Unmanned Maritime Autonomy Architecture (UMAA) Support Operations (SO) Interface Control Document (ICD) (UMAA-SPEC-SOICD)

MDE Version 4.2.1 Commit 7c2b513

UMAA Spec Commit f2fa426

Version 3.0.1
25 February 2021

UMAA Support Operations ICD

(UMAA-SPEC-SO-ICD)

Signature Page

Submitted by:	Rothgeb.Mark.E.ORC3011008580.ID Digitally signed by Rothgeb.Mark.E.ORC3011008580.ID Date: 2021.05.05 11:56:24 -04'00'	**Date Signed:**	
	Mark Rothgeb Unmanned Maritime Autonomy Architecture Standards Board Chair		
PMS 406 Approvals:	ANDERSON.JEREMIAH.P.1254266481 Digitally signed by ANDERSON.JEREMIAH.P.1254266481 Date: 2021.05.10 14:54:11 -04'00'	**Date Signed:**	
	CDR Jeremiah Anderson PMS 406 Advanced Autonomous Capabilities, Principal Assistant Program Manager		
	SMALL.PETER.D.1048778646 Digitally signed by SMALL.PETER.D.1048778646 Date: 2021.06.03 17:19:13 -04'00'	**Date Signed:**	
	CAPT Pete Small Program Manager PMS 406 Unmanned Maritime Systems, PEO Unmanned and Small Combatants		

Contents

List of Figures

List of Tables

1 Scope

1.1 Identification

This document defines a set of services and interfaces as part of the Unmanned Maritime Autonomy Architecture (UMAA). The services and corresponding interfaces covered in this ICD encompass the functionality to provide support operations for an Unmanned Maritime Vehicle (UMV) (surface or undersea). As such, it provides services used across functional boundaries of UMAA ICDâ€™s such as logging, supporting startup and shutdown, providing emissions control (EMCON) services, and resource control. This document is generated automatically from data models that define its services and interfaces as part of the Unmanned Systems (UxS) Control Segment (UCS) Architecture as extended by UMAA to provide autonomy services for UMVs.

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.

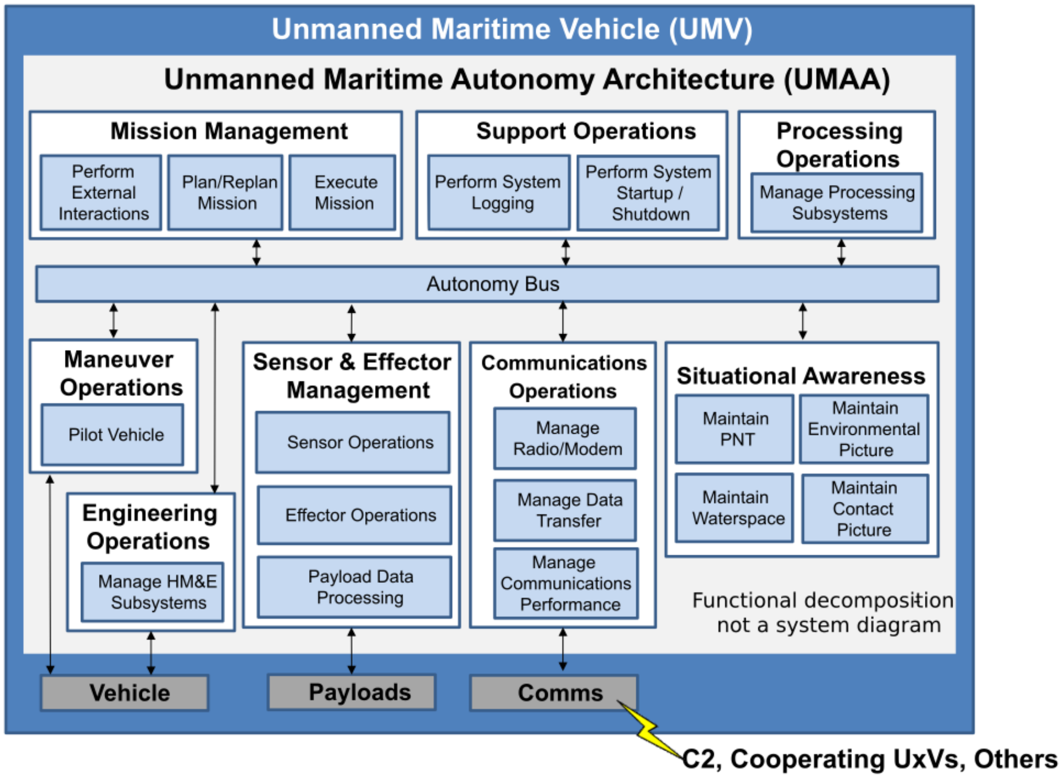

Figure 1: UMAA Functional Organization

1.2 Overview

The fundamental purpose of UMAA is to promote the development of common, modular, and scalable software for UMV's that is independent of a particular autonomy implementation. Unmanned Maritime Systems (UMSs) consist of Command and Control (C2), one or more UMVs, 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 UMV. This includes the autonomy for all classes of UMVs and must support varying levels of communication in mission (i.e., constant, intermittent, or none) with its C2 System. 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 platform interface.

Support Operations provides capabilities for services that are shared across all of the other functional areas within UMAA. This support includes the ancillary infrastructure services and interfaces required to operate a UMV. Standard interfaces are defined for startup and shutdown, logging of time-stamped event and attribute data, operational mode control (e.g. operational, simulation, maintenance, training), and resource control (i.e. managing which client is in control of a component).

Unlike the primary concerns of a vehicle system, such as propulsion control and sensor data processing, the support operations are not typically seen in an external view of the system. Standardization of these services provides a consistent way to manage internal modes and control hierarchies across platforms and programs.

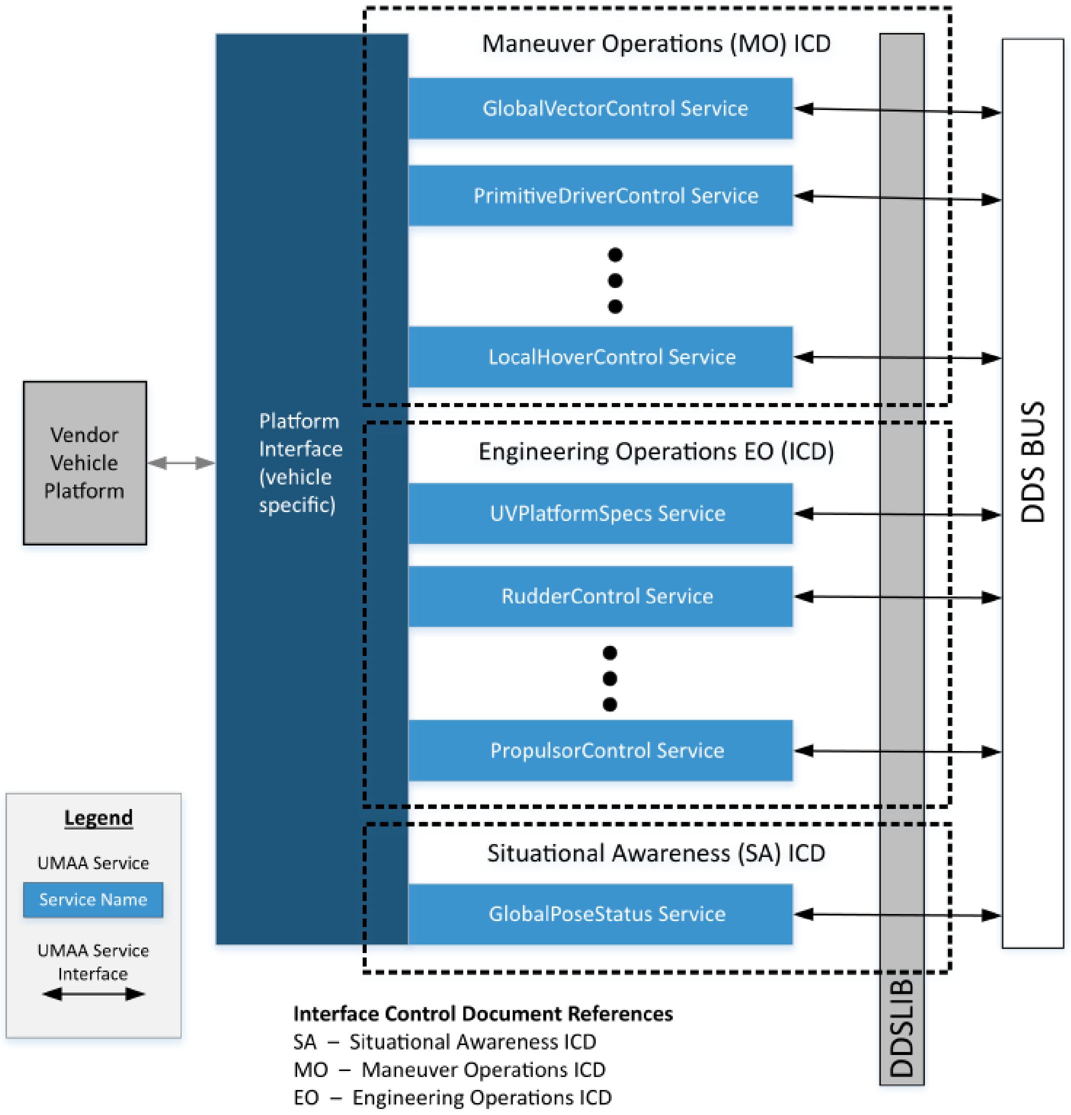
Maneuver Operations (MO) ICD
GlobalVectorControl Service
PrimitiveDriverControl Service
LocalHoverControl Service
Vendor Vehicle Platform
Platform Interface (vehicle specific)
Engineering Operations EO (ICD)
UVPlatformSpecs Service
RudderControl Service
PropulsorControl Service
Situational Awareness (SA) ICD
GlobalPoseStatus Service
DDS BUS
DDSLIB
Legend
UMAA Service
Service Name
UMAA Service Interface
Interface Control Document References
SA – Situational Awareness ICD
MO – Maneuver Operations ICD
EO – Engineering Operations 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 – Flow Control: A description of different flow control patterns used throughout UMAA.

Section 5 – Support Operations (SO) 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 3: 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

Table 4: 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, 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 to assure they are consistent with the data model and to ensure delivered software matches its interface specification.

The data model is maintained as a maritime extension to the UCS Architecture and will be maintained under configuration control by the UMAA Board. Section 5 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 compliance with a UMAA ICD is to provide service and interface specification compliance. Components may group services and their 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 represent 1 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. Compliance is satisfied at the individual service level. Given this context, section 5 correspondingly defines services with their interfaces and not components.

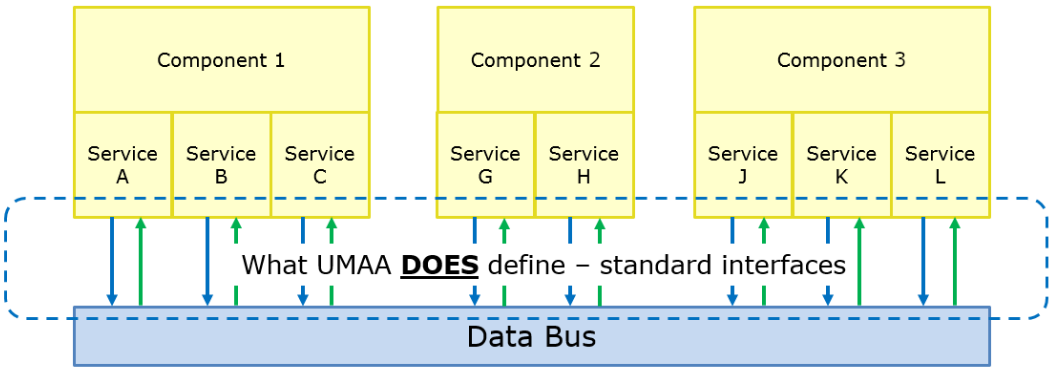

Figure 3: Services and Interfaces Exposed on the UMAA Data Bus

Services may use other services within this ICD or in other UMAA defined ICDs in order to provide their capability. Additionally, components for acquisition and development may span ICDs. An example of this would be a vehicle control system on a UMV. The control of the vehicle would be found in the Maneuver Operations ICD. However, an Inertial Navigation Unit (INU) that gives dynamic vehicle status is found in the Situational Awareness ICD. These are often organic to a vehicle and in that case are provided together with the vehicle as a component.

3.3 Data Distribution Service (DDS^{TM})

The data bus supporting autonomy messaging as depicted 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 components of a system 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 a 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. (UCSArchitecture,ArchitectureTechnicalGovernance)

In order to promote commonality across service definitions, a common way of naming services and their set 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 will not be broken up as indicated below 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 in implementation. The conventions used to define UCS-MDE services are as follows:

Service Name
- <SBN>Config
- <SBN>Control
- <SBN>Specs
- <SBN>Status

where the SBN should be descriptive of the task or information provided by the service.

Table 5: Service Requests and Associated Responses

	Service Requests (Inputs)	**Service Responses (Outputs)**
Config	query<SBN><FN>Config	report<SBN><FN>Config
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	query<SBN><FN>	report<SBN><FN>

Service Requests (operation:message)
- 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
- query<SBN><FN>ExecutionStatus:<SBN><FN>ExecutionStatusRequestType[1]
- query<SBN><FN>Specs:<SBN><FN>SpecsRequestType[1]
- query<SBN><FN>:<SBN><FN>RequestType [1] [2]

[1] These message types are required for compatibility with the UCS model but are not used by the UMAA specification.
[2] At this time there are no Requests in the specification but when they have been added, this will be the message format.

Service Responses (operation:message)
    report<SBN><FN>Config:<SBN><FN>ConfigReportType
    report<SBN><FN>CommandStatus:<SBN><FN>CommandStatusType
    report<SBN><FN>CommandAck:<SBN><FN>CommandAckReportType
    report<SBN><FN>CancelCommandStatus:<SBN><FN>CancelCommandStatusType
    report<SBN><FN>ExecutionStatus:<SBN><FN>ExecutionStatusReportType
    report<SBN><FN>Specs:<SBN><FN>SpecsReportType
    report<SBN><FN>:<SBN><FN>ReportType

where,

- Config (Configuration) Report – 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, maximum allowable radio transmit power.
- Command Status – the current state of a particular command (either control or configuration)
- Command – the ability to influence or direct the behavior of a resource during operation of a particular task. Attributes are variable. Examples include a vehicleâ€™s speed, engine RPM, antenna raising/lowering, controlling a light or gong.
- Command Ack (Acknowledgement) Report – the command currently being executed.
- Cancel – the ability to cancel a particular command that has been issued.
- Execution Status Report – the status related to executing a particular command. Examples associated with a waypoint command include cross track error, time to achieve, distance remaining.
- Specs (Specifications) Report – 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 UMV, cycle time of a radar.
- Report – the current information provided by a resource. Examples include a vehicle speed, rudder angle, current waypoint, 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 service under Maneuver Operations:
    UMAA::MO::PrimitiveDriver
Access the Feature Service under Situational Awareness:
    UMAA::SA::Feature

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 Report Message Fields:
    UMAA::UMAARpt
Access the common UMAA Position2D (i.e., latitude and longitude) structure:
    UMAA::Measurement::Position2D

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 Sets

Some reports under the UMAA standard utilize Large Sets, which are unordered sets of related data. The purpose of a Large Set is to provide the ability to update one or more elements of the set without having to republish the entire set on the DDS bus and consuming more resources as a set is appended or updated. In a given DDS topic, each element of the set is tracked to its identifier through the use of the <service>SetID identifier (a key). Additionally, users will be able to trace an element in a set by its source attribute (a NumericGUID) to the Service Provider that is generating the report with this set.

When elements of the set are updated, the timestamp of the metadata must be updated as well to signal a change in the set. The element timestamp for the update must be later than the current metadata timestamp. Once the element is updated, the timestamp of the metadata must be updated to a time equal to or later than the timestamp of the individual element update. The set can be updated as a batch (multiple elements in a single "update cycle," as determined by the provider) provided the metadata timestamp is updated to a time that is no earlier than the the most recent timestamp of all element updates in the batch. This allows for a coarse synchronization: data elements with timestamps later than the current metadata timestamp 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 metadata timestamp is advanced beyond the element's timestamp to signal the complete update cycle has finished and consider the set as a whole.

4 Flow Control

4.1 Command / Response

This section defines the flow of control for command/response over the DDS bus. A command/response is used to control 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 unique `sessionID` for the command when commanded. The `sessionID` is used to track the command execution as a key into other command-related messages. 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`. 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 4. 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.

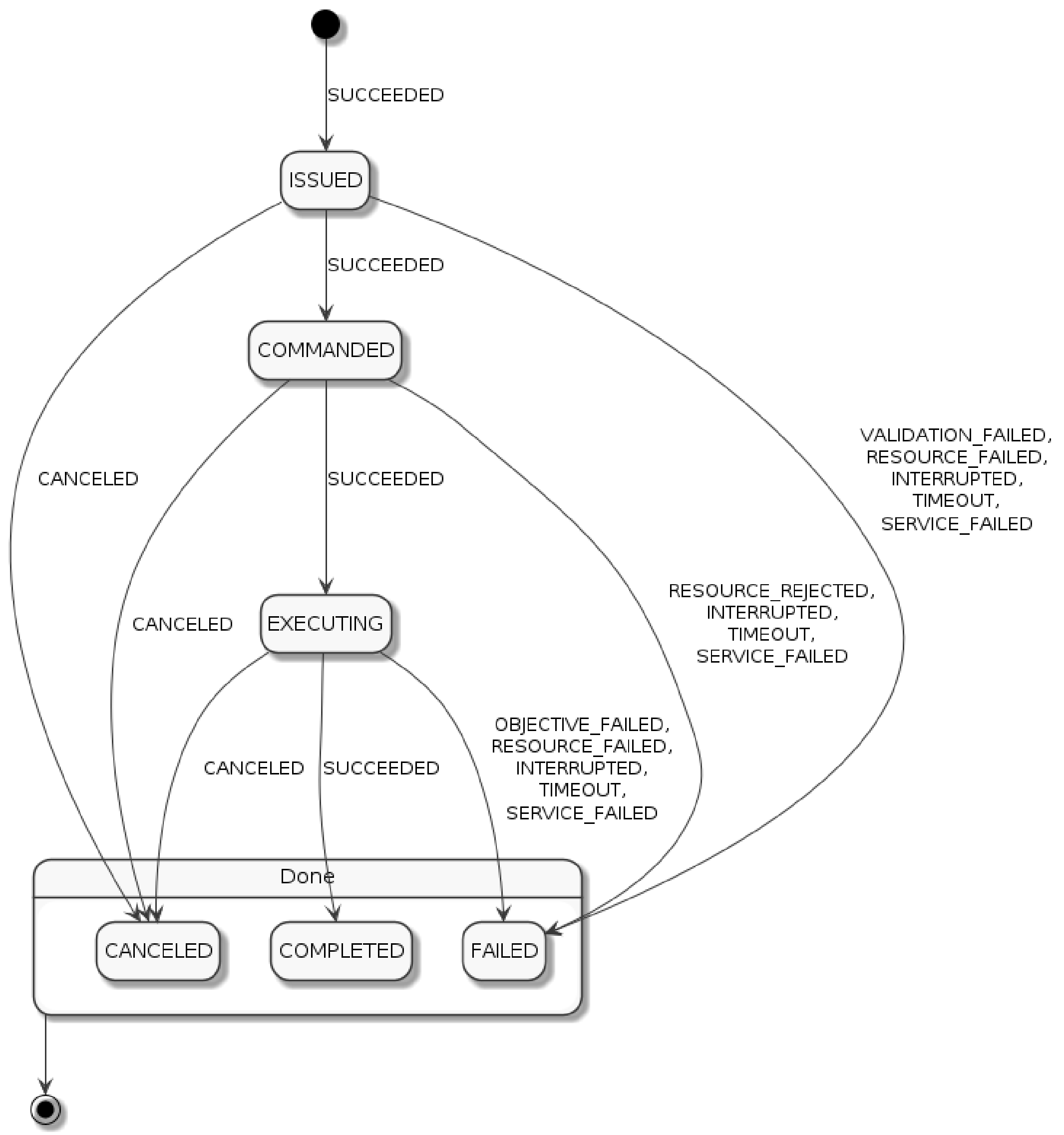
SUCCEEDED
ISSUED
SUCCEEDED
COMMANDED
CANCELED
SUCCEEDED
VALIDATION_FAILED, RESOURCE_FAILED, INTERRUPTED, TIMEOUT, SERVICE_FAILED
RESOURCE_REJECTED, INTERRUPTED, TIMEOUT, SERVICE_FAILED
CANCELED
EXECUTING
CANCELED
SUCCEEDED
OBJECTIVE_FAILED, RESOURCE_FAILED, INTERRUPTED, TIMEOUT, SERVICE_FAILED
Done
CANCELED
COMPLETED
FAILED

Figure 4: The state transitions of the `commandStatus` as commands are processed. Labels on the arrows represent valid `commandStatusReason` values for each transition.

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 similiar 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.

4.1.1 High-Level Flow

The high-level flow of a command sequence is shown in Figure 5 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 my not be identical to the overall system execution duration. For example, providers may only be available to execute certain commands during specific phases of a misison or when certain hardware is in specific configurations. This Command Startup Sequence is not required to happend during a system startup phase. The only requirement is 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 anytime the `FunctionCommandType` will no longer be supported. There is no requirement the Command Shutdown Sequence only be performed during a system shutdown phase.

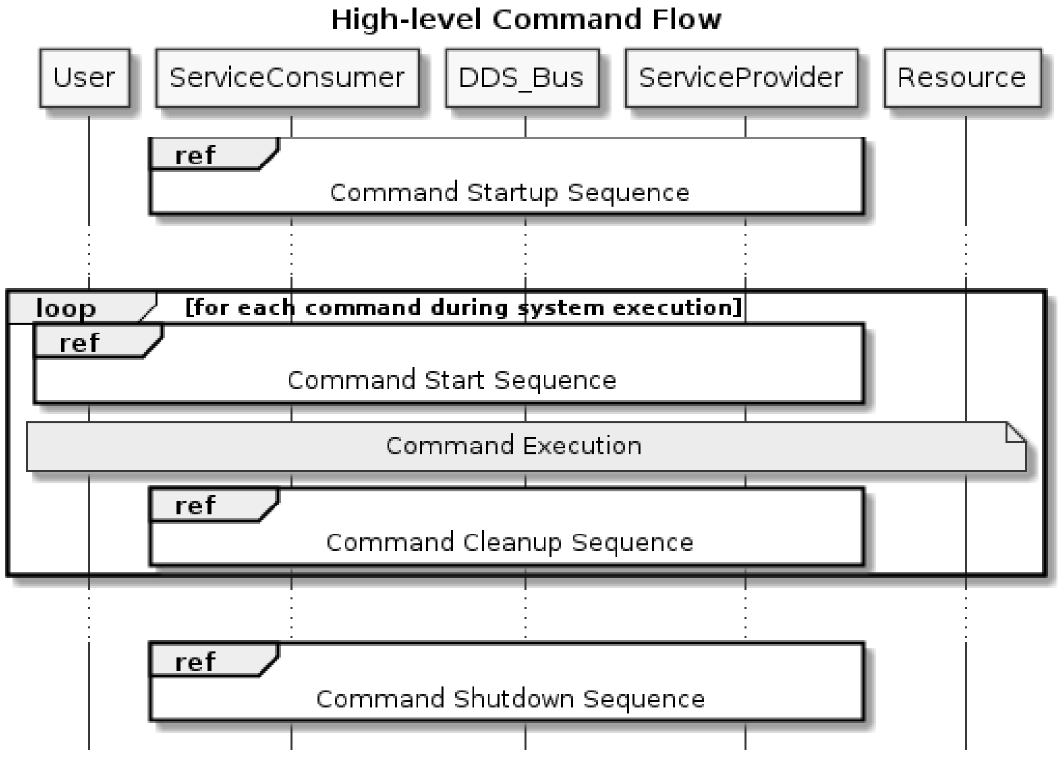

Figure 5: The sequence diagram for the high-level description of a command exeuction.

4.1.2 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 6.

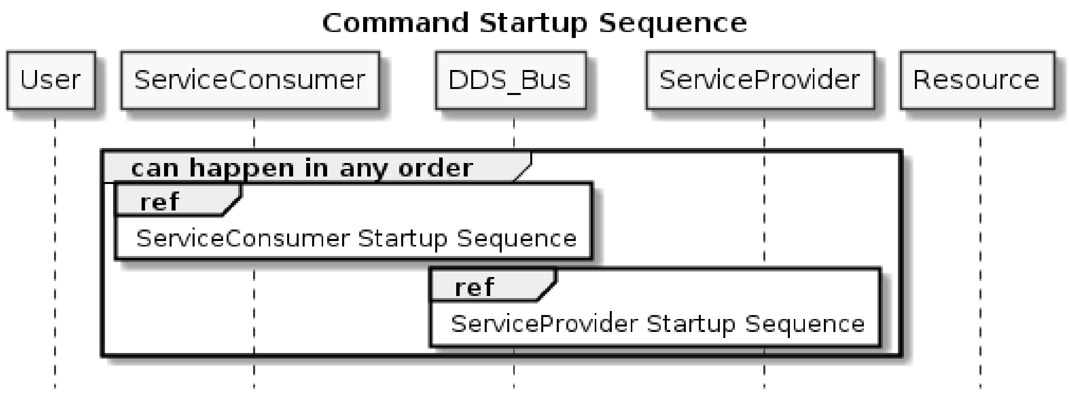

Figure 6: The sequence diagram for command startup.

4.1.2.1 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 `FunctionExecutionStatus` 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 choses 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 7.

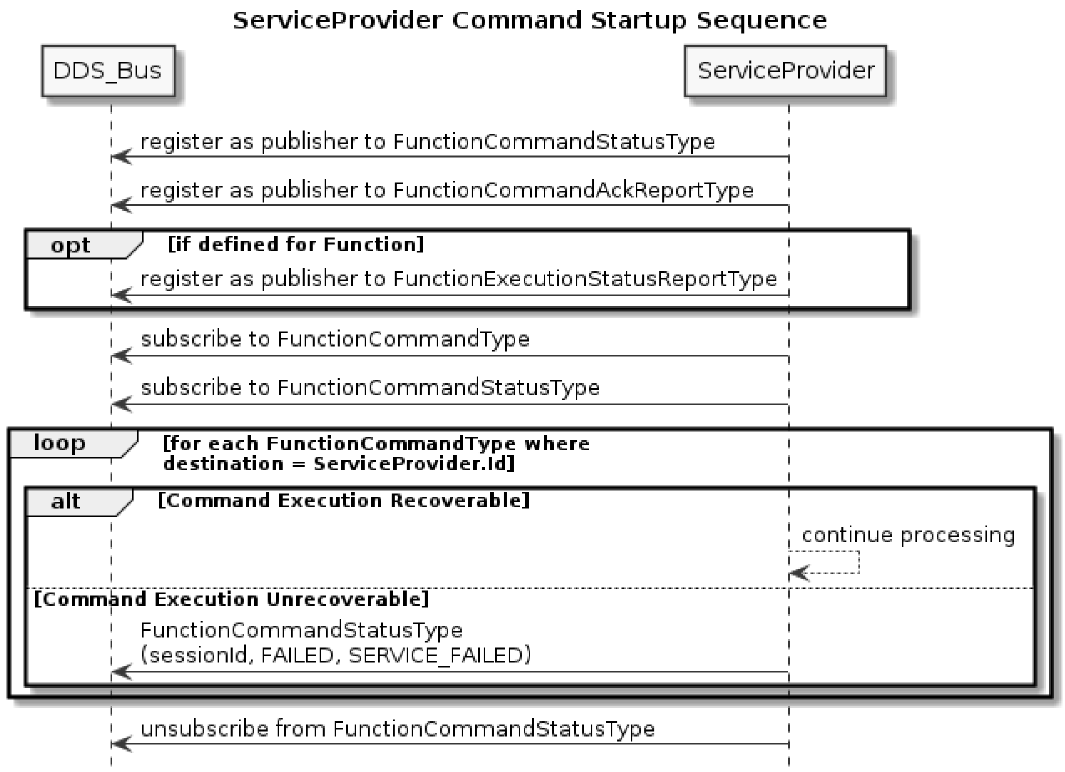
ServiceProvider Command Startup Sequence
DDS_Bus
ServiceProvider
register as publisher to FunctionCommandStatusType
register as publisher to FunctionCommandAckReportType
opt [if defined for Function]
register as publisher to FunctionExecutionStatusReportType
subscribe to FunctionCommandType
subscribe to FunctionCommandStatusType
loop [for each FunctionCommandType where destination = ServiceProvider.Id]
alt [Command Execution Recoverable]
continue processing
[Command Execution Unrecoverable]
FunctionCommandStatusType (sessionId, FAILED, SERVICE_FAILED)
unsubscribe from FunctionCommandStatusType

Figure 7: The sequence diagram for command startup for Service Providers.

4.1.2.2 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 choses 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 8.

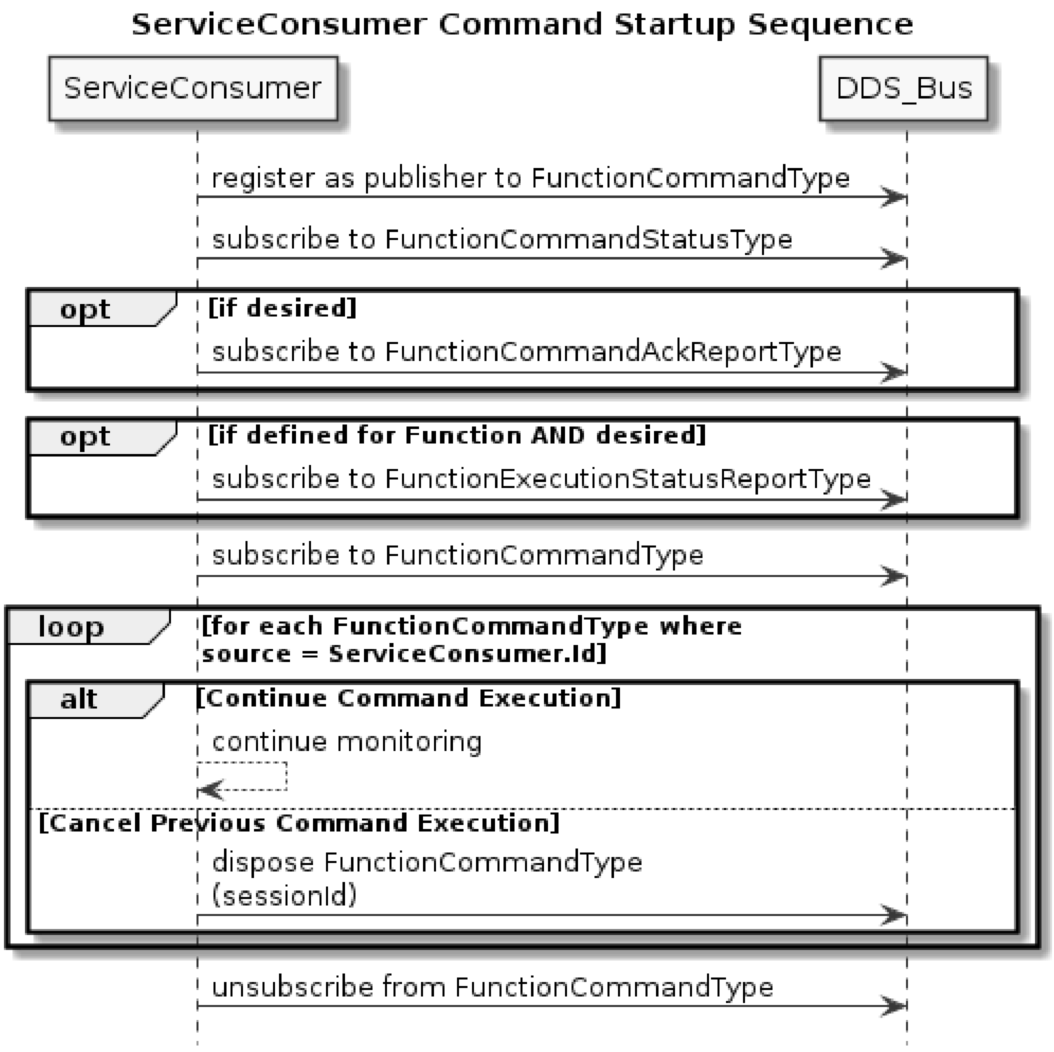
ServiceConsumer Command Startup Sequence
ServiceConsumer
DDS_Bus
register as publisher to FunctionCommandType
subscribe to FunctionCommandStatusType
opt
[if desired]
subscribe to FunctionCommandAckReportType
opt
[if defined for Function AND desired]
subscribe to FunctionExecutionStatusReportType
subscribe to FunctionCommandType
loop
[for each FunctionCommandType where source = ServiceConsumer.Id]
alt
[Continue Command Execution]
continue monitoring
[Cancel Previous Command Execution]
dispose FunctionCommandType (sessionId)
unsubscribe from FunctionCommandType

Figure 8: The sequence diagram for command startup for Service Consumers.

4.1.3 Command Execution Sequences

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

4.1.4 Command Start Sequence

The initial start sequence to execute a single 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 the same session ID as the new `FunctionCommandType` and the status of `ISSUED` and reason of `SUCCEEDED` to notify the Service Consumer it has received the new command.

The Command Start Sequence is shown in Figure 9. This pattern will be repeated each time a new command is requested. After the Command Start Sequence, the sequence can take different paths depending on the actual execution of the command. Some possible paths are detailed in the following sections, but they do not enumerate all of the possible execution paths. Other paths (e.g., an objective failing) will follow a similiar pattern to other failures; all are required to follow the state diagram shown in Figure 4 and eventually end with the Command Cleanup Sequence (as shown in Figure 15).

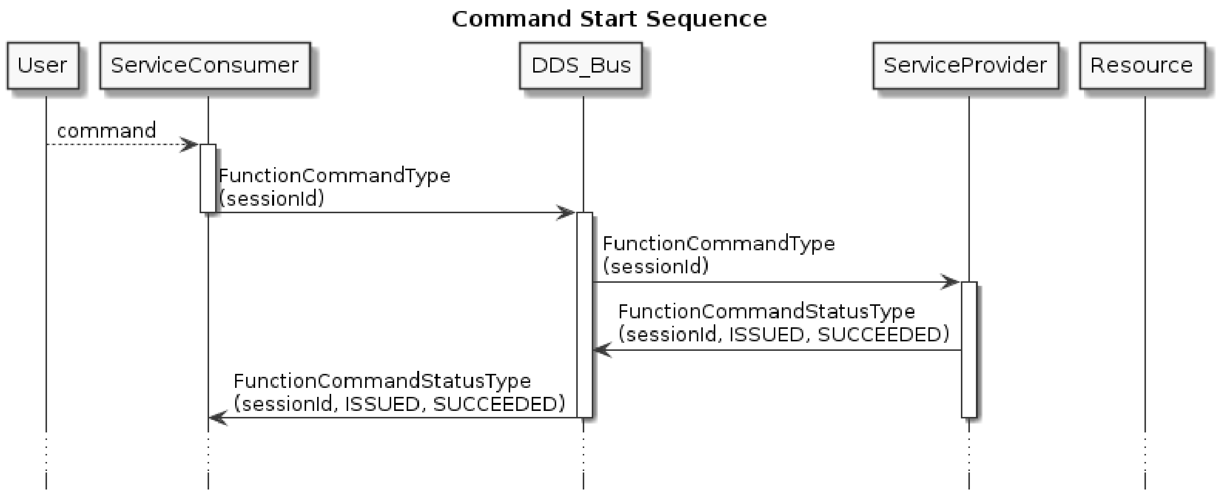
Command Start Sequence
User
ServiceConsumer
DDS_Bus
ServiceProvider
Resource
command
FunctionCommandType (sessionId)
FunctionCommandType (sessionId)
FunctionCommandStatusType (sessionId, ISSUED, SUCCEEDED)
FunctionCommandStatusType (sessionId, ISSUED, SUCCEEDED)

Figure 9: The sequence diagram for the start of a command execution.

4.1.4.1 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 `EXECUTED` 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 defintions of the command-specific attributes in the execution status report.

The command execution sequence is shown in Figure 10. This sequence holds until the command completes execution.

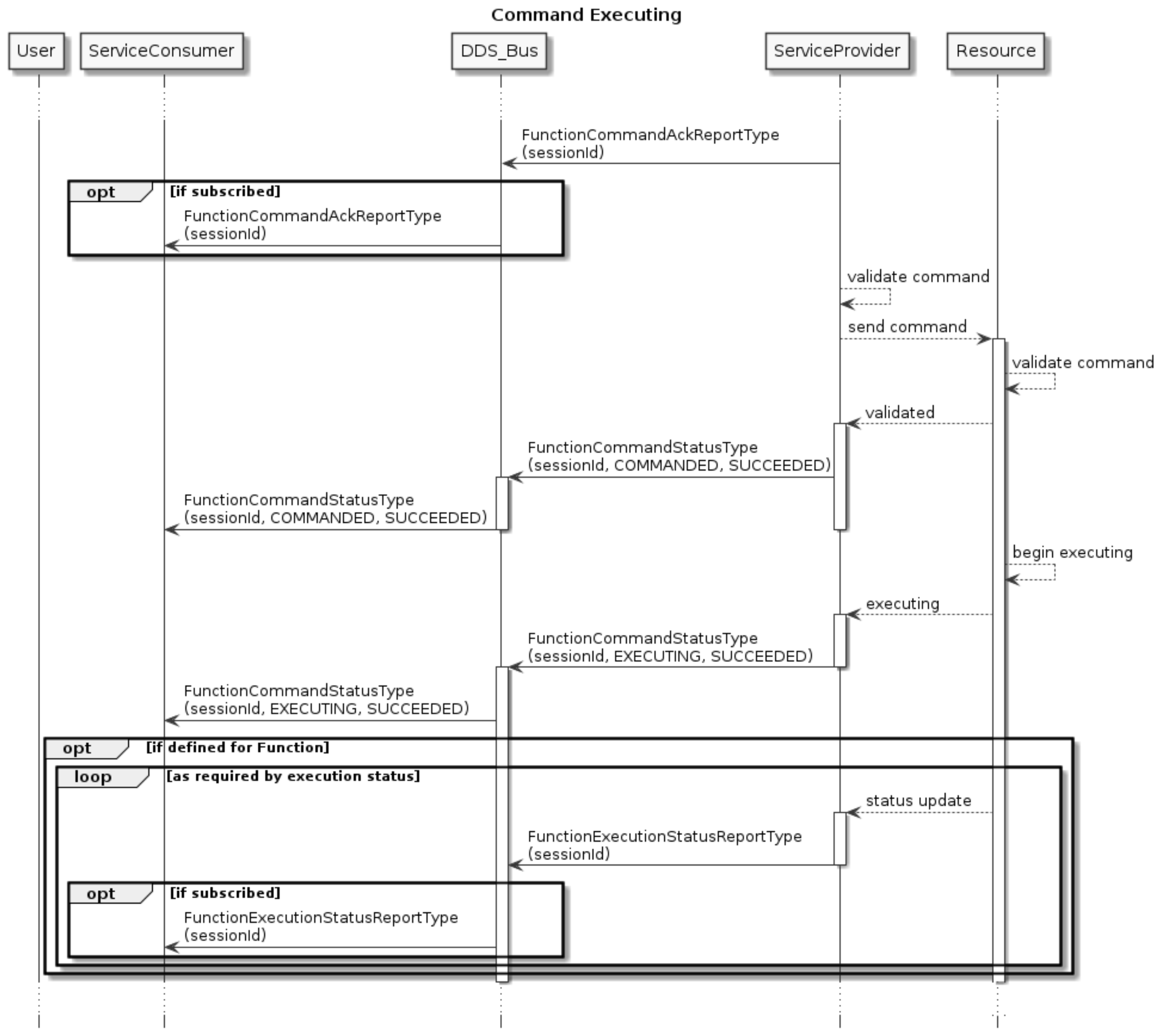
Command Executing
User
ServiceConsumer
DDS_Bus
ServiceProvider
Resource
FunctionCommandAckReportType (sessionId)
opt [if subscribed]
FunctionCommandAckReportType (sessionId)
validate command
send command
validate command
validated
FunctionCommandStatusType (sessionId, COMMANDED, SUCCEEDED)
FunctionCommandStatusType (sessionId, COMMANDED, SUCCEEDED)
begin executing
executing
FunctionCommandStatusType (sessionId, EXECUTING, SUCCEEDED)
FunctionCommandStatusType (sessionId, EXECUTING, SUCCEEDED)
opt [if defined for Function]
loop [as required by execution status]
status update
FunctionExecutionStatusReportType (sessionId)
opt [if subscribed]
FunctionExecutionStatusReportType (sessionId)

Figure 10: The 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.

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

The Command Execution Success sequence is shown in Figure 11.

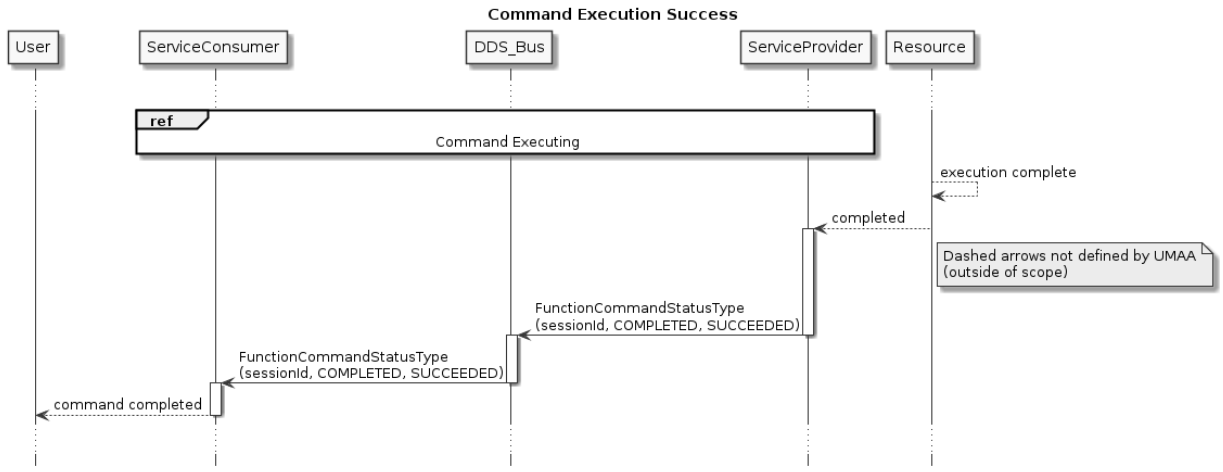
Command Execution Success
User
ServiceConsumer
DDS_Bus
ServiceProvider
Resource
ref
Command Executing
execution complete
completed
Dashed arrows not defined by UMAA (outside of scope)
FunctionCommandStatusType (sessionId, COMPLETED, SUCCEEDED)
FunctionCommandStatusType (sessionId, COMPLETED, SUCCEEDED)
command completed

Figure 11: The sequence diagram for a command that completes successfully.

4.1.4.3 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).

The following figures provide examples of cases where a command has failed.

In the first example, the backing Resource has failed and the Service Provider in 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 12.

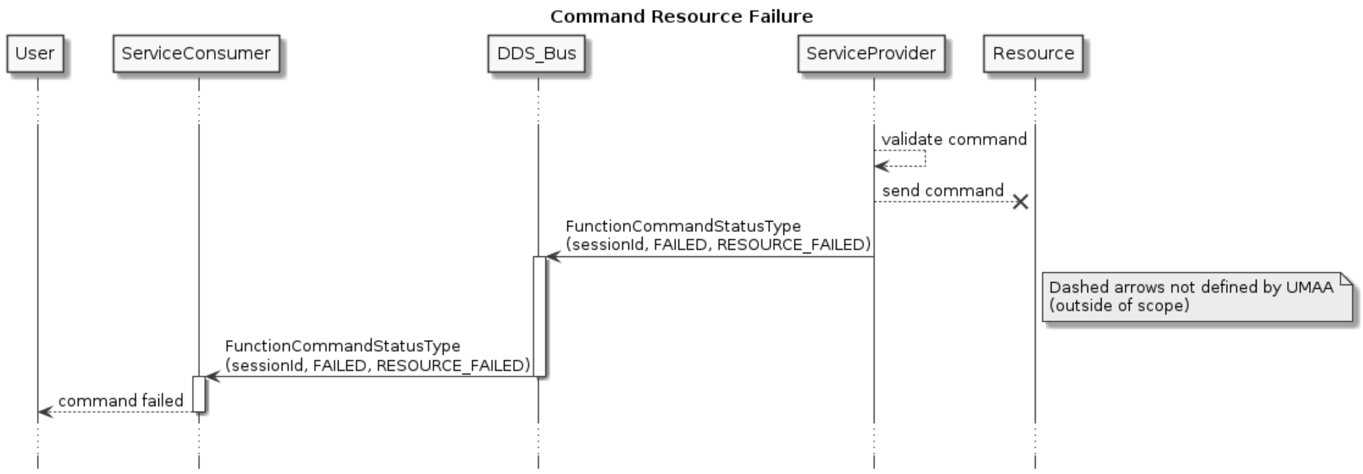
Command Resource Failure
User
ServiceConsumer
DDS_Bus
ServiceProvider
Resource
validate command
send command
FunctionCommandStatusType (sessionId, FAILED, RESOURCE_FAILED)
Dashed arrows not defined by UMAA (outside of scope)
FunctionCommandStatusType (sessionId, FAILED, RESOURCE_FAILED)
command failed

Figure 12: The sequence diagram for a command that fails due to Resource failure.

In the second example, the Resource takes too long to response, 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 13.

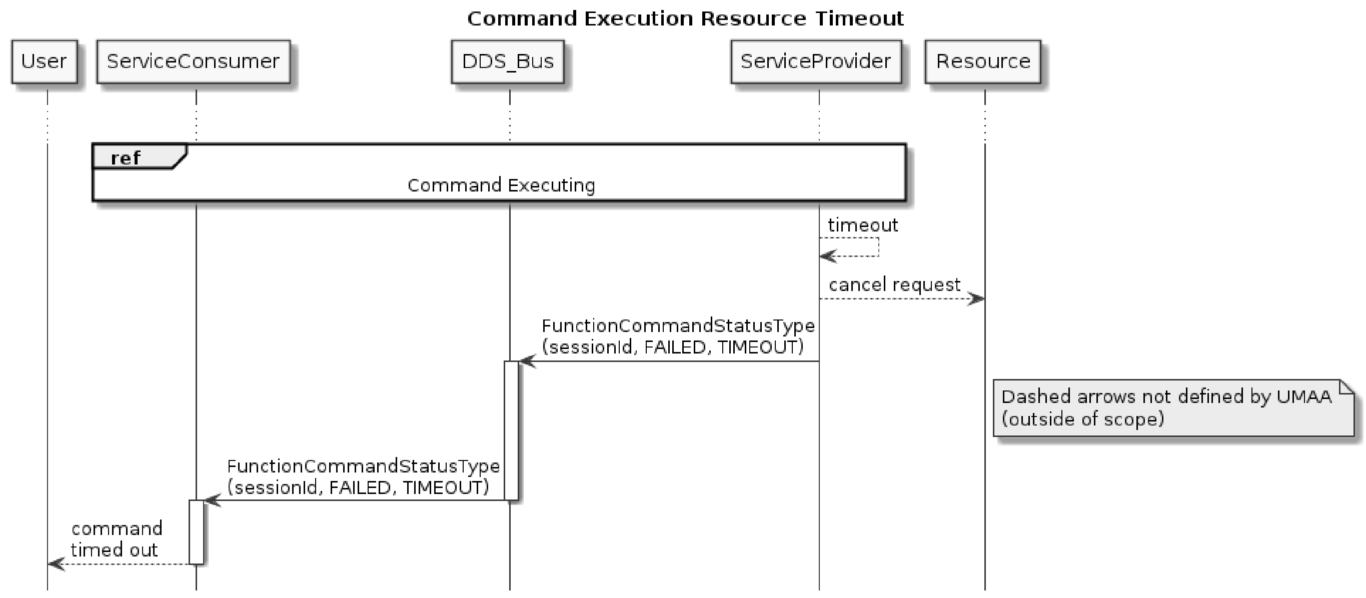
Command Execution Resource Timeout
User
ServiceConsumer
DDS_Bus
ServiceProvider
Resource
ref
Command Executing
timeout
cancel request
FunctionCommandStatusType (sessionId, FAILED, TIMEOUT)
Dashed arrows not defined by UMAA (outside of scope)
FunctionCommandStatusType (sessionId, FAILED, TIMEOUT)
command timed out

Figure 13: The 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.

4.1.4.4 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 the existing `FunctionCommandType` from the DDS bus before the execution is complete. When notified of the command disposal, 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` and then dispose the `FunctionCommandStatusType` and `FunctionCommandAckReportType` and, if defined for the Function service, the `FunctionExecutionStatusReportType` from the bus. This is shown in Figure 14. If the command cannot be canceled the Service Provider can continue to update the command status until the execution is completed, reporting `FunctionCommandStatusType` with a status of `COMPLETED` and a reason of `SUCCEEDED`, and then dispose the `FunctionCommandStatusType` and `FunctionCommandAckReportType` and, if defined for the Function service, the `FunctionExecutionStatusReportType` from the DDS bus.

There is no new unique 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.

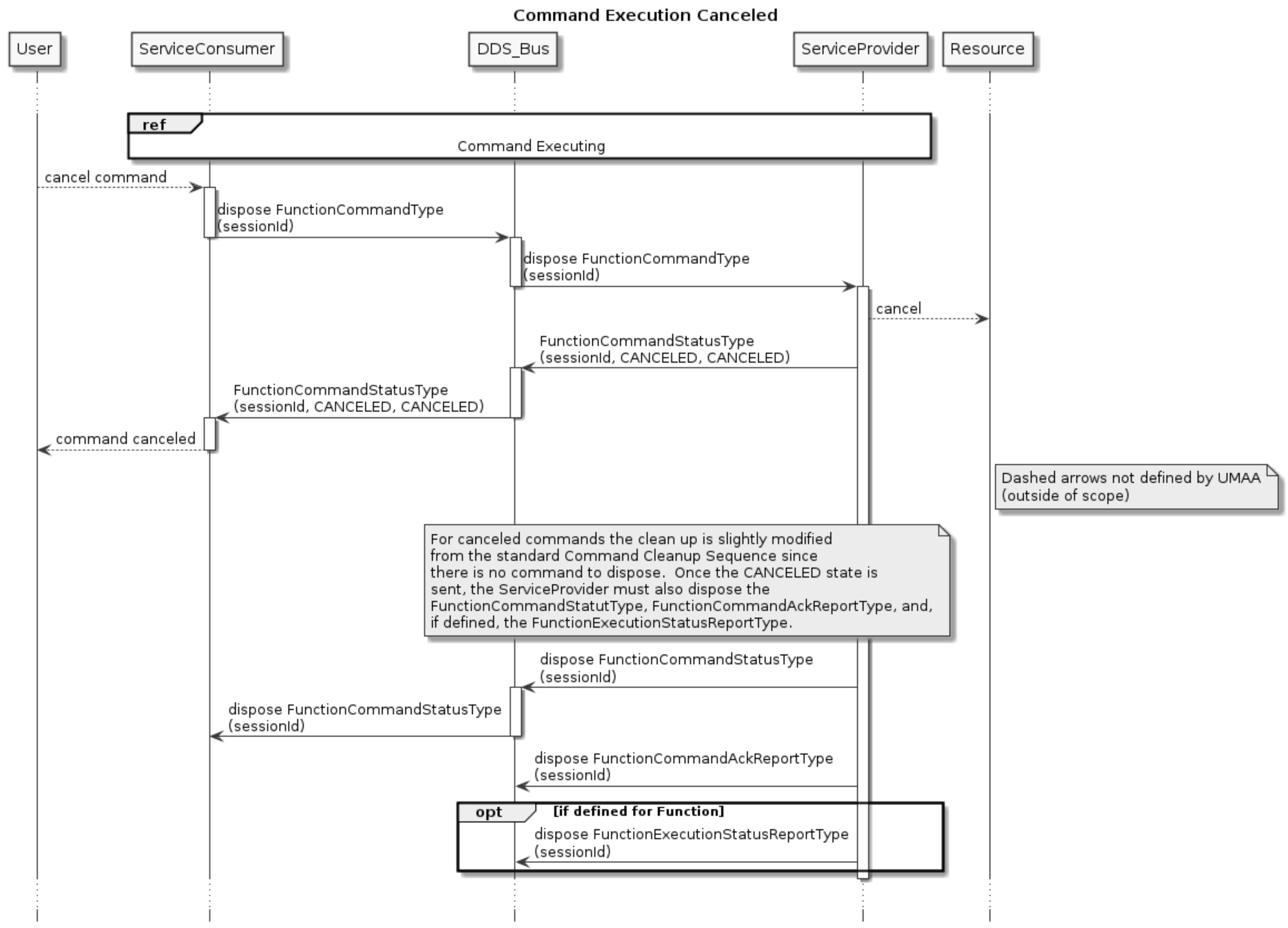
Command Execution Canceled
User
ServiceConsumer
DDS_Bus
ServiceProvider
Resource
ref
Command Executing
cancel command
dispose FunctionCommandType (sessionId)
dispose FunctionCommandType (sessionId)
cancel
FunctionCommandStatusType (sessionId, CANCELED, CANCELED)
FunctionCommandStatusType (sessionId, CANCELED, CANCELED)
command canceled
Dashed arrows not defined by UMAA (outside of scope)
For canceled commands the clean up is slightly modified from the standard Command Cleanup Sequence since there is no command to dispose. Once the CANCELED state is sent, the ServiceProvider must also dispose the FunctionCommandStatutType, FunctionCommandAckReportType, and, if defined, the FunctionExecutionStatusReportType.
dispose FunctionCommandStatusType (sessionId)
dispose FunctionCommandStatusType (sessionId)
dispose FunctionCommandAckReportType (sessionId)
opt [if defined for Function]
dispose FunctionExecutionStatusReportType (sessionId)

Figure 14: The sequence diagram for a command that is canceled by the Service Consumer before the Service Provider is able to complete it.

4.1.5 Command Cleanup

The Service Consumer and Service Provider are responsible for disposing corresponding data 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 15.

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

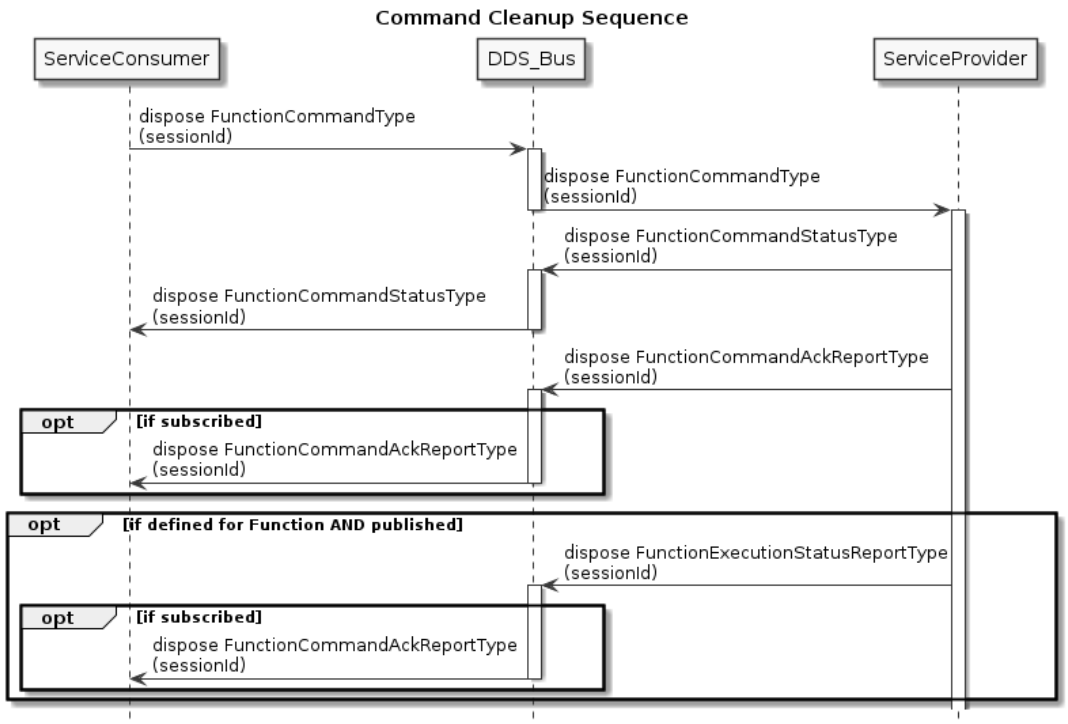
Command Cleanup Sequence
ServiceConsumer
DDS_Bus
ServiceProvider
dispose FunctionCommandType (sessionId)
dispose FunctionCommandType (sessionId)
dispose FunctionCommandStatusType (sessionId)
dispose FunctionCommandStatusType (sessionId)
dispose FunctionCommandAckReportType (sessionId)
opt [if subscribed]
dispose FunctionCommandAckReportType (sessionId)
opt [if defined for Function AND published]
dispose FunctionExecutionStatusReportType (sessionId)
opt [if subscribed]
dispose FunctionCommandAckReportType (sessionId)

Figure 15: The 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.

4.1.6 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 shutdown in any order. The sequence diagram is shown in Figure 16.

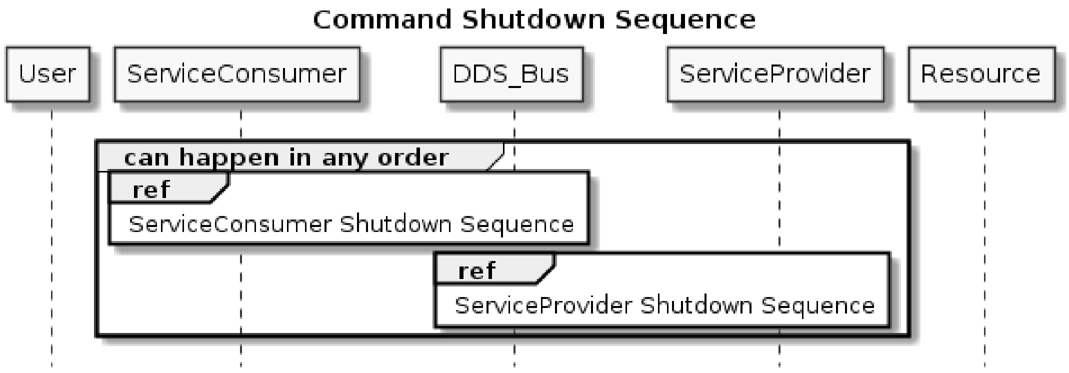
Command Shutdown Sequence
User
ServiceConsumer
DDS_Bus
ServiceProvider
Resource
can happen in any order
ref
ServiceConsumer Shutdown Sequence
ref
ServiceProvider Shutdown Sequence

Figure 16: The sequence diagram for command shutdown.

4.1.6.1 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 17.

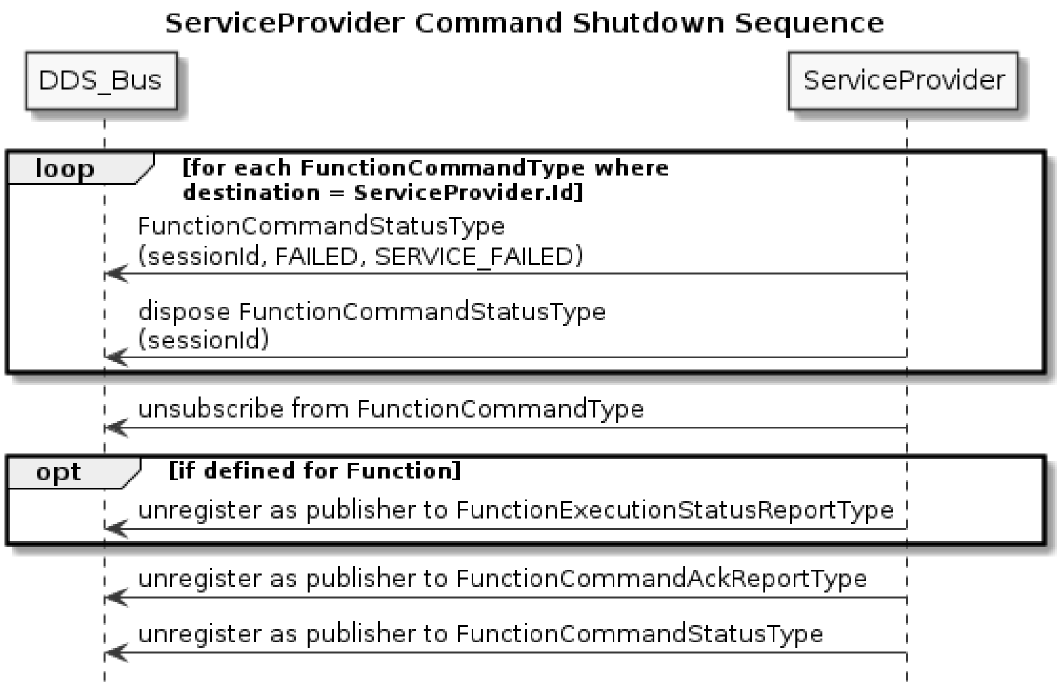
ServiceProvider Command Shutdown Sequence
DDS_Bus
ServiceProvider
loop
[for each FunctionCommandType where destination = ServiceProvider.Id]
FunctionCommandStatusType (sessionId, FAILED, SERVICE_FAILED)
dispose FunctionCommandStatusType (sessionId)
unsubscribe from FunctionCommandType
opt
[if defined for Function]
unregister as publisher to FunctionExecutionStatusReportType
unregister as publisher to FunctionCommandAckReportType
unregister as publisher to FunctionCommandStatusType

Figure 17: The sequence diagram for command shutdown for Service Providers.

4.1.6.2 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 18.

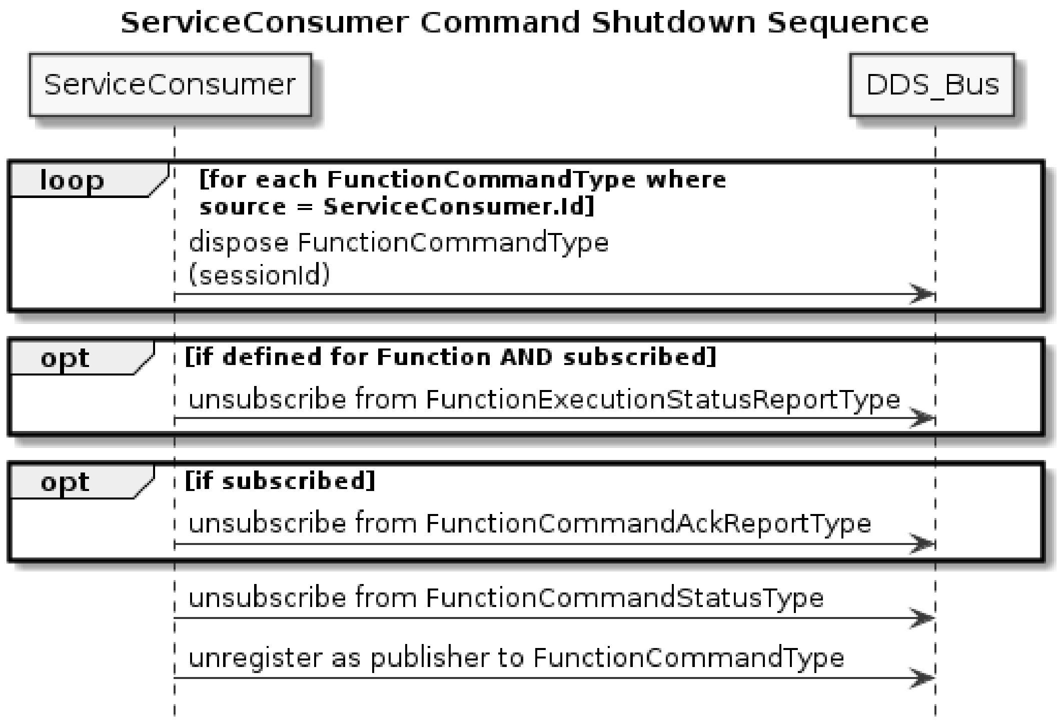
ServiceConsumer Command Shutdown Sequence
ServiceConsumer
DDS_Bus
loop
[for each FunctionCommandType where source = ServiceConsumer.Id]
dispose FunctionCommandType (sessionId)
opt
[if defined for Function AND subscribed]
unsubscribe from FunctionExecutionStatusReportType
opt
[if subscribed]
unsubscribe from FunctionCommandAckReportType
unsubscribe from FunctionCommandStatusType
unregister as publisher to FunctionCommandType

Figure 18: The sequence diagram for command shutdown for Service Consumers.

4.2 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.

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. Additionally, 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 external Resource). In all implementations, however, UMAA-defined exchanges with the DDS bus between the Service Consumer and Service Provider must happen in the order shown within the sequence diagrams.

4.2.1 Request/Reply without Query Data

In the case where there is no specific query data (i.e., the service is always just providing the current data to the bus) the sequence of exchanges is show in Figure 19.

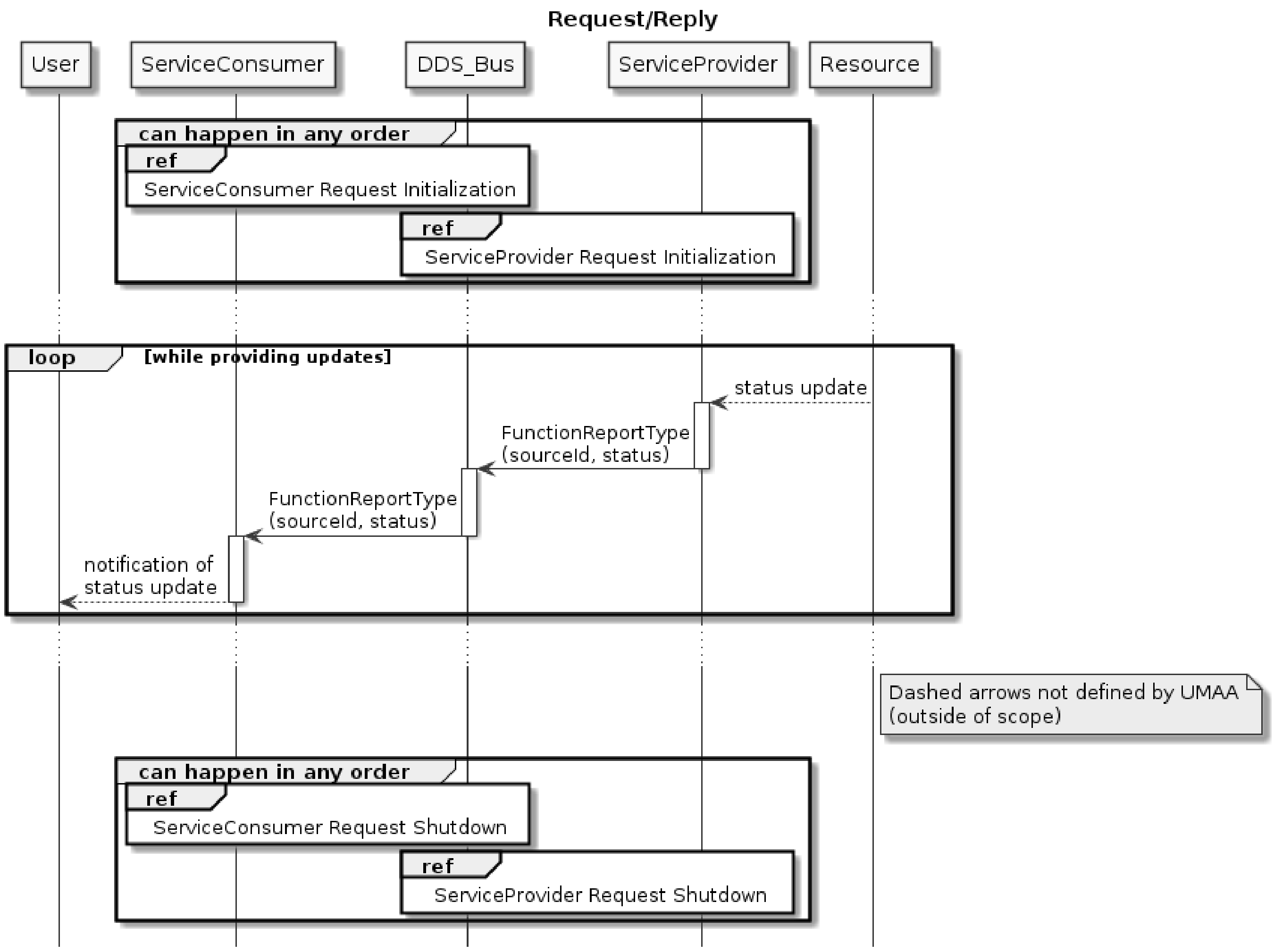
Request/Reply
User
ServiceConsumer
DDS_Bus
ServiceProvider
Resource
can happen in any order
ref
ServiceConsumer Request Initialization
ref
ServiceProvider Request Initialization
loop [while providing updates]
status update
FunctionReportType (sourceId, status)
FunctionReportType (sourceId, status)
notification of status update
Dashed arrows not defined by UMAA (outside of scope)
can happen in any order
ref
ServiceConsumer Request Shutdown
ref
ServiceProvider Request Shutdown

Figure 19: The sequence diagram for a request/reply for report data that does not require any specific query data.

4.2.1.1 Service Provider Startup Sequence

The Service Provider registers as a publisher of `FunctionReportTypes` 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 20.

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.

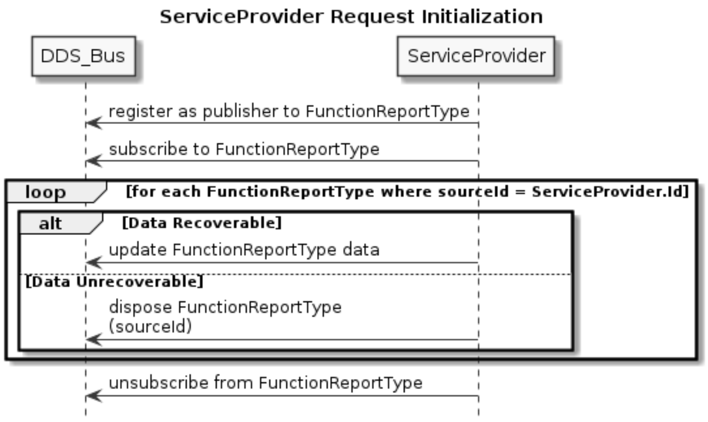
ServiceProvider Request Initialization
DDS_Bus
ServiceProvider
register as publisher to FunctionReportType
subscribe to FunctionReportType
loop
[for each FunctionReportType where sourceId = ServiceProvider.Id]
alt
[Data Recoverable]
update FunctionReportType data
[Data Unrecoverable]
dispose FunctionReportType (sourceId)
unsubscribe from FunctionReportType

Figure 20: The sequence diagram for initialization of a Service Provider to provide FunctionReportTypes.

4.2.1.2 Service Consumer Startup Sequence

The Service Consumer subscribes to the `FunctionReportType` to signal an outstanding request for updates. This is shown in Figure 21.

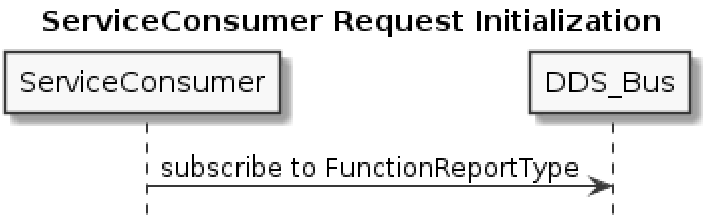
ServiceConsumer Request Initialization
ServiceConsumer
DDS_Bus
subscribe to FunctionReportType

Figure 21: The sequence diagram for initialization of a Service Consumer to request FunctionReportTypes.

4.2.1.3 Service Provider Shutdown

To no longer provide `FunctionReportTypes`, the Service Provider disposes the `FunctionReportType` and unregisters as a publisher of the data as shown in Figure 22.

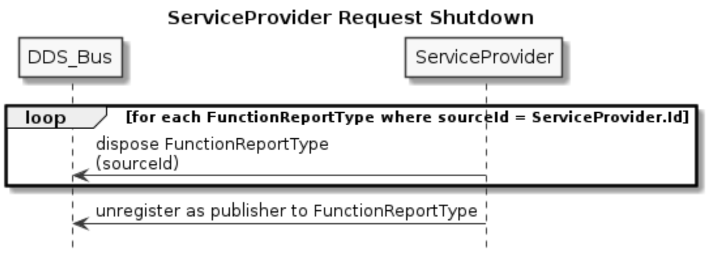
ServiceProvider Request Shutdown
DDS_Bus
ServiceProvider
loop
[for each FunctionReportType where sourceId = ServiceProvider.Id]
dispose FunctionReportType (sourceId)
unregister as publisher to FunctionReportType

Figure 22: The sequence diagram for shutdown of a Service Provider.

4.2.1.4 Service Consumer Shutdown

To no longer request `FunctionReportTypes`, the Service Consumer unsubscribes from `FunctionReportType` as shown in Figure 23.

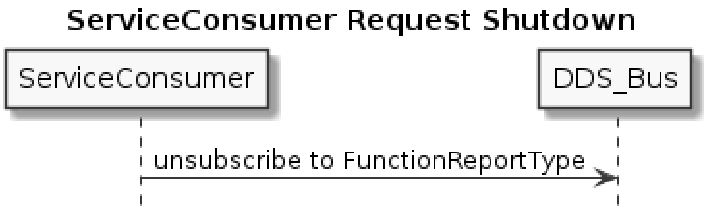
ServiceConsumer Request Shutdown
ServiceConsumer
DDS_Bus
unsubscribe to FunctionReportType

Figure 23: The sequence diagram for shutdown of a Service Consumer.

4.2.2 Request/Reply with Query Data

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

5 Support Operations (SO) Services and Interfaces

5.1 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

The following operations are all handled directly by DDS. They are marked in the operations tables with a ⊕.

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.

5.1.1 HealthReporter

The purpose of this service is to provide health details and summary.

Table 6: HealthReporter Operations

Service Requests (Inputs)	Service Responses (Outputs)
setHealthReporter	reportHealthReporterCommandStatus
queryHealthReporterCommand⊕	reportHealthReporterCommand
cancelHealthReporterCommand⊕	reportHealthReporterCancelCommandStatus⊕
setResetReminder	reportResetReminderCommandStatus
queryResetReminderCommand⊕	reportResetReminderCommand
cancelResetReminderCommand⊕	reportResetReminderCancelCommandStatus⊕
queryHealthDetails⊕	reportHealthDetails
queryHealthSummary⊕	reportHealthSummary
queryHourMeter⊕	reportHourMeter
queryReminder⊕	reportReminder
queryReminderSummary⊕	reportReminderSummary

See Section 5.1 for an explanation of the inputs and outputs marked with a ⊕.

5.1.1.1 reportHealthDetails

Description: This operation is used to report the most recent health detail status for each system or subsystem.

Namespace: UMAA::SO::HealthReporter

Topic: HealthDetailsReport

Data Type: HealthDetailsReportType

Table 7: HealthDetailsReportType Message Definition

Attribute Name	Attribute Type	Attribute Description
Additional fields included from UMAA::UMAAStatus		
healthDetails	sequence<HealthDetailsStatusType>	A list of health detail of the service
resourceID*	NumericGUID	Unique Identifier of the resource

5.1.1.2 reportHealthReporterCommand

Description: This operation is used to report the current built-in-test command.

Namespace: UMAA::SO::HealthReporter

Topic: HealthReporterCommandReport

Data Type: HealthReporterCommandReportType

Table 8: HealthReporterCommandReportType Message Definition

Attribute Name	Attribute Type	Attribute Description
Additional fields included from UMAA::UMAAStatus		
builtInTest	BuiltInTestStatusType_All	builtInTest is used to run a built-in test to an unmanned vehicle and its subsystems.

5.1.1.3 reportHealthReporterCommandStatus

Description: This operation is used to report the status of the built-in-test command.

Namespace: UMAA::SO::HealthReporter

Topic: HealthReporterCommandStatus

Data Type: HealthReporterCommandStatusType

Table 9: HealthReporterCommandStatusType Message Definition

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

5.1.1.4 reportHealthSummary

Description: This operation is used to report the health report summary of the system or subsystem.

Namespace: UMAA::SO::HealthSummaryStatus

Topic: HealthSummaryReport

Data Type: HealthSummaryReportType

Table 10: HealthSummaryReportType Message Definition

Attribute Name	Attribute Type	Attribute Description
Additional fields included from UMAA::UMAAStatus		
resourceIDs	sequence<NumericGUID>	A list of unique identifier of the resources.
severities	sequence<ErrorConditionEnumType>	The error list reporting for each subsystem. A zero-valued severity code indicates normal operation of the system or subsystem; otherwise the severity code reported for a system or subsystem will be the highest severity code.

5.1.1.5 reportHourMeter

Description: This operation is used to report the cumulative operational (powered-on) minutes of the system or subsystem.

Namespace: UMAA::SO::HealthReporter

Topic: HourMeterStatus

Data Type: HourMeterStatusType

Table 11: HourMeterStatusType Message Definition

Attribute Name	Attribute Type	Attribute Description
Additional fields included from UMAA::UMAAStatus		
elapsedMin	Duration_Hours	The operational (powered-on) minutes for each system or subsystem

5.1.1.6 reportReminder

Description: This operation is used to report the status and configuration for the maintenance reminder.

Namespace: UMAA::SO::HealthReporter

Topic: ReminderStatus

Data Type: ReminderStatusType

Table 12: ReminderStatusType Message Definition

Attribute Name	Attribute Type	Attribute Description
Additional fields included from UMAA::UMAAStatus		
descriptor	StringShortDescription	A human-readable string describing the maintenance reminder
elapsedTime	Duration_Hours	Current elapsed subsystem powered-on time since last reset
reminderConfig	BooleanEnumType	Indicate whether the associated maintenance reminder is not configured (set to 1)
reminderID	NumericGUID	Unique identifier for each maintenance reminder
reminderStatus	BooleanEnumType	Status indication whether the reminder is expired (set to 1) or not expired (set to 0)
serviceInterval	Duration_Hours	A reminder timer in minutes

5.1.1.7 reportReminderSummary

Description: This operation is used to report a simple status for each maintenance reminder.

Namespace: UMAA::SO::HealthReporter

Topic: ReminderSummaryStatus

Data Type: ReminderSummaryStatusType

Table 13: ReminderSummaryStatusType Message Definition

Attribute Name	Attribute Type	Attribute Description
Additional fields included from UMAA::UMAAStatus		
reminderExpireds	sequence<BooleanEnumType>	Status indication whether the reminder is expired (set to 1) or not expired (set to 0)
reminderIDs	sequence<NumericGUID>	A list of unique identifiers for maintenance reminder

5.1.1.8 reportResetReminderCommand

Description: This operation is used to current reminder reset command.

Namespace: UMAA::SO::HealthReporter

Topic: ResetReminderCommandReport

Data Type: ResetReminderCommandReportType

Table 14: ResetReminderCommandReportType Message Definition

Attribute Name	Attribute Type	Attribute Description
Additional fields included from UMAA::UMAAStatus		
reminderID	NumericGUID	An unique identifier of the reminder

5.1.1.9 reportResetReminderCommandStatus

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

Namespace: UMAA::SO::HealthReporter

Topic: ResetReminderCommandStatus

Data Type: ResetReminderCommandStatusType

Table 15: ResetReminderCommandStatusType Message Definition

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

5.1.1.10 setHealthReporter

Description: This operation is used to command built-in-test for the system or subsystem.

Namespace: UMAA::SO::HealthReporter

Topic: HealthReporterCommand

Data Type: HealthReporterCommandType

Table 16: HealthReporterCommandType Message Definition

Attribute Name	Attribute Type	Attribute Description
	Additional fields included from UMAA::UMAACommand	
builtInTest	BuiltInTestStatusType_All	builtInTest is used to run a built-in test to an unmanned vehicle and its subsystems.

5.1.1.11 setResetReminder

Description: This operation is used to reset the maintenance reminder.

Namespace: UMAA::SO::HealthReporter

Topic: ResetReminderCommand

Data Type: ResetReminderCommandType

Table 17: ResetReminderCommandType Message Definition

Attribute Name	Attribute Type	Attribute Description
	Additional fields included from UMAA::UMAACommand	
reminderID	NumericGUID	An unique identifier of the reminder

5.1.2 HealthSummaryControl

The purpose of this service is to provide a health status summary report.

Table 18: HealthSummaryControl Operations

Service Requests (Inputs)	Service Responses (Outputs)
setHealthSummary	reportHealthSummaryCommandStatus
cancelHealthSummaryCommand⊕	reportHealthSummaryCancelCommandStatus⊕

See Section 5.1 for an explanation of the inputs and outputs marked with a ⊕.

5.1.2.1 reportHealthSummaryCommandStatus

Description: This operation is used to report the status of the system and/or subsystem health report summary request.

Namespace: UMAA::SO::HealthSummaryControl

Topic: HealthSummaryCommandStatus

Data Type: HealthSummaryCommandStatusType

Table 19: HealthSummaryCommandStatusType Message Definition

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

5.1.2.2 setHealthSummary

Description: This operation is used to request an update to the systems and/or subsystems health report summary.

Namespace: UMAA::SO::HealthSummaryControl

Topic: HealthSummaryCommand

Data Type: HealthSummaryCommandType

Table 20: HealthSummaryCommandType Message Definition

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

5.1.3 HealthSummaryStatus

The purpose of this service is to gather health status and fault of the systems and/or subsystems.

Table 21: HealthSummaryStatus Operations

Service Requests (Inputs)	Service Responses (Outputs)
queryHealthSummary⊕	reportHealthSummary

See Section 5.1 for an explanation of the inputs and outputs marked with a ⊕.

5.1.3.1 reportHealthSummary

Description: This operation is used to report the health report summary of the system or subsystem.

Namespace: UMAA::SO::HealthSummaryStatus

Topic: HealthSummaryReport

Data Type: HealthSummaryReportType

Table 22: HealthSummaryReportType Message Definition

Attribute Name	Attribute Type	Attribute Description
Additional fields included from UMAA::UMAAStatus		
resourceIDs	sequence<NumericGUID>	A list of unique identifier of the resources.
severities	sequence<ErrorConditionEnumType>	The error list reporting for each subsystem. A zero-valued severity code indicates normal operation of the system or subsystem; otherwise the severity code reported for a system or subsystem will be the highest severity code.

5.1.4 ResourceAllocation

This service provides the interfaces necessary for attempting exclusive control of a resource, setting the priority ordering of resource consumers, and retrieving the current configuration and control information for a resource.

Resource Definition
For UMAA, a resource is defined as a logical grouping of UMAA service providers whose execution is mutually exclusive within the group. For example, when executing a GlobalWaypointControl command, a FinControl command cannot execute at the same time. A resource is identified by its resourceId, and the relationship of service providers to their resource is reported by the resource configuration message.

Resource Consumer Definition
Resource consumers are simply consumers of UMAA services whose access is controlled by ResourceAllocation. A resource consumer is identified by the source attribute in the header of the intended command message. This identifier is used by ResourceAllocation when determining the consumer's priority (set using the priority command).

Configuring Resource Consumer Priority
Priority is configured at the resource level, based on resourceId. Priority is modelled using a sequence of resource consumer identifiers (see above), ordered by priority (low to high). Priority may be changed during runtime using the ResourceAllocation priority command, or it may be treated as static configuration. In either case, the priority ordering for all resource consumers in the system must be configured before any resource allocation can take place. Similarly, if a resource consumer attempts a resource allocation and its identifier is not present in the priority configuration, the request must be rejected. Priority is primarily used internally by the ResourceAllocation service, but is published to the DDS bus for persistence. Note that service providers do not rely on this report data to determine whether a command can execute; rather, they use the ResourceAllocation control report to make this determination.

Additions to Flow Control - Resource Consumer
Implementing ResourceAllocation adds additional steps to the flow control for UMAA commands. Before sending a command to a service provider, the consumer must first command ResourceAllocation to attempt to allocate the resource for control. The consumer may only proceed with its service command only if it receives a ResourceAllocation command status of EXECUTING, indicating that it now has control of the resource. This process is detailed below:

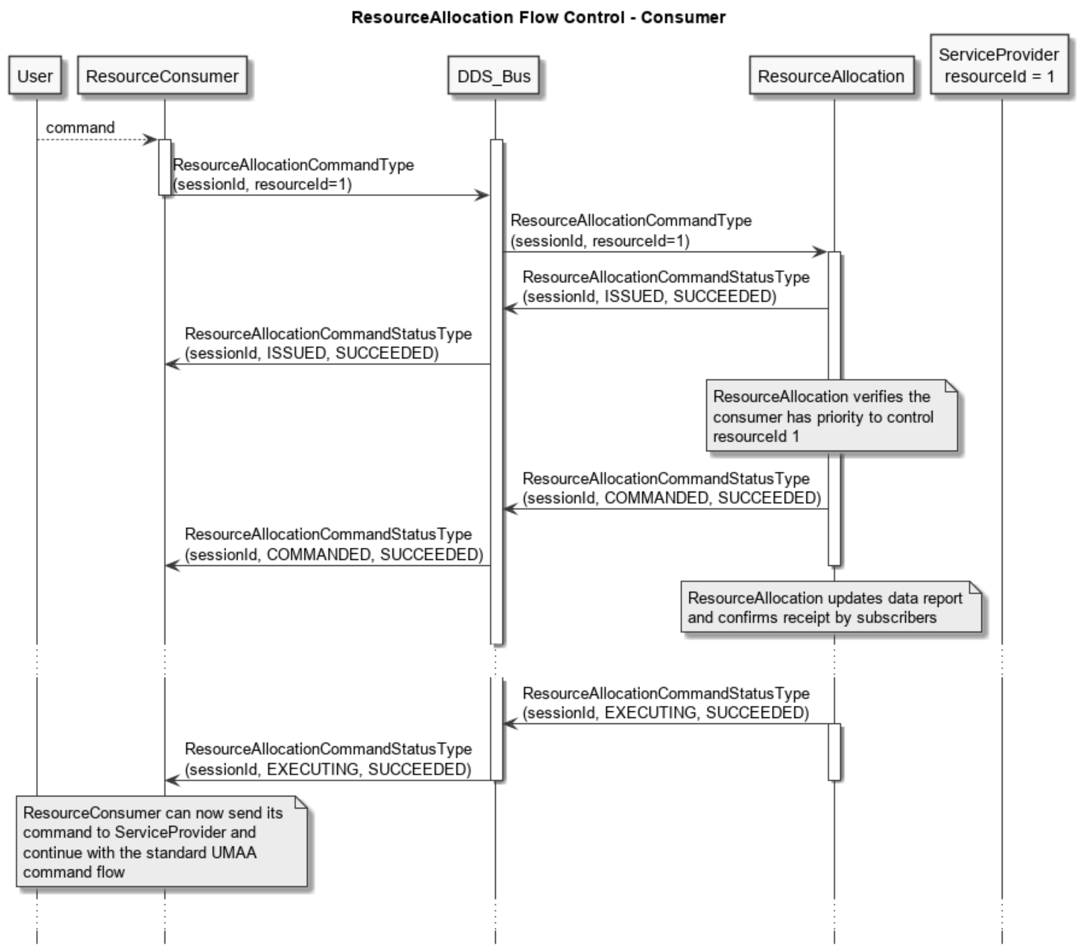
ResourceAllocation Flow Control - Consumer
User
ResourceConsumer
DDS_Bus
ResourceAllocation
ServiceProvider resourceId = 1
command
ResourceAllocationCommandType (sessionId, resourceId=1)
ResourceAllocationCommandType (sessionId, resourceId=1)
ResourceAllocationCommandStatusType (sessionId, ISSUED, SUCCEEDED)
ResourceAllocationCommandStatusType (sessionId, ISSUED, SUCCEEDED)
ResourceAllocation verifies the consumer has priority to control resourceId 1
ResourceAllocationCommandStatusType (sessionId, COMMANDED, SUCCEEDED)
ResourceAllocationCommandStatusType (sessionId, COMMANDED, SUCCEEDED)
ResourceAllocation updates data report and confirms receipt by subscribers
ResourceAllocationCommandStatusType (sessionId, EXECUTING, SUCCEEDED)
ResourceAllocationCommandStatusType (sessionId, EXECUTING, SUCCEEDED)
ResourceConsumer can now send its command to ServiceProvider and continue with the standard UMAA command flow

Figure 24: Sequence Diagram of Resource Consumer Requesting Control of a Resource

Additionally, resource consumers can set a duration when attempting control of a resource. If a duration is provided, the consumer must continue to ask for control within the timeout period so control allocation is not lost. This enables robust recovery for software failures and recovery.

Additions to Flow Control - Service Provider
Service providers must subscribe to the ResourceAllocation report. When a service provider receives a command, it uses this report to determine whether the consumer has control of the resource by checking the source of the incoming command message against the consumer currently in control. If the identifiers do not match, or the end time of the control session has elapsed, then the request must be rejected. This process is detailed below:

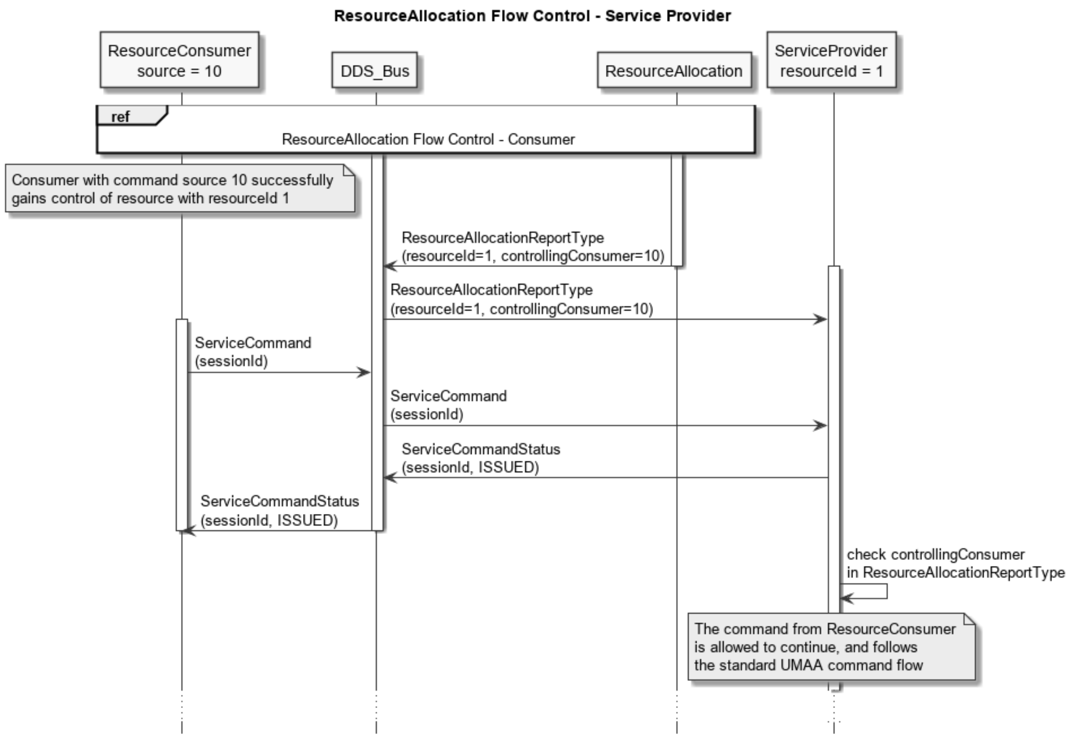
ResourceAllocation Flow Control - Service Provider
ResourceConsumer source = 10
DDS_Bus
ResourceAllocation
ServiceProvider resourceId = 1
ref
ResourceAllocation Flow Control - Consumer
Consumer with command source 10 successfully gains control of resource with resourceId 1
ResourceAllocationReportType (resourceId=1, controllingConsumer=10)
ResourceAllocationReportType (resourceId=1, controllingConsumer=10)
ServiceCommand (sessionId)
ServiceCommand (sessionId)
ServiceCommandStatus (sessionId, ISSUED)
ServiceCommandStatus (sessionId, ISSUED)
check controllingConsumer in ResourceAllocationReportType
The command from ResourceConsumer is allowed to continue, and follows the standard UMAA command flow

Figure 25: Sequence Diagram of Service Provider Verifying a Command using ResourceAllocation

Additions to Flow Control - Nested Service Provider

The ResourceAllocation command flow control has an additional step for nested services. A nested service provider is commanded by other service providers (and potentially by service consumers as well). For example, in the case where a GlobalWaypoint service implementation can achieve its functionality by sending a series of GlobalVector commands, the GlobalVector service provider is a nested service. When a nested service provider receives a command, it checks the configuration report to determine if the command was sent from another service in the same resource (using the command's source header field). If it was, the command can continue as normal, since the commanding service would have already followed the flow control above. If the command is received directly from the resource consumer originating the command, then the nested service must perform additional verification that the consumer has control of the resource. This process is detailed below:

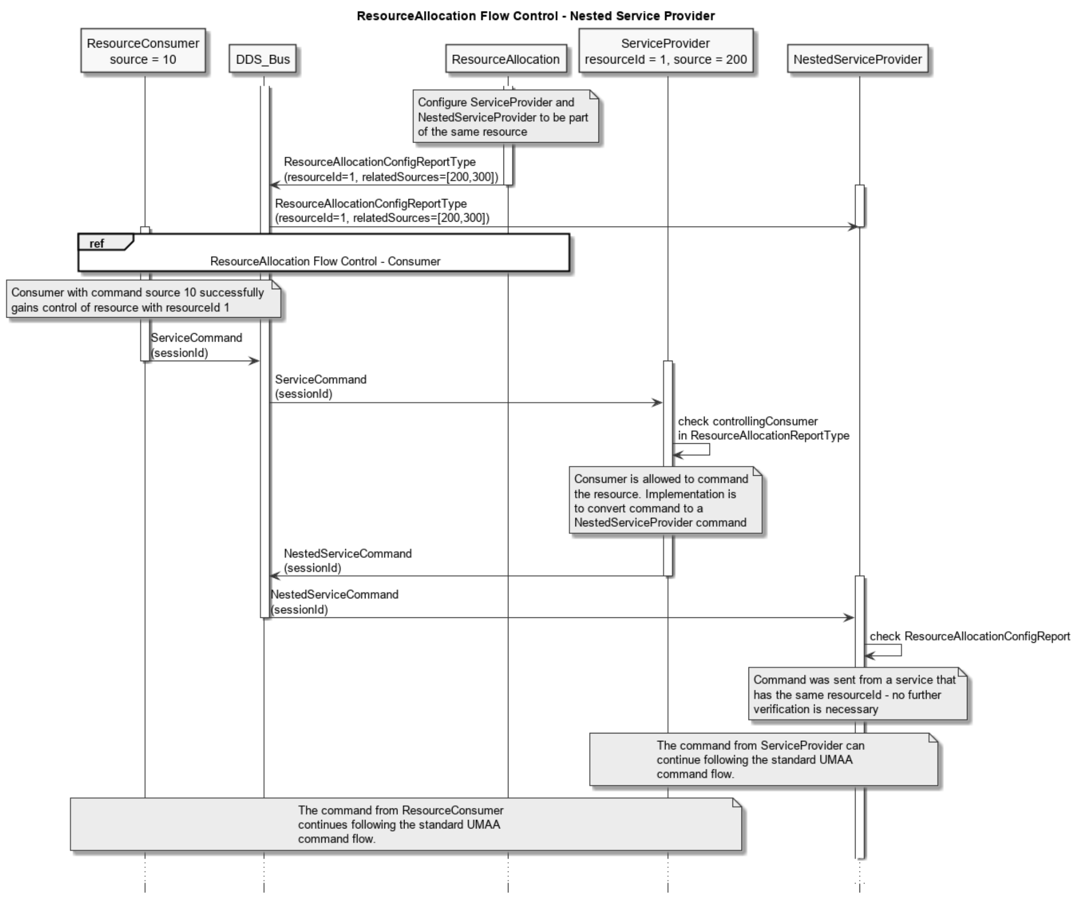
ResourceAllocation Flow Control - Nested Service Provider
ResourceConsumer source = 10
DDS_Bus
ResourceAllocation
ServiceProvider resourceId = 1, source = 200
NestedServiceProvider
Configure ServiceProvider and NestedServiceProvider to be part of the same resource
ResourceAllocationConfigReportType (resourceId=1, relatedSources=[200,300])
ResourceAllocationConfigReportType (resourceId=1, relatedSources=[200,300])
ref
ResourceAllocation Flow Control - Consumer
Consumer with command source 10 successfully gains control of resource with resourceId 1
ServiceCommand (sessionId)
ServiceCommand (sessionId)
check controllingConsumer in ResourceAllocationReportType
Consumer is allowed to command the resource. Implementation is to convert command to a NestedServiceProvider command
NestedServiceCommand (sessionId)
NestedServiceCommand (sessionId)
check ResourceAllocationConfigReport
Command was sent from a service that has the same resourceId - no further verification is necessary
The command from ServiceProvider can continue following the standard UMAA command flow.
The command from ResourceConsumer continues following the standard UMAA command flow.

Figure 26: Sequence Diagram of Nested Service Provider Verifying a Command using ResourceAllocation

Ensuring Strong Data Consistency
To avoid report data race conditions, the ResourceAllocation service implementation must take additional steps to ensure strong consistency of ResourceAllocationReport data. Before allowing control of a resource, the ResourceAllocation service must ensure that all subscribers of ResourceAllocationReport have received the most recent sample. This means that service providers will have the data they need to verify the service consumer's command before it is sent. This consistency can be achieved in a number of ways, such as

- Using standard DDS capabilities by configuring the ResourceAllocation report DataWriter to be reliable and using wait_for_acknowledgements()
- Using implementation-specific DDS extensions
- Implementing multiple ResourceAllocation services and using a combination of flow control and other methods above to communicate across a proxy (see examples online)

Avoiding Pitfalls
The ResourceAllocation service exists to deconflict requests coming from multiple service consumers, which would otherwise cause a fight over a particular resource. There is no mechanism in place to prevent a single consumer who has gained control of a resource to issue concurrent commands to multiple service providers within the resource. Doing so is bad engineering practice and should be avoided to ensure command determinism.

Table 23: ResourceAllocation Operations

Service Requests (Inputs)	**Service Responses (Outputs)**
setResourceAllocation	reportResourceAllocationCommandStatus
queryResourceAllocationCommand⊕	reportResourceAllocationCommand
cancelResourceAllocationCommand⊕	reportResourceAllocationCancelCommandStatus⊕

Service Requests (Inputs)	Service Responses (Outputs)
queryResourceAllocation⊕	reportResourceAllocation
setResourceAllocationPriority	reportResourceAllocationPriorityCommandStatus
queryResourceAllocationPriorityCommand⊕	reportResourceAllocationPriorityCommand
cancelResourceAllocationPriorityCommand⊕	reportResourceAllocationPriorityCancelCommandStatus⊕
queryResourceAllocationPriority⊕	reportResourceAllocationPriority
queryResourceAllocationConfig⊕	reportResourceAllocationConfig

See Section 5.1 for an explanation of the inputs and outputs marked with a ⊕.

5.1.4.1 reportResourceAllocation

Description: This operation is used to report the current resources and what consumer currently owns each resource.

Namespace: UMAA::SO::ResourceAllocation

Topic: ResourceAllocationReport

Data Type: ResourceAllocationReportType

Table 24: ResourceAllocationReportType Message Definition

Attribute Name	Attribute Type	Attribute Description
Additional fields included from UMAA::UMAAStatus		
controlInfo→controlInfoSetID	LargeSet<ResourceAllocationControlInfo>	A list of the control information for every resource defined by ResourceAllocation. This attribute is implemented as a large set, see this section for an explanation. The associated topic is UMAA::SO::ResourceAllocation::ResourceAllocationReport_-ControlinfoSet.

5.1.4.2 reportResourceAllocationCommand

Description: This operation is used to report the current command.

Namespace: UMAA::SO::ResourceAllocation

Topic: ResourceAllocationCommandReport

Data Type: ResourceAllocationCommandReportType

Table 25: ResourceAllocationCommandReportType Message Definition

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

Attribute Name	Attribute Type	Attribute Description
duration†	Duration_Seconds	Specifies the end of the valid time period for the control session. Once the control session ends, the resource will become available for other requesters to control. If this field is empty, then the duration is assumed to be infinite.
resourceId*	NumericGUID	The identifier of the resource to attempt control of

5.1.4.3 reportResourceAllocationCommandStatus

Description: This operation is used to report the status of the current resource allocation command

Namespace: UMAA::SO::ResourceAllocation

Topic: ResourceAllocationCommandStatus

Data Type: ResourceAllocationCommandStatusType

Table 26: ResourceAllocationCommandStatusType Message Definition

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

5.1.4.4 reportResourceAllocationConfig

Description: This operation is used to report all service provider identifiers that are sharing a single resource.

Namespace: UMAA::SO::ResourceAllocation

Topic: ResourceAllocationConfigReport

Data Type: ResourceAllocationConfigReportType

Table 27: ResourceAllocationConfigReportType Message Definition

Attribute Name	Attribute Type	Attribute Description
Additional fields included from UMAA::UMAAStatus		
resources→resourcesSetID	LargeSet<ResourceAllocationDefinitionType>	The configuration of each resource. This attribute is implemented as a large set, see this section for an explanation. The associated topic is UMAA::SO::ResourceAllocation::ResourceAllocationConfigReport_ResourcesSet.

5.1.4.5 reportResourceAllocationPriority

Description: This operation is used to report the current priority ordering of resource consumers.

Namespace: UMAA::SO::ResourceAllocation

Topic: ResourceAllocationPriorityReport

Data Type: ResourceAllocationPriorityReportType

Table 28: ResourceAllocationPriorityReportType Message Definition

Attribute Name	Attribute Type	Attribute Description
Additional fields included from UMAA::UMAAStatus		
priorities→prioritiesSetID	LargeSet<ResourceAllocationPriorityInfo>	The priority ordering of resource consumers for each resource defined by ResourceAllocation. This attribute is implemented as a large set, see this section for an explanation. The associated topic is UMAA::SO::ResourceAllocation::ResourceAllocationPriorityReport PrioritiesSet.

5.1.4.6 reportResourceAllocationPriorityCommand

Description: This operation is used to report the current command.

Namespace: UMAA::SO::ResourceAllocation

Topic: ResourceAllocationPriorityCommandReport

Data Type: ResourceAllocationPriorityCommandReportType

Table 29: ResourceAllocationPriorityCommandReportType Message Definition

Attribute Name	Attribute Type	Attribute Description
Additional fields included from UMAA::UMAAStatus		
priorities	sequence<NumericGUID>	The priority-ordered (low to high) sequence of resource consumer source identifiers
resourceId*	NumericGUID	The identifier of the resource the priority is being set for.

5.1.4.7 reportResourceAllocationPriorityCommandStatus

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

Namespace: UMAA::SO::ResourceAllocation

Topic: ResourceAllocationPriorityCommandStatus

Data Type: ResourceAllocationPriorityCommandStatusType

Table 30: ResourceAllocationPriorityCommandStatusType Message Definition

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

5.1.4.8 setResourceAllocation

Description: This operation is used to set the current resource allocation command. The source attribute in the header of this command is used to identify the consumer that is requesting the resource to be allocated

Namespace: UMAA::SO::ResourceAllocation

Topic: ResourceAllocationCommand

Data Type: ResourceAllocationCommandType

Table 31: ResourceAllocationCommandType Message Definition

Attribute Name	Attribute Type	Attribute Description
Additional fields included from UMAA::UMAACommand		
duration†	Duration_Seconds	Specifies the end of the valid time period for the control session. Once the control session ends, the resource will become available for other requesters to control. If this field is empty, then the duration is assumed to be infinite.
resourceId*	NumericGUID	The identifier of the resource to attempt control of

5.1.4.9 setResourceAllocationPriority

Description: This operation is used to set the ordered priority of consumers who will be requesting access. All potential consumers must be on this list to be enabled to request control allocation.

Namespace: UMAA::SO::ResourceAllocation

Topic: ResourceAllocationPriorityCommand

Data Type: ResourceAllocationPriorityCommandType

Table 32: ResourceAllocationPriorityCommandType Message Definition

Attribute Name	Attribute Type	Attribute Description
Additional fields included from UMAA::UMAACommand		
priorities	sequence<NumericGUID>	The priority-ordered (low to high) sequence of resource consumer source identifiers
resourceId*	NumericGUID	The identifier of the resource the priority is being set for.

5.2 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.

5.2.1 UCSMDEInterfaceSet

Namespace: UMAA::UCSMDEInterfaceSet

Description: Defines the common UCSMDE Interface Set Message Fields.

Table 33: UCSMDEInterfaceSet Structure Definition

Attribute Name	Attribute Type	Attribute Description
timeStamp	DateTime	The time at which the data was derived.

5.2.2 UMAACommand

Namespace: UMAA::UMAACommand

Description: Defines the common UMAA Command Message Fields.

Table 34: UMAACommand Structure Definition

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

5.2.3 UMAAStatus

Namespace: UMAA::UMAAStatus

Description: Defines the common UMAA Status Message Fields.

Table 35: UMAAStatus Structure Definition

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

5.2.4 UMAACommandStatusBase

Namespace: UMAA::UMAACommandStatusBase

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

Table 36: UMAACommandStatusBase Structure Definition

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

5.2.5 UMAACommandStatus

Namespace: UMAA::UMAACommandStatus

Description: Defines the common UMAA Command Status Message Fields.

Table 37: 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.

5.2.6 DateTime

Namespace: UMAA::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 38: 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

5.2.7 BuiltInTestStatusType_All

Namespace: UMAA::Common::Measurement::BuiltInTestStatusType_All

Description: Realizes BuiltInTestStatusType: the Selector for the BuiltInTestStatusEnumType.

Table 39: BuiltInTestStatusType_All Structure Definition

Attribute Name	Attribute Type	Attribute Description
domain	sequence<BuiltInTestStatusEnumType>	An EnumerationSet which defines the allowable values for the Selector.
setPoint	BuiltInTestStatusEnumType	An EnumerationSet which specifies the desired value of the Selector.
value	BuiltInTestStatusEnumType	An EnumerationSet which specifies the actual value of the Selector.

5.2.8 HealthDetailsStatusType

Namespace: UMAA::SO::HealthReporter::HealthDetailsStatusType

Description: This structure is used to report the health details status of the unmanned vehicle and/or its subsystems.

Table 40: HealthDetailsStatusType Structure Definition

Attribute Name	Attribute Type	Attribute Description
code	ErrorCodeEnumType	The types of system or subsystems associated with the error report
descriptor	StringShortDescription	A description of the system or subsystem reporting the error
logTime	DateTime	Log time when the error occurs
resourceURN	StringShortDescription	An uniform resource name of the service
severity	ErrorConditionEnumType	The types of error is reporting
detailID*	NumericGUID	Unique Identifier of the health detail of the resource

5.2.9 Quaternion

Namespace: BasicTypes::Quaternion

Description: Defines a four-element vector that can be used to encode any rotation in a 3D coordinate system.

Table 41: Quaternion Structure Definition

Attribute Name	Attribute Type	Attribute Description
a	double	Real number a.
b	double	Real number b.
c	double	Real number c.
d	double	Real number d.

5.2.10 ResourceAllocationControlInfo

Namespace: UMAA::SO::ResourceAllocation::ResourceAllocationControlInfo

Description: This structure is used to define attributes related to the controller of a resource

Table 42: ResourceAllocationControlInfo Structure Definition

Attribute Name	Attribute Type	Attribute Description
controlSession†	ResourceAllocationControlSession	Information on the consumer currently controlling the resource. If empty, this resource is not currently allocated for control.
resourceId*	NumericGUID	The identifier of the resource being controlled.

5.2.11 ResourceAllocationControlSession

Namespace: UMAA::SO::ResourceAllocation::ResourceAllocationControlSession

Description: This structure is used to define attributes related to the current controller of a resource

Table 43: ResourceAllocationControlSession Structure Definition

Attribute Name	Attribute Type	Attribute Description
controllingConsumer	NumericGUID	The source identifier of the resource consumer in control of the resource
endTime†	DateTime	The absolute end time of the consumer's control. After this time is reached, the resource is available to be controlled by another process. If this field is empty, then the duration is assumed to be infinite.

5.2.12 ResourceAllocationDefinitionType

Namespace: UMAA::SO::ResourceAllocation::ResourceAllocationDefinitionType

Description: This structure is used to define the attributes associated with a resource - that is, a collection of related service providers whose functionality cannot be executed simultaneously.

Table 44: ResourceAllocationDefinitionType Structure Definition

Attribute Name	Attribute Type	Attribute Description
relatedSources	sequence<NumericGUID>	The source identifiers of each service that is logically part of this resource. For instance, this resource could represent driving-related services at large. This field would then contain the source of each driving-related service provider active in the system.
resourceId*	NumericGUID	The identifier of the resource.

5.2.13 ResourceAllocationPriorityInfo

Namespace: UMAA::SO::ResourceAllocation::ResourceAllocationPriorityInfo

Description: This structure is used to define the configuration of resource control priority for a particular resource

Table 45: ResourceAllocationPriorityInfo Structure Definition

Attribute Name	Attribute Type	Attribute Description
priorities	sequence<NumericGUID>	The priority-ordered (low to high) sequence of client identifiers
resourceId*	NumericGUID	The identifier of the resource being controlled.

5.3 Enumerations

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

5.3.1 BuiltInTestStatusEnumType

Namespace: UMAA::Common::Enumeration::BuiltInTestStatusEnumType

Description: BuiltInTestStatusEnumTypeLDM is a Realization of BuiltInTestStatusEnumType which is a mutually exclusive set of values that defines the state of a Built-In Test.

Table 46: BuiltInTestStatusEnumType Enumeration

Enumeration Value	Description
BIT_SUSPENDED	The built-in test (BIT) has been suspended.
BIT_FAILED	The built-in test (BIT) has failed.
BIT_PASSED	The built-in test (BIT) has passed.
RUNNING_BIT	The built-in test (BIT) is current executing.
OFF_ABORT	The built-in test (BIT) is off or has been aborted.

5.3.2 CommandStatusReasonEnumType

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

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

Table 47: CommandStatusReasonEnumType Enumeration

Enumeration Value	Description
CANCELED	Indicates a transition to the CANCELED state when the command is canceled successfully.
VALIDATION_FAILED	Indicates a transition to the FAILED state when the command contains missing, out-of-bounds, or otherwise invalid parameters.
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.
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.
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.
INTERRUPTED	Indicates a transition to the FAILED state when the command has been interrupted by a higher priority process.
TIMEOUT	Indicates a transition to the FAILED state when the command is not acknowledged within some defined time bound.
SUCCEEDED	Indicates the conditions to proceed to this state have been met and a normal state transition has occurred.

5.3.3 ErrorCodeEnumType

Namespace: UMAA::Common::MaritimeEnumeration::ErrorCodeEnumType

Description: A mutually exclusive set of values that defines the error codes.

Table 48: ErrorCodeEnumType Enumeration

Enumeration Value	Description
ACTUATOR	Actuator
FILESYS	File system
NONE	None
POWER	Power
PROCESSOR	Processor
RAM	RAM
ROM	ROM
SENSOR	Sensor
SOFTWARE	Software

5.3.4 ErrorConditionEnumType

Namespace: UMAA::Common::MaritimeEnumeration::ErrorConditionEnumType

Description: A mutually exclusive set of values that defines the error condition.

Table 49: ErrorConditionEnumType Enumeration

Enumeration Value	Description
INFO	An error condition is reported, but impact on operation and performance is minimal.
WARN	An error condition is reported and expected to have significant impact on component or device performance.
ERROR	An error condition is reported and expected to seriously compromise use of the reporting component or device.
FAIL	An error condition is reported with severity indicating component or device failure.
NONE	Indicates that no error condition exists.

5.3.5 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 50: CommandStatusEnumType Enumeration

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

5.4 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 51: Type Definitions

Type Name	**Primitive Type**	**Range of Values**	**Description**

BooleanEnumType	boolean	units=N/A
		minInclusive=N/A
		maxInclusive=N/A
		fractionDigits=N/A
		length=N/A
			BooleanEnumTypeLDM is a Realization of BooleanEnumType which is a mutually exclusive set of values that defines the truth values of logical algebra.

DateTimeNanoseconds	long	units=Nanoseconds
		minInclusive=0
		maxInclusive=999999999
		fractionDigits=0
			number of nanoseconds elapsed within the current second.

DateTimeSeconds	longlong	units=Seconds
		minInclusive=0
		maxInclusive=18446744073709500000
		fractionDigits=0
			seconds offset from the standard POSIX (IEEE Std 1003.1-2017) epoch reference point of January 1st, 1970 00:00:00 UTC.

Duration__Hours	double	units=Hour
		minInclusive=0
		maxInclusive=10505
		fractionDigits=3
			Represents a time duration in hours.

Duration__Seconds	double	units=Seconds
		minInclusive=0
		maxInclusive=37817280
		fractionDigits=6
			Represents a time duration in seconds.

NumericGUID	octet[16]	units=N/A
		minInclusive=0
		maxInclusive=(2^128)-1
		fractionDigits=0
			Represents a 128-bit number according to RFC 4122 variant 2

StringLongDescription	string	fractionDigits=N/A
		length=4095
		maxExclusive=N/A
		maxInclusive=N/A
		minExclusive=N/A
		minInclusive=N/A
		units=N/A
			Represents a long format description.

StringShortDescription	string	fractionDigits=N/A
		length=1023
		maxExclusive=N/A
		maxInclusive=N/A
		minExclusive=N/A
		minInclusive=N/A
		units=N/A
			Represents a short format description.

A Appendices

A.1 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
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
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
WMO	World Meteorological Organization