAck
cancelMastCommand⊕	reportMastCancelCommandStatus⊕

See Section 6.1 for an explanation of the inputs and outputs marked with a ⊕.

6.1.12.1 reportMastCommandAck

Description: This operation is used to provide the Mast commanded values.

Namespace: UMAA::EO::MastControl

Topic: MastCommandAckReportType

Data Type: MastCommandAckReportType

Table 35: MastCommandAckReportType Message Definition

Attribute Name	Attribute Type	Attribute Description
Additional fields included from UMAA::UMAACommandStatusBase		
command	MastCommandType	The source command.

6.1.12.2 reportMastCommandStatus

Description: This operation is used to report the current commanded mast action.

Namespace: UMAA::EO::MastControl

Topic: MastCommandStatusType

Data Type: MastCommandStatusType

Table 36: MastCommandStatusType Message Definition

Attribute Name	Attribute Type	Attribute Description
Additional fields included from UMAA::UMAACommandStatus		

6.1.12.3 setMast

Description: This operation is used to set the control parameters for the Mast service.

Namespace: UMAA::EO::MastControl

Topic: MastCommandType

Data Type: MastCommandType

Table 37: MastCommandType Message Definition

Attribute Name	Attribute Type	Attribute Description
	Additional fields included from UMAA::UMAACommand	
action	MastActionEnumType	The desired mast action.

6.1.13 MastStatus

The purpose of this service is to provide the operations and interfaces to provide current status of the mast on the vehicle.

Table 38: MastStatus Operations

Service Requests (Inputs)	Service Responses (Outputs)
queryMast⊕	reportMast

See Section 6.1 for an explanation of the inputs and outputs marked with a ⊕.

6.1.13.1 reportMast

Description: This operation is used to report the data parameters for the Mast service.

Namespace: UMAA::EO::MastStatus

Topic: MastReportType

Data Type: MastReportType

Table 39: MastReportType Message Definition

Attribute Name	Attribute Type	Attribute Description
	Additional fields included from UMAA::UMAAStatus	
state	MastStateEnumType	The state of the mast.

6.1.14 UVPlatformSpecs

The purpose of this service is to report the physical and operational capabilities of the vehicle. For example, this information should be used in UMAA services to validate and reject commands received that are not able to be performed by the vehicle.

Table 40: UVPlatformSpecs Operations

Service Requests (Inputs)	Service Responses (Outputs)
queryUVPlatformCapabilities⊕	reportUVPlatformCapabilities
queryUVPlatformSpecs⊕	reportUVPlatformSpecs

See Section 6.1 for an explanation of the inputs and outputs marked with a ⊕.

6.1.14.1 reportUVPlatformCapabilities

Description: This operation is used to report the current operational capabilities of the vehicle.

Namespace: UMAA::EO::UVPlatformSpecs

Topic: UVPlatformCapabilitiesReportType

Data Type: UVPlatformCapabilitiesReportType

Table 41: UVPlatformCapabilitiesReportType Message Definition

Attribute Name	Attribute Type	Attribute Description
Additional fields included from UMAA::UMAAStatus		
minWaterDepth	DistanceBSL	The smallest distance from the water surface to the sea floor required by the vehicle to operate.
surfaceCapabilities	SurfaceCapabilityLimitsType	The capabilities of the vehicle while operating on the surface.
towingCapacity†	MassMetricTon	The largest weight that may be towed by the vehicle.
underwaterCapabilities†	UnderwaterCapabilityLimitsType	The capabilities of the vehicle while operating submerged.

6.1.14.2 reportUVPlatformSpecs

Description: This operation is used to report the physical specifications of the vehicle.

Namespace: UMAA::EO::UVPlatformSpecs

Topic: UVPlatformSpecsReportType

Data Type: UVPlatformSpecsReportType

Table 42: UVPlatformSpecsReportType Message Definition

Attribute Name	Attribute Type	Attribute Description
Additional fields included from UMAA::UMAAStatus		
aftDistance	Distance	The distance from the vehicle reference origin to the most aft of the vehicle, measured along the negative X-axis of the vehicle coordinate frame.
beamAtWaterline	Distance	The distance along the Y-axis of the widest point of the hull where it meets the waterline.
bottomDistance	Distance	The distance from the vehicle reference origin to the maximum depth of the vehicle, measured along the positive Z-axis of the vehicle coordinate frame.
centerOfBuoyancy	Position3DBodyXYZ	The measurements on the vehicle coordinate frame at which the center of buoyancy of the vehicle is located.
centerOfGravity	Position3DBodyXYZ	The measurements on the vehicle coordinate frame at which the center of gravity of the vehicle is located.
diameter†	Distance	The outer diameter as a clean, cylindrical pressure hull of the unmanned vehicle.
displacement	MassMetricTon	The weight of the volume of displaced fluid up to the waterline in which the vehicle (including fuel, cargo, payloads, etc) is floating.
draft	Distance	Specifies the distance from the waterline to the bottom of the vehicle.
forwardDistance	Distance	The distance from the vehicle reference origin to the most forward of the vehicle , measured along the positive X axis of the vehicle coordinate frame.
lengthAtWaterline	Distance	(LWL) The measured distance of the vehicle at the level where it sits in the water. Measured along the X axis.
name	StringShortDescription	The name of the unmanned vehicle.
portDistance	Distance	The distance from the vehicle reference origin to the most port of the vehicle, measured along the negative Y axis of the vehicle coordinate frame.
referenceFrameOrigin	ReferenceFrameOriginEnumType	The origin from which all distance measurements are taken.
starboardDistance	Distance	The distance from the vehicle reference origin to the most starboard of the vehicle, measured along the positive Y axis of the vehicle coordinate frame.
topDistance	Distance	The distance from the vehicle reference origin to the top most part of the vehicle, measured along the negative Z axis of the vehicle coordinate frame.
weightInWater†	MassMetricTon	The weight of a vehicle in water.
weightLight	MassMetricTon	The weight of a vehicle on land with empty ballast tanks and suspended from a crane.
weightLoaded	MassMetricTon	The weight of a dry surface vehicle on land with tanks (fuel, ballast, etc) and cargo/payloads at designed capacity and suspended from a crane.

6.2 Common Data Types

Common data types define DDS types that are referenced throughout the UMAA model. These DDS types are considered common because they can be re-used as the data type for many attributes defined in service interface topics, interface topics, and other common data types. These data types are not intended to be directly published to/subscribed as DDS topics.

6.2.1 UCSMDEInterfaceSet

Namespace: UMAA::UCSMDEInterfaceSet

Description: Defines the common UCSMDE Interface Set Message Fields.

Table 43: UCSMDEInterfaceSet Structure Definition

Attribute Name	Attribute Type	Attribute Description
timeStamp	DateTime	The origination time of the data being conveyed in the message, or as close to the data or command generation time as is reasonably possible.

6.2.2 UMAACommand

Namespace: UMAA::UMAACommand

Description: Defines the common UMAA Command Message Fields.

Table 44: UMAACommand Structure Definition

Attribute Name	Attribute Type	Attribute Description
Additional fields included from UMAA::UCSMDEInterfaceSet		
source*	IdentifierType	The unique identifier of the originating source of the command interface.
destination*	IdentifierType	The unique identifier of the destination of the command interface.
sessionID*	NumericGUID	The unique identifier for the session.

6.2.3 UMAAStatus

Namespace: UMAA::UMAAStatus

Description: Defines the common UMAA Status Message Fields.

Table 45: UMAAStatus Structure Definition

Attribute Name	Attribute Type	Attribute Description
Additional fields included from UMAA::UCSMDEInterfaceSet		
source*	IdentifierType	The unique identifier of the originating source of the status interface.

6.2.4 UMAACommandStatusBase

Namespace: UMAA::UMAACommandStatusBase

Description: Defines the common UMAA Command Status Base Message Fields.

Table 46: UMAACommandStatusBase Structure Definition

Attribute Name	Attribute Type	Attribute Description
Additional fields included from UMAA::UCSMDEInterfaceSet		
source*	IdentifierType	The unique identifier of the originating source of the command status interface.
sessionID*	NumericGUID	The unique identifier for the session.

6.2.5 UMAACommandStatus

Namespace: UMAA::UMAACommandStatus

Description: Defines the common UMAA Command Status Message Fields.

Table 47: UMAACommandStatus Structure Definition

Attribute Name	Attribute Type	Attribute Description
Additional fields included from UMAA::UMAACommandStatusBase		
commandStatus	CommandStatusEnumType	The status of the command.
commandStatusReason	CommandStatusReasonEnumType	The reason for the status of the command.
logMessage	StringLongDescription	Human-readable description related to response. Systems should not parse or use any information from this for processing purposes.

6.2.6 DateTime

Namespace: UMAA::Common::Measurement::DateTime

Description: Describes an absolute time. Conforms with POSIX time standard (IEEE Std 1003.1-2017) epoch reference point of January 1st, 1970 00:00:00 UTC.

Table 48: DateTime Structure Definition

Attribute Name	Attribute Type	Attribute Description
seconds	DateTimeSeconds	The number of seconds offset from the standard POSIX (IEEE Std 1003.1-2017) epoch reference point of January 1st, 1970 00:00:00 UTC.
nanoseconds	DateTimeNanoSeconds	The number of nanoseconds elapsed within the current DateTimeSecond.

6.2.7 BatteryCellDataType

Namespace: UMAA::EO::BatteryStatus::BatteryCellDataType

Description: This structure is used to report the current status of a cell in the battery system.

Table 49: BatteryCellDataType Structure Definition

Attribute Name	Attribute Type	Attribute Description
current	PowerBusCurrent	The runtime current of the battery cell.
temperature	Temperature	The temperature of the battery cell.
voltage	PowerBusVoltage	The voltage of the battery cell.

6.2.8 IdentifierType

Namespace: UMAA::Common::IdentifierType

Description: This structure defines a two-level hierarchical identifier, where the parent is defined to be a group or collection of entities.

Table 50: IdentifierType Structure Definition

Attribute Name	Attribute Type	Attribute Description
id	NumericGUID	Provides the identifier of an entity.
parentID	NumericGUID	Provides the identifier of the parent, which is a group or collection of one or more entities. If the entity has no parent (it is the root of the tree), this value will be the Nil UUID.

6.2.9 OrientationAcceleration3D

Namespace: UMAA::Common::Measurement::OrientationAcceleration3D

Description: OrientationAcceleration3D specifies the acceleration for each axis of an Orientation.

Table 51: OrientationAcceleration3D Structure Definition

Attribute Name	Attribute Type	Attribute Description
pitchAccelY	PitchAcceleration	pitchAccelY specifies the acceleration of the platform's rotation about the lateral axis (e.g. the axis parallel to the wings) in a locally level, XYZ coordinate system centered on the platform.

Attribute Name	Attribute Type	Attribute Description
rollAccelX	RollAcceleration	rollAccelX specifies the acceleration of the platform's rotation about the longitudinal axis (e.g. the axis through the body of an aircraft from tail to nose) in a locally level, XYZ coordinate system centered on the platform.
yawAccelZ	YawAcceleration	yawAccelZ specifies the acceleration of the platform's rotation about the vertical axis (e.g. the axis from top to bottom through an aircraft) in a locally level, XYZ coordinate system centered on the platform.

6.2.10 Position3DBodyXYZ

Namespace: UMAA::Common::Measurement::Position3DBodyXYZ

Description: Specifies a three-dimensional location on a Cartesian coordinate system relative to the origin of the body.

Table 52: Position3DBodyXYZ Structure Definition

Attribute Name	Attribute Type	Attribute Description
xAxis	XPosition	The position on the body x-axis, which extends out the front of the reference body.
yAxis	YPosition	The position on the body y-axis, which extends out the right (starboard) of the reference body.
zAxis	ZPosition	The position on the body z-axis, which is perpendicular to the x and y axes and is directed downward from the center of the body.

6.2.11 PropulsionType

Namespace: UMAA::Common::Propulsion::PropulsionType

Description: Union Type. Propulsion value in either effort or RPM.

Table 53: PropulsionType Union(s)

Type Name	Type Description
PropulsiveEffortType	Defines the propulsive value as an effort.
PropulsiveRPMType	Defines the propulsive value as an RPM value.

6.2.12 PropulsiveEffortType

Namespace: UMAA::Common::Propulsion::PropulsiveEffortType

Description: Defines the propulsive value as an effort.

Table 54: PropulsiveEffortType Structure Definition

Attribute Name	Attribute Type	Attribute Description
propulsiveEffort	Effort	The desired propulsive effort, as a percent value. Negative values will cause motion in the port direction for fixed bow or stern thrusters.

6.2.13 PropulsiveRPMType

Namespace: UMAA::Common::Propulsion::PropulsiveRPMType

Description: Defines the propulsive value as an RPM value.

Table 55: PropulsiveRPMType Structure Definition

Attribute Name	Attribute Type	Attribute Description
RPM	FrequencyRPM	The desired RPM of the propulsor thruster.

6.2.14 SurfaceCapabilityLimitsType

Namespace: UMAA::EO::UVPlatformSpecs::SurfaceCapabilityLimitsType

Description: This structure describes the capability limits for any vehicle operating on the surface.

Table 56: SurfaceCapabilityLimitsType Structure Definition

Attribute Name	Attribute Type	Attribute Description
cruisingSpeed†	SpeedLocalWaterMass	The speed that results in the maximum range for the vehicle.
maxAcceleration†	AccelerationLocalWaterMass	The highest rate of increase of vehicle linear velocity as a function of time.
maxDeceleration†	AccelerationLocalWaterMass	The highest rate of decrease of vehicle linear velocity as a function of time.
maxForwardSpeed†	SpeedLocalWaterMass	The largest possible forward horizontal displacement of the vehicle as a function of time.
maxReverseSpeed†	SpeedLocalWaterMass	The largest possible reverse horizontal displacement of the vehicle as a function of time.
maxTowingSpeed†	SpeedLocalWaterMass	The fastest linear velocity that is allowed when dragging a payload.
maxTowingTurnAcceleration†	AngleAcceleration	The highest rate of linear velocity change as a function of time when towing.

Attribute Name	Attribute Type	Attribute Description
maxTowingTurnRate†	TurnRate	The fastest vehicle direction change rate when towing a payload.
maxTurnAcceleration†	AngleAcceleration	The highest rate of linear velocity change as a function of time.
maxTurnRate†	TurnRate	The fastest vehicle direction change rate.
minSpeedInMedium†	SpeedLocalWaterMass	The slowest linear velocity that is required to enable control surfaces to operate.
minTowingSpeed†	SpeedLocalWaterMass	The slowest linear velocity that is allowed when dragging a payload.

6.2.15 UnderwaterCapabilityLimitsType

Namespace: UMAA::EO::UVPlatformSpecs::UnderwaterCapabilityLimitsType

Description: This structure describes the capability limits for any vehicle operating under the surface.

Table 57: UnderwaterCapabilityLimitsType Structure Definition

Attribute Name	Attribute Type	Attribute Description
cruisingSpeed†	SpeedLocalWaterMass	The speed that results in the maximum range for the vehicle.
maxAcceleration†	AccelerationLocalWaterMass	The highest rate of increase of vehicle linear velocity as a function of time.
maxAttitudeAcceleration†	OrientationAcceleration3D	The highest rate of increase of vehicle rotational velocity as a function of time in three dimensions.
maxAttitudeDeceleration†	OrientationAcceleration3D	The highest rate of decrease of vehicle rotational velocity as a function of time in three dimensions.
maxDeceleration†	AccelerationLocalWaterMass	The highest rate of decrease of vehicle linear velocity as a function of time.
maxDepthAcceleration†	SpeedBSLAcceleration	The highest rate of vertical velocity change as a function of time.
maxDepthChangeRate†	SpeedBSL	The largest possible vertical displacement of the vehicle as a function of time.
maxForwardSpeed†	SpeedLocalWaterMass	The largest possible forward horizontal displacement of the vehicle as a function of time.
maxPitchRate†	PitchRate	The highest angular rate of change in the rotation of a vehicle about the transverse axis.
maxReverseSpeed†	SpeedLocalWaterMass	The largest possible reverse horizontal displacement of the vehicle as a function of time.
maxTowingSpeed†	SpeedLocalWaterMass	The fastest linear velocity that is allowed when dragging a payload.
maxTowingTurnAcceleration†	AngleAcceleration	The highest rate of linear velocity change as a function of time when towing.
maxTowingTurnRate†	TurnRate	The fastest vehicle direction change rate when towing a payload.
maxTurnAcceleration†	AngleAcceleration	The highest rate of linear velocity change as a function of time.

Attribute Name	**Attribute Type**	**Attribute Description**
maxTurnRate†	TurnRate	The fastest vehicle direction change rate.
maxVehicleDepth†	DistanceBSL	The largest vehicle operating distance below the water surface.
minSpeedInMedium†	SpeedLocalWaterMass	The slowest linear velocity that is required to enable control surfaces to operate.
minTowingSpeed†	SpeedLocalWaterMass	The slowest linear velocity that is allowed when dragging a payload.

6.3 Enumerations

Enumerations are used extensively throughout UMAA. This section lists the values associated with each enumeration defined in UCS-UMAA.

6.3.1 AnchorActionEnumType

Namespace: UMAA::Common::MaritimeEnumeration::AnchorActionEnumType

Description: Defines a mutually exclusive set of values for the anchor action.

Table 58: AnchorActionEnumType Enumeration

Enumeration Value	Description
LOWER	Lower the anchor.
RAISE	Raise the anchor.
STOP	Stop anchor from lowering or raising.

6.3.2 AnchorKindEnumType

Namespace: UMAA::Common::MaritimeEnumeration::AnchorKindEnumType

Description: Defines a mutually exclusive set of values for the anchor type.

Table 59: AnchorKindEnumType Enumeration

Enumeration Value	Description
COMMERCIAL_STOCKLESS	Anchor type is commercial stockless.
DANFORTH	Anchor type is danforth.
FOUR_FLUKE	Anchor type is four-fluke.
GENERAL	Anchor type is general.
LIGHTWEIGHT	Anchor type is lightweight.
MARK_2_LWT	Anchor type is mark 2 lightweight.
MARK_2_STOCKLESS	Anchor type is mark 2 stockless.
MUSHROOM	Anchor type is mushroom.
NAVY_TYPE_STOCK	Anchor type is navy type stock.
NONMAGNETIC	Anchor type is nonmagnetic.
STANDARD_NAVY_STOCKLESS	Anchor type is standard navy stockless.
TWO_FLUKE_BALANCED_FLUKE	Anchor type is two-fluke balanced fluke.
WEDGE_BLOCK_LWT	Anchor type is wedge block lightweight.

6.3.3 AnchorLocationEnumType

Namespace: UMAA::Common::MaritimeEnumeration::AnchorLocationEnumType

Description: Defines the location of the anchor.

Table 60: AnchorLocationEnumType Enumeration

Enumeration Value	Description
BOWER	A bower anchor is carried on the bow.
KEEL	A keel anchor is housed within the hull neer the keel.
STERN	A stern anchor is carried on the stern.

6.3.4 AnchorRodeEnumType

Namespace: UMAA::Common::MaritimeEnumeration::AnchorRodeEnumType

Description: A mutually exclusive set of values that defines the rode type.

Table 61: AnchorRodeEnumType Enumeration

Enumeration Value	Description
CHAIN	Chain
ROPE	Rope

6.3.5 AnchorStateEnumType

Namespace: UMAA::Common::MaritimeEnumeration::AnchorStateEnumType

Description: Defines a mutually exclusive set of values of the anchor state.

Table 62: AnchorStateEnumType Enumeration

Enumeration Value	Description
DEPLOYED	Anchor is deployed.
LOWERING	Anchor is lowering.
RAISING	Anchor is raising.
STOPPED	Anchor is neither DEPLOYED nor STOWED, but is not in the process of LOWERING or RAISING.
STOWED	Anchor is stowed.

6.3.6 CommandStatusReasonEnumType

Namespace: UMAA::Common::MaritimeEnumeration::CommandStatusReasonEnumType

Description: Defines a mutually exclusive set of reasons why a command status state transition has occurred.

Table 63: CommandStatusReasonEnumType Enumeration

Enumeration Value	Description
CANCELED	Indicates a transition to the CANCELED state when the command is canceled successfully.
INTERRUPTED	Indicates a transition to the FAILED state when the command has been interrupted by a higher priority process.
OBJECTIVE_FAILED	Indicates a transition to the FAILED state when the commanded resource is unable to achieve the command's objective due to external factors.
RESOURCE_FAILED	Indicates a transition to the FAILED state when the commanded resource is unable to achieve the command's objective due to resource or platform failure.
RESOURCE_REJECTED	Indicates a transition to the FAILED state when the commanded resource rejects the command for some reason.
SERVICE_FAILED	Indicates a transition to the FAILED state when the commanded resource is unable to achieve the command's objective due to processing failure.
SUCCEEDED	Indicates the conditions to proceed to this state have been met and a normal state transition has occurred.
TIMEOUT	Indicates a transition to the FAILED state when the command is not acknowledged within some defined time bound.
UPDATED	Indicates a transition back to the ISSUED state from a non-terminal state when the command has been updated.
VALIDATION_FAILED	Indicates a transition to the FAILED state when the command contains missing, out-of-bounds, or otherwise invalid parameters.

6.3.7 IgnitionControlEnumType

Namespace: UMAA::Common::MaritimeEnumeration::IgnitionControlEnumType

Description: Defines a mutually exclusive set of values that defines the state of engine control.

Table 64: IgnitionControlEnumType Enumeration

Enumeration Value	Description
OFF	Stop the engine.
RUN	Run the engine.

6.3.8 IgnitionStateEnumType

Namespace: UMAA::Common::MaritimeEnumeration::IgnitionStateEnumType

Description: Defines a mutually exclusive set of values that defines the state of engine ignition.

Table 65: IgnitionStateEnumType Enumeration

Enumeration Value	Description
OFF	The engine is off.

Enumeration Value	Description
RUN	The engine is running.
START	The engine is starting.

6.3.9 MastActionEnumType

Namespace: UMAA::Common::MaritimeEnumeration::MastActionEnumType

Description: A mutually exclusive set of values that defines the action of the mast.

Table 66: MastActionEnumType Enumeration

Enumeration Value	Description
LOWER	set to lower the mast down
RAISE	set to raise the mast up
STOP	set to stop the mast

6.3.10 MastStateEnumType

Namespace: UMAA::Common::MaritimeEnumeration::MastStateEnumType

Description: A mutually exclusive set of values that defines the state of the mast.

Table 67: MastStateEnumType Enumeration

Enumeration Value	Description
DOWN	set when the mast is down
MOVING_DOWN	set when the mast is moving down
MOVING_UP	set when the mast is moving up
STOPPED	set when the mast is not in motion, but between fully up and fully down
UP	set when the mast is up

6.3.11 CommandStatusEnumType

Namespace: UMAA::Common::MaritimeEnumeration::CommandStatusEnumType

Description: Defines a mutually exclusive set of values that defines the states of a command as it progresses towards completion.

Table 68: CommandStatusEnumType Enumeration

Enumeration Value	Description
CANCELED	The command was canceled by the requestor before the command completed successfully.

Enumeration Value	Description
COMMANDED	The command has been placed in the resource's command queue but has not yet been accepted.
COMPLETED	The command has been completed successfully.
EXECUTING	The command is being performed by the resource and has not yet been completed.
FAILED	The command has been attempted, but was not successful.
ISSUED	The command has been issued to the resource (typically a sensor or streaming device), but processing has not yet commenced.

6.3.12 OnOffStatusEnumType

Namespace: UMAA::Common::Enumeration::OnOffStatusEnumType

Description: A mutually exclusive set of values that defines the on/off status of a device or subsystem.

Table 69: OnOffStatusEnumType Enumeration

Enumeration Value	Description
OFF	The device or subsystem is off.
ON	The device or subsystem is on.

6.3.13 PowerPlantStateEnumType

Namespace: UMAA::Common::MaritimeEnumeration::PowerPlantStateEnumType

Description: A mutually exclusive set of values that defines the power state of each power plant on the vehicle.

Table 70: PowerPlantStateEnumType Enumeration

Enumeration Value	Description
FAULT	Faulted
OFF	Off
ON	On

6.3.14 ReferenceFrameOriginEnumType

Namespace: UMAA::Common::MaritimeEnumeration::ReferenceFrameOriginEnumType

Description: A mutually exclusive set of values that defines the origin from which all distance measurements are taken.

Table 71: ReferenceFrameOriginEnumType Enumeration

Enumeration Value	Description
BOW_WATERLINE_INTERSECTION	Bow Waterline Intersection.
CENTER_OF_BUOYANCY	Center of buoyancy.
CENTER_OF_GRAVITY	Center of gravity.
INS_LOCATION	INS Location
KEEL_TRANSOM_INTERSECTION	Keel transom intersection

6.4 Type Definitions

This section describes the type definitions for UMAA. The table below lists how UMAA defined types are mapped to the DDS primitive types.

Table 72: Type Definitions

Type Name	Primitive Type	Range of Values	Description
AccelerationLocalWaterMass	double	maxInclusive=299792458 minInclusive=-299792458 units=MeterPerSecondSquared	The change in velocity over time relative to local water mass.
AmpHours	double	maxInclusive=500 minInclusive=0 units=AmpereHours	Represents the nominal capacity of a battery at 1C rate.
AngleAcceleration	double	maxInclusive=10000 minInclusive=-10000 units=RadiansPerSecondSquared referenceFrame=PlatformXYZ	Represents the rate of change of angular velocity.
BatteryCurrent	double	maxInclusive=1000 minInclusive=0 units=Ampere	Represents the current of a battery.
BatteryCurrentDuration	double	maxInclusive=20 minInclusive=0 units=Seconds	Represents the duration for which a battery can supply a given amount of current.
BatteryCycles	double	maxInclusive=10000 minInclusive=0	Represents an integer number of battery cycles.
BooleanEnumType	boolean		A mutually exclusive set of values that defines the truth values of logical algebra.
Charge	double	maxInclusive=3600000 minInclusive=0 units=Coulomb referenceFrame=Counting	Represents physical property of matter that causes it to experience a force when placed in an electromagnetic field. Measured in Coulomb.
DateTimeNanoseconds	long	units=Nanoseconds minInclusive=0 maxInclusive=999999999	The number of nanoseconds elapsed within the current second.
DateTimeSeconds	longlong	units=Seconds minInclusive=-9223372036854775807 maxInclusive=9223372036854775807	The seconds offset from the standard POSIX (IEEE Std 1003.1-2017) epoch reference point of January 1st, 1970 00:00:00 UTC.
Distance	double	maxInclusive=401056000 minInclusive=0 units=Meter referenceFrame=Counting	This type stores a distance in meters.
DistanceBSL	double	maxInclusive=10000 minInclusive=0 units=Meter referenceFrame=BSL	The distance below sea level in meters.

Type Name	Primitive Type	Range of Values	Description
DurationHours	double	maxInclusive=10505 minInclusive=0 units=Hour referenceFrame=Counting	Represents a time duration in hours.
DurationSeconds	double	maxInclusive=37817280 minInclusive=0 units=Seconds referenceFrame=Counting	Represents a time duration in seconds.
Effort	double	maxInclusive=100 minInclusive=-100 units=Percent referenceFrame=PlatformXYZ	Represents the level of effort measured in percent.
ElectricalPower	double	maxInclusive=100000000 minInclusive=0 units=Watt referenceFrame=None	Represents the rate at which electric energy is transferred by an electric circuit measured in watts.
EnergyPercent	double	maxInclusive=1000 minInclusive=0 units=Percent referenceFrame=Counting	Defines a percentage where 100% = 100.0. Values greater than 100% are allowed.
EngineSpeed	double	referenceFrame=Counting units=RevolutionsPerMinute minInclusive=-100000 maxInclusive=100000	This type stores number of occurrences in revolutions per minute (RPM). Negative number is used for reverse RPM.
Force	double	maxInclusive=100000000 minInclusive=0 units=Newton referenceFrame=Counting	Represents the degree of force measured in Newtons.
FrequencyRPM	long	maxInclusive=100000 minInclusive=-100000 units=RevolutionsPerMinute referenceFrame=Counting	This type stores number of occurrences in revolutions per minute (RPM). Negative number is used for reverse RPM.
LargeCollectionSize	long	maxInclusive=2147483647 minInclusive=0	Specifies the size of a Large Collection.
Mass	double	maxInclusive=100000000 minInclusive=0 units=Kilogram referenceFrame=Counting	This type stores mass in kilograms.
MassMetricTon	double	maxInclusive=100000 minInclusive=0 units=MetricTon referenceFrame=Counting	Represents the property of physical body measured in non-SI derived unit, metric ton.
NumericGUID	octet[16]	minInclusive=0 maxInclusive=(2^128)-1	Represents a 128-bit number according to RFC 4122 variant 2.
PitchAcceleration	double	maxInclusive=10000 minInclusive=-10000 units=RadianPerSecondSquared referenceFrame=Counting	Specifies the platform's angular acceleration about the lateral axis in a locally level, North-East-Down coordinate system centered on the platform.

Type Name	Primitive Type	Range of Values	Description
PitchRate	double	maxInclusive=32.767 minInclusive=-32.767 units=RadianPerSecond referenceFrame=Counting	Specifies the rate of change of the platform's pitch angle.
PowerBusCurrent	double	maxInclusive=100000 minInclusive=-100000 units=Ampere referenceFrame=None	Represents the time rate of flow of electric charge measured in amperes.
PowerBusVoltage	double	maxInclusive=100000 minInclusive=-100000 units=Volt referenceFrame=None	Represents the potential difference in charge between two points in an electrical field measured in volts.
PressureKiloPascals	double	maxInclusive=51200 minInclusive=0 units=KiloPascal referenceFrame=STP	Represents barometric pressure and is stored in KiloPascals.
PressurePercent	double	maxInclusive=200 minInclusive=0 units=Percent referenceFrame=Counting	Represents the weight or force per unit area that is produced when something presses or pushes against something else.
Ratio	double		Represents the real number ratio.
RollAcceleration	double	maxInclusive=10000 minInclusive=-10000 units=RadianPerSecondSquared referenceFrame=Counting	Specifies the angular acceleration of the platform about the longitudinal axis (e.g. the axis through the body of the vehicle from tail to nose) in a locally level, North-East-Down coordinate system centered on the platform.
SpeedBSL	double	maxInclusive=299792458 minInclusive=-299792458 units=MeterPerSecond referenceFrame=BSL	This type stores speed in meters/s in a below sea level reference frame.
SpeedBSLAcceleration	double	maxInclusive=299792458 minInclusive=-299792458 units=MeterPerSecondSquared	Describes change in velocity over time below sea level.
SpeedLocalWaterMass	double	maxInclusive=299792458 minInclusive=0 units=MeterPerSecond referenceFrame=LocalWaterMass	This type stores speed in meters/s.
StringLongDescription	string	length=4095	Represents a long format description.
StringShortDescription	string	length=1023	Represents a short format description.
Temperature	double	maxInclusive=1000 minInclusive=-273 units=Celsius referenceFrame=Counting	Represents the degree or intensity of warmness or coldess presence in a substance. Measured in Celsius.

Type Name	Primitive Type	Range of Values	Description
TurnRate	double	maxInclusive=32.767 minInclusive=-32.767 units=RadianPerSecond referenceFrame=Counting	Specifies the rate of change of the heading angle of a platform.
VolumeCubicMeter	double	maxInclusive=1000 minInclusive=0 units=VolumeCubicMeter referenceFrame=Counting	Represents the quantity of three-dimensional space enclosed by some closed boundary
VolumePercent	double	maxInclusive=1000 minInclusive=0 units=Percent referenceFrame=Counting	Defines a percentage where 100% = 100.0. Values greater than 100% are allowed.
WattHours	double	maxInclusive=900000 minInclusive=0 units=WattHours	Represents the nominal energy of a battery at 1C rate.
XPosition	double	units=Meter	Represents the x axis position.
YawAcceleration	double	maxInclusive=10000 minInclusive=-10000 units=RadianPerSecondSquared referenceFrame=Counting	Specifies the platform's angular acceleration about the vertical axis in the body coordinate system.
YPosition	double	units=Meter	Represents the y axis position.
ZPosition	double	maxInclusive=100000 minInclusive=-100000 units=Meter	Represents the z axis position.

A Appendices

A.1 Glossary

Note: This glossary aims to define terms that are uncommon, or have a special meaning in the context of UMAA and/or the DoD. This glossary covers the complete UMAA specification. Not every word defined here appears in every ICD.

Almanac Data (GPS)	A navigation message that contains information about the time and status of the entire satellite constellation.
Coulomb	The SI unit of electric charge, equal to the quantity of electricity conveyed in one second by a current of one ampere.
Ephemeris Data (GPS)	A navigation message used to calculate the position of each satellite in orbit.
Glowplug or Glow Plug	A heating device used to aid in starting diesel engines.
Interoperability	1) The ability to act together coherently, effectively, and efficiently to achieve tactical, operational, and strategic objectives. 2) The condition achieved among communications-electronics systems or items of communications-electronics equipment when information or services can be exchanged directly and satisfactorily between them and/or their users.
Mean Sea Level	The average height of the surface of the sea for all stages of the tide; used as a reference for elevations.
Middleware	A type of computer software that provides services to software applications beyond those available from the operating system. Middleware makes it easier for software developers to implement communication and input/output, so they can focus on the specific purpose of their application.
SoaML	The Service oriented architecture Modeling Language (SoaML) specification that provides a metamodel and a UML profile for the specification and design of services within a service-oriented architecture. The specification is managed by the Object Management Group (OMG).

A.2 Acronyms

Note: This acronym list is included in every ICD and covers the complete UMAA specification. Not every acronym appears in every ICD.

ADD	Architecture Design Description
AGL	Above Sea Level
ASF	Above Sea Floor
BSL	Below Sea Level
BWL	Beam at Waterline
C2	Command and Control
CMD	Command
CO	Comms Operations
CPA	Closest Point of Approach
CTD	Conductivity, Temperature and Depth
DDS	Data Distribution Service
DTED	Digital Terrain Elevation Data
EGM	Earth Gravity Model
EO	Engineering Operations
FB	Feedback
GUID	Globally Unique Identifier
HM&E	Hull, Mechanical, & Electrical

ICD	Interface Control Document
ID	Identifier
IDL	Interface Definition Language Specification
IMO	International Maritime Organization
INU	Inertial Navigation Unit
LDM	Logical Data Model
LOA	Length Over All
LRC	Long Range Cruise
LWL	Length at Waterline
MDE	Maritime Domain Extensions
MEC	Maximum Endurance Cruise
MM	Mission Management
MMSI	Maritime Mobile Service Identity
MO	Maneuver Operations
MRC	Maximum Range Cruise
MSL	Mean Sea Level
OMG	Object Management Group
PIM	Platform Independent Model
PMC	Primary Mission Control
PNT	Precision Navigation and Timing
PO	Processing Operations
PSM	Platform Specific Model
RMS	Root-Mean-Square
ROC	Risk of Collision
RPM	Revolutions per minute
RTPS	Real Time Publish Subscribe
RTSP	Real Time Streaming Protocol
SA	Situational Awareness
SEM	Sensor and Effector Management
SO	Support Operations
SoaML	Service-oriented architecture Modeling Language
STP	Standard Temperature and Pressure
UCS	Unmanned Systems Control Segment
UMAA	Unmanned Maritime Autonomy Architecture
UML	Unified Modeling Language
UMS	Unmanned Maritime System
UMV	Unmanned Maritime Vehicle
UxS	Unmanned System
WGS84	Global Coordinate System
WMM	World Magnetic Model
WMO	World Meteorological Organization